ATTACKS ON WEB SERVICES AND MITIGATION SCHEMES

Vipul Patel, Radhesh Mohandas and Alwyn R. Pais
Information Security Research Lab, National Institute of Technology Karnataka, Surathkal, India
{vip04pat, radhesh, alwyn.pais}@gmail.com

Keywords: Attacks on Web Services, XML Injection, XSS Injection, Schema Validation, Schema Hardening, Attachment Scanner, WS-Trust, WS-Security, Frankenstein Message.

Abstract: Web Services have become dependable platform for e-commerce and many B2B models. Extensive adaptation of Web Services has resulted in a bunch of standards such as WS-Security, WS-Trust etc. to support business and security requirements for the same. Majority of the web services are offered over Http with Simple Object Access Protocol (SOAP) as an underlying exchange infrastructure. This paper describes attacks targeted at Web Services such as XML injection, XSS injection, HTTP header manipulation, sending stale message and other protocol specific attacks. We have used XML Re-Writing mechanism to perform "timestamp modification attack" and WS-Trust, WS-SecureConversation protocols attack. Schemas stated in WSDL file may not be accurate enough to validate messages effectively; Schemas should reflect structure of all possible genuine requests. Hence, we have proposed a new self-adaptive schema hardening algorithm to obtain fine-tuned schema that can be used to validate SOAP messages more effectively. We have also proposed mitigation techniques to counter attacks using MIME/DIME attachments.

1 INTRODUCTION

A Web Service is used as a basic building block for the implementation of SOA (Service Oriented Architecture) based system. Ease of interoperability and loose coupling are attributed to exchange infrastructure that SOA based system relies on.

Web Service engine needs a XML parser to extract the required parameters from an incoming message. Exploiting this parser can successfully lead to Denial of Service attacks. Often additional nodes are injected or existing nodes are tweaked so as to change the operation parameters. We have tried these attacks on web services and also developed mitigation techniques for some of these attacks. To ensure confidentiality and integrity of SOAP message, WS-Security standard is used that relies on XML Encryption and XML Signature. Improper use of these primitives gives scope for new attacks. XML Signature Re-Writing attack is one such attack. This attack paves way to further attacks. We have shown attacks on WS-Trust and Timestamp field in Microsoft's implementation of WS-Security. Many attacks can easily be circumvented by having strong schema generation and validation system.

Section 2 reviews related work pertaining to attacking and defending Web Services. In section 3, we have discussed attacks carried out by us. In section 4, we propose new mitigation techniques for some attacks on Web Services. Section 5 lists our future work and Section 6 concludes the paper.

2 RELATED WORK

There has been considerable effort in exploring different kind of attacks on Web Services. Probing Attacks, Coercive Parsing, External Reference Attacks and SQL Injections attacks have been discussed by (Negm, 2004). (Vorobiev, 2006) has classified attacks as XML attacks, SOAP based attacks or Semantic WS attacks based on exploits used. (Jensen, 2007) has demonstrated SOAPAction spoofing, oversized payload and cryptography based attacks. The message validation as a technique to thwart DoS attack is shown by (Gruschka, 2006). It mentions the use of schema embedded in WSDL (Web Service Description Language) to validate messages. (McIntosh and Austel, 2005) have discussed XML Re-Writing attack which shall serve as a baseline for the attacks that we have described here. Use of context sensitive signature as a countermeasure of XML Re-Writing attack is outlined by (Gajek, 2009).

3 ATTACKS ON WEB SERVICES

In this section, we describe different attacks performed on Web Services.

3.1 Injection Attacks

The Web Service engine parses SOAP request and converts extracted parameters to native types. Injection attacks alter parameters within SOAP request that get processed by the business logic. If a message generated by a client is tweaked a bit then validation on the server side can easily be bypassed without getting noticed.

3.1.1 XML Injection

This attack modifies existing tags or injects new XML tags inside operation parameters. We developed .Net Web Service as depicted below:

```
[WebMethod]
public string Add(int no1, int no2,
string name);
```

We could override the value of parameter 'no2' by injecting XML tags inside first method parameter 'no1' as shown in figure 1. Also, we could reset the value of the first parameter by injecting a XML node 'no23'.

```
<Add>                                          <Add>
 <no1>10</no1><no2>30</no2><no1></no1>          <no1><no23>10</no23></no1>
 <no2>20</no2>                                  <no2>20</no2>
 <name>NITK</name>                              <name>NITK</name>
</Add>                                         </Add>
```

Figure 1: Overriding values of 'no2' and 'no1'.

