Indian Journal of Science and Technology, Vol 9(29), DOI: 10.17485/ijst/2016/v9i29/91685, August 2016 ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Custom Security in Web Services

Balika J. Chelliah[1*], K. Vivekanandan[2] and P. Jeni[3]

[1]Department of Computer Science and Engineering, SRM University, Chennai - 603203, Tamil Nadu, India; balika888@gmail.com
[2]Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry - 605014, Tamil Nadu, India; k.vivekanandan@pec.edu
[3]Department of Computer Science, and Engineering, SRM University, Chennai - 603203, Tamil Nadu, India; jenipadhu@gmail.com

Abstract

Background/Objectives: Service oriented Architecture (SOA) infrastructures using web services are deployed by many firms worldwide. Web Services provide a standard means of inter-operation between heterogeneous software applications that run on a variety of platforms. Most of the web services are offered with HTTP over Simple Object Access Protocol (SOAP) as the underlying infrastructure. The greatest web security threat is accepting the request from the client without proper validation. The objective is to separate the application logic and the security or validation procedures which offers more advantage for software reuse since it is not necessary to recompile, when the validation or security requirements change. **Methods:** An Interceptor is created for validation which has the token based authentication procedures along with the steps for validating the data. The system is devised in such a way that the business logic will be triggered if and only if the data is validated and passed by the interceptor procedures. **Findings:** The proposed system provides a way to keep the validation and security mechanism out of application logic and hence this does not modify the existing functionality. Thus, combining all custom security as one unit of validation before hitting the business logic is the basic idea of the proposed system.

Keywords: Custom Security, SOA, Validation Model, Web Service

1. Introduction

Web services may expose business critical systems and information and hence a proper security should be applied to it. Though security is implemented through many standards, policies, firewalls, XML security standards, XML encryption, XML signatures, it depends on the developer who utilizes the security concepts along with the business logic. Hence, the procedure that secure web services against unwanted input are therefore given high importance. Mostly, developers who work on business logic tend to neglect validation. Since, they are keen in implementation of the logic and sometimes given least priority to security considerations which shows the security issues are not under centralised control.

A tool has been created for verifying generated schemas and an self-adaptive schema hardening mechanism which makes the restriction in schema level so that the attacks on webservices such as XML injection, XSS injection and HTTP header manipulation, timestamp[1]. Security is insisted by applying XML encryption and signature for data in transaction and in storage form[2]. To prevent the Distributed Denial of Service attack (DDoS), a XSD DDoS trace handler and a totient encryption algorithm is used[3]. A validation tool is created for the configuration file definition[4]. An input validation model is created which compares the schema with a predefined standards and has the concept of building it against the request. This does not carry any authentication information or a signature to verify the client authenticity[5].

Using local copies of schema to validate the xml[6]. Generic flooding attack is solved based on client puzzles[7]. Certificate generation, attestation key, private key usage is determined on authentication for web based ser-

*Author for correspondence

vices[8]. Federated identity and three step process for WS authentication and authorization in a maritime environment[9]. A metering system is created for WS security based implementation[10]. A reusable approach by the use of XML files and an XML schema for the security parameter specification is created for validation[11]. An input validation framework for validating the schema in web service is implemented[12]. A capture, comparison, and decision making module are defined for preventing Denial of service attack in web service. A token system was created to prevent DDoS attack in REST web services[13].

Threshold value, selection, payload and authenticated connection to counter DDoS attacks[14]. A rule based WS input validation is created for schema[15]. A constraint level validation manager is defined for a run time monitoring and validation framework for WS interaction[16]. A security API is defined for a security solution in a dynamic composition scenario[17]. A load balancing and a protecting environment for a Linux based OS to prevent DDoS Attacks[18]. A strong authentication for web services using a smart card system[19]. A public WS security framework is defined through product generation[20]. Message replay attack, over sized payload, coercive parsing and XML based injection attacks are experimentally tested with separate algorithms[21]. Parses that function based on grammars and permuttions[22]. Without modifying the complete system, service oriented architectures with webservices are used for easy integration of services[23].

