WS-SM: Web Services - Secured Messaging Framework with Pluggable APIs

Kanchana Rajaram, Chitra Babu

WS-SM: Web Services - Secured Messaging Framework with pluggable APIs

Kanchana Rajaram and Chitra Babu

Department of Computer Science and Engineering,
SSN College of Engineering, Anna University,
Chennai - 603110. Tamil Nadu, India
{rkanch,chitra}@ssn.edu.in
ORCID: 0000-0002-2591-2482, 0000-0002-9343-2288

Abstract. Dynamic composition of web services is important in B2B applications where user requirements and business policies change and new services get added to the service registry frequently. In a dynamic composition environment, ensuring the security of messages communicated among the web services becomes challenging since, several attacks are possible on SOAP messages in the public network due to their standardized interfaces. Most of the existing works on web services security provide solutions to ensure basic security features such as confidentiality, integrity, authentication, authorization, and non-repudiation. Few existing works that provide solutions such as schema validation and schema hardening for attacks on web services do not provide attack-specific solutions. The web services security standard and all the existing works have addressed only the security of messages between a client and a single web service but not the security for messages between two services which is quite challenging. Hence, a security framework for secured messaging among web services has been proposed to provide attack-specific solutions. Since new types of web service attacks are evolving over time, the proposed security solutions are implemented as APIs that are pluggable in any server where the web service is deployed. The proposed framework has been tested for compliance with WSI-BP to demonstrate its interoperability and subjected to vulnerability testing which proved its immunity to attacks. The stress testing results revealed that the throughput decreased only by 35% achieving a good trade-off between performance and security.

Keywords: Web services, Composition, SOAP messages, Security, Threats

1 Introduction

Service Oriented Architecture (SOA) [1] represents an open, agile and composable architecture comprised of autonomous, interoperable and potentially reusable services, implemented as web services [2] . The use of web services implies crossing of trust boundaries and the involvement of software with uncertain reliability that leads to mitigation of risks. The security mechanisms that aim to

mitigate risks are applied at four different levels in an SOA, namely user, message, service, and transport. At the service level, the objective is to ensure the availability and correct functioning of a service. Due to standardized interfaces, web services are prone to security attacks, since attackers know more about the format of these interfaces. At the transport level, the goal is to guarantee a seamless and reliable communication between parties.

The web services communicate using SOAP protocol. In General, SOAP message is transmitted over HTTP [3], which can flow freely through a firewall and it cannot protect SOAP messages that transmit in application layer. Secure Socket Layer (SSL)/Transport Layer Security (TLS) [4] is inadequate for protecting SOAP messages since it is designed to operate between two endpoints. Most of the times, SOAP messages might be processed by SOAP intermediaries. If the SOAP intermediaries are compromised, the security provided by SSL/TLS becomes insufficient to ensure the end-to-end integrity and confidentiality of SOAP messages. Hence, SOAP message communication in the application layer is prone to attacks.

Services can be composed in two ways viz., statically or dynamically. In a dynamic composition scenario, the concrete workflow is created during run-time while the abstract workflow is created during design-time. In such scenario, the web service is discovered and composed at run-time and SOAP messages are exchanged between web services. When these messages travel through a public network, they become vulnerable to attacks and hence, it is essential to protect these messages.

Lemos et al. [5] surveyed a variety of techniques and tools for web service compositions as well as provided a systematic analysis of the most representative service composition approaches by evaluating and classifying them against the proposed taxonomy. Mouli et al. [8] presented a systematic review on the studies of web service security and observed that the solutions were mainly proposed using dynamic analysis, closely followed by static analysis. Masood et al.[7] review techniques and tools to improve services security by detecting vulnerabilities and discuss the potential static code analysis techniques to discover these vulnerabilities. Singhal et al. [4] describe various web service threats and the basic security standards that provide solution to few of the web service attacks.The various attacks possible on SOAP messages are tabulated in Table. 1. Jensen et al. [8] surveyed the vulnerabilities in the context of web services. Their methods provide solutions based on accessing WSDL of the domain service for each attack which is a time-consuming process.

