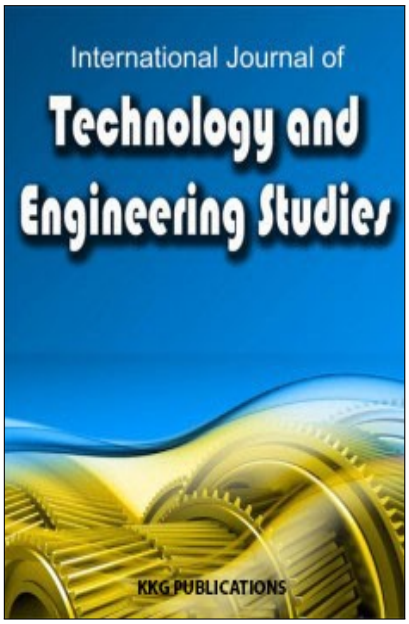
This article was downloaded by:
Publisher: KKG Publications



Key Knowledge Generation
Publication details, including instructions for author and
subscription information:
http://kkgpublications.com/technology/

Self-Aware Message Validating Algorithm for Preventing XML-Based Injection Attacks

E. UMA[1], A. KANNAN [2]

Department of Information Science and Technology Anna University Chennai-25, India

Published online: 18 June 2016

To cite this article: E. Uma, A. Kannan, "Self-aware message validating algorithm for preventing XML-based injection attacks." *International Journal of Technology and Engineering Studies,* vol. 2, no. 3, pp. 60-69, 2016.
DOI: https://dx.doi.org/10.20469/ijtes.2.40001-3

To link to this article: http://kkgpublications.com/wp-content/uploads/2016/2/Volume2/IJTES-40001-3.pdf

PLEASE SCROLL DOWN FOR ARTICLE

KKG Publications makes every effort to ascertain the precision of all the information (the "Content") contained in the publications on our platform. However, KKG Publications, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the content. All opinions and views stated in this publication are not endorsed by KKG Publications. These are purely the opinions and views of authors. The accuracy of the content should not be relied upon and primary sources of information should be considered for any verification. KKG Publications shall not be liable for any costs, expenses, proceedings, loss, actions, demands, damages, expenses and other liabilities directly or indirectly caused in connection with given content.

This article may be utilized for research, edifying, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly verboten.

KKG PUBLICATIONS

International Journal of Technology and Engineering Studies
vol. 2, no. 3, pp. 60-69, 2016

IJTES

SELF-AWARE MESSAGE VALIDATING ALGORITHM FOR PREVENTING XML-BASED INJECTION ATTACKS

E. UMA [1*], A. KANNAN [2]

[1, 2] Department of Information Science and Technology Anna University Chennai-25, India

Keywords:
XML
Web Services Security
Injection Attacks

Received: 06 February 2016
Accepted: 04 April 2016
Published: 18 June 2016

Abstract. A new XML-based injection filter is proposed in this research work to prevent and detect injection attacks. The validation approach is used as a security mechanism in this research work and presents an XML-based injection filter framework. The self-aware message validating algorithm estimates the causes of incoming queries dynamically. The proposed algorithms have been tested and evaluated against various XML-based attacks. The conventional firewall and filters are lacking in detecting and preventing XML-based attacks because these attacks contain huge volumes of data and are cluttered in nature. Therefore, a new XML-based injection filter is proposed in this research work to prevent and detect attacks. The results show that the algorithm is very robust against attacks. This filter prevents significant attacks by using a parameter tampering filter, coercive parsing filter, oversized message filter, message replay filter, and semantic URL filter.

INTRODUCTION

Web Service attacks, generally called XML-based attacks, occur at the SOAP message level and thus they are not readily handled by existing security mechanisms in earlier firewalls. So as to provide robust security mechanisms for Web Services, XML filters have recently been introduced for Web Services security. In this research, a framework for dynamic XML filters is proposed, called self-aware message validating filter for XML-based attacks, which supports detection and protection of XML-based attacks in real-time.

A detailed design of the injection filter security model has been provided by validating schema information of the message with detection and protection policies. An oversized message attack is a type of flooding attacks, where an attacker creates enormous level of traffic to a Web Service to exhaust its resources at the server side and parameter tampering attack can crash the server by sending unacceptable parameters.