Among the related works, there is a lack of studies specially addressing both authentication and validation together distinguishing from the business logic. Thus, in the proposed system, we try to cover all the security and validation issues in the separate model before hitting the business logic. This system will have a token based system for authentication along with traditional username/pwd, and further checks the client IP, no of attempts and an Interceptor for validation.

2. Proposed System

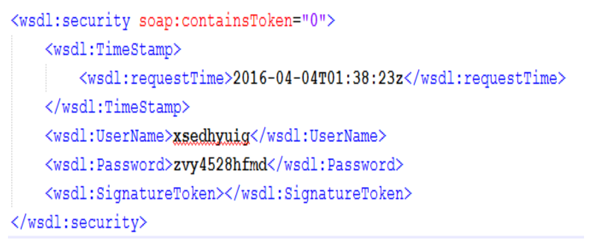
In the proposed system, the interceptor (INT) which has a token based authentication along with the traditional username/password system is being used to confirm the client authenticity. Once the authentication scheme is passed, the request is passed to the sanitizer module. If all the validation is passed in the sanitizer module, then the business logic is invoked. Imagine, the service requester finds the intended service using the registry and hence it binds with the provider.

Figure (1) shows the interceptor will authenticate and validate the input requests and hits the service only when the request is valid. The process involved in Interceptor are Registration, Signature Verification, IP Verification, Attempts Check , Sanitizer.

Registration:

The new tags which is added for this process are requestTime (login time of the req), containsToken (valid values 0,1), SignatureToken ST_{client}.

```
If(containsToken == 0)
        Perform Registration(username,pwd);
Else if(containsToken == 1)
        Perform SignatureValidation(username, pwd,STclient);
```



```
<wsdl:security soap:containsToken="0">
    <wsdl:TimeStamp>
        <wsdl:requestTime>2016-04-04T01:38:23z</wsdl:requestTime>
    </wsdl:TimeStamp>
    <wsdl:UserName>xsedhyuig</wsdl:UserName>
    <wsdl:Password>zvy4528hfmd</wsdl:Password>
    <wsdl:SignatureToken></wsdl:SignatureToken>
</wsdl:security>
```

```
Registration(username,pwd)
[Whitelist:LoginTime:$L_T$,Token/seed,LoginIP:$L_{IP}$]{
INT sends Token ;
Records $L_T$, Token / seed, $L_{IP}$ in whitelist;
}
```

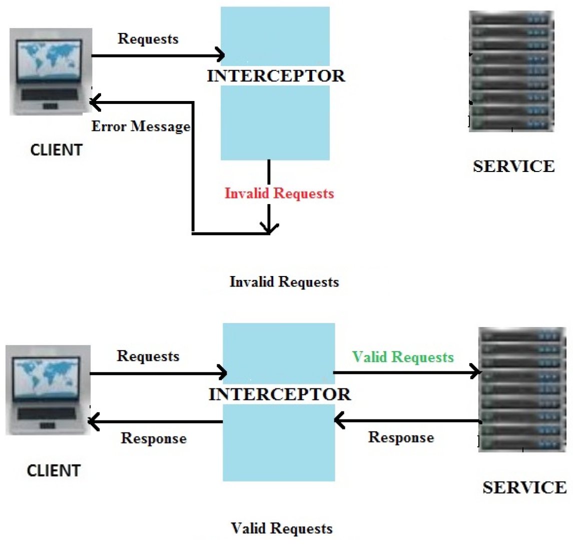
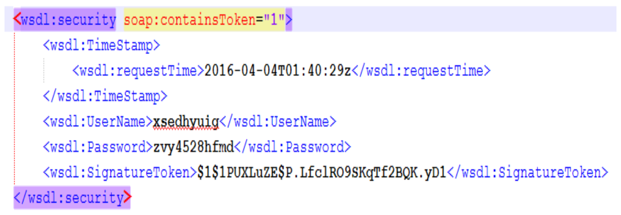


Figure 1. Operations.