3.1.2 XSS Injection

In XML, a CDATA section is used to escape a block of text that would otherwise be parsed as mark up. Characters like "<" and "&" are considered illegal if appear as values of XML nodes. An attacker injects JavaScript by embedding it inside a CDATA tag and inserts CDATA within operation parameter.

3.2 Header Manipulation

The *SOAPAction* Http header helps uniquely identify target operation among multiple operations available at the same end point. In our setup, two operations named "AddIntegers" and "SubtractIntegers" were available at an endpoint: http://myservice.com/WebService/Service.asmx. We changed the *SOAPAction* attribute of "AddIntegers" operation to http://company.com/samples/wse/SubtractIntegers without modifying the Http body. We could successfully invoke the "SubtractIntegers" operation and obtained zero as a result value. Hence, it is still possible to cause unintended behaviour without changing SOAP body but through manipulation of *SOAPAction* header.

3.3 Attacks through SOAP Attachment

The SOAP with Attachments specification allows transmitting attachments using MIME/DIME package. The SOAP message package is constructed using MIME's Multipart/related media type. The SOAP message can refer to attachments through an URI.

Binary files containing malware can be posted as an attachment along with the SOAP message. If a web service happens to store this attachment as a file on the server or distribute it further to other entities then it can result in serious consequences if it does not check for the presence of virus in the attachment.

An attacker can attach extra attachments to the original MIME package so as to make it busy by forcing it to extract these attachments which may result in a DoS attack. In an obfuscation attack, an attacker encrypts content of an attachment and places <xenc:EncryptedData> element inside the <wsse:Security> header. If this encrypted attachment contains malicious code then it becomes difficult to scan it.

3.4 Frankenstein Message: Modify Timestamp

A common concern in message-oriented systems relates to the timeliness of data. To handle the time-related issues Microsoft introduced the wsu:Timestamp element in their implementation of WS-Security. By knowing the creation and expiration time, a receiver can decide if the data is new or stale. Following elements appear inside Timestamp element:

Created: The time when the message was created.
Expires: Identifies when the message should expire.

The replaying captured message would result in a SOAP fault if it is sent past an expiration time. An attacker can modify "Expires" field and then send the message successfully. To mitigate such scenario, digital signature over timestamp is used. The XML Re-Writing attack comes handy for an attacker to send stale messages.

Figure 2: Signature section of stale message.

As shown in figure 2, we injected a new Timestamp node immediately under Security element. “Expires” element bears the value one day ahead in future. We wrapped original Timestamp element inside Dummy node qualified with “mustUnderstand=0”. The Signature section refers to the original timestamp element whose id equals “Timestamp-553996f7-762b-4ff9-a0e4-5aeb1b4d53bf” that has been moved intact inside Dummy node. Again, server does not make sure that the signed timestamp element is the one against which server’s timestamp is compared. We could send this message successfully without causing any SOAP fault.

3.5 Attack on WS-Security

The WS-Security standard specifies the use of XML Encryption and Signature operations for a SOAP message which can be applied to the SOAP message in either order.

Assume that the message is encrypted first and then signed. Let’s assume an online marketing company that sells products online. There are several registered sellers who sell goods through the aforementioned company. Company has made a provision for these sellers to upload details of their products they offer through a Web Service. Also, each registered supplier is provided a digital certificate. All the sellers have a copy of the digital certificate of the marketing company with them. The Body of a SOAP message contains product details which are encrypted with the public key of an online marketing company so that it is not legible to unintended recipients.

Figure 3: Genuine Encrypted and Signed Message.

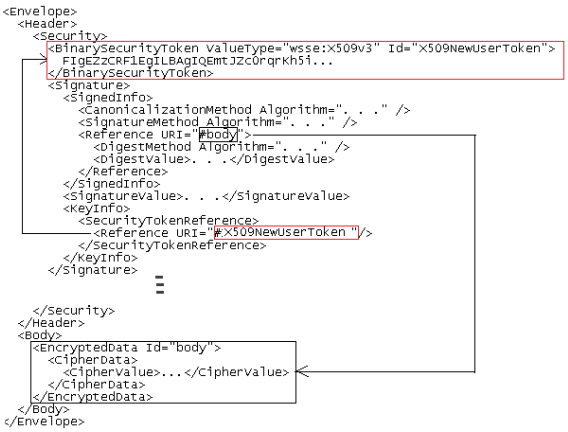

Figure 4: Message with modified Signature.

