Research Article

iMedPub Journals
http://www.imedpub.com

American Journal of Computer Science and Information Technology

2020
Vol. 8 No. 5: 64

A New Approach to Prevent the DDOS Attack and XML Injection Attacks Using XSD Trace Handler in Web Service

Hasan Hashim* and Alzighaibi AR

Department of Computer Science and Engineering, Taibah University, Yanbu, KSA

Corresponding author:
Hasan Hashim, Department of Computer Science and Engineering, Taibah University, Yanbu, KSA

satlam@taibahu.edu.sa

Citation: Hashim H, Alzighaibi AR (2020) A New Approach to Prevent the DDOS Attack and XML Injection Attacks Using XSD Trace Handler in Web Service. Am J ComptSci Inform Technol Vol.8 No.5: 64.

Abstract

Web management today plays a major role in developing Business-To-Business (B2B) and business-to-customer (B2C) applications. Web services protection is facing a major threat due to Distributed Denial of Service (DDoS), XML Injection and Cross Site Scripting (XSS) attacks by injecting. Protection of Web sites Security is also at risk. It is therefore monumental that the sensitive Web Service is provided with substantial safety. Security components such as XML encryption, advanced marks and customer tokens are a key part of communication in the business process in web administrations. The attacker can use the situation and make administrative changes to hack the information secured via web management. The main objective is to provide a Safety System in Service- orientated Architecture for the prevention of XML attacks and DDoS. The research aims to develop a framework to detect and prevent attack on web service-based applications by XML-based distributed denial of service, DDoS, etc. A monitory of the source parameter that is performed is applied to prevent DDoS attacks by the historical traffic attack detection mechanism. The algorithm detects the user request number in a specific day and time. It also calculates the number of bytes saved by preventing a DDoS attack. If the number of requests from a single client is greater than the threshold value, the client IP is blocked and the incoming request IP from the client to the server is retained. It also serves to generate a Captcha to check whether the application is from a BOT, illegal user or a legally binding user. The user's IP address is blocked when an illegitimate user is identified. The application is then sent to the XML injection filter. In order to filter the incoming request, static and dynamic filtrations are used.

Keywords: Web service; XML injection; DDoS attack; Service-based design

Received: November 5, 2020; **Accepted:** November 19, 2020; **Published:** November 26, 2020

Introduction

The security of web application is an information security branch devoted to website, web applications and web services specifically. At a high level, web security applies but specifically applies to internet and web systems, based on principles of application security.

Materials and Methods

Web service

Web Server is ideally built in Standard General Mark-Up Language (SGML) XML, so attacks such as XML 2, XSS, and Xpath injections are highly prevalent and even Web Services are provided over http using a Single Object Access Protocol (SOAP) [1]. The attacks above insert more nodes or alter the current nodes to adjust operating parameters. For Web Service protection and business model to function properly, mitigation of these attacks is important. Confidentiality, availability and integrity are the fundamental requirements of a secured system. The assault may take place in a way that deviates from the above activities and is said to be vulnerable to the attack. Such attacks may be performed by UN sanitized web service input or unauthorized source code updating through malicious code injecting into web services. The main threat of a web service is, among other things, inserting malicious XML code into the web service through UN sanitized

data. In the following sections, the detection and corrective actions of different kinds of attack, including the attacks referred to above, will be listed [2].

Security issues in web services

The advances of web Services technology impact the world's web and business community extensively and considerably. Open Web Platform standards such as XML, Single Object Access Protocol (SOAP) allow user data and application to interact directly and with complex connections without human interference. In different types of architecture, the web service technology is applied and cooperates with accessible software and the design process. They can therefore be accepted in progress without major modifications in both databases and legacy applications.

Distributed denial of service (DDoS) attack

An attempt to deny service (DOS) if the attacker tries to reduce the service's availability thus preventing the authorized user from using the operating system. In a web service, 16 forms are based on external attacks by entities and recursive attacks by entities [3]. In a DDoS attack, numerous requests for service access from different network flood points with competitor hit [4]. We must know the difference between bad and legitimate requests quickly to recognize the attacks from DDoS. In an earlier step of the network and at different points in the network, this can be identified [5]. A security system should be used to filter traffic from each access point to the networks. This strategy can be seen in **Figure 1** [6].