The token will be generated based on the concept of cryptographically secure pseudo random number generator. Either token or seed used to generate the token is saved in the whitelist. The client receives the token and computes the SignatureToken.

SignatureToken = MD5(Token+pwd)



```
<wsdl:security soap:containsToken="1">
    <wsdl:TimeStamp>
        <wsdl:requestTime>2016-04-04T01:40:29z</wsdl:requestTime>
    </wsdl:TimeStamp>
    <wsdl:UserName>xsedhyuig</wsdl:UserName>
    <wsdl:Password>zvy4528hfmd</wsdl:Password>
    <wsdl:SignatureToken>$1$1PUXLuZE$P.LfclRO9SKqTf2BQK.yD1</wsdl:SignatureToken>
</wsdl:security>
```

Whitelist will also contains other attributes such as username,pwd,Threshold limit –T_L, Last Request Time – L_{RT}, Timestamp skew – TS_S, No of Attempts – NoA, Interval – I_T.
Default validity of the token / seed is 12 hours & if NoA reaches threshold, it will expire.

Rather than using traditional authentication schemes, using a signatureToken helps in addressing Denial of Service attack. Even if the frequency of the request is increased by the attacker, the token/seed will be valid only for the certain time period or till it is active.

Signature Verification

It verifies the client authenticity by checking the value in the SignatureToken tag received from the request.

```
SignatureValidation(username,pwd, $ST_{client}$)
{
        1.      Consider Token / seed from Whitelist
        2.      If (seed)
                Generate Token;
                Else if (Token)
                Perform step 3.
        3.      $ST_{INT}$ = MD5 (Token + pwd);
        4.      If ($ST_{client}$ = $ST_{INT}$)
        {
                Success;
                Perform IPverfication($IP_{client}$);
        }
        Else
        {
                Invalid Requests;
                Error message to client;
        }
}
```

The option of selecting a hash algorithm such as MD5 is making the computation easier rather than sending or maintaining a public / private key pair and again have to apply another cryptographic algorithm to secure it.

IP Verification

It verifies whether the client login IP is valid from both whitelist and blacklist. Blacklist contains the set of IP address which caused threat to the system.

```
IPVerfication($IP_{client}$)
{
        1.      Consider $IP_{whitelist}$, $IP_{blacklist}$.
        2.      If[($IP_{client}$ = $IP_{whitelist}$) || ($IP_{client}$ != IPblacklist )]
        {
                        Success;
                        AttemptsCheck();
                }
                Else
                {
                        Invalid Requests;
                        Error message to client;
                }
}
```

The above processes rejects the unauthorized entry.

Attempts Check

This process evaluates the request on the timestamp related information based on the algorithm given below.

Constraints:

a. Time references should be in standard timeformat (UTC).

b. Time references are recommended in xsd:dateTime format. If any other format, it should be specified in ValueType attribute

c. Threshold Limit:

$$T_L < \sum_{i=0}^{t1} (Requests)\, t_1, IP \quad (for\ heavy\ traffic)$$

$$T_L < \sum_{i=0}^{t1} (Requests)\, t_1, IP / n \quad (Average\ traffic)$$

$$T_L < \sum_{i=0}^{t1} (Requests)\, t_1, IP, k \quad (Requests\ from\ same\ IP)$$

n – No of requested URI during time period t_1

k – particular k URI from same IP.

d. TS_S - maximum tolerance limit for the clock skewed between the sender and the receipient.

e. I_T - allowed time gap between L_{RT} and L_T.

```
AttemptsCheck()
    {
        Current time of INT - $C_T$
        If [$C_T < (L_T – TS_S * 1000)$] //1000 – a constant
        that can be changed.
    {
        Invalid Requests;
        Error message to client;
    }
    Else
    {
    If [$(L_T - L_{RT}) > I_T$ ]
    {
    If(NoA < $T_L$)
        Sanitizer();
    Else
    {
        Invalid Requests;
        Error message to client;
        }
    }
  }
}
```

This will avoid the message replay attack.