In a dynamic composition environment, a composer service in the middleware discovers and invokes services based on the user requirements specified at run-

[1] SOAPAction Spoofing, (2017), `http://www.ws-attacks.org/SOAPAction_Spoofing`

[2] WS-Addressing Spoofing, (2015), `https://www.ws-attacks.org/WS-Addressing_spoofing`

[3] Web Services Addressing,(2004), `http://www.w3.org/Submission/ws-addressing/`

[4] Replay of Messages Attack,(2010) `http://msdn.microsoft.com/en-us/library/ff649371.aspx`

Table 1. Attacks on SOAP Message Communication Among Web Services

Attack	**Description**
SOAPAction Spoofing [1]	Changes the operation in SOAPAction header that leads to execution of unintended operation
WS-Addressing Spoofing [2]	Changes the address of addressing field which leads to flooding of web service [3]
Replay of Messages [4]	Resending the message to the same web service
Message Alteration Attack	Modifying the content of SOAP message
Loss of Confidentiality	Discloses the information to the unauthorised person
XML Injection Attack	Changes the structure of SOAP message
Principal Spoofing	Changes the credentials of SOAP message
Forged Claims	Construct SOAP Message using false credentials
Falsified Messages	Sends fictitious message to receiver

time. The composer communicates with client programs as well as domain web services. It is quite challenging to provide security for the messages exchanged between the composer and domain services since, the composer handles multiple communication channels. None of the existing security standards and solutions address security of messages exchanged between web services during dynamic composition and execution of services. Hence, a novel **W**eb **S**ervices - **S**ecured **M**essaging (**WS-SM**) framework is proposed in this paper, with pluggable APIs for attack-specific security solutions in a dynamic composition scenario.

The rest of the paper is organized as follows, Section 2 describes existing works on web service security. Section 3 explains the architecture and methodology of the proposed system. Testing of the proposed solution and the results are discussed in Section 4 and Section 5 concludes the paper.

2 Existing Work

The defacto standard for providing security on SOAP message communication, WS-Security [9] provides basic security features such as confidentiality, integrity, authentication, authorization, and non-repudiation. Moreover, it detects the presence of a few web service attacks on the communicated SOAP message. However, it does not provide solutions to overcome any of these attacks and does not address some of the attacks like SOAPAction Spoofing, WS-Addressing Spoofing, etc. Alotaibi [10] implemented a secure Web Service using WS-Security specifications such as Signature, Timestamp and Username Token. Thelin et al. [11] proposed a security framework to achieve end-to-end propagation of security credentials throughout the SOAP processing stack.

Hua Yue et al. [12] discussed about the security issues of Web based services on heterogeneous platforms. This approach uses WS-Security to provide security solution with asymmetric cryptography algorithms. Since it is provided as a

wrapper in Axis2 platform, it is difficult to provide the solution for upcoming attacks without changing the functionality of rampart.

Layer 7 SecureSpan and CloudSpan [5] provide APIs against attacks that occur in SOA. Layer 7 XML Gateway implements WS-* standards to ensure integrity between the transactions. Layer 7 SecureSpan XML Firewall provide developers the ability to define and enforce security policy through a simple graphical policy language. The web services Security Programming Application Programming Interfaces (WSS API) [6] is used for securing the SOAP messages. However, the functionality of the application needs to be changed for providing security. All the above solutions consider SOAP message communication between a client program and a web service.

Kishore Kumar et al. [13] proposed an API based solution that protects the SOAP messages that are communicated between web services from Message Alteration Attack (MAA). However, this solution is not generic and pluggable.

3 WS-SM Framework

In a dynamic composition environment, user requirements from clients are submitted to the composer which dynamically selects and composes services according to user requirements. While invoking domain web services, the composer sends back SOAP request message and in turn the domain web services send SOAP response message that contains the output or the fault information. During the exchange of these messages via a public network, they are prone to attacks. The security gateway adds (removes) basic security functionality to (from) SOAP request (response) messages irrespective of the attack type. The secured SOAP message is intercepted by different APIs to protect the message from the particular web service attack. When a specific attack occurs, the respective API in the composer or in the domain server side detects it and overcomes. Certain web service attacks are prevented by the corresponding APIs installed on both sides. The proposed framework is depicted in Fig. 1. The design of security gateway is detailed in the next subsection and the design of proposed APIs are described in the subsequent subsections.

3.1 Security Gateway

The security gateway implemented as a web service is deployed in composer as well as in the domain server side. The security gateway in composer side remembers a copy of the SOAP request and redirects it to the security gateway in the domain server. The security gateway in domain server side receives the SOAP request and invokes the domain service. The response from domain service is redirected by the gateway to its counterpart in the composer side after

[5] Layer7 Technology, (2013), `http://www.layer7tech.com/solutions/web-api-attack-protection`

[6] IBM WSSAPI, (2014), `https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.base.doc/ae/cwbs_wss_api.html`