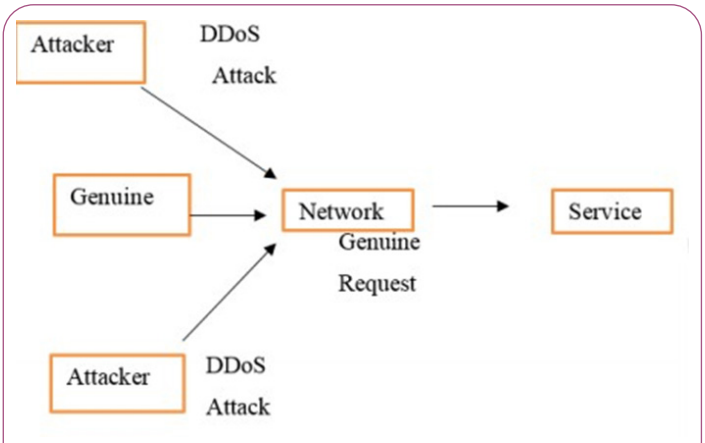

Figure 1: DDoS attack.

Filtering of a trafficking generated multiple points of access in the earlier phase can avoid the DDoS attack [7]. Six multiple requests for a service may be provided for specific attacker flood requests, the service can also be accessed in a different network direction, which means that all firewalls must be supplied to the access point within the network, in order to handle the service request. It helps the server to process the legitimate application without DDoS attacks.

The main goal of this research consists of proposing and implementing a new XSD trace processor and secure web service message communication pattern, including the use of static and dynamic filters. This work provides a new approach for transmitting protected messages with improved encryption methods. This newly proposed system offers solutions for many problems, such as malicious code prevention and identification of sensitive web-based content. The framework suggested incorporates attacks based on injection, and secures web-based messages. It ensures the data shared are authenticated, detailed and confidential. The main contribution of this work is the sharing of knowledge between business processes to avoid and protect them. End-to-end protection must be given to optimize the scope of web services. When intermediaries don't trust communication endpoints. Although the security requirements for web services are complex, the needs of SOAP-oriented messaging are not protected by new security frameworks.

Related work

The advancement of Internet services has speeded up the integration of various business processes across a number of platforms spread across the globe alongside the advantages provided by internet services for online transactions, which together pose some security threats. It offers a comprehensive safety study for web services and service-based design (SOA) [8]. In addition, all the latest research areas in the fields of access management for easy, conversation-based Internet service and web-based workflow access management briefly address the safety requirements of internet services. A robust WS Security system uses stream based approaches to boost web service delivery performance and enhance the various types of denial of service attacks [9]. The system is a comprehensive WS security system. A new approach for the study of business processes and risk services. We focused mainly on vulnerability identification through previous research into vulnerability and added more to information sources vulnerability compilation [10]. Classified XSS (Cross Site Scripting) exploits reflected or Stored Object Mode (SOM) and categorized XSS defenses into different types of testing, defense coding [11], vulnerability detection and weakness and strengths. In addition, online methods for implementing such XSS protections have been studied. In order to counteract signature wrapping attack, the use of context sensitive XML signature was proposed [12]. Signed nodes are sometimes removed from the SOAP message and inserted into a new SOAP message to lose their original setting. Introduced the sigfree tool, an online tool to avoid injection of code and buffer overflow attacks for a range of web and web services [13]. Sigfree uses a new method called data-flow code abstraction. Double Guard, intrusion detection systems that mimic the network's use of the front-end server and back-end application for user behavior, are both proposed and used to combat new and unknown buffer overflow attacks. The user interface as well as the database request was also evaluated by the web client. It also allows us to identify attacks which could not be found by independent IDS [14,15]. A study of XML rewrite attack detection technology on web services explains web services, SOAP messages, headers and XML rewrite attacks

in great detail. Studied the advantages and disadvantage of various mechanisms such a policy-based approach for detection/ mitigation of attacks [16,17].