In this research work, the validation approach is used as a security mechanism and presented a framework for XML-based injection filter.

RELATED WORKS ON XML INJECTION FILTER FOR WEB SERVICES

The dynamic programming algorithm used has its roots in an algorithm introduced by [1]. The sequence comparison problem has been mapped to shortest path problem in edit graph. Now, the problem of finding a minimum-cost edit script between two sequences is reduced to the problem of finding a shortest path from one end of the edit graph to the other. DTD schema comparison solution [2] was extended to XML Schema Definition (XSD) schemas [3]. This work takes into account semantic similarity of element as well as attributes names in addition to considering structural similarity.

[4] detailed the importance of mod-security apache server module to safeguard Web Services. The author has shown configuring mod-security to extract required parameters from the SOAP request using regular expressions and then throw away requests containing suspicious values.

[5] attempted to integrate XML firewall with existing Web Services security specifications.

[6] described the threat profile of Web Services environment. A set of conceptual attacks was introduced by another to compromise Web Services.

[7] provided a comprehensive guide to security for Web Services and Service-Oriented Architecture (SOA). They explained all recent standards that address Web Service security standards, as well as recent research areas on access control for simple and conversation-based Web Services and access control for Web-based work flows.

ARCHITECTURE OF XML-BASED INJECTION FILTER SERVICE

The architecture of the XML-based injection filter service model is illustrated in Figure 1. As shown in the figure, an injection filter lies between service consumers and a service provider, and can be installed either on the same or a different machine where the actual Web Services are deployed. It interacts with service consumers through its User Interface (UI), which is responsible for receiving requests from and sending responses back to the login service.

*Corresponding author: E. Uma
†Email: euma@annauniv.edu

© 2016 The Author(s). Published by KKG Publications. This is an Open Access article distributed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

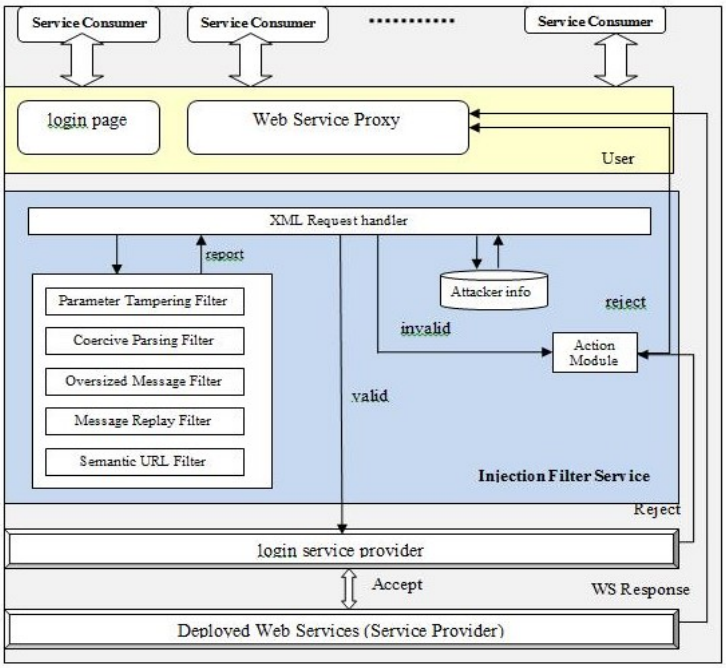


Fig. 1 . A framework for injection filter service

There are five major components supporting the filters, namely oversized message filter, message replay filter, parameter tampering filter, coercive parsing filter and semantic URL filter, which process the incoming request and state-based information, respectively. In the injection filter security model, input validation and protection are the major features for providing user access control, which ensures that only valid users are allowed to access certain Web Services.

Parameter Tampering Filter

The attacker adjusts the parameters in a SOAP message in an attempt to redirect the input validation in order to access unauthorized information. A change in integrity of the parameters is to detour the input validation and gain unauthorized access of some confidential functionality of Web Services. This filter checks the XML schema definition of received message for data type, null values. This filter checks the parameter for valid data. If it fails, then, it throws an error to the sender once.