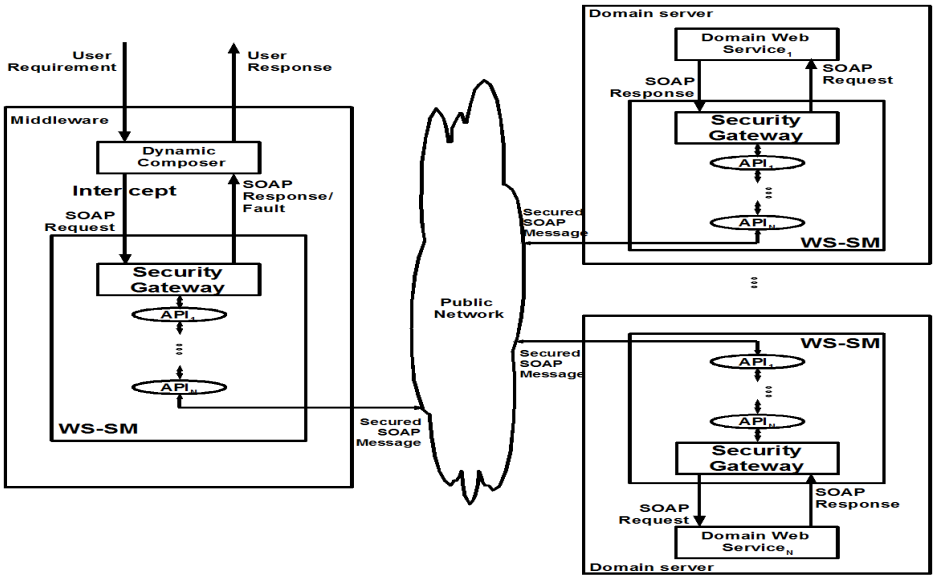


Fig. 1. Architecture of WS-SM Framework

remembering a copy. The security gateway in the composer side forwards the response to the composer. In case of an attack on the communicated message in the network, the receiving gateway generates a fault message and in response, the sender gateway sends the remembered copy of the original message. If the communication channel is subject to an attack repeatedly, resending the copy of the original message several times would incur increased overhead. Hence, a *threshold* for maximum number of times the message copy can be re-transmitted is designed and it is made configurable. After the message is resent a specific number of times equal to the pre-defined *threshold*, the security gateway notifies the composer. Based on this, the composer selects an alternative service with an alternative communication channel so that the attack can potentially be avoided. The design of security gateway service is described in Algorithm 1.

Algorithm 1 Design of Security Gateway Service

SecurityGateway (InSOAPMsg, ResponseSOAPMsg)
INPUT: InSOAPMsg // Input SOAP message
Threshold // Configured by WS-SM
OUTPUT: ResponseSOAPMsg // Output SOAP message
1. NumberofResends = 0
2. EndPointReference ← InSOAPMsg.SOAPBody.endpoint
3. OutSOAPMsg.SOAPBody ← InSOAPMsg.SOAPBody
// Redirect *InSOAPMsg* to Service Gateway in Domain Server side
4. OutSOAPMsg.To ← concat(GetIPAddress(EndPointReference), GetPath(SecurityGateway))

5. ResponseSOAPMsg ← Sendmessage(OutSOAPMsg)
6. **WHILE**(ResponseSOAPMsg.SOAPFault = TRUE **AND** NumberofResends <Threshold)
    NumberofResends = NumberofResends + 1
    ResponseSOAPMsg ← Sendmessage(OutSOAPMsg)
  ENDWHILE
7. RETURN(ResponseSOAPMsg)

3.2 SOAPAction Spoofing API

In a public network, an attacker can tamper the SOAP request, change the operation in the header part of the SOAP request and allow it to transmit in the same communication channel. In effect, an unintended wrong operation would be invoked. In order to avoid this SOAPAction Spoofing attack, the SOAPAction Spoofing API (SAS API) is proposed as a part of WS-SM framework. It is installed in the domain server end. It intercepts the SOAP request and compares the content in the header field and the operation in the SOAP Body. If they are not the same, it responds with a SOAP fault message. On receiving the fault message, the security gateway service at the composer end resends a copy of the request. If the attacker hacks the communication channel more than the *threshold* number of times that is pre-configured, the composer will be notified regarding the attack. The message communication sequence is depicted in Figure. 2.

