International Journal of Engineering & Technology, 7 (4.7) (2018) 322-329

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Protection of XML-Based Denail-of-Service and Httpflooding Attacks in Web Services Using the Middleware Tool

Abbas Alasri[1], Rossilawati Sulaiman[2]

[1]Center for Cyber Security Faculty of Information Science and Technology, UniversitiKebangsaan Malaysia,43600 Bandar BaruBangi, Selangor,Malaysia

[2] Center for Cyber Security Faculty of Information Science and Technology, UniversitiKebangsaan Malaysia, 43600 Bandar BaruBangi, Selangor, Malaysia

Abstract

A web service is defined as the method of communication between the web applications and the clients. Web services are very flexible and scalable as they are independent of both the hardware and software infrastructure. The lack of security protection offered by web services creates a gap which attackers can make use of. Web services are offered on the HyperText Transfer Protocol (HTTP) with Simple Object Access Protocol (SOAP) as the underlying infrastructure. Web services rely heavily on the Extended Mark-up Language (XML). Hence, web services are most vulnerable to attacks which use XML as the attack parameter. Recently, a new type of XML-based Denial-of-Service (XDoS) attacks has surfaced, which targets the web services. The purpose of these attacks is to consume the system resources by sending SOAP requests that contain malicious XML content. Unfortunately, these malicious requests go undetected underneath the network or transportation layers of the Transfer Control Protocol/Internet Protocol (TCP/IP), as they appear to be legitimate packets.In this paper, a middleware tool is proposed to provide real time detection and prevention of XDoS and HTTP flooding attacks in web service. This tool focuses on the attacks on the two layers of the Open System Interconnection (OSI) model, which are to detect and prevent XDoS attacks on the application layer and prevent flooding attacks at the Network layer.The rule-based approach is used to classify requests either as normal or malicious,in order to detect the XDoS attacks. The experimental results from the middleware tool have demonstrated that the rule-based technique has efficiently detected and prevented theattacks of XDoS and HTTP flooding attacks such as the oversized payload, coercive parsing and XML external entities close to real-time such as 0.006s over the web services. The middleware tool provides close to 100% service availability to normal request, hence protecting the web service against the attacks of XDoS and distributed XDoS (DXDoS).\

Keywords: *Web Service, SOA, SOAP, XML, Denial-of-Service,XDoS, DXDoS*

1. Introduction

The dynamic use of the Internet services and applications has led to an increase in conducting businesses online. The web technologies and service-oriented architectures areconsidered as among the recent most innovative inventions. From the early 1990s until now, the Web Service (WS) has evolved and came along way by offering rich functionalities, isuser friendly, easyto use and its integration with other applications and services irrespective of the different platforms. This trend has fascinated governments, banks and large organizations to offer their services via web applications[1].

Most of the web services today use XML messages for transferring information between the web services and their clients.However, many of the vulnerabilities of XML have been recently discovered and reported [2-6], as these vulnerabilities provide the chances for DoS attacks. Therefore, many systems that depend on XML protocol are at risk if the vulnerabilities are not properly mitigated [7].Based on the Open Web Application Security Project (OWASP), the XMLinjection, and XDoS such as XML External Entities (XXE) attacks are the most-frequently occurring type of attacks in Web and WS applications. They have been classified as the top attacks since the year 2017[8, 9].

An XML parser is required for the web service engine to extract the information from an incoming SOAP message. An attacker can exploit this parser to successfully establish XDoS attacks by sending malicious SOAP requests also the parser behavior can be influenced by adding a Document Type Definition (DTD) on top of the XML document [10]. According to Späth, Mainka [10], a systematically analyse of 30 different XML parsers of six programming languages. The evaluation results have demonstrated that the vulnerabilities in 66% of all tested parses in their default configurations.

XDoS and DXDoS are found to be more destructive than the traditional DoS [11]. The purpose of these attacks is to consume the system resources by sending SOAP requests that contain malicious XML content.Unfortunately, these malicious requests go undetected underneath the network or the transportation layers of the Transfer Control Protocol/Internet Protocol (TCP/IP), as they appear to be legitimate packets[12]. It is difficult to block malicious traffic using the firewalls, since the firewalls are unable to check the XML content for any suspicious packets [13].