Coercive Parsing Filter

This filter verifies the namespaces and version mismatch received in the WSDL and SOAP files. This filtering policy used the fault values in SOAP fault code. The filter verifies the received message for wrong format of SOAP message by generating SOAP fault code. This filter blocks the input that has a strange format.

Oversized Message Filter

The XML parsing of the service provider is directly affected by the size of the SOAP message. As a consequence, large amounts of Central Processing Unit (CPU) cycles are consumed when presented with large documents to process. A hacker can send a payload that is in alarming rate to exhaust system's resources. So, the filter is designed to refine the size of the message, requisition resources presented in the incoming message.

Message Replay Filter

A hacker can resend the SOAP message requests to access the Web Service using other's login credentials. This kind of Web hacking will be escaped as a legitimate request because the source IP address is valid, the network packet attributes are valid and the HTTP request is well-formed. Hence, the filter assigns an identifier for each incoming message and stored into the database to identify the replayed message. After that, the filter catches and matches the identifier of incoming messages and uses the replay detection policy to identify and reject messages which match an entry in the database of replay detection filter.

Semantic URL Filter

The semantic URL attack is when the client manually retypes the parameters of its request by keeping the URL's structure but altering its semantic meaning. This is protected by giving token

and timestamp for expiration.

ALGORITHM OF SELF-AWARE MESSAGE VALIDATION FOR XML-BASED INJECTION ATTACK FILTER

A Web Service communicates with other applications over a network. There is no surety that the incoming message is requested from legitimate user, though the incoming request is coming from authorized IP address. Meanwhile, the intruder includes some parameter to gain some data or to redirect the flow to some proprietary Web links. Based on the input information, the XML Request Handler can detect and verify XML-based attack in real-time.

Detection of XML-Based Injection Attacks

The XML request handler module is responsible for the dynamic detection and verification of the XML-based injection attacks by checking both the SOAP message and the parameters passed to a Web Service operation. The algorithm proposed by the XML request handler module is depicted in Figure 2 and explained how the input is treated as malformed input or not. As shown in the figure, when a SOAP message with a valid user request is sent to the XML request handler module, there the input is refined in all filters to verify the attack.

Algorithm for XML-Based Injection Attack Filter Service	
Input: SOAP message with parameters for service invocation Output: accept / reject	
1.	Receive SOAP message
2.	Switch (Input)
3.	Case Parameter tampering attack:
4.	Verify parameter tampering attack using tamper filtering policies
5.	Case Coercive parsing attack:
6.	Verify coercive parsing attack using parsing filtering policies
7.	Case Oversized message attack:
8.	Verify oversized message attack using oversized message filtering policies
9.	Case Message replay attack:
10.	Verify message replay attack using replay filtering policies
11.	Case URL semantic attack:
12.	Verify URL semantic attack using URL semantic filtering policies
13.	If any attack confirmed
14.	Then forward incoming request to XML request handler
15.	Output reject the attacker's request
16.	Else output is valid and invoke a login service

Fig. 2 . Algorithm for XML-based injection filter

The process of detecting other types of XML-based attacks involves two major steps, which are detection of malformed SOAP messages and protection from attacks [8-10]. Malformed SOAP messages are detected using the SOAP message validator. For example, to detect XML attacks, the handler module analyzes for possible flooding requests and keeps track of the allowable message size and the nesting depth in the incoming XML messages.

Parameter Tampering Filter

In this filter, the received parameters are checked for data type, number of parameters and null values. This filter checks the parameter for valid data; if it fails, then it throws an error to the sender once. Even if the sender continues, his misbehaving with parameters leads to the disconnection of communication. To solve this problem, the proposed XSS filter was created with tamper checker function. It checks the arguments for null values, data type, start element and end element of the received request from the client.

Advantages

This filter protects the server from anomaly parameters and unstructured form of XML document. This assists to guard against the injection attacks, even for business Web Services that do not implement any validation control. The Web Service performs validation autonomously for the client.

Coercive Parsing Filter

This attack is targeted on the standard structure of XML documents. In Web Services, every document must be formatted according to the rules and regulations mentioned in the W3C consortium. If any misplaced namespaces, any disparity in version of SOAP or WSDL document will affect the communication. This malformed attack stops the service provider suddenly, after affected by enormous level of attacks.