Proposed work

This proposed Web Service Security (WSS) architecture is complemented with three main components, DDOS attack detection, dynamic injection filter service, and fixed XML- based filtering service. To detect and prevent the attack of the intruder, the filtering policies are implemented. Customer requests are passed through DDOS detection filter services. Upon validating a DDOS attack, it searches XML for a fixed template injection attack. The filter checks the wrong type of script and validates the sort, length, format and range of inputs in the received text (**Figure 2**).

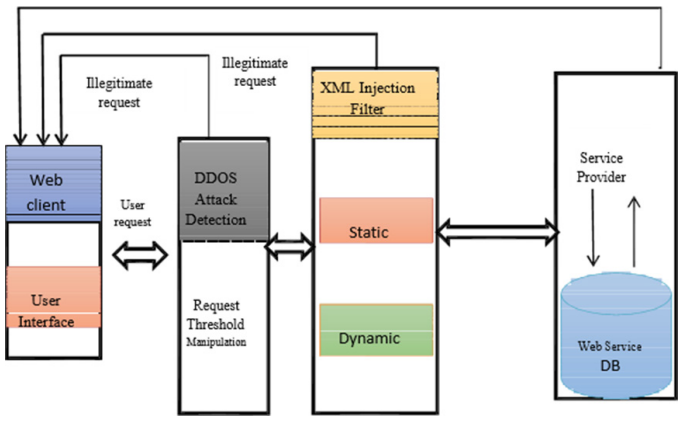

Figure 2: XSD handle architecture of XSD handle.

The server-side code is designed to restrict income from other sources such as query strings or XML injection cookies, SQL injection, X path Injection and cookie replay detections. This is supported by client-side controls or inputs. If no injection is undetected, the application is validated via dynamic patterns based on application severity. Using a single attack detection system is not simple to identify legitimate requests or illegitimate requests. For this reason, a large group of attack detection systems have to be installed by service providers because the attacks can be of any form. Therefore, before accessing the service provider in business, the filtration and detection mechanism is installed.

Web client

The client must be from different machines. It is the user interface where the user inputs the data in the user interface screen. The request is then validated in the next level DDoS attack detection.

DDoS attack detection

In case of an order with the historic traffic aggregation DDoS detection algorithm, which calculates the threshold value for the historical value, the demand is passed to the DDoS detection element. The request is analyzed. The request is passed with the captcha generation algorithm for detection of DDoS based on the threshold value.

XML injection filter

The code was evaluated here using the static pattern in the XML injection static filter. If no injection is found, Dynamic Filter has assessed the request based on the severity by the programmer of the application design.

Web service DB

The data shall be compiled and sent to the Service user once a valid request has been processed and the request enters the web service (Database) DB.

DDoS attack detection

The following explains the algorithm and procedure for DDOS detection and prevention mechanism.

Procedure for historic traffic surge aggregation DDoS detection

This section uses an algorithm for the avoidance of DDoS, the historical analysis of DDoS surges. This algorithm draws input from the application's hosting application's historical data and logs.

Three main inputs are proposed to be used.

- Total number of Hourly requests (HR0, HR2…. HR23)
- Time of the requests coming into the server
- IP address of the origin of the request

This algorithm is divided into three major sections.

a. Threshold request count computation (TR_{cnt})

b. Threshold time computation (TH_t)

c. Threshold bytes computation provided for each user (TB_t)

Threshold request count computation

The first step is to set the total time under consideration on the basis of the historical data available and other realistic constraints, usually for a few weeks to a month. The total number of requests from the start of the period to the end of the period considered shall be determined after the time limit of the data is set. The next step is to calculate the average daily demand according to the following formula.

