International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435(Online) Volume 11, Issue 10, October 2022)

An Analysis of Web Services Security Issues, Solutions, and Restrictions

Sanidayal Gupta[1], Prof. Vishal Paranjape[2]

Department of Computer Science & Engineering Global Nature Care Sangathan's Group of Institutions, India

Abstract-- **One message at a time, the WS-Security standard describes the fundamental SOAP traffic-securing techniques. However, using WS-Security separately for each message in a normal web service is fairly wasteful; in addition, it is frequently crucial to guarantee the integrity of the entire session in addition to each message. More SOAP-level techniques are available for this purpose in recent standards.**

I. INTRODUCTION

Among the most significant technology advancements from the previous decade are frequently regarded as Web Services and Service-Oriented Architectures (SOAs). However, these new methods' benefits do help to counteract certain very major drawbacks that these new technologies have. The most important problems include Web services security[19]. A secure system should typically have integrity, confidentiality, and availability. An attack is any action intended to compromise one of these characteristics, and vulnerability is the term for that action. A collection of security issues within the Web Services domain are presented in this article. The list is not intended to be exhaustive; rather, it is only a collection of the most noteworthy attacks that we have looked at in our research. The majority of the attacks fall within the Denial-of-Service (DoS) attack category[22] because the availability of services was the focus of this research.

Daily news reports can be used to gauge the frequency of DoS assaults. For example, in April and May 2007, DDoS attacks on official and commercial websites in Estonia were reported[25]. These assaults were carried out by botnets utilising methods for flooding the network layer. In this post, we'll demonstrate how much less resource effort is required to conduct DoS attacks against web services than against non-web-service systems.

There are many different facets to the attacks. We'll start by discussing attacks on individual Web Services that lack security precautions, then go on to attacks on WS-Security-enabled Web Services, and ultimately, attacks on Web Services that are employed in compositions of Web Services. Although the latter covers all varieties of Web Service compositions, we have chosen WS-BPEL (often known as BPEL for short) to illustrate an attack since it seems to be taking the lead in Web Service composition standards.

The remainder of this article is structured as follows. The essential terms and concepts of safety are explained in the part that follows, along with Web services and BPEL. Web Ser attack flaws and vulnerabilities are listed in Section 3. Part Section 4 Following the discussion of general countermeasure principles, Section 5 provides the classification structure for attacks. Finally, we wrap up in Section 6 on the research given in this survey report.

II. FUNDAMENTALS

WS-Security 2.1

WS-Security [21] is the most significant specification addressing the security requirements of Web Services. It works in conjunction with the SOAP requirements to provide Web Services with integrity, confidentiality, and authentication. WS-

The so-called security header, which carries the WS-Security extensions, is defined in Security as a SOAP header block. Additionally, it specifies how SOAP communications should be encrypted using current XML security standards like XML Encryption [13] and XML Signature [2].

With XML Signature, XML fragments can be digitally signed to prove their authenticity or to guarantee their integrity. A Signature element is created, which is once more appended to the security header, and contains the output of the signature procedure, or the encrypted digest.

XML Encryption enables the encryption of XML fragments to guarantee data confidentiality. An Encrypted Data element that has the ciphertext of the encrypted fragment as its content is substituted for the encrypted fragment.

Additionally, an EncryptedKey element is defined by XML Encryption for the purpose of transporting keys. The usual use of an encrypted key is a hybrid encryption, in which an XML fragment is encrypted using a symmetric key that was created at random and then encrypted using the recipient of the message's public key. In SOAP messages, the security header must contain the EncryptedKey element, if it is present. In addition to encryption and signatures, WS-Security also specifies security tokens like the UsernameToken or X.509 certificates that can be used to transfer digital identities.

The high degree of flexibility of the mechanisms utilised in WS-Security is a significant feature. They are applied to any part of the SOAP message at random, leaving the rest untouched. As a result, Web server clients and servers must agree on a security policy that specifies the WS-Security components that will be used.