Oversized Message Filter

Denial of Service attacks happened by exhausting resources available in server side. Such attacks aim at reducing service availability by exhausting the resources of the service's host system, like memory, processing resources or network bandwidth. It is performed through query, a service using a very large request message, which is called as over sized message. Because of over sized message, the Web Service resources such as CPU time, memory usage and database connections keep busy. The filter was implemented to measure the size of the incoming message.

Message Replay Filter

An attacker who attempts to resend SOAP requests to repeat sensitive transactions is called the message replay attack. Here, the client side message is assigned with an identifier and time stamp then sent to the server for validation purpose. Thereafter, the filter captures the identifier of incoming messages and rejects messages that match an entry in the replay detection database. If the message identifier is valid because of its nonexistence, the filter compares the message timestamp to its clock time value for synchronization. If the message identifier has unacceptable identifier or any time stamp mismatch then, the message is rejected. This can be done by calculating elapsed Time, cache Life Span and max Message Period. The time tolerance is the acceptable value of time difference between the sender and the maximum message period is configured as 600 seconds.

Semantic URL Attack Filter

This filter service acts to handle password reset request from the client. In addition it is implemented to handle semantic URL attack which is triggered by the legitimate user. The legitimate user can also attack the server to retrieve other's password through the URL link by changing its parameters when he received URL link for password reset. In this way the attacker tries to modify the password of the other user. This filter sets the nonce by generating a random number and assigns the time stamp to use the request URL for limited time. The random number generator needs a seed value to create the nonce. For that, the system takes the process id as seed value as given below.

IMPLEMENTATION OF XML-BASED INJECTION ATTACK FILTER

Parameter Tampering Filter

The message validation validates each incoming request message to ensure that it is well-formed XML, that it contains all of the parts required by the service, and that the contents of the message conform to an expected structure as defined by an XML Schema Definition (XSD). Then the regular expression checks to ensure that input contains only valid data and does not contain malicious SQL, HTML, or Java Script code that could lead to code injection attacks.

Then the service processes the request and responds back to the client. If the request passes all validation checks performed by the message validator, the service processes the message. In this policy, the request size must be checked before any other step. This implementation pattern uses policy assertions to check for required message parts and to validate the message schema. The following example policy file provides an example of policy assertions for the service.

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
<extensions>
...
<extension name="bodyValidator"
type="MessageValidation.CustomAssertions.BodyValidatorAssertion,MessageValidation.CustomAssertions"/>
</extensions>
<policy name="MessageValidationService">
<bodyValidator xsdPath="Configuration\GetCustomers.xsd" />
...
<requireSoapHeader name="MessageID" namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
<requireSoapHeader name="To" namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
<requireActionHeader />
</policy>
...
</policies>
```

Fig. 3 . Policy assertions for the service

In this policy file example, the <Action>, <MessageID>, and <To>elements are required on all incoming request messages. XML document reader method and Exception handler are shown in Figures 3 and 4.

```
public override void ReadXml(System.Xml.XmlReader reader, IDictionary<string, Type> extensions)
{
if (reader == null)
throw new ArgumentNullException("reader");
if (extensions == null)
throw new ArgumentNullException("extensions");
bool isEmpty = reader.IsEmptyElement;
string xsdPath = reader.GetAttribute("xsdPath");
if (!string.IsNullOrEmpty(xsdPath))
{
this.xsdPath = xsdPath;
}
else
{
throw new ConfigurationErrorsException(Messages.MissingXsdPath);
}
reader.ReadStartElement("bodyValidator");
if(!isEmpty)
reader.ReadEndElement();