$Rcnt_{ave}$= Total user requests in all days under consideration/Total days considered

Based on the per day average figure calculated above, the requisite threshold value is calculated using the following formulae

$$TRcnt\ per\ sec = Rcnt_{ave}/86400$$

Threshold bytes provided for each user

For the next phase, in the normal few weeks to a month, the total time under consideration should be set. The total number of

bytes accessed between the beginning of the period and the end of the time period under consideration shall be established after the date of data is fixed. The next step is to calculate the user-accessed average volume of data (in bytes) during the reporting time. The IP address is used as unique identifier for the purpose of determining the data accessed by the user. The average data per program is determined using the following formats:

TB_t=Total bytes accessed by the users/Total number of users

If IP address/user has exceeded the quantum threshold bytes, the use of any system resource is not permitted and the request is considered to be DDoS. The following method, which defines the validity of the user/demand by assigning the above parameters as baseline. First of all, the characteristics of the current request are extracted EX: IP, request form and request time stamp. Then you can download the historical data of the same user/application.

Now update the request threshold number for that specific user/demand (IP adresse) in the table with the time differential count. Using the following formulae, measure the time between the last successful applications for the current application.

$$T_{diff}=t_{curr}-t_{pre}$$

If Tdiff is greater that the THt and the Threshold count of that particular user (IP address) don't exceed the TRcnt per second then the user is considered as a legitimate user and the request is not flagged as DDoS attack.

Step 1: If $t_{diff} > TH_t$

Step 2: if TR_{diff} < req/sec from IP address request from user is legitimate

Step 3: Then DDoS attack detected

The next step of the algorithm is to enter comprehensive logs about the DDoS attack observed, including IP address, time of attack etc. This is an input to the following stage in which a novel ' Prolific Captcha Reconstruction Algorithm' is used to deter the ongoing DDoS attack.

Step 1: If ip address in attack table $A_{ipcnt} = A_{ipcnt}+1$

Step 2: Then, attach severity count $S_{ivi}=A_{ipcnt}$

First step is to measure the severity of the attack by determining the attack count of the IP address in consideration, within a period of time (say 60 seconds). Increment the attack count by one (for the current attack) and determine the updated severity of attack. Based on the severity of the attack, a profilic captcha is generated by the following algorithm.

Captcha generation algorithm

This algorithm is used to generate a captcha page where a text will be displayed randomly within the position of the box in the screen.

Step 1: Initialize bitmap object "bmap" with basic parameters like width, height and pixel format.

Step 2: Initialize graphics-> assigned to bitmap with smoothing mode property.

Step 3: Initialize rect object for setting boundary, hatch=brush initialization to style the content with fore and background color.

Step 4: Fill graphic object with brush and rectangle

Step 5: Intelligent captcha text size definer

Step 6: Validate the condition of font size>rectangle width

Step 7: If Yes

Step 8: Initialize font width size, family and type

Step 9: Font size Val=Customize based on the random text generated by generate random code member of random image custom class

Step 10: Initialize string formatter with alignment details

Step 11: Initialize graphics path to write out the captcha string by finalizing the "x" and "y" location co-ordinates by manipulating the rectangle zone initialized in step 3

Step 12: Graphics path warped the graphics object with the co-ordinate point and rectangle object initialized in

Step 13: Initialize the hatch brush style, fore and back color

Step 14: Fill path method applied on graphics object using the hatch brush & graphics path

Step 15: Make the visualization harden by filling with the ellipse method for the graphics object. The strength is based on the pre-requisite value Slvl obtained

Step 16: Final captcha image is extracted

Static filter based attack detection

Without the required level of security clearance, compromise database protection, harm or delete da-deletion in any web service, which has the effect of taking one or more parameters as input and returning one or more values as result after performing percussive data/database operations, etc.

Web services are therefore maintained by static pattern matching technology as shown in **Table 1** to protect most of the current filtering mechanisms (**Table 1**).

Rule number	Description
Rule 1	Field-> Type checks-> Dynamic XSD decision -> Valid -> Grant
Rule 2	Field-> length checks-> Length lesser than expected-> Dynamic XSD decision-> Invalid-> Deny
Rule 3	Field-> length checks -> Length greater than expected -> Dynamic XSD decision -> Invalid -> Deny
Rule 4	Field> length Checks -> meets length criteria -> Dynamic XSD decision -> Valid -> Grant

Table 1: Rule Set of Static Filter.