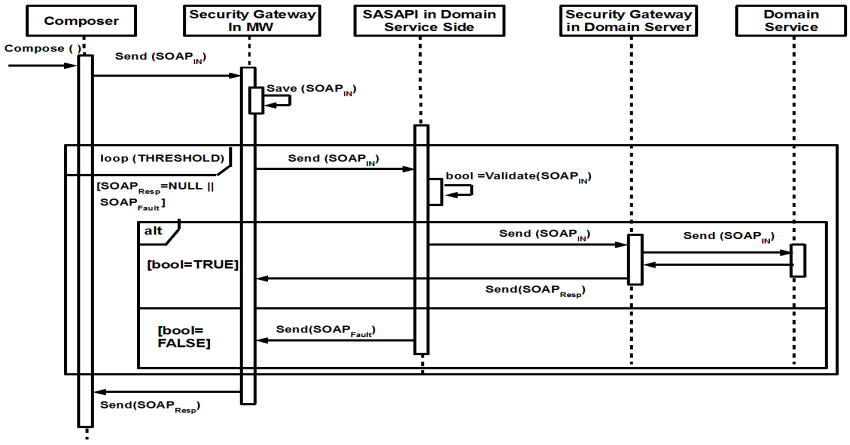


Fig. 2. Securing message communication from SAS attack

3.3 WS-Addressing Spoofing API

An attacker can tamper the SOAP message in transit and add an invalid address in *ReplyTo*, *FaultTo* or *To* fields which causes redirection of SOAP message to an unintended receiver. This WS-Addressing Spoofing attack is addressed by proposing WS-Addressing Spoofing API (WSAS API) in WS-SM framework. The WSAS API is installed in domain server end to intercept the SOAP request and validate the *RelyTo*/*FaultTo*/*To* fields against the *whitelist*, a list of valid URLs registered with the domain service that can communicate with it. If the address is not present in the *whitelist*, the WSAS API sends a fault response to the composer as shown in Figure 3.

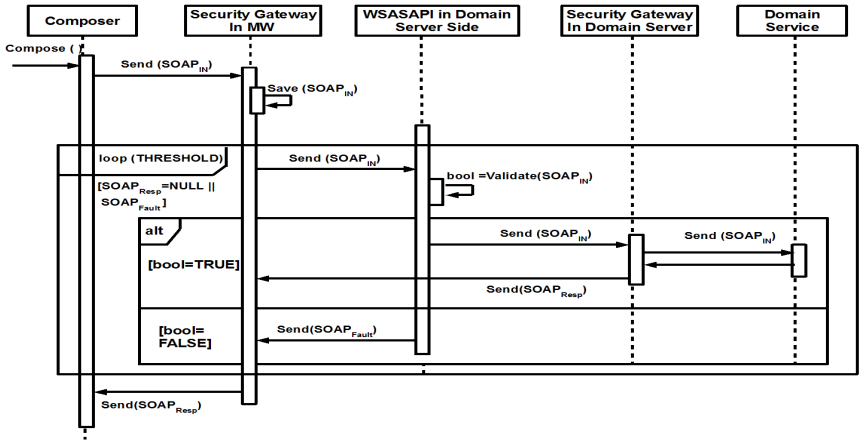


Fig. 3. Securing message communication from WSAS attack

3.4 Replay of Messages API

Every SOAP message has a unique *MessageId* to differentiate it from another message. An attacker can tamper this *MessageId* and send the same message several times as if it is a request from a valid client. The proposed Replay of Messages API (RoM API) intercepts and encrypts the *MessageId* field before sending the request through the network so that, the attacker cannot change it. The RoM API at the receiver end decrypts the *MessageId* value of the received message and validates with *MessageIdList* file that contains all the *MessageIds* of previously executed SOAP requests. If the *MessageId* is not present in the list, then the RoM API allows the request to invoke the domain service. Otherwise, it drops the request.

3.5 Message Alteration Attack API

The Message Alteration Attack (MAA) API is proposed in WS-SM framework to ensure the confidentiality of SOAP messages. This API intercepts the outgoing SOAP message from security gateway and encrypts [7] its entire body so that the attack is prevented. If the attacker changes the encrypted content of the message during transit, decryption of the SOAP message by MAA API in the receiver end will result in an error. Due to this, the request will not be executed in the domain server. To detect this attack, the MAA API at the domain server end intercepts the request and decrypts the content of the incoming request. If this decrypted content is some junk value, then it is sent back to the security gateway to resend the original message. The MAA API also addresses the following attacks.

- **Loss of Confidentiality Attack:** Loss of Confidentiality occurs when information within a SOAP Message is disclosed to an unauthorized individual in the network. By encrypting the entire SOAP message body using MAA API, this attack is prevented.
- **XML Injection Attack:** Even though a SOAP message is encrypted, an attacker can tamper it by adding additional XML tags or deleting encrypted contents. The MAA API can detect this attack by intercepting and decrypting the received encrypted SOAP message. The decrypted message is validated against the schema of the SOAP message. In case of a decryption or a validation error, a fault message is sent back. The design of MAA API is described in Algorithm 2.