}
```

Fig. 4 . Custom policy assertion class validator for the received SOAP

Coercive Parsing Attack Filter

This filter protects Web Service attacks from intruders by verifying the attack from exception handler and throws exceptions to the client. The exception with corresponding IP address is stored and maintained by the coercive parsing handler. To identify the attacks the following attributes are required to set. if (customHeader.MustUnderstand!=true l)
throw new MustUnderstandException (“SOAP header entry not understood by processing party”); if (nameSpace.VersionMismatch!=true)
throw new VersionMismatchException(“Invalid namespace defined in SOAP envelope element r”);

Oversized Message Attack Filter

This filter prevents the service from processing request messages that are larger than a specified size. This message validation protects against denial of service attacks, but the message validation must be very efficient when it conducts its validation checks. This policy checks any access of local resources of such as CPU. To achieve that, it sets the maximum request size in the service’s configuration file to limit the size of messages that the service will process. Then it compares the size of the request to the value established for the max Request Length attribute of the <http Runtime>element in the application’s configuration file, which is specified in kilo bytes. To limit the response length of the request, a value for the timeout attribute of the response time element must be set in the service’s Web config. file as shown in Figure 5. This value should be set according to the largest response length that it can reasonably expect the service to process. The following XML code set the timeout value for Web Service process.

```
Configuring Timeout Values
private const int TIMEOUT = 10000;  // 10 seconds

public override object GetEntity(Uri absoluteUri, string role,
   Type ofObjectToReturn)
{
   System.Net.WebRequest request = WebRequest.Create(absoluteUri);
   request.Timeout = TIMEOUT;

   System.Net.WebResponse response = request.GetResponse();
   if (response == null)
      throw new XmlException("Could not resolve external entity");
   Stream responseStream = response.GetResponseStream();
   if (responseStream == null)
      throw new XmlException("Could not resolve external entity");
   responseStream.ReadTimeout = TIMEOUT;
   return responseStream;
```

Fig. 5 . Configuration of over sized message filter

The service uses a protocol other than HTTP (such as TCP), the filter <maxMessageLength>setting is used to limit the size (in kilobytes) of incoming requests as shown in Figure 6. The following configuration example shows the <MaxMessageLength>set to 1024 KB for a service that uses the SoapClient/SoapService model.

```
<configuration>
...
<microsoft.Web.services3>
...
<messaging>
<maxMessageLength value="1024" />
</messaging>
...
</microsoft.Web.services3>
...
</configuration>
```

Fig. 6 . Configuration of maximum message length

Message Replay Attack Filter

The implementation of message replay detection is an identifier assigned from client side for the message. This identifier provides assurance that the message has not been replayed in transit. Next, the client sends the message to the recipient. The filter verifies the client's identifier and the message time stamp. The Web Service verifies the message identifier to ensure that the message contents have not been replayed in transit. If the message identifier is valid, then the Web Service compares the message time stamp to its own current clock value. If either the identifier is invalid or the message was received beyond the acceptable time span, the message is rejected. Lastly, the service checks the replay cache for the Identifier Value field. The Web Service checks the replay cache for the Identifier Value that is used to uniquely identify the incoming message. If the Identifier Value is already in the cache, the message is rejected as a duplicate. If the message identifier is not in the cache, the message identifier and time stamp are added to the cache.

Service Policy

The following code example is an example of the configuration for the custom replay detection policy assertion on the service shown in the Figure 7.

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
<extensions>
...
<extension name="replayDetection"
type="ReplayDetection.CustomAssertions.ReplayDetectionAssertion,ReplayDetectio
n.CustomAssertions"/>
</extensions>
<policy name="ReplayDetectionService">
<replayDetection cacheLifeSpanInSeconds="1200"
maxMessagePeriodInSeconds="600" />
...
</policy>
</policies>
```

Fig. 7 . Configuration for the custom replay detection policy assertion on the service

The Replay Detection assertion has two important parameters; they are cache life span and maximum message periods configuration parameter.

Cache Life Span in Seconds

This parameter specifies how long in seconds identifiers will remain in the replay cache. In Figure, this parameter is configured for 1,200 seconds or 20 minutes.

Max Message Period in Seconds

This parameter specifies the maximum message age in seconds that is tolerated by the assertion without accounting for clock skew. In the preceding example, this parameter is configured for 600 seconds or 10 minutes. The following code configuration snippet provides an example of this setting in the service's Web config file. The value is set to 300 seconds as shown in Figure 8.

```
<microsoft.Web.services3>
...
<security>
<elapsedTimeInSeconds value="300" />
...
</security>
</microsoft.Web.services3>
```

Fig. 8 . Code configuration for time tolerance

Messages are held in the cache for at least as long as the value that is defined in the cache Life Span in Seconds setting. To ensure that the server cannot accept a message after a duplicate message has been removed from the cache, the cache Life Span In Seconds setting must be set to at least the Maximum Message Age + elapsed Time*2.