The filters are planned, designed and implemented at the time

of original design with static pattern matching filters. This form of pattern match is rigid in nature and can't be changed any later because the structure of the program is hardly coded as shown in **Figure 3**. In comparison, static pattern matching strategies, potential attack models cannot be accurately predicted and if the filters are to be updated, they are resource- intensive and time-consuming. The following code demonstrates the static filter XSD pattern (**Figure 3**).

Static pattern with Length Restriction

```
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:elementname="UserName">
<xsd:simpleType>
<xsd:restrictionbase="xsd:string">
<xsd:patternvalue="[A-Z]*[0-9]*[a-z]*[^\-/]"/>
<xsd:lengthvalue="12"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:schema>
```

Figure 3: Static pattern validation.

Dynamic filter based attack detection

The above narrated drawbacks in the legacy system can be mitigated by this proposed dynamic pattern matching technique wherein the validation logic and then filter component can be separated into two different entities (**Figure 4**).

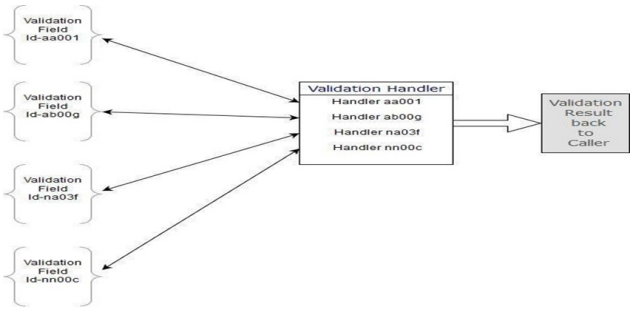

Figure 4: Dynamic validation handling-software architecture.

When designing and coding a web service, regardless of its application type or platform, some kind of placeholder together with a level indicator can be used for multiphase validation by the validation mechanism if appropriate. This is an integral part of the web service and does not have to be modified during the lifecycle of the application. The validation placeholder that is part of the original web service code. A unique ID can be used for the field required to be validated and generally made globally unique to the server and/domain, is contained in each validation placeholder. The testing logic and processes usually have to take place on a separate server or VM and can be a different platform. This method of defining must be done. Such validation logic separation is also useful in situations where the entire validation process of many different web sites/servers can be linked to a single validation server that can be designed to carry out complex mathematical calculations and functions that are too expensive to duplicate to all web servers or services.

Since most Web services return some value directly from a database or by calculating values based on data bases tables, and when the validation is performed on a separate validation VM, the request can be quarantined in case of a suspected attack pattern, simulation of execution can be carried out with the validation VM without affecting the actual web service.

The admin or security expert the dynamic-frame the required levels and type of validation, determine the complexity of the validation, and allocate the validation unique ID against the validation as shown in figure in the proposed two entity dynamic validation techniques. More than one validation entity can be selected against one validation ID and based on the requesting entity/user ID profile, geo-location, etc. the program can handle this automatically without any manual intervention in real-time (**Figure 5**).

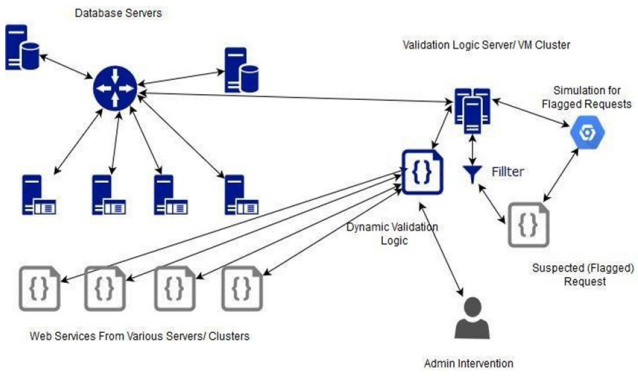

Figure 5: Dynamic validation handling-implementation scenario.