Such security policies can be declared using an XML syntax provided by WS-SecurityPolicy [17]. A server may declare its security requirements in a WS-Security-Policy document as an addition to the Web Service definition. The SOAP message components that must be encrypted or signed, the algorithms to apply, and the necessary security tokens can all be specified using the WS- SecurityPolicy.

Engine BPEL These jobs can be divided into three categories: communication tasks that represent incoming or outgoing Web Service calls, structural tasks that describe the execution sequence, and other fundamental duties like process variable access, workflow execution time restrictions, or fault management. Each deployed BPEL process may have several process instances—concurrent execution contexts of the same process—at any given time.

The ability to employ asynchronous communication is a crucial component of BPEL-based Web service composition. In a typical Web Service call, a request message is followed immediately by a reply message. Until the reply message is received, the requester must maintain the connection to the server. BPEL provides asynchronous behaviour, allowing the requester to disconnect after sending its request, by using a particular language construct. In this scenario, the Web Service server creates a new connection and requests a Web Service on behalf of the original requester to provide the reply message. Long-running processes that cannot be finished within the timeout parameters of a single Web Service call benefit from this communication style.

The WS-Addressing [11] specification is used to specify the callback destination and enables the requester to include an abstract endpoint reference in the request message. This reference contains all the information required for the BPEL engine to invoke the Web Service on behalf of the requester.

Message correlation is an additional task that a BPEL engine must complete. It becomes necessary to use designated message data fields to identify the target process instance for an incoming Web Service message because a BPEL engine may run multiple instances of one BPEL process concurrently. In the context of BPEL, these are referred to as correlation sets.

III. ATTACKS

In this section we present a list of attacks on Web Ser- vices. For each attack an abstract attack methodology and impact is given, demonstrated by a concrete attack execution where appropriate. Additionally, countermeasures against the particular attacks are discussed.

3.1 Oversize Payload

One important category of Denial-of-Service attacks is called Resource Exhaustion [24]. Such attacks target at eliminating a service's availability by exhausting the re- sources of the service's host system, like memory, processing resources or network bandwidth. One "classic" way to perform such a Resource Exhaustion attack is to query a service using a very large request message. This is called an Oversize Payload attack [19].

Against Web Services, an Oversize Payload attack is quite easy to perform, due to the high memory consumption of XML processing. 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 (DOM [12]). Using this model, an XML document like a SOAP message is completely read, parsed and transformed in- to an in-memory object representation, which occupies much more memory space than the original XML document. For common Web Service frameworks, we observed a raise in memory consumption of factor 2 to 30.

Example: An Axis Web Service was attacked using a large SOAP message document, which consisted of a long list of elements considered as parameter values of the Web Service operation1:

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

The SOAP message had a total size of approx. 1.8 MB. The message processing induced a full CPU load for more than one minute and an additional memory usage of more than 50 MB. Enlarging the message to approx. 1.9 MB even resulted in an out-of-memory exception.

An obvious countermeasure against Oversize Payload attacks consists in restriction of the total buffer size (in bytes) for incoming SOAP messages. In this case, it is sufficient to check the actual message size and reject any message exceeding the predefined limit. This method is used by the .NET 2.0 frameworks, which discards all SOAP messages larger than 4 MB (in the default configuration). While this countermeasure is very simple to implement, it is not suitable for Web Service messages.

A more appropriate approach uses restrictions on the XML info set. This can be realized by modifying the XML schema inside the Web Service description and validating incoming SOAP message to this schema [7]. Details of this approach can be found in section 4.

3.2 Coercive Parsing

One of the first steps in processing a Web Service request is parsing the SOAP message and transforming the content to make it accessible for the application behind the Web Service. Especially when using namespaces, XML can become verbose and complex in parsing, compared to other message encodings. Thus, the XML parsing process allows other possibilities for a special kind of Denial-of- Service attacks, which is called Coercive Parsing attacks [19].