Replay Cache

The Cache Manager class interacts with a replay cache database. Cache expiration is calculated to centralize all policy on the Web Service. The replay cache database table is named as Replay Database.

This method provides a solution to prevent the service from processing replayed messages. It does this by rejecting messages that the service has previously received within the valid processing time for them.

Cache Cleaner

The filter must clear the cache at regular intervals to regulate its size. A cache cleanup policy in the filter clears the database cache. The task is scheduled to execute the Clear Old Messages stored procedure at approximately the same interval as the cache life span value configured in the replay detection policy assertion. It executes every 22 minutes to keep the cache reasonably clear.

Semantic URL Attack Filter

In this filter, the mitigation approach is implemented by two significant components of semantic URL filter. Those are random nonce generator and time stamp calculator. The random nonce generator produces random values and from getting values from identity of task. Afterwards, the nonce value is concatenated with time stamp values to maintain its freshness.

RESULTS

The injection filter has been configured and embedded in administrator's login as shown in Figure 9. The administrators can enable or disable the filter based on their requirement. This feature will improve the speed of the server.

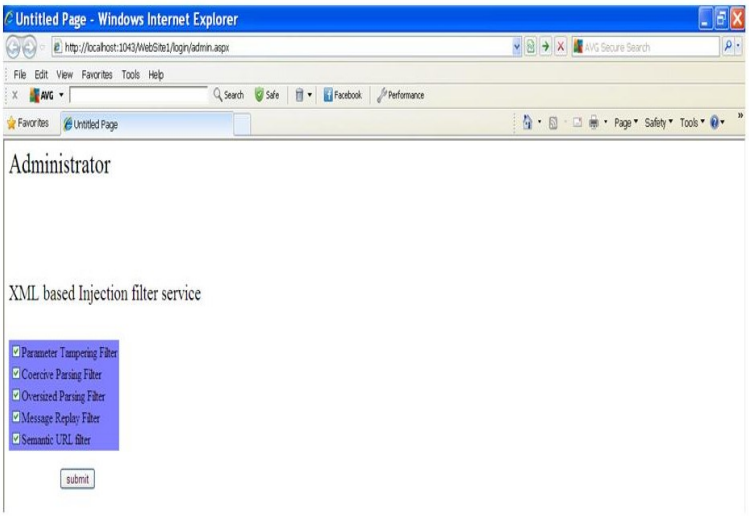


Fig. 9 . The filter settings in administrators page

Performance Comparison

The proposed system is compared with various existing systems namely input validator, AntiXSS filter, IE explorer, Opera and Firefox. The comparative analysis has been carried out with respect to number of attacks prevented and number of failures as shown in the Figures 5.21 5.26.