During dynamic validation, the system tends to be trained to detect attacks over time and this type of knowledge can be generalized and the learned dataset can directly be incorporated into other future design validation machines, i.e. when the system has been adequately trained with a complex set of values and even in real time, the time required for the system for the detection of attacks can be minimized. It makes the system stable and reliable over a period of time against attacks. Such an attack detection/validation without disrupting existing web services and code may in future even include some kind of deep learning mechanism due to this separation of the code with the validation mechanism as two separate entities.

n(D)=> Today number of records available in the dimension D

F_k=> Fraction of records meant to be deviated

Cumulative fraction of records=N.F_k

Standard deviation $\sqrt{\Sigma}$ (Fraction record-Inverse of factional record)2/n(D)-1 Sparse Outlier Boundary Detection=$F_k1+F_k2+F_k3+.....F_kn$/Standard Deviation

The sparse outlier limit is calculated using the above formulae. For all entry value fields, it is calculated and if the value is within the computed boundary, the input field value is then considered safe to operate and passed for further processing to the web service (**Figure 6**).

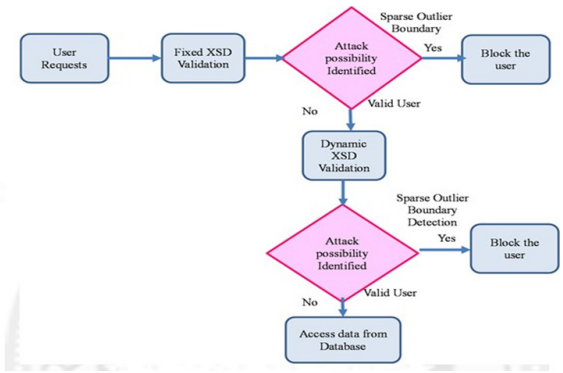

Figure 6: Established protocol for dynamic validation.

In the event that the calculated sparse outlier boundary value for the field does not fall within the value, it is regarded as an attack and the IP address is blocked and further action against the user (IP) is taken, as shown in figure under the established protocol for dynamic validation technique.

Algorithm for the dynamic filter

Input:

User request

Output:

1. IP Address of the attacker.

2. Valid response for the valid user.

Procedure:

Step 1: User input requests received.

Step 2: Validation with fixed XSD generated schema.

Step 3: Check for attack identified-calculate the sparse outlier boundary value.

Step 4: If SOBV less than 1 validation with dynamic XSD generated schema.

Step 5: Check for attack identified with additional filtered result set calculate the sparse outlier boundary value.

Step 6: If SOBV exceeds the value 1 block the request. Register the IP address for future reference.

Step 7: Else respond the user with valid input.

Step 8: Repeat the process till the entire user request is processed.

Implementation of dynamic and static filter

Tables 2 and 3 represent set of rules to allow data both integer and string data type with the following restriction (**Tables 2 and 3**).

	Fixed grid XSD	Dynamic grid XSD	
S. No.	**Integer**		**Additional info**
1	Type will be checked	Type will be checked	
2	Length will be checked	Length will be checked	
3	Space permitted or not	Space permitted or not	If space not permitted, input string will be space nullified
4		Negative character checks	
5		Float value checks	
6		Max permitted and min permitted values	

Table 2: Comparison of static validation Vs dynamic validation (Integer data type).

	Fixed grid XSD	Dynamic grid XSD	
S. No.	**String**		**Additional info**
1	Type will be checked	Type will be checked	
2	Length will be checked	Length will be checked	
3	Space permitted or not	Space permitted or Not	If space not permitted, Input String will be space nullified
4		Special character existence will be checked	
5		First character of the column can be caps or some special indications	
6		Numeric values permitted in the column	

Table 3: Comparison of static vs. dynamic validation (String data type).

Evaluation of DDoS attack experimental setup

In a simulation environment, the proposed system was tested. An interactive web page is built and many virtual machines are developed to access the same page by using multiple sources. The attack is exploited and repeated over a number of days and times by means of the target link. The experimental results for attack were calculated without the system proposed and with the defense system proposed, and by defending the attack that was generated the number of bytes was rescued.

Discussion about DDoS attack detection using historic traffic surge

Table 4 indicates the number of requests each user makes to access a web page for a given time and day. The results show that around 95% of attacks are detected on the basis of the thresholds DDoS attack results in comparison to and without an XSD handler (**Figure 7**).