Sanitizer

The sanitizer must maintain the schema caching process which is defined as maintaining the local copies of the schema. This copy needs to be refreshed whenever an update is made to the schema. The local schema repository can be created as simple as creating a folder by using namespaces names to provide guidance on naming the folders.

This process tries to harden the schema as possible to prevent it from attacks. Hence, the sanitizer maintains whitelist with a set of valid parameters that needs to validate the client request against XML specification, which limits XML injection and oversize payload attacks.

The whitelist should contain,

Element Names / Parameter Names - *All the elements used in the schema should be listed.*

Data Type – *The data type of each element name should be defined.*

Size – *The size of each element along with its maximum and minimum size restrictions should be listed.*

Operation Names – *The operation names present in the schema should be listed.*

ComplexTypes – *The complex types should be mentioned with the number of occurrence of this block.*

Unbounded Occurrence – *On designing the schema, the unbounded occurrence should be restricted as much as possible, as this will be the root cause of coercive parsing attack. Hence, we prefer to restrict this occurrence depending on the type of application used. (Say, if a flight reservation / bus booking, we may not require a parameter with this unbound occurrence.)*

Regular Expression – *specified for validation.*

An attack of namespace injection can occur and hence the namespaces are monitored along with the tags that are defined or valid for each namespace. These can also be included in the whitelist for validation.

Policy Enforcement should not be made manual and hence the system will enforce specified input policies for a defined set of services and ports.

Verify the XSDPath from the XML request. If the path is empty, the request can be rejected. Else, check for the startElement and EndElement. If it is empty then the request can be rejected and the error message can be sent to the client. It avoids Parameter tampering attack.

If needed, the size of the request message is also defined. Hence, if the size of the total request exceeds the size defined, we can reject the client request, which will handle coercive parsing attack.

Thus the sanitizer validates the data against the XML specifications and maintain the data completeness, correctness and structure.

3. Experimental Results

The proposed work is implemented in Java and weblogic server. The system results are obtained and compared in the form of CPU usage.

A comparative study is done without the implementation of this approach and with this implementation of the proposed system.

Figure (2) shows the performance boost up with the implementation of this proposed system compared to the system which does not implement this module.

Figure (3) shows the CPU usage of SOAP based web services for 100,300,500 number of requests. It shows CPU usage is reduced using the existing algorithm.

4. Figures

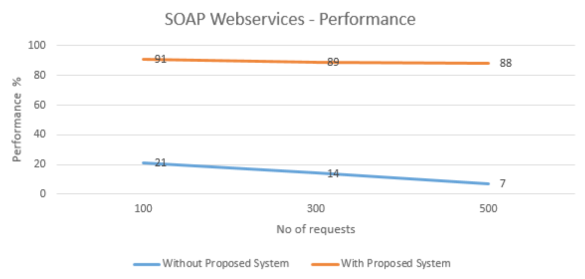


Figure 2. Performance – SOAP.

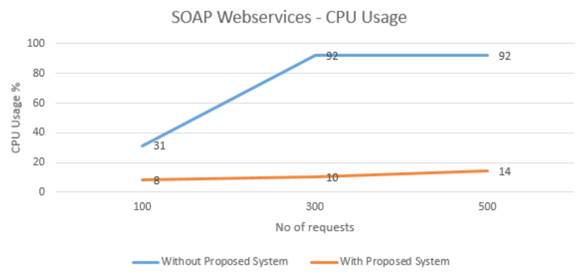


Figure 3. CPU usage – SOAP.

5. Conclusion and Future Works

Results from the proposed system shows that there is a steady performance and reduction in CPU usage when compared to the services which did not use the same.

The future work includes making the system function for REST web services with few modifications and can try making the references with DB index instead of maintaining a whitelist or a blacklist. Since, the REST web service does not have a predefined schema, we need to generate the schemas based on the input and try validating it via the DB index within the specified process of interceptor.