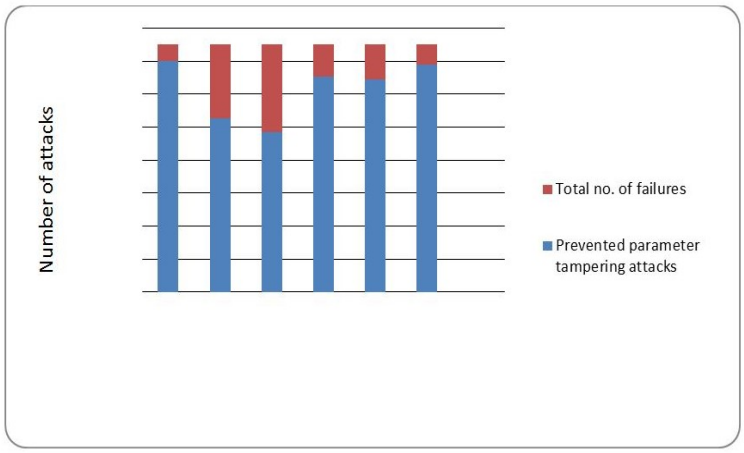


Fig. 10 . Number of failures for parameter tampering attacks

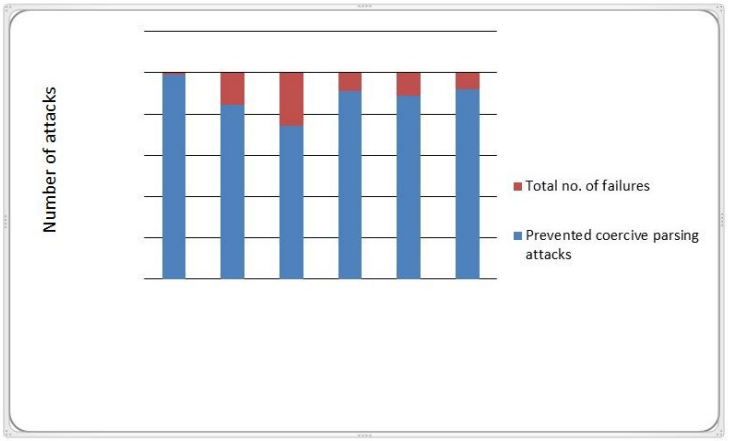


Fig. 11 . Number of failures for coercive parsing attacks

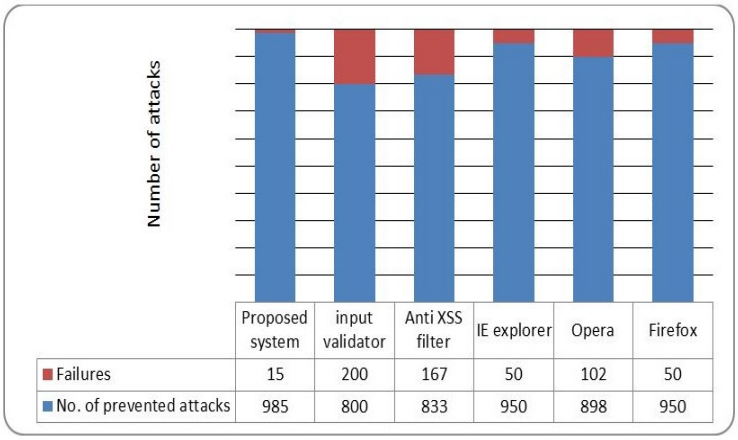


Fig. 12 . Number of failures for over sized message attacks

KKG PUBLICATIONS

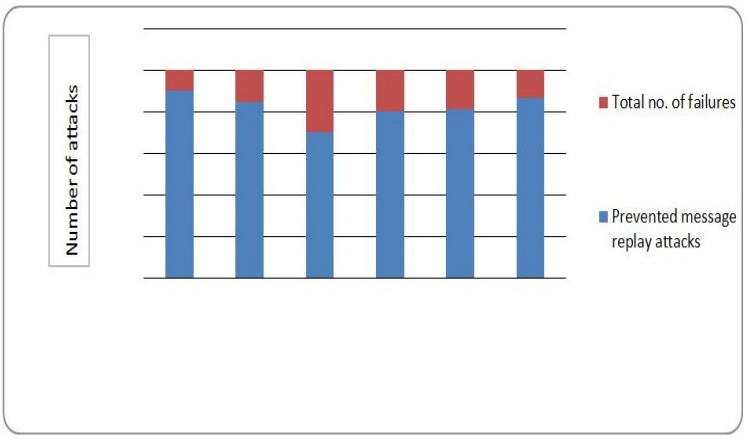


Fig. 13 . Number of failures for message replay attacks

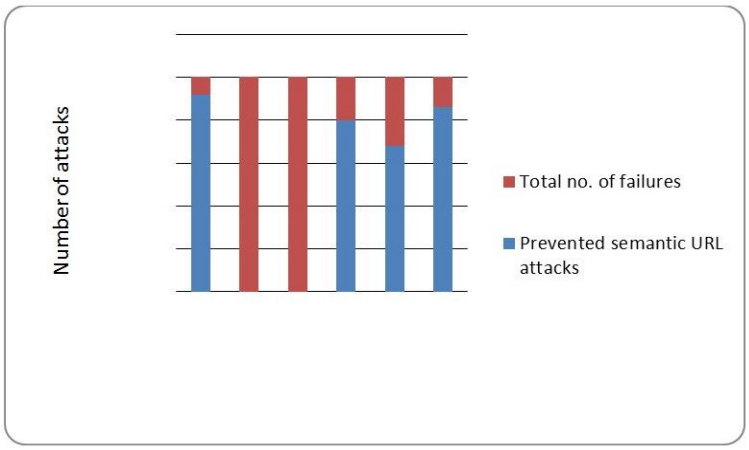


Fig. 14 . Number of failures for semantic URL attacks

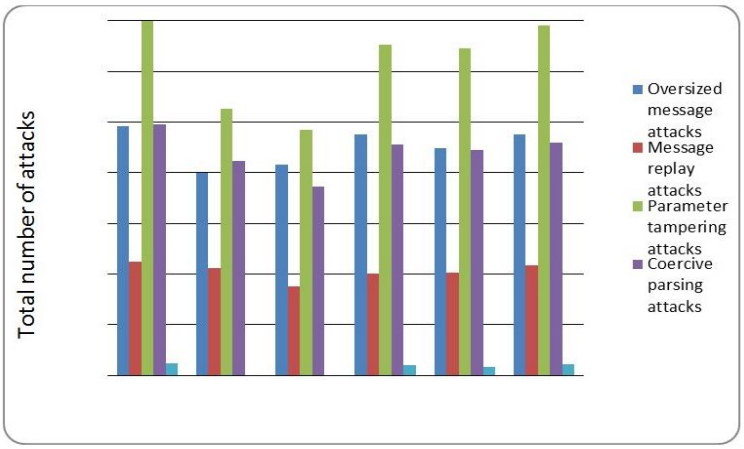


Fig. 15 . Overall comparative analysis: Number of failures

CONCLUSION AND RECOMMENDATIONS

The conventional firewall and filters are lacking in the detection and prevention of the XML-based attacks due to the fact that these attacks contain huge volume of data and are cluttered in nature. Therefore, a new XML-based injection filter is proposed in this research work to prevent and detect the attacks. All incoming requests are passed through this filter to access the service provider. It is implemented using input

KKG PUBLICATIONS

validation approach in XML documents. This filter prevents significant classes of attacks by using parameter tampering filter, coercive parsing filter, over sized message filter, message replay filter and semantic URL filter. This model has been tested against various XML-based attacks to check whether it meets the requirement of the Web Service provider and proved that the proposed approach detects attacks efficiently.

Declaration of Conflicting Interests

No conflicting interests are present.

REFERENCES

[1] S. Chawathe, "Comparing hierarchical data in external memory," in *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases*, pp. 90-101, 1999.