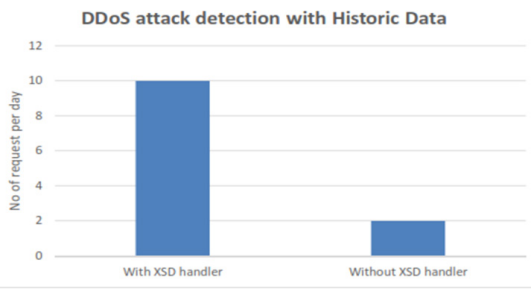

Figure 7: Historic date in of DDoS detection using XSD handler.

Table 5 shows the request for write byte counts detected and tested according to the number of hits on the various user requests. The specific byte counts for different times and number of hits (**Tables 4-6**).

User	No of request/ day	Avg request/ time in (ms) THt	Threshold request count (TRcnt)	TR diff	Attack detection
1	300	30	20	10	Detected
2	500	45	20	15	Detected
3	400	40	20	20	Detected
4	300	20	20	0	No attack
5	650	70	20	50	Detected
6	500	40	20	20	Detected
7	700	10	20	-10	No attack

Table 4: DDoS attack detection using historic traffic surge.

Total hits	Threshold time	Threshold valuec	Prevention hits stopped	Preventive write bytes count
20	5	8	7	18288
40	5	8	7	85532
60	5	8	7	222165
80	5	8	7	87832
100	5	8	10	194552

Table 5: DDoS detection with byte count saved.

Evaluation of static and dynamic filter

In a simulation environment, the proposed system has been tested. A webpage is designed for experimentation and a number of malicious code requests are being tested. In a given time, for several requests, the attack was manipulated and replicated with the target page (**Figures 8-10**).

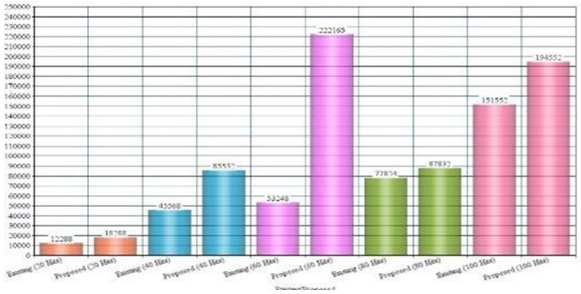

Figure 8: Byte saved on existing and proposed.

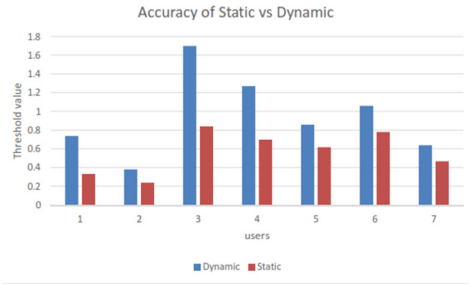

Figure 9: Comparison of accuracy-static versus dynamic validation techniques.

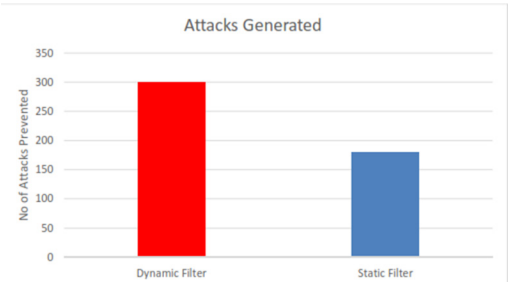

Figure 10: Attacks prevented using filters.

User ID	Fraction of records	Dynamic XSD filtration additional records	Inverse	Standard deviation	Sparse outlier value	Validity of the user
102	20	24	3642	59.27	0.74	Valid
104	15	8	3663	59.96	0.38	Valid
105	50	48	3588	57.49	1.70	Invalid
106	42	32	3612	58.28	1.27	Invalid
107	37	14	3635	59.04	0.86	Valid
108	46	16	3624	58.68	1.06	Invalid
110	28	10	3648	59.47	0.64	Valid
10	10	10	10	10	10	10

Table 6: Static vs. dynamic filter threshold detection.