Example: The following attack was performed targeting an Axis2 Web Service. The attack used a continuous sequence of opening tags:

```
<x>
<x>
<x>
...
```

The attack resulted in a CPU usage of 100% on the target system. The service's availability was massively reduced, and the incoming message was finally received with a constant rate of 150 byte/s. Thus, the attack would per- form well even if the attacker has a very low bandwidth connection. The Web Service server did not abort the connection, thus this attack could apparently be continued infinitely. In our experiment, we stopped the attack after one hour.

Typical Coercive Parsing attacks targeting at resource exhaustion use a large number of namespace declarations, oversized prefix names or namespace URIs or very deeply nested XML structures.

These types of attacks require different countermeasures.

An attack that is based on complex or deeply nested XML documents (like the one in the example above) can be fended by using schema validation (compare section 4).

Attacks misusing namespace declarations are harder to prevent. As the XML specification does neither limit the number of namespace declarations per XML element nor the length of the namespace URIs, any restriction on the number or length of namespace declarations would be arbitrary and could lead to unpredictable rejection of messages.

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 "SOAPAction" can be used for operation identification. Although this value only represents a hint to the actual operation, the SOAPAction field value is often used as the only qualifier for the requested operation. This is based on the bogus optimization that evaluating the HTTP header does not require any XML processing.

This twofold operation identification enables two classes of attacks. The first one is executed by a man-in-the- middle attacker and tries to invoke an operation different from the one specified inside the SOAP body. It is based on modification of the HTTP header.

Example: The following attack was performed targeting a .NET Web Service. The deployed service provi- ded two operations: op1(string s) and op2(int x)— with the respective SOAP Action and message element also named opn. The following message (including the HTTP header) was sent to the service:

```
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(0). This shows that the operation is selected solely by the SOAP Action value from the HTTP header. Even worse, the "wrong" operation was executed despite of incompatible parameter names and types.

The example shows how modifications of the HTTP header can invoke methods that were not intended by the SOAP message creator. As the HTTP header is not secured by WS-Security and is newly created at every SOAP intermediary, it can easily be modified.

The second class of SOAP Action spoofing attacks is executed by the Web Service client and tries to bypass an HTTP gateway.

Example: The following attack was performed tar- geting an Axis2 Web Service. The deployed service pro- vided two operations: hidden and visible—with the respective SOAP Action and message element equally na- med. The following message (including the HTTP header) was sent to the service:

```
POST /axis2/testService HTTP/1.1
...
SOAPAction: "visible"
<Envelope>
<Body>
<hidden />
</Body>
</Envelope>
```

The Axis2 server actually ignored the SOAP Action value and invoked the hidden operation instead. If an HTTP border gateway—which of course operates on the HTTP header only—is configured to reject hidden and accept visible accesses, this attack allows calling hidden anyway.

A countermeasure to SOAP Action Spoofing attacks would be to determine the operation by the SOAP body content. Additionally, the operations determined by the HTTP header and by the SOAP body must be compa red and any difference should be regarded as threat and result in rejecting the Web Service request.

3.4 XML Injection

An XML Injection attack tries to modify the XML structure of a SOAP message (or any other XML document) by inserting content—e.g. operation parameters containing XML tags. Such attacks are possible if the special characters ”<” and ”>” are not escaped appropriately. At the Web Service server side, this content is regarded as part of the SOAP message structure and can lead to undesired effects.

Example: The following attack was executed against a .NET Web Service. The deployed service method has two parameters a and b, both of type xsd:int. This service was invoked using the following SOAP message:

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

Such a message could result from an XML Injection attack: 1 was inserted as parameter content without escaping ”<” and ”>”. As the SOAP message obviously violates the Web Service schema, it should be rejected. But in fact, not only that the message was accepted by .NET, the resulting parameter values inside the service application for this request were: a = 1, b =0. Thus, the attacker was able to modify the value of b just by modifying the content of a. It is easy to imagine a scenario in which this can lead to unintended service behaviour, e.g. access to restricted data.

An important step in detecting such attacks is a strict schema validation on the SOAP message, including data type validation as possible (see section 4). This would have rejected the message from the example attack.

IV. GENERAL COUNTERMEASURE APPROACHES

Attacks on Web Services—as on any other system rely on a large number of different vulnerabilities. Therefore, countermeasures against attacks are also very wide ranging. Nevertheless, there exist several general defense mechanisms.

4.1 Schema Validation

Schema validation can be used against attacks, which use messages that are not conform to the Web Service description. Such attacks are called deviation from protocol message syntax [18]. By validating incoming messages to the XML schema generated from the WSDL, the attack can be detected—like shown in section 3.2 and 3.4. Nevertheless, in current Web Service frameworks schema validation is not used or not activated by default. This is mainly due to performance reasons, as schema validation is expensive regarding CPU load and memory consumption.

Schema validation is also effective against some other attacks on Web Service applications, like SQL Injection or Parameter Tampering [19], which also use non-valid messages3.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435(Online) Volume 11, Issue 10, October 2022)

Additionally, schema validation can be used as a foundation for other countermeasures. One important exam- ple is restricting the XML info set to limit the memory needed for processing the message like discussed in section 3.1. This is what we call Schema Hardening.

4.2 Schema Hardening

The general idea is to analyze a schema e.g. from a Web Service description for constructs allowing unbounded large or complex XML trees. These constructs are modified to fulfill finite boundaries.

For example, if the Web Service description defines an unbounded list of elements4 , the list is con verted into a list with limited number of elements. Inside the XML schema, the entry <element maxOccurs="unbounded"> is replaced by <element maxOccurs="n">, where n is a finite number. For most services such a limit is easy to define. An advantage of this restriction—compared to a limit of the network buffer size—is that this limit can be included in the service's "official" Web Service descrip- tion and thus becomes visible to clients in advance.

A second application of schema hardening could be removal of non-public operations from the schema inside the Web Service description (see section 3.5).

There are a number of further possibilities for har- dening the Web Service description—and thus the XML schema generated. Details can be found in [7]. The sa- me article also discusses problems raised by processing schemas containing large "maxOccurs" values.