6. References

1. Patel V, Mohandas R, Pais AR. Attacks on web services and mitigation schemes. Proceedings of the 2010 International Conference on Security and Cryptography (SECRYPT). 2010 Jul 1–6.
2. Menaka R, Wahida Banu RSD, Ashadevi B. Survey on Signatured Xml Encryption for Multi-Tier Web Services Security. Indian Journal of Science and Technology. 2016
3. Murugan A, Vivekanandan K. Xsd ddos trace handler in web service environment. Journal of Software. 2015; 10:086–1095.
4. Gupta AN, Thilagam DPS. Attack on web services need to secure xml on web. Computer Science Engineering: An International Journal. 2013; 03:1–11.
5. Brinhosa RB, Westphall CM, Westphall CB. Proposal and development of the web services input validation model. IEEE Network Operations and Management Symposium (NOMS). 2012; 03:262–6.
6. CDISC, XML Schema Validation for Define.xml, CDISC INC.
7. Suriadi S, Stebila D, Clark A, Liu H. Defending web services against denial of service attacks using client puzzles. 2011 IEEE International Conference on Web Services (ICWS). 2011; 01. p. 25–32.
8. Sheng Y, Lu Z. A online user authentication scheme for web-based services. Business and Information Management, 2008. ISBIM '08. International Seminar. 2008; 02:173–6.
9. Kim A, Khashnobish A, Kang M. An architecture for web services authentication and authorization in a maritime environment. International Conference on Information Technology, IEEE. 2007; 14. p. 560–6.
10. Auletta V, Blundo C, Cimato S. Authenticated web services: A wssecurity based implementation. European Commission through the IST program under Contract IST-2002-507932. 2002; 01:1596–608.
11. Brinhosa RB, Westphall CB, Westphall CM. A security framework for input validation. The Second International Conference on Emerging Security Information, Systems and Technologies. 2008; 01. p. 88–92.
12. Jensen H. Input Validation Framework for Web Services. NTNU Innovation and creativity.
13. Lad N, Baria J. Ddos prevention on rest based web services. International Journal of Computer Science and Information Technologies. 2014; 05:7314–7.
14. Prabu SS, Kumar DVS. Countering the ddos attacks for a secured web service. Indian Journal of Computer Science and Engineering. 2013; 04:149–54.
15. Kalman M. Rule-based web service validation. 2014 IEEE International Conference on Web Services (ICWS). 2014; 01. p. 542–9.
16. Li Z, Jin Y, Han J. A runtime monitoring and validation framework for web service interactions. Proceedings of the 2006 Australian Software Engineering Conference. 2006; 01. p. 79–89.
17. Sindhu S, Kanchana R. Security solutions for web service attacks in a dynamic composition scenario. IEEE

International Conference on Advanced Communication Control and Computing Teclmologies. 2014; 01. p. 624–8.

18. Kargl F, Maier J, Weber M. Protecting web servers from distributed denial of service attacks. Proceedings of the 10th International Conference on World Wide Web. 2001; 10. p. 514–24.
19. Stienne DS, Clarke N, Reynolds P. Strong authentication for web services using smartcards. Proceedings of the 7th Australian Information Security Management Conference. 2013; 03. p. 55–62.
20. Thelin J, Murray PJ. A Public Web Services Security Framework based on Current and Future usage Scenarios. Proceedings of the International Conference on Internet Computing.
21. Uma E, Kannan A. Self-aware message validating algorithm for preventing XML-based injection attacks. International Journal of Technology and Engineering Studies. 2016; 2(3):60–9.
22. Zhang W, van Engelen RA. High-Performance XML Parsing and Validation with Permutation Phrase Grammar Parsers. IEEE International Conference on Web Services. 2008.
23. RajKumar N, Vinod V. Integrated Educational Information Systems for Disabled Schools via a Service Bus using SOA. Indian Journal of Science and Technology. 2015.