Recently, several security standards are established to provide protection for web services, such as WS-Security, but they only address integrity and confidentiality aspects of web service security[14]. The lack of availability protection made it easy for attack-

ers to launch XDoS attacks that overwhelm servers and prevent them from delivering service to legitimate customers.
In this paper, a solution is proposed for preventing XDoS and HTTPflooding attacks on the web services. In the context of thispaper, since DXDoS is derived from XDoS techniques, the phraseXDoS attacks will refer to both XDoS as well as DXDoS attacks.
The rest of this paper is arranged as follows; Section **II** discusses the related works,Section**III**illustrates the XDoS attacks which target the web services, Section **IV** clarifies the proposed system, Section **V** discusses the experimental results andSection**VI**presents the conclusion.

2. Related Work

Falkenberg, Mainka [15]conducted a comprehensive study on the vulnerabilities in the SOAP-based Web services. Many of these attacks target the availability of web services, such as XML external entity DoS, coercive parsing, and oversized XML attack.Further, the paper proposes apenetration testing tool which simulates known XDoS attacks.
Chan, Chua [11] presented a fuzzy association rule-based (FAR) and fuzzy associative pattern-based (FAP) intrusion detection and prevention (IDP) system to defend and prevent web service attacks. In this work, authors extend their examination on the use of fuzzy logic, associative pattern matching and association rules in helping for detection and prevention of existing signature-based attacks, as well as prediction of new anomaly-based attacks on a web service-based e-commerce application deployed over the Internet.The FAR IDP system provided almost 95% of service availability to normal transactions.
Vissers, Somasundaram [12]proposed a defensesystem using Gaussian model to prevent XDoS and HTTP flooding attacks on web services. This model is presented by means and standard deviations of many features, such as number of elements, content-length, nesting depth, attribute, and namespace.The authors aim to detect all reported vulnerabilities and might be able to detect unknown attacks.
Utsai and Joshi [16] proposed application layer filters to detect and mitigate DoS attacks. Based on author's experiment, this work provides good results in detecting XDoS attacks but it requires extended processing time for attributes retrieval from request message, insertion, updating of data to and from databases, and so on. However, prevention and detection of new attacks are not processed in real-time.
In Anitha and Malliga [17], a rule set-based detection system called CLASSIE is used to detect XDoS attacks. It used the packet marking method to prevent spoofing attacks. CLASSIE system should be placed one hop away from the host, and its rule set-based should be generated over time to identify known HTTP flooding and XDoS attacks. Moreover, when a packet matches one of the rule-sets then it is dropped by the CLASSIE. Packets are marked on the edge and core routers after passing tests by CLASSIE. At the edge router, one bit is required for demonstrating that the packet is marked.
Karnwal, Sivakumar [18] proposed a Filtering Tree and Trace Back approach. The approach uses filtering tree technique to filter suspicious IP addresses. Suspicious IP addresses are stored in a Trace-Back module. A Cloud Defender then detects for HTTP or XML DDoS attacks. The approach focuses on detection of web service vulnerabilities, e.g. SOAP coercive parsing, HTTP, and XML DDoS attacks. No preventive measure being mentioned and no performance evaluation results are discussed.
Through the related works, various researchesin detecting and preventing XDoS and HTTP flooding attacks have been presented. However, these researches are somehow limited in preventing XDoS attacks only because they mainly considered the application layer to prevent these attacks. Other researches which focus on the network layer also has limitation in detecting this kind of attacks because the researches do not provide a deep XML analysis, which is to find more detail features to classify packets. These solutions can be considered as not enough because an attacker could flood the server with thousands of requests per second using flooding attack, which causes the web service to be unavailable due to consumption of server's resources.
Therefore, this paper aims to address this gap by proposing a middleware tool whichworks on the two layers of the OSI model which are the application layer to detect attacks and network layer to prevent flooding attacks, and uses deep XML analysis techniques. The key concept behind this approach is to analyse the XML packets according to either their content, structure or both. The middleware tool will use a rule-based classification method in determining the classification of these packets, into either normal or malicious.

3. XML-based DoS Attacks

The XDoS attacks flood a web service usingXML-based messages in order to use up all of the server-side resources[19]. The DXDoS attacks can be described as the distributed variant of the XDoS attacks, where multiple hosts are used to launch the attack[20]. In this type of attack, message content manipulation is used, which in turn causes a crash in the web server. Due to the complexity and parsing of XML documents, even a small malformed XML message can cause a huge consumption of server resources [21].
There are numerous typesof XDoS attacks. In this section, anoverview of the XDoS attacks will be discussed. The following XDoS attacks are described in [15] and are also implemented in this work.