Algorithm 2 Design of MAA API

MAAAPI(InSOAPMsg, ResponseSOAPMsg)
INPUT: InSOAPMsg // Incoming SOAP message
OUTPUT: ResponseSOAPMsg // Outgoing SOAP Message
1. **IF** Equals(getElementsByTagName(InSOAPMsg.SOAPBody), EncryptedData) **THEN**
ResponseSOAPMsg ← Decrypt(InSOAPMsg.SOAPBody)
IF DECRYPTERROR **OR NOT** Validate(ResponseSOAPMsg, SOAP-Schema) **THEN**
THROW SOAPFault // If MAA is detected, a fault is thrown
ELSE ResponseSOAPMsg ← Encrypt(InSOAPMsg.SOAPBody)
2. **RETURN** (ResponseSOAPMsg)

3.6 Principal Spoofing Attack API

Principal Spoofing Attack (PSA) [8] occurs when an attacker hacks the SOAP message and sends it as if it is sent by another authorized client, by stealing

[7] Encryption Algorithm, (2014), `https://www.princeton.edu/~ota/disk2/1987/8706/870612.PDF`
[8] Principal Spoofing, (2014), `https://capec.mitre.org/data/definitions/195.html`

its credentials. The attack results in the execution of an unintended service. In order to avoid this attack, the proposed PSA API intercepts the message and adds the *username* token containing *UserName*, *Password*, *Created* and *Nonce* tags. Then, a digital signature [9] is added to the message and is sent through the network. The PSA API in the other end verifies the incoming SOAP message and sends a fault message in case it is tampered during transit. The PSA API also addresses the following attacks.

- Forged Claims Attack
- Falsified Messages Attack

Forged Claims Attack This type of attack is a variation of PSA where an attacker constructs a new SOAP message using credentials of a different authorized client. The attacker sends the SOAP message as if it is sent by an authorized client. The PSA API prevents this attack by creating the digest of the password of an identity. The password digest is calculated using plain password, nonce and created time. It is not possible for the attacker to obtain the password from the password-digest and hence the credentials.

Falsified Messages Attack This type of attack is also another variation of PSA where a fictitious message is created by an attacker with the same credentials so that the receiver believes that the SOAP message comes from the original authorized client. The PSA API prevents this attack by making digest of sender password so that the attacker will not be able to extract password of a valid sender. Thus, the PSA API prevents principal spoofing attack. The design of PSA API is described by Algorithm 3.

Algorithm 3 Design of PSA API

PSAAPI(InSOAPMsg, ResponseSOAPMsg)
INPUT: InSOAPMsg //Incoming SOAP message
OUTPUT: ResponseSOAPMsg //Outgoing SOAP Message
//Check if header of InSOAPMsg contains UserNameToken
1. **IF** $InSOAPMsg.SOAPHeader.getChildrenWithLocalName() \notin UsernameToken$ **THEN**
InSOAPMsg.SOAPHeader.addChild(UserName)
InSOAPMsg.SOAPHeader.addChild(Password)
InSOAPMsg.SOAPHeader.addChild(Nonce)
InSOAPMsg.SOAPHeader.addChild(CreatedTime)
InSOAPMsg.SOAPHeader.addChild(Signature) //Add Digital Signature to InSOAPMsg
ResponseSOAPMsg ← InSOAPMsg
ELSE Validate PasswordDigest and Digital Signature
IF NOT(Validate(InSOAPMsg.SOAPHeader.UserNameToken) **AND**

[9] XML Signature, (2013), `http://www.xml.com/pub/a/2001/08/08/xmldsig.html`

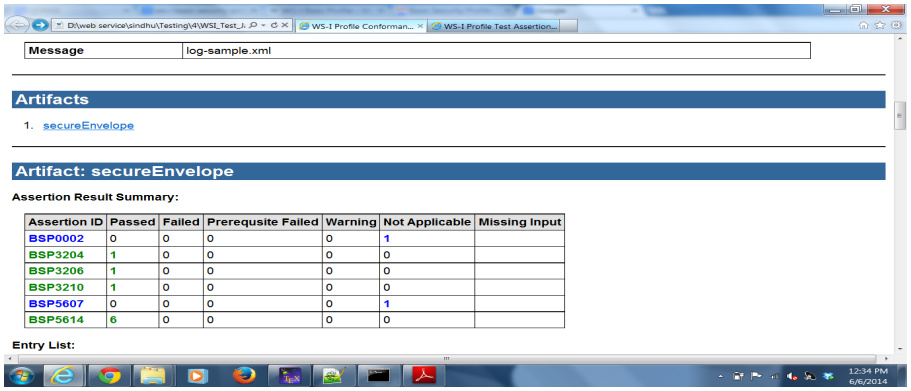
Validate(InSOAPMsg.SOAPHeader.Signature)) **THEN**
THROW SOAPFault //Principal Spoofing is detected
2. **RETURN** (ResponseSOAPMsg)

4 Testing and Discussion

The proposed WS-SM Framework has been tested with the following three objectives:

- To test the compliance of WS-SM framework for interoperability to prove suitability of WS-SM for SOA based applications.
- To test the immunity of WS-SM approach against web service attacks on communication among web services
- To analyze the trade-off between providing security using WS-SM approach and degradation of application performance.