Figure 3 shows a genuine message sent by a seller whose certificate is identified by ‘X509OrigUserToken’. Now, let’s assume that there exists a seller with mala fide intentions who possesses a digital certificate identified as ‘X509NewUserToken’. The malicious seller captures SOAP message sent by the other seller and removes a signature node from the security header and embeds his own signature node to sign original message body using his own key as shown in the figure 4. On processing this message, server assumes as if message has arrived from the malicious seller because the key used for signature points to the malicious seller’s certificate.

3.6 Attack on WS-Trust, WS-SecureConversation

The Web service may expect an incoming SOAP message to prove a set of claims. If the requestor does not have the necessary tokens to prove the

required claims to a service, it can contact an entity called “security token service” to obtain a security token using a mechanism specified by WS-Trust. The requestor can now produce obtained security token to get an access to the web service. The WS-SecureConversation specifies the use of such obtained token for multiple message exchange between the client and a web service.

```
<soap:Body wsu:Id="Id-af446ec7-1096-46f9-9721-7b8059230867">
  <wst:RequestSecurityToken>
    <wst:TokenType>
      http://schemas.xmlsoap.org/ws/2004/04/security/sc/sct
    </wst:TokenType>
    <wst:RequestType>
      http://schemas.xmlsoap.org/ws/2004/04/security/trust/Issue
    </wst:RequestType>
  </wst:RequestSecurityToken>
</soap:Body>
```

Figure 5: Request Security Token Message.

The client requests for security token by sending RST (Request Security Token) message to the token service as shown in figure 5. Token Service sends back token response containing secret to be used for subsequent message exchange.

3.6.1 Attack: Changing Service End-point

The Requestor may include “AppliesTo” element inside “RequestSecurityToken” message. This element specifies the web service for which the token is being requested. An obtained security token can only be used to communicate with the web service identified by “AppliesTo” element. The token service would include the same “AppliesTo” element in a response when issuing the security token.

The figure 6 shows the token service response after modification. Issuing service has included “AppliesTo” element in signature calculation. In this attack, an attacker changes “AppliesTo” element of a response so that the client discards obtained security token and there won’t be any context establishment with the Web Service. An attacker would add a Dummy node qualified with “mustUnderstand=0” and move original “AppliesTo” element inside it. The signature verification would be successful as original “AppliesTo” element is still present though it has been moved inside Dummy node. However, an injected new “AppliesTo” element no longer points to the web service for which token was requested (Forged node points to http://www.Hacker.com/NoStockWatch.asmx web service). The token requestor would discard the token since it didn’t receive token corresponding to the service it asked for (http:// www.myservice.com/StockWatch.asmx).

```
<Envelope>
  <Header>
    <Action mustUnderstand="1" Id=" 2">
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT
    </Action>
    <RelatesTo Id="_3">urn:uuid:87d07326-7663-4b4b-8df6-162133c2ec60</RelatesTo>
    <AppliesTo Id="_AppliesToForged">http://www.Hacker.com/NoStockWatch.asmx</AppliesTo>
    <Dummy mustUnderstand="0">
      <AppliesTo Id="_AppliesToOrig">http://www.myservice.com/StockWatch.asmx</AppliesTo>
    </Dummy>
    <Security mustUnderstand="1">
      <Timestamp>.....</Timestamp>
      <Signature>
        <SignedInfo>
          <Reference URI="#_0">.....</Reference>
          ..............
          <Reference URI="#_AppliesToOrig">.....</Reference>
        </SignedInfo>
        <SignatureValue>V5chavlKdb+ttjSbHP7ZLAU+BEQ=</SignatureValue>
        <KeyInfo>.....</KeyInfo>
      </Signature>
    </Security>
  </Header>
  <Body Id="_0">
    <RequestSecurityTokenResponseCollection>
      <RequestSecurityTokenResponse>
        <TokenType>
          http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
        </TokenType>
        <RequestedSecurityToken>·····</RequestedSecurityToken>
        <Entropy> ····· </Entropy>
        <Lifetime>.....</Lifetime>
        <KeySize>256</KeySize>
      </RequestSecurityTokenResponse>
    </RequestSecurityTokenResponseCollection>
  </Body>
</Envelope>
```

Figure 6: Modified ‘AppliesTo’ element.

4 MITIGATION TECHNIQUES

Here, we shall see SOAP message validation mechanism along with self-adaptive hardened schema generation algorithm. Also, we shall see solution to thwart SOAP attachment based attacks.