Table 1: Overview of the XDoS attacks.

Oversized Payload	XML Entity Expansion
Coercive Parsing	XML External Entity
XML Attribute Count	XML Overlong Names
XML Element Count	

3.1 OversizedPayload Attack.

Very large valid SOAP payload could be sent by the attackers to a web service server, whereby it may cause the server to consume a considerable amount of resources to parse the given SOAP payload, which leads to the DoS attack[12].

3.2 Deeply Nested Payload Attack (Coercive Parsing Attack)

This type of attack is a variant of the oversized-payload attack, where a hacker sends an XML request whichincludes a deeply-nested XML structure. The message will be parsed by an XML parser which will parse the deeply-nested elements continuously[15].

3.3 XML Attribute Count Attack

This type of attack will send a SOAP request with a large number of attributes in order to attack the server[15].

3.4 XML Element Count Attack

This type of attack can attack the web service by sending a SOAP request with a large number of non-nested elements[15].

3.5 XML Entity Expansion Attack

This type of attack forces the web service parser to repeatedly resolve entities defined within a DTD which then causes the DoS.

It is also known asthe Recursive Entities, XML Bomb attack[4].In Figure 1, the parser resolves the entity a to a reference of b and the entity b resolves to a reference of a. Therefore, the parser will loop indefinitely and consumes the CPU resources.

```
<!DOCTYPE data [
<!ENTITY a "&b;">
<!ENTITY b "&a;"> ]>
<data>&a;</data>
```

Figure 1: An example of theXML Bomb attack

3.6 XML External Entity Attack

In this type of attack, the web service parser is forced to resolve external entity defined within a DTD causing a DoS. At the time of parsing, theXML external entity attack (XXE) allows the implication of information from a specific resource either locally or remotely [7]. In Figure 2, the parser will replace the external entity '&xe;' with the content of the system file '/etc/passwd', since it contains confidential data.

```
<?XML version="1.0"?>
<!DOCTYPE myFile [
  <!ELEMENT myFile ANY >
  <!ENTITY xe SYSTEM "file:///etc/passwd">
]>
<myFile>&xe;</myFile>
```

Figure 2: An example of the XX E attack

3.7 XML Overlong Names Attack.

In this type of attack, an attacker injects overlong XML nodes into the XML document. Overlong nodes can be overlong element names, attribute names, attribute values, or namespace descriptions[15].

4. The Proposed System

The proposed tool is developed in such a way that intelligent detection of and prevention from XDoS and HTTP flooding attacks in web servicescan be achieved. The proposed tool provides an easy wayfor the approach to be deployed on any public or e-commerce web service-based application. The proposed tool can be deployed into the same server of web service or in the middleware server. The proposed tool consists of two main components as shown in Figure 3, including the rule-based classification module and the mitigation module.

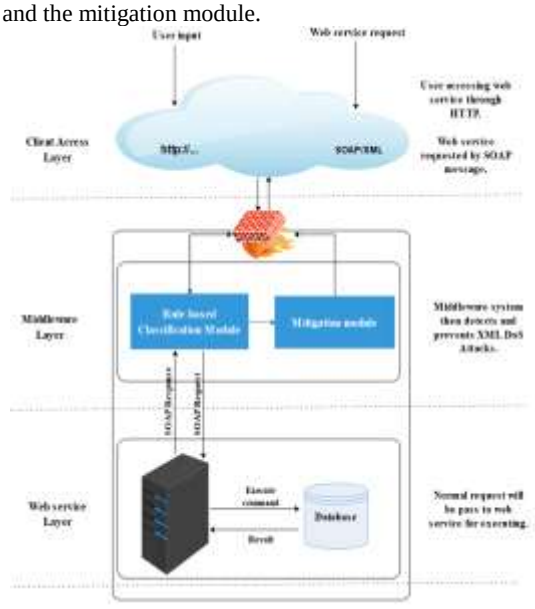

Figure 3: System Architecture

As shown in Figure1, the system architecture consists of three main layers including the client access layer, the middleware layer and the web servicelayer. In the web service, the client access layer can be a human user, a web application or another web service.The second layer, which is the middleware layer, aims to receive the HTTP/SOAP request from the client layer and classifies the request based on a set of rules. The third layer,which is the web servicelayer, aims to receive SOAP requests and execute each request before sending back a SOAP response to the requester with the appropriate results.The middleware layer will be presented in the coming sections with further details.