4.1 Compliance Testing



Assertion ID	Passed	Failed	Prerequsite Failed	Warning	Not Applicable	Missing Input
BSP0002	0	0	0	0	1	
BSP3204	1	0	0	0	0	
BSP3206	1	0	0	0	0	
BSP3210	1	0	0	0	0	
BSP5607	0	0	0	0	1	
BSP5614	6	0	0	0	0	

Fig. 4. Complaince Testing using WSI-BP

The WS-I Basic Profile (WSI-BP) [10], a specification from the Web Services Interoperability (WS-I), industry consortium, provides interoperability guidance within its scope of standards like SOAP, WSDL, UDDI, etc.

Conformance to WSI-BP is defined by adherence to the set of requirements for a specific target, within the scope of WSI-BP. Requirements state the criteria for conformance to WSI-BP and consist of refinements, interpretations and clarifications that improve interoperability. Targets allow for the description of conformance in different contexts, to allow conformance testing and certification of

[10] WS-I Basic Security Profile, (2007), `http://ws-i.org/Profiles/BasicProfile-2.0-2010-11-09.html`

artifacts such as SOAP messages and WSDL descriptions, web service instances, and web service consumers. WSI-BP makes requirement statements about artifacts such as SOAP messages, WSDL Descriptions, and registry elements. A web service is allowed to advertise conformance to WSI-BP by annotating these artifacts with conformance claims, which use a URI to assert conformance with a particular profile. An instance of an artifact is considered conformant when all of the requirements associated with it are met.

The proposed WS-SM Framework was adapted to an example case study of banking application where dynamic composition of services is involved. The compliance of WS-SM framework to WSI-BP has been checked using an existing tool [11] by submitting the SOAP messages communicated in the network as input. The snapshot of the conformance report from the tool is depicted in Figure. 4. It indicates that the assertions related to the artifact of SOAP messages have been passed. Hence, it is established that WS-SM Framework is compliant with WSI-BP and thus interoperable.

4.2 Vulnerability Testing

Most of the existing methodologies and tools for vulnerability testing either do not work properly, are poorly designed, or do not fully test for real world web service vulnerabilities [12]. Typically, web application penetration tests are not scoped properly to include the related web services. Testing methodologies should include not only technical details on how to test web services, but also non-technical information such as proper scoping as well as pre-engagement requirements, which often are overlooked by penetration testers. In addition, current methodologies lack information on a complete threat model for web services. Depending on the data being exchanged, threats need to be carefully identified. SOAPUI [14], WS-Attacker [13] [15] and WSBang [14] are some of the testing tools used to test the functionality and few of the security features like SQL Injection, Cross Site scripting, etc. However, these tools are used for testing the attacks against a single web service [16] and they do not work in a dynamic composition environment. Hence, a specific penetration testing tool has been designed based on ATLIST (Attentive Listener) [17] methodology. ATLIST is a vulnerability analysis method developed during and for the analysis of SOA service orchestrations. It facilitates the detection of known vulnerability types and enables derivation of vulnerability patterns for tool support. ATLIST is applicable to business processes composed of services as well as single services. ATLIST offers better transferability by guiding the analysis with a set of analysis elements

[11] WS-I Basic Security Profile Tool, (2009), `http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools`

[12] Eston, T., J. Abraham, and K. Johnson.: Dont Drop the SOAP: Real World Web Service Testing. Retrieved July 6, 2013

[13] WS-Attacker, (2013), `http://sourceforge/p/ws-attacker/wiki/Home`

[14] WSBang Testing Tool, (2014), `https://www.isecpartners.com/tools/application-security/wsbang.aspx`

such as Point of View, Attack Effect, Active Component, Involved Standard, and Triggering Property.

The proposed ATLIST based penetration tool considers banking application as a business process (POV), domain web server as an active component, and SOAP as an involved standard. For each of the nine web service attacks that have been addressed by the proposed WS-SM framework, the triggering property (Precondition) and test case(s) have been identified and tabulated in Table. 2. Each attack was simulated in the dynamic composition environment in accordance with the test case. For the input SOAP message generated according to the test case, the attack effect (Postcondition) was observed. The banking application was then enabled with WS-SM approach, the vulnerability testing was repeated with the same test case, and the outcome (Protection measure) of the proposed approach was observed and tabulated. It is found that the proposed WS-SM approach is not vulnerable to any of the specified nine attacks. It detects and overcomes five of the attacks and prevents four of the attacks.