4.3 Strict WS-Security Policy Enforcement

A WS-Security Policy policy defines a minimum set of security tokens that have to be included within a SOAP message to fulfill the policy. The specification does not provide a possibility for declaring their maximum usage. So as discussed before an attacker may add an unbounded number of additional tokens, engaging the targeted system in costly cryptographic computations and forcing high memory consumption.

To avoid this, a good strategy is to consider the requirements from the WS-Security Policy document not only as a minimum requirement, but also as a maximum requirement. This means, a SOAP message must contain exactly the security tokens specified by the security policy not less, not more.

As pointed out in [6], this limitation does not restrict the functionality, but enables the detection of attacks using oversized cryptography and can help to mitigate their effects.

V. CLASSIFICATIONS

In an effort to categorize and systemize these numerous attacks, we took a closer look at their specific impacts. Table 1 shows a classification of the attacks described here, based on the following parameters.

Category: Describes the security property that is viola- ted by the attack. Possible values are confidentiality (C), data integrity (I), avaliability/Denial-of-Service (A) or access control issues (AC).

Level: This value indicates whether the attack resides on messaging layer (M) or on process layer (P) as defined in [27].

Spreading: Attacks can be application specific (A), targeting a specific Web Service framework only, or they can be due to a conceptional (C) flaw of the under- lying protocol specifications.

Size: Some attacks target single Web Services, others involve several communication partners. The Size va- lue gives the usual or minimal number of involved systems—apart from the attacker.

Deviation: Describes whether the attack generally uses syntactical (S), sequential (O), or semantical/appli- cation specific (A) protocol deviation techniques. A [•] indicates potential, but not necessary deviation.

Dependencies: This parameter indicates how far an attack relies on prerequisites at the targeted Web Ser- vice server, e.g. the existence of a specific operation or a necessary flaw in the Web Service description.

Fendability: A measure on how effective potential countermeasures can be in terms of mitigating (m) or even completely fending (f) the particular attacks. The intended countermeasure concepts are given as well. Note that the general countermeasure of performing access control is applicable to any of the attacks presented here, but it only mitigates the attack, and does not completely annihilate the possibility for an attack.

Amplification: This factor as defined in [16] is only applicable for flooding attacks and describes the rela- tion of attack performance workload to attack impact workload.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435(Online) Volume 11, Issue 10, October 2022)

TABLE 1
Attack Classification

Attack	Category	Level	Spreading	Size	Deviation	Dependencies	Fendability	Amplification
Oversize Payload	A	M	C	1	[S]	none	Schema validation (m), Schema hardening (m-f)	28
Coercive Parsing	A	M	C	1	[S]	none	Schema validation (m), Schema hardening (m-f)	
SOAPAction Spoofing	AC	M	A	1	S	missing comparison of ope-ration and SOAPAction	match values (f)	
XML Injection	I	M	A	2+	A,[S]	faulty processing of client-side input data	Schema validation (m), client-side message validation (f)	
WSDL Scanning	AC,C	M	A	1	A	existence of secret operati-ons in public WSDL	WSDL reduction (f)	
Metadata Spoofing	all	M	C	1+	A	ability to access/modify metadata documents	authenticity check of metadata documents (f)	
Attack Obfuscation	A	M	C	1	A,[S]	WS-Security processing enabled	none	90
Oversized Cryptography	A	M	C	1	[S]	WS-Security processing enabled	strict security policy enforcement (m-f)	
BPEL State Deviation	A	P	A	1	O	BPEL process with more than one inbound endpoint	stateful firewalling (m)	700
Instantiation Flooding	A	P	C	1	A,[O]	none (knowledge of correla-tion sets fortifies impact)	efficient correlation set matching (m)	
Indirect Flooding	A,AC	P	A	2+	A,[O]	appropriate BPEL process	none	
WS-Addressing Spoofing	A,C	M	C	2	A	WS-Addressing processing enabled	address validity / access authorization check (m-f)	
Middleware Hijacking	A,AC	P	C	2+	A,[O]	asynchronous BPEL pro-cess	address validity / access authorization check (m-f)	

VI. CONCLUSIONS

Like every upcoming technology, Web Services are challenged by several security issues. The attacks presented in this article illustrate how easily an insufficiently secured Web Service server can be affected with a single or few messages. While some of the vulnerabilities are caused by implementation weaknesses, most of them exploit fundamental protocol flaws, abusing the given flexibility within WS-related standards.