4.1 Rule-Based Classification Module

The rule-based classification module is responsible for extracting features from incoming requests and for classifying the requests into either normal or malicious requests based on the matching of20 rules. Basically, the rules are obtained by validating 8 features, namely the number of requests, packet size, SOAP size, the number of nested attributes and entity, DTDdeclaration in the SOAP request and the overlong name of attributes or entities.
First, the firewall checks the IP address and compares it to the list of blocked IPs.If it is found within the blocked IPs, then the firewall will reject the connection. If the IP address is not found among the blocked IPs, then the client request will be forwarded to the function which checks for request attempts. Figure 4 depicts thefunctions of the rule-based classification module.

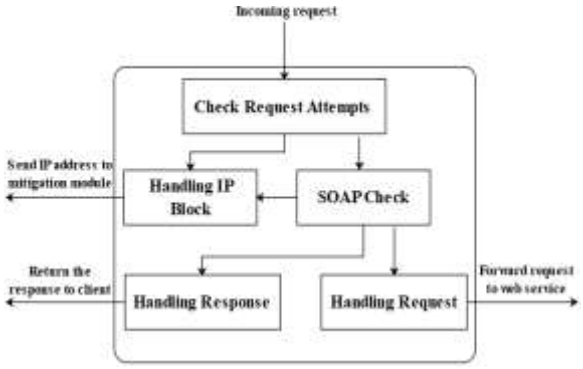

Figure 4: Rule-based Classification Module

4.1.1 Check Request Attempts Function

The middleware tool limits the number of requests to avoid flooding attacks. It only permits a single request from a client in a particular timespan. The default is set at 1000ms. To accomplish this, a monitoring list is used to save clients IPs and the last time that they made a request to the web service.The number of attempts is calculated using the following equation:

$$T_t \leq (R_c - R_l) \quad (1)$$

T_t *Time Threshold.*
R_c *Time of thecurrent request.*
R_l *Time of thelast request.*

4.1.2 SOAP Check Function

At this stage, the features are extracted from the request.The features are checked and verified to ascertain that they arefree from the XML attack, which may cause the service to be unavailable. After the features are extracted, the decision is madebased on the match with the 20 rules. If the decision denies, the request will then be dropped and forwarded to the Handling Response Function.On theother hand, if the decision allows, it will be forwarded to the Handling Request Function. Table 2 shows a sample of the generated rules.

4.1.3 Handling Response Function

The middleware tool returns the following HTTP response to the clients:

a. 200 Ok for normal request.
b. 400 Bad Request for invalid request.
c. 404 Not Found for an URL which could not be found.
d. 405 MethodNotAllowed for a method that is not allowed.
e. 500 Internal Server Error a Server error.

4.1.4 Handling Request Function

This function is responsible for forwarding the client's request to the web service for execution.

4.1.5 Handling IP Block Function

At this stage, the current client's session will be closed and the client's IP will be sent to the mitigation module.

4.2 Mitigation Module

The mitigation module employs blacklisting of suspicious IPs to eliminate the clients' IP addresses which send huge requests to the web server in a short period of time. The mitigation module calculates the ratio of the clients' requests with the baseline ratio before deciding whether to allow or deny the clients' requests. The mitigation module also forwards malicious IPs to the blacklist and it also has the ability to modify firewall rules to block further attempts.

Table 2: Sample of Classification Rules

No	Features								Decision
	Number Request	No Invalid Request	Request Size	SOAP Size	Nested Attribute	Nested Entity	DTD Declaration	Overlong Name	
1	Valid	Valid	Normal	Normal	No	No	No	No	Allow
2	Valid	Valid	Normal	Normal	No	No	No	Yes	Deny
3	Valid	Valid	Normal	Normal	No	No	Yes	No	Deny
4	Valid	Valid	Normal	Normal	No	Yes	No	No	Deny
5	Valid	Valid	Normal	Normal	Yes	No	No	No	Deny
6	Valid	Valid	Normal	Large	No	No	No	No	Deny
7	Valid	Valid	Large	Normal	No	No	No	No	Deny
8	Valid	Exceed	Normal	Normal	No	No	No	No	Deny
9	Exceed	Valid	Normal	Normal	No	No	No	No	Deny
10	Valid	Valid	Normal	Normal	No	No	Yes	Yes	Deny
11	Valid	Valid	Normal	Normal	No	Yes	Yes	Yes	Deny
12	Valid	Valid	Normal	Normal	Yes	No	Yes	Yes	Deny
13	Valid	Valid	Normal	Normal	Yes	Yes	Yes	Yes	Deny
14	Valid	Valid	Normal	Large	Yes	Yes	Yes	Yes	Deny
15	Valid	Valid	Large	Large	Yes	Yes	Yes	Yes	Deny
16	Valid	Exceed	Large	Large	Yes	Yes	Yes	Yes	Deny
17	Exceed	Exceed	Large	Large	Yes	Yes	Yes	Yes	Deny
18	Valid	Valid	-	-	-	-	-	-	Allow
19	Valid	Exceed	-	-	-	-	-	-	Deny
20	Valid	Valid	Normal	-	-	-	-	-	Allow