4.3 STRESS TESTING

A prototype banking application involving dynamic composition of services was implemented on a 2.4 GHz Intel Core2 Duo processor with 4 GB RAM. The domain web services, composer and client were developed on different machines with different platforms such as Linux Ubuntu 10.04, Ubuntu 11.04, and Fedora 12. The services were deployed in different web servers like Glassfish, Apache Tomcat, etc. The composer and various domain services with WS-SM framework were all deployed in different LAN terminals.

The performance of the banking application was assessed in terms of throughput as the number of requests per second by increasing the number of concurrent user requests from 25 to 200. The throughput was measured 10 times at different workloads of network and the maximum values have been plotted as a graph shown in Figure. 5. Then, the banking application was enabled with WS-SM framework by installing the Security APIs and security gateway in both composer as well as in each of the domain server end. The throughput readings were taken and the impact of security APIs over the performance of the application was analyzed. It is found that even with 200 concurrent user requests, the throughput decreased by merely 35% when compared to not providing the security using the proposed approach.

In general, the attacks are generated automatically and repeatedly. The proposed security APIs prevent or detect the attacks every time they affect the communication. In order to decrease the turnaround time of the user request, the proposed WS-SM approach allows to configure the maximum number of times a particular instance of an attack is prevented or overcome. When this threshold reaches, the composer is notified so as to select an alternate service and thus an alternate communication channel to potentially avoid the attack. The impact of this threshold over performance of the WS-SM enabled banking application prototype has been analyzed. The threshold was configured as 1 and 3. For each of these configuration, the number of concurrent user requests was

Table 2. ATLIST Based Vulnerability Testing of Proposed Security APIs

Attack	**Triggering Properties**	**TestCase**	**Attack Effect**	**Outcome from Proposed Security API**
SOAP Action Spoofing	Action field in soapenv:Header	Different operations in action field of soapenv:Header and operation field of soapenv:Body	Execution of incorrect SOAP operation	Prevention of invoking incorrect service and responding with soapenv:Fault message
WS-Addressing Spoofing	1. ReplyTo addressing field in soapenv:Header 2. FaultTo addressing field in soapenv:Header 3. To addressing field in soapenv:Header	1. Blacklisted address in ReplyTo 2. Blacklisted address in FaultTo 3. Blacklisted address in To	1. Redirection of SOAP Response to blacklisted address	Preventing the redirection of SOAP message and responding with soapenv:Fault message
Replay of Messages	MessageId in soapenv:Header	SOAP Messages with same/different MessageId and same soapenv:Body	Unintended execution of same service more than once	All resent SOAP Messages are dropped except the original message
Message Alteration Attack	Plaintext in soapenv:Body	soapenv:Body with unintended contents in plaintext	SOAP Message with unintended contents	Testcase not allowed
Loss of Confidentiality		SOAP Message with plaintext	Exposure of SOAP Message to unintended users	
XML Injection	Encrypted/ Unencrypted Message in soapenv:Body	Encrypted/ Unencrypted Message in soapenv:Body not conforming to XML Schema	soapenv:Fault or SOAP Message with unintended contents	Invalidating SOAP Message and responding with soapenv:Fault message
Principal Spoofing	1. Username in soapenv:Header 2. Password in soapenv:Header 3. Nonce in soapenv:Header 4. Created in soapenv:Header	UsernameToken with unregistered credentials in soapenv:Header	Service provisioning to user with false credentials	Preventing service provision to user with false credentials and responding with soapenv:Fault
Forged Claims				Testcase Not allowed
Falsified Messages				Testcase Not allowed

varied from 25 to 200 and the throughput in terms of number of requests per second was measured. Three samples were taken for each reading at different workloads of network and the maximum values have been plotted as a graph shown in Figure. 6. It is observed that the throughput decreased by 46 % when the threshold is increased from 1 to 3.