Results and Discussion

The comparison of the set of fixed and dynamic pattern results by means of the threshold values that have been evaluated for about 7 different users at different times with different IP addresses with different causes of input fields in both static, dynamical filters. The Sparse description of the dynamic filter is shown in below.

The dynamic filter was used to stop nearly 95 percent of attacks in comparison with the performance. Displays total number of attacks prevented using a filter that prevents attacks by dynamic filters more than static filters.

Conclusion

The new filtering policies to avoid XML-based attack were suggested in this paper. This work proposes a dynamic filter approach to effectively prevent attack. In addition, the service provider will not be allowed directly to enter. It instead transmits the application to the filter services, analyzes the service demand and answers the user. It also validates the application and blocks illegal users and prevents the IP from accessing the server application.

References

1. Sajjad SM, Bouk SH, Yousaf M (2015) Neighbor node trust based intrusion detection system for WSN. Procedia Comput Sci 63: 183-188.
2. Ezhilarasi M, Krishnaveni V (2019) an evolutionary multipath energy-efficient routing protocol (EMEER) for network lifetime enhancement in wireless sensor networks. Soft Comput 23: 8367-8377.
3. Arumugam GS, Ponnuchamy T (2015) EE-LEACH: Development of energy-efficient LEACH protocol for data gathering in WSN. J on Wirel Communi and Netw 76: 1-9.
4. Kaur A, Kaur H (2017) a review on a hybrid approach using mobile sink and fuzzy logic for region based clustering in WSN. Int J Comput Technol 16: 7586-7590.
5. Kumar A (2017) Handling DDoS attacks in cloud computing based on SDN system. Int J of Comput Sci & Eng Inf technol 2: 62- 68.
6. Shameli-Sendi A, Pourzandi M, Fekih-Ahmed M, Cheriet M (2015) Taxonomy of distributed denial of service mitigation approaches for cloud computing. J Netw Comput. Appl 58: 165-179.
7. Ezhilarasi M, Krishnaveni V (2019) An evolutionary multipath energy-efficient routing protocol (EMEER) for network lifetime enhancement in wireless sensor networks. Soft Comput 23: 8367-8377.
8. Sqalli MH, Al-Haidari F, Salah K (2011) Edos-shield-a two-steps mitigation technique against edos attacks in cloud computing. In fourth IEEE international conference on utility and cloud computing 49-56.
9. Nagarajan M. A New Approach to Improve Life Time Using Energy Based Routing in Wireless Sensor Network. Int J Sci Res 3: 1734-1738.
10. Zhang ZH, Wang Q, Zhou XP (2005) Research and development of distributed workflow based ERP system. Indu Con Compu 18: 5-7.
11. Ribeiro F, Metrôlho J, Leal J, Martins H, Bastos P (2018) A mobile application to provide personalized information for mobility impaired tourists. In world conference on information systems and technologies 164-173.
12. Nagarajan M, Karthikeyan S (2012): A new approaches to increase the life time and efficiency of wireless sensor network. In International conference on pattern recognition, informatics and medical engineering 231-235.
13. Nguyen HQ, Taniar D, Rahayu JW, Nguyen K (2011) Double-layered schema integration of heterogeneous XML sources. J Syst Softw 84: 63-76.
14. Zhao Z, Liu W, Qian Y, Nie L, Yin Y et al (2018) Identifying advisor-advisee relationships from co-author networks via a novel deep model. Inform Sci 466: 258-269.
15. Shanmugapriya M, Munusamy N (2018) an approach to increase energy and life time using power control optimization in wireless sensor networks. Int J Pure Appl Math 119: 1171-1181.
16. Xu P, Sugano Y, Bulling (2016) Spatio-temporal modeling and prediction of visual attention in graphical user interfaces. Proc 34th Annu ACM Conf Human Factors Comput Syst 3299-3310.
17. Gnanaprasanambikai L, Munusamy N (2017) Survey of genetic algorithm effectiveness in intrusion detection. International conference on intelligent computing and control 1-5.