5. Experimental Results

This section aims to identify the parameters usedin the experiments. First, the type of web service used in this study is the*asmx* web service, and the programming language selected is .Net C#. Microsoft SQL 2014 is chosen as the database. The web service is deployed on the Internet Information Service (IIS) version 10. The middleware tool is developed using the C# programming language. In addition, the extension used to enable for the network layer protection is the windows firewall.
In this work, four computers are used.The first computer uses Windows service provider and the second computer uses Windows Server 2016 operating system as the middleware server.The other two computers pose as a legitimate user and as an attacker respectively. The specifications of the computers that are used in evaluating the proposed method are depicted in Table 3.

Table 3: Experiment specifications

	Web Server	Middleware server	Clients
Operating system	Windows server 2016	Windows server 2016	Windows 10
CPU	Intel(R) Xeon CPU E-52620 v3	Intel(R) Xeon CPU E-52620 v3	Intel Core i5
RAM	16 GB	8 GB	8 GB

Figure 5 show an experimental topology diagram

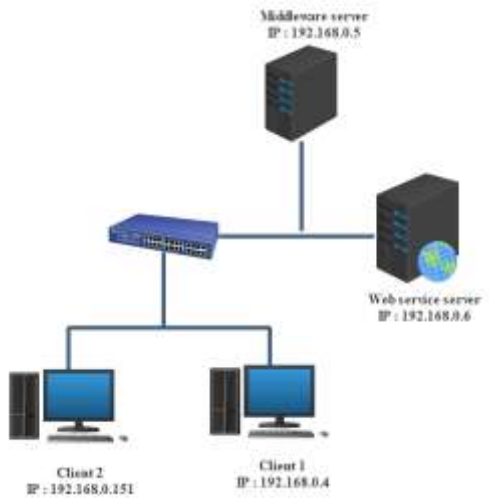

Figure 5: Experimental topologydiagram

Basically, the experiment is conducted to assess the middleware tool in terms of its efficiency in detecting and preventing XDoS and flooding attacks. The evaluation is performed based on the calculation of the server response time, which is adapted from the work of [15, 22]. Both workshad examined the efficiency of the adaptive algorithm on web services, which was based on the server's response time. They concluded that the web service is vulnerable to XDoS attacks, by observing the difference between the request time and the response time. This work will use the same concept for the evaluation process.

To evaluate the efficiency of the proposed method, two experiment scenarioswereconducted. In the first scenario, the web service used the security settings provided by the Internet Information Service (IIS), such as the limited number of requests and the web service security (e.g. username and password). In the second scenario, the web service appliedthe proposed middleware tool. A comparison based on the response time was conducted between the two scenarios in order to observe the results. Therefore, the two experiments were labelled as Scenario 1 and Scenario 2.

Hence, in order to conduct the experiments, a set of normal and malicious SOAP requests were prepared. The malicious SOAP included multiple types of XDoS attacks including the oversized payload, deep nested payload, XML external entity, recursive entities, XML attribute count and the XML entity count attacks. Table 4 shows the details of these SOAP requests.

Table 4: Details of theSOAP request set.

Request	Quantity
Normal	20
Oversized payload	20
Deep nested payload	20
XML external entity	20
XML entity expansion	20
XML attribute count	20
XML entity count	20
Overlong Names	20

The server response time for the prepared SOAP requests was calculated during a normal mode (no attack), and under Scenario 1 and Scenario 2.

Basically, the middleware tool should be initialized with the normal requests.On the first use, the middleware tool would ask the administrator to configure normal requests including the names of the web service methods, the names and types of the parameter for each method, the maximum depth allowed as well asthe maximum number of entities allowed for each web method. These configurations were then saved in the middleware tool setting file. Figure 6 depicts the initialization phase of the middleware tool.