5 Conclusion

A framework for secured messaging among web services that either prevents or detects and overcomes some of the web service attacks has been implemented in the context of dynamic web service composition. Compliance testing of WS-SM for WSI-BP revealed that it is interoperable. From the vulnerability testing of WS-SM, it was found that the proposed WS-SM approach is not vulnerable to

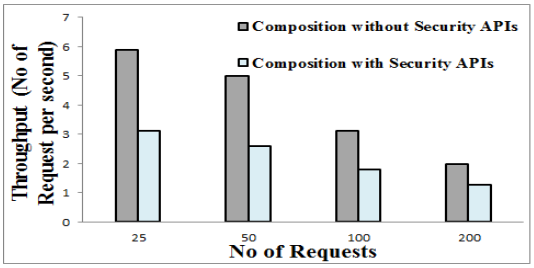


Fig. 5. Security Vs Performance

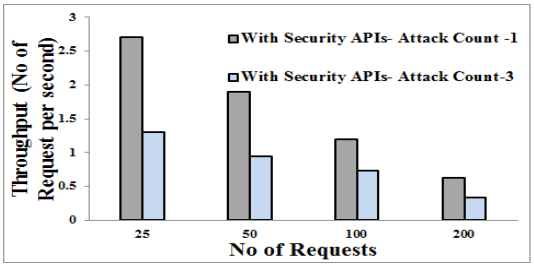


Fig. 6. Impact of Repeated Attacks on Performance

any of the specified nine attacks. It detects and overcomes five of the attacks and prevents four of the attacks. A banking application involving dynamic composition of services was adapted with WS-SM framework and subjected to stress testing. Even with 200 concurrent requests, the throughput decreased by merely 35% when compared to not providing the security using the proposed approach demonstrating a good tradeoff between security and performance.

The proposed WS-SM framework with pluggable APIs can be adapted for any application involving dynamic composition by installing it in the web servers where the composer as well as domain services are deployed. The proposed solution is generic since new APIs for other kinds of attacks can be added on both sides as and when new types of attacks emerge. Moreover, the solution is entirely transparent to the client.

The future work involves creating APIs for addressing and overcoming the other web service attacks such as denial of service.

Acknowledgments

Authors would like to thank S.M. Sindhu, postgraduate student for her coding efforts.

References

1. Erl T.: Service-Oriented Architecture concept, Technology, and Design. Pearson Education, (2006)
2. Schmelzer R., and Vandersypen T.: XML and Web Services Unleashed. Sams publication, (2002)
3. web services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly Media, Inc, (2002)
4. Singhal A., Winograd T., and Scarfone K.: Guide to Secure Web Services. Tech report of National Institute of Standards and Technology, Special Publication, 800(95), (2007)
5. Lemos, A. L., Daniel, F., and Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR), 48(3), Article no. 33, (2016)
6. Varsha R. Mouli, K.P. Jevitha.: Web Services Attacks and Security- A Systematic Literature Review. In: Procedia Computer Science, vol. 93, pp. 870-877, (2016)
7. A. Masood, J. Java.: Static analysis for web service security - Tools & techniques for a secure development life cycle. IEEE International Symposium on Technologies for Homeland Security (HST). pp. 1–6, (2015)
8. Jensen M., Gruschka N., and Herkenhoner R.: A Survey of Attacks on Web Services- Classification and Countermeasures. Computer Science Research and Development (CSRD), 24(4), pp. 189–197, (2009)
9. Nordbotten N. A.: XML and Web Services Security Standards. IEEE Communications Surveys and Tutorials. 11(3), pp. 4–21, (2009)
10. Alotaibi, S. J.: Toward a Secure Web Service by Using WS-Security Specifications. Journal of Computational and Theoretical Nanoscience. 14(8), pp. 3837–3842, (2017)
11. Thelin, J., and P. J. Murray.: A Public Web Services Security Framework Based on Current and Future Usage Scenarios. In: International Conference on Internet Computing, pp. 825–833, (2002)
12. Yue H., and Tao X.: Web Services Security Problem in Service-Oriented Architecture. In: International Conference on Applied Physics and Industrial Engineering, 24(9), pp. 1635–1641, (2001)
13. Kishore Kumar R., Kanchana R. and Chitra Babu.: Security for SOAP based Communication among Web Service. In: IJCA Proceedings on International Conference on Science, Engineering and Management (ICSEM'13), pp. 46–51. Foundation of Computer Science, USA (2013)
14. Noor A. Altaani, Ameera S. Jaradat.: Security Analysis and Testing in Service Oriented Architecture. International Journal of Scientific & Engineering Research, 3(2), pp. 1–9, (1981)
15. Mainka C., Somorovsky J., and Schwenk J.: Penetration Testing Tool for Web Service Security. IEEE 8th World Congress on Services, pp. 163–170, (2012)
16. M.I.P. Salas, E. Martins.: Security Testing Methodology for Vulnerabilities Detection of XSS in Web Services and WS-Security. In: Electronic Notes in Theoretical Computer Science, 302, pp. 133–154, (2014)
17. Lowis L., and Accorsi R.: Vulnerability analysis in SOA-based business processes. In: IEEE Transactions on Services Computing, 4(3), pp. 230–242, (2011)