4.1 Schema Validation

To mitigate issues such as XML Injection and *SOAPAction* manipulation discussed before, we need an extra layer of input validation in addition to the default validation functionality provided by the web service engine. We implemented a solution in the form of SOAP Extension that works with Internet Information Services (IIS) server. We developed a tool that scans through a WSDL file to identify SOAP ports and then associated operations along with the value of *SOAPAction*. Having identified operations, it generates a set of Xml Schema Definition (XSD) corresponding to each operation by processing schema section of WSDL file. These generated XSDs are then used by our extension for performing validation.

Our extension captures the SOAP message when it is available in a raw XML form and subjects it to the validation against XSDs before it gets converted to native objects with wrong values. Our Extension maintains repository of schemas corresponding to each operation stored according to *SOAPAction* attribute. On intercepting incoming SOAP request, extension obtains *SOAPAction* HTTP header value. Then it uses this value to perform lookup in to schema repository to extract appropriate schema. The message is then subjected to validation against retrieved schema. If it is learnt that the schema is violated then it may be an instance of attack vector and we do not allow it to be de-serialized and

prevent it from reaching a web service. This technique also helps us thwart any discrepancy between *SOAPAction* and message body as schema is retrieved in accordance with the value of *SOAPAction* attribute.

In addition to validating messages against schemas, extension should also do the following:

1. An XML node can have text content along with CDATA section within it. The resultant text of a node is the concatenation of content inside CDATA and an actual text content of the node. This resultant text is matched against a set of regular expressions that check a presence of potentially dangerous JavaScript or html mark-up.
2. For XML-Rewriting, mitigation technique would be to incorporate check that makes sure that signed nodes do not reside within a node marked with 'mustUnderstand=0'.

For IIS, such a solution can be implemented in the form of SOAP Extension as demonstrated earlier. For servers like apache/tomcat, we can write an input module which does the similar job.

4.2 Self Adaptive Schema Hardening

Efficacy of validation mechanism largely depends on an accuracy of schemas used. Easiest way would be to manually harden schema mentioned in WSDL file. The quality schema allows all good requests to pass validation while blocks bad requests. Such a schema can be obtained by scanning through the SOAP messages encountered by the web service and categorizing them into bad and good requests. Then an XML schema (XSD) can be hand crafted that would allow good messages to pass validation. However, this process is cumbersome and an error prone as schema has to take into account all likely patterns of the good messages. Here, we propose a theoretical model of self-adaptive algorithm to automate hardened schema generation process. Our algorithm has its roots in the fact that XSDs corresponding to a good SOAP requests would either be same or differ insignificantly. These good schemas can be merged together to obtain a single schema. The web server extension as described before would log all incoming SOAP messages in a data store. Extension (Schema Validator) shall initially rely on XSD Schemas generated from a WSDL file (using our tool as mentioned before). The goal is to deduce a set of schemas based on the measure of similarity among messages. Figure 7 shows overall architecture of the system.

This algorithm shall run periodically and operates as follows:

1. *Sampling:* Pick subset of SOAP requests from the corpus of SOAP requests logged by an extension.
2. *Schema Generation:* SOAP message comparison is easier in XSD domain then in XML domain. Hence, we generate schema (XSDs) from each of the sampled SOAP request.
3. *Cluster Messages:* Messages sharing identical or similar generated schema become part of the same cluster. Corresponding schemas are merged together to obtain a resultant schema that would become representative of the cluster.
4. *Select Winner Cluster:* All good requests would have been mapped to the same cluster due to the fact that they all share similar schemas. Hence, cluster representing relatively large number of SOAP requests would be considered winner cluster.
5. *Refine Schemas stored in repository:* Fine tune schemas installed in a repository in line with the schema corresponding to the winner cluster.
6. *Purge* logged SOAP requests that have already been processed.

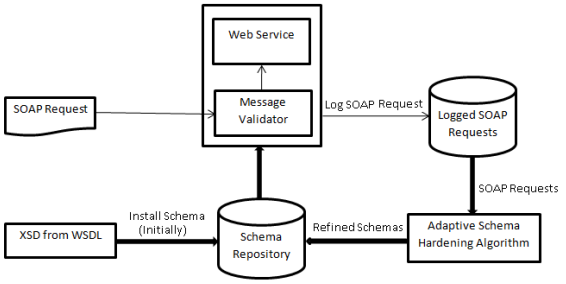

Figure 7: Schematic of Adaptive Schema Hardening Algorithm.