Figure 6: Initialization phase

After the initialization phase, the middleware tool waited for the incoming requests. Each request was saved in a log file for evaluation analysis. Figure 7 shows the middleware tool console.

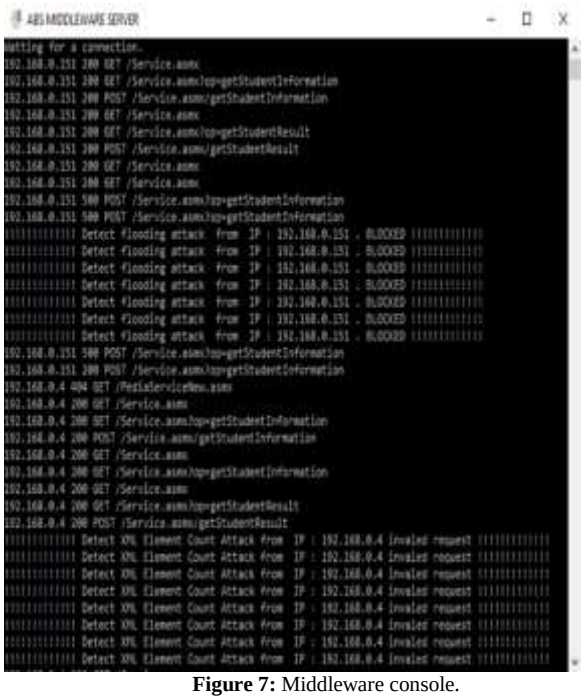

Figure 7: Middleware console.

5.1 Data Collection

The evaluation of the proposed method is based on the server response time. Hence, the server response timefor the prepared SOAP requests (160 requests that are mentioned in Table 4) are calculated during a normal mode (no attack). Figure 8 depicts the method of collecting the response time in the normal mode. In the normal mode, the web service will use the security settings provided by Microsoft IIS version 10, and the countermeasures provided by the asmx web service. In addition, the username and password are added to increase security. There will be no attack in this scenario.

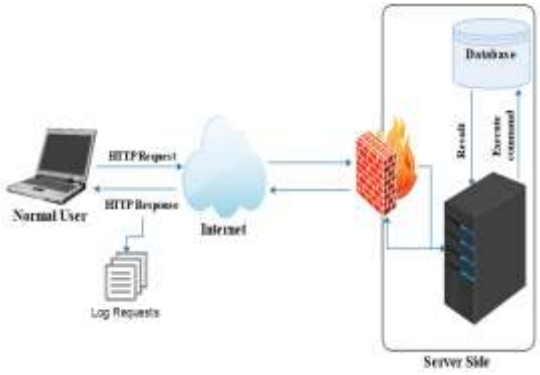

Figure 8: Collection of the server response time in Normal mode

In Scenario 1, the experiment setting was the same as the Normal mode. However, it was simulated to be under attack. The attacker would send malicious SOAP requests to the web service to cause a denial of the service. At the same time, the normal user will send the prepared SOAP requests. Therefore, every request would be calculated. Figure 9 depicts this scenario.

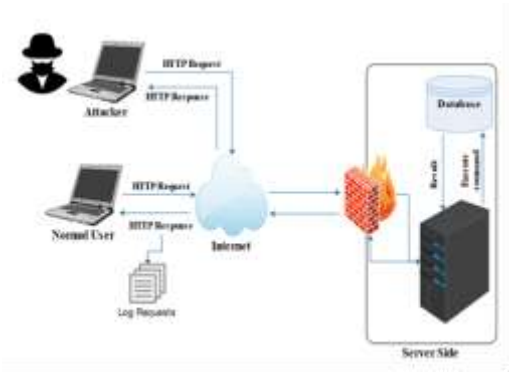

Figure 9: Scenario 1

In Scenario 2, the web service was at the back end, and the users were not able to access it directly. The users had to go through the middleware tool to access the web service. Figure 10 depicts the second scenario.

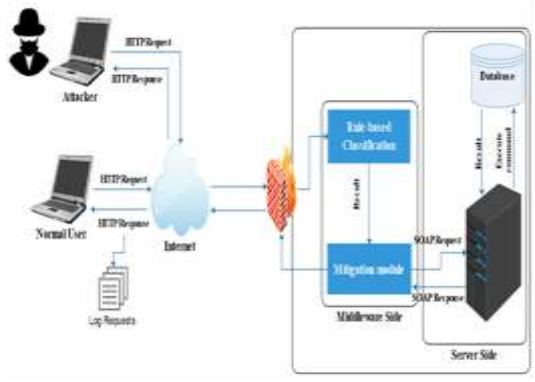

Figure 10: Scenario 2

5.2 Evaluation

This section aims to discuss the results of both scenarios. The evaluation of the two scenarios is divided based on the type of XDoS attacks that are mentioned in Table 4.

5.2.1 Oversized Payload Attack

As shown in Figure11, the results observed from Scenario 2 which usedthe proposeddefense method has outperformed the results observed from Scenario 1 for the 20 'Oversized Payload' requests. On average, Scenario 1 response time is 0.459 second compared with 0.006 second for Scenario 2.

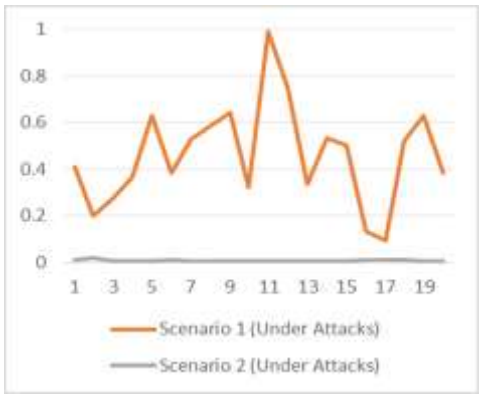

Figure 11: Result of the 'Oversized Payload' request

5.2.2 Deeply Nested Payload Attack

As shown in Figure12, Scenario 2 has outperformed Scenario 1 for the 20 'Deeply Nested Payload' requests. On average, Scenario 1 response time is 0.171 second compared with 0.007 second in Scenario 2.

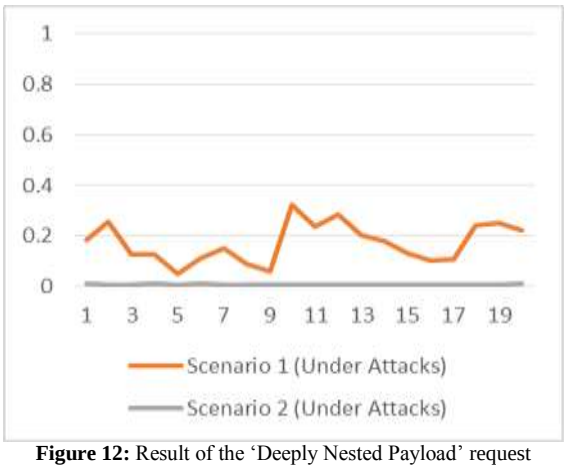

Figure 12: Result of the 'Deeply Nested Payload' request

5.2.3 XML Attribute Count Attack

For the 'XML attribute count attack' requests, the results of both scenarios are depicted in Figure 13. Based onthe results, Scenario 2 has also outperformed Scenario 1 for the 20 'XML attribute count attack' requests. Scenario 1 has an average response time of 0.130 second compared with 0.007 second for Scenario 2.

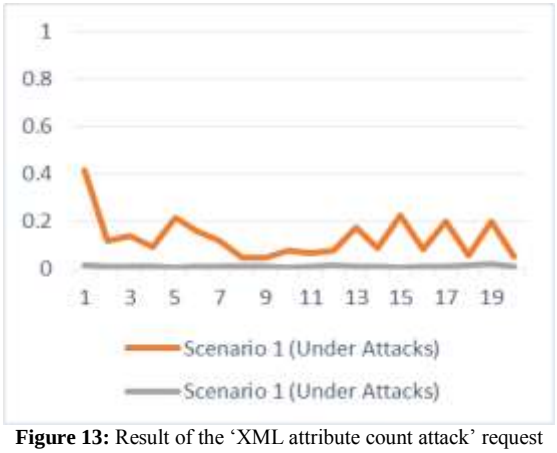

Figure 13: Result of the 'XML attribute count attack' request

5.2.4 XML Element Count Attack

Similarly, for the 'XML element count attack' requests, the results of both scenarios are depicted in Figure 14. The same results are found for this experiment where Scenario 2 has outperformed Scenario 1 and the Normal scenario. On average, Scenario 1 has a response time of 0.118 second compared with 0.006 second in scenario 2.