Thus, in order to cope with these threats, Web Ser- vice developers and adopters must be aware of the vulnerabilities and their potential impact. Further, researchers need to examine the existing Web Service standards for further vulnerabilities in order to develop more accurate countermeasures. Only improvement of attack mitigation techniques along with integration into every Web- Service-driven system will face up with these challenges and help to make Web Services as secure as possible.

REFERENCES

[1] Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Tricko-vic I, Weerawarana S (2003) Business Process Execution Language for Web Services Version 1.1. Oasis Standard

[2] Bartel M, Boyer J, Fox B, LaMacchia B, Simon E (2002) XML-Signature Syntax and Processing. W3C Recom-mendation

[3] Bhargavan K, Fournet C, Gordon AD, O'Shea G (2005) An advisor for Web Services security policies. In: SWS'05: Proceedings of the 2005 workshop on Secure web services, ACM Press, New York, NY, USA, pp 1–9

[4] Fernando R (2006) Secure web services with apache ram-part. Tech. rep., WSO2 Oxygen Tank

[5] Gruschka N (2008) Schutz von Web Services durch erwei terte und effiziente Nachrichtenvalidierung. PhD thesis, Christian-Albrechts-University of Kiel, Germany

[6] Gruschka N, Herkenh¨oner R (2006) WS-SecurityPolicy Decision and Enforcement for Web Service Firewalls. In: Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435(Online) Volume 11, Issue 10, October 2022)

[7] Gruschka N, Luttenberger N (2006) Protecting Web Ser- vices from DoS Attacks by SOAP Message Validation. In: Proceedings of the IFIP TC-11 21. International In- formation Security Conference (SEC 2006)

[8] Gruschka N, Luttenberger N, Herkenh¨oner R (2006) Event-based SOAP message validation for WS- SecurityPolicy-Enriched web services. In: Proceedings of the 2006 International Conference on Semantic Web & Web Services

[9] Gruschka N, Herkenh¨oner R, Luttenberger N (2007) Ac- cess Control Enforcement for Web Services by Event- Based Security Token Processing. In: Braun T, Carle G, Stiller B (eds) 15. ITG/Gi Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007), pp 371–382

[10] Gruschka N, Jensen M, Luttenberger N (2007) A Stateful Web Service Firewall for BPEL. Proceedings of the IEEE International Conference on Web Services (ICWS 2007)

[11] Gudgin M, Hadley M, Rogers T (2006) Web Services Ad- dressing 1.0 - SOAP Binding. W3C Recommendation

[12] Hors AL, Hegaret PL, Wood L, Nicol G, Robie J, Cham- pion M, Byrne S (2004) Document Object Model (DOM) Level 3 Core Specification. W3C Recommendation

[13] Imamura T, Dillaway B, Simon E (2002) XML Encryp- tion Syntax and Processing. W3C Recommendation

[14] Jayasinghe D (2006) SOA development with Axis2: Un- derstanding Axis2 basis. IBM developerWorks

[15] Jensen M (2008) BPEL Firewall Abwehr von Angriffen auf zustandsbehaftete Web Services (german). VDM Verlag Dr. Mu¨ller, ISBN 9783836485517

[16] Jensen M, Gruschka N, Luttenberger N (2008) The Impact of Flooding Attacks on Network-based Services. In: Proceedings of the IEEE International Conference on Availability, Reliability and Security

[17] Kaler C, Nadalin (editors) A (2005) Web Services Security Policy Language (WS-SecurityPolicy) 1.1

[18] Leiwo J, Nikander P, Aura T (2000) Towards network denial of service resistant protocols. In: Proc. of the 15th International Information Security Conference (IFIP/SEC)

[19] Lindstrom P (2004) Attacking and Defending Web Service. A Spire Research Report

[20] McIntosh M, Austel P (2005) XML signature element wrapping attacks and countermeasures. In: SWS '05: Proceedings of the 2005 workshop on Secure web services, ACM Press, New York, NY, USA, pp 20–27

[21] Nadalin A, Kaler C, Monzillo R, Hallam-Baker P (2006) Web Services Security: SOAP Message Security 1.1 (WS- Security 2004)

[22] Needham RM (1994) Denial of service: an example. Commun ACM 37(11):42–46

[23] Noga ML, Schott S, L¨owe W (2002) Lazy XML proces- sing. In: DocEng '02: Proceedings of the 2002 ACM symposium on document engineering, ACM Press, New York,NY, USA, pp 88–94

[24] Sch¨afer G (2005) Sabotageangriffe auf Kommunika-tionsstrukturen: Angriffstechniken und Abwehrmanah men. PIK 28 pp 130–139

[25] Smith A (2007) Estonia: Under siege on the web. Time Magazine URL http://www.time.com/time/magazine/article/0,9171,1626744,00.html

[26] The SAX Project (2002) Simple API for XML–SAX 2.0.1 URL http://www.saxproject.org

[27] Weerawarana S, Curbera F, Leymann F, Storey T, Ferguson DF (2005) Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS- Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall PTR