Web Services are often used with standards like WS-Security, WS-Routing etc. The use of these standards introduces additional nodes within SOAP header and body. However, schemas stated in WSDL do not necessarily highlight presence of these nodes in SOAP messages. Our algorithm helps refine schema to include such nodes which would be difficult to anticipate otherwise if done manually. Also, it replaces generic data types used with the concrete data types based on the values of operation parameters encountered while processing logged SOAP messages.

4.3 Thwart SOAP Attachment Attacks

The best mitigation would be to have a SOAP message interceptor that scans incoming SOAP request for the presence of an attachment. Then extract and scan attachment for the presence of virus signature before being given to a web service. We used open source ClamAV antivirus to scan attachments. To prevent inclusion of extra attachments, mitigation would be to ensure that all attachments are signed. If any attachment referred from SOAP message is not found to be signed then it's probably an instance of an attachment insertion attack.

Table 1: Summary of Attacks and Mitigation Schemes.

Attack on/through	Newly mentioned	Mitigation by others	Mitigation by us
XML Injection	No	Yes	Yes[1]
XSS Injection	No	Yes	Yes
XML Re-Writing	No	Yes	Yes[2]
Header manipulation	No	No	Yes
SOAP Attachments	No	Yes	Yes[3]
Frankenstein Message	Yes	-	-
WS-Security	Yes	-	-
WS-Trust	Yes	-	-

[1] Our proposed solution also highlights mechanism to automate the process of hardened schema generation.

[2] We have shown the check that makes sure that the signed nodes do not reside within a dummy node.

[3] We have specifically described the use of open source antivirus ClamAV for SOAP attachment scanning.

5 FUTURE WORK

Many of the attacks discussed above are direct consequence of lack of thorough validation. The XML Injection and Header Manipulation attacks can be mitigated if we have strong validation logic in place which in turn depends on quality of schema. Our future work will focus on materializing efficient schema hardening algorithm.

6 CONCLUSIONS

Table 1 highlights our contribution. In this paper, we have shown injection based attacks. We have also introduced Frankenstein message attack and attacks on WS-Security and WS-Trust standards. We have suggested mitigation techniques for subset of these attacks. From the attacks discussed, it is apparent that the mere use of security primitives does not always evade all possible attacks. Use of these security and other standards in a mature way can supress new kind of attacks. Also, we have introduced self-adaptive schema hardening algorithm to automate the process of hardened schema generation.

REFERENCES

Lindstrom, P., 2004. "Attacking and Defending Web Services", *Spire Research Report*.

Vorobiev, A., 2006. "Security Attack Ontology for Web Services", *IEEE Proceedings of the Second International Conference on Semantics, Knowledge, and Grid.*

Gruschka, N., 2009. "Vulnerable Cloud: SOAP Message Security Validation Revisited", *IEEE International Conference on Web Services*.

Negm, W., 2004. "Anatomy of a Web Services Attack", *Forum Systems.*

McIntosh, M. and Austel, P., 2005. "XML signature element wrapping attacks and countermeasures", *In Workshop on Secure Web Services.*

Gajek, S., Jensen, M., Liao, L., and Schwenk, J., 2009. "Analysis of signature wrapping attacks and countermeasures", *In IEEE International Conference on Web Services.*

Jensen, M., Gruschka, N., Herkenhoner, R., Luttenberger, N., 2007. "SOA and Web Services: New Technologies, New Standards – New Attacks", *Fifth European Conference on Web Services.*

Gruschka, N., and Luttenberger, N., 2006. "Protecting Web Services from DoS Attacks by SOAP Message Validation", *In Proceedings of IFIP International Federation for Information Processing*, pp 171–182.

Orrin, S., "The SOA/XML Threat Model and New XML/SOA/Web 2.0 Attacks & Threats", *Intel Corporation.*

Bidou, R., 2009. "Attacks on Web Services", *OWASP*.

Testing for XML Injection (OWASP-DV-008), [online], Available: http://www.owasp.org/index.php/Testing_for_XML_Injection_(OWASP-DV-008) "*Web Services Security: SOAP Message Security 1.0*", OASIS Security Standard, March 2004.

"*WS-SecureConversation 1.3*", OASIS Standard, March 2007.

"*WS-Trust 1.3*", OASIS Standard, March 2007.

Understanding WS-Security, [online], Available: http://msdn.microsoft.com/en-us/library/ms977327.aspx

"*Web Services Security: SOAP Messages with Attachments (SwA) Profile 1.1*", OASIS Standard, Feb 2006.

ClamAV Anti-Virus, [online], Available: http://www.clamav.net/