[2] A. Nierman and H. V. Jagadish, "Evaluating structural similarity in XML documents," in *Proceedings of the Fifth International Workshop on the Web and Databases*, vol. 2, pp. 61-67, 2002.

[3] I. Mlynkova, "Equivalence of XSD constructs and its exploitation in similarity evaluation," in *On the Move to Meaningful Internet Systems: OTM 2008* (pp. 1253-1270). Springer Berlin Heidelberg, 2008.

[4] S. Shah, "Defending web services using mod security (apache): Methodology and filtering techniques," Several Advisories On Security Flaws, 2002.

[5] M. Cremonini, S. Vimercati, E. Damiani and P. Samarati, "An XML-Based approach to combine firewalls and web services security specifications," in *Proceedings of ACM Workshop XML Security*, Virginia, pp. 69-78, 2003.

[6] P. Lindstrom, "Attacking and defending web services, 2013," A Spire Research Report, 2004.

[7] E. Bertino, L. Martino, F. Paci and A. Squicciarini, *Security for Web Services and Service-Oriented Architectures*, 1st ed. Berlin: Germany, Springer Publisher.

[8] V. R. Mouli, and K. P. Jevitha, "Web services attacks and security-a systematic literature review," *Procedia Computer Science,* vol.93, pp.870-877, 2015.

[9] G. Y. Chan, C. S. Lee, and S. H. Heng, "Defending against XML-related attacks in e-commerce applications with predictive fuzzy associative rules," *Applied Soft Computing,* vol. 24, pp. 142-157, 2014.

[10] M. Yampolskiy, P. Horvath, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "A language for describing attacks on cyber-physical systems," *International Journal of Critical Infrastructure Protection,* vol. 8, pp. 40-52, 2015.

— This article does not have any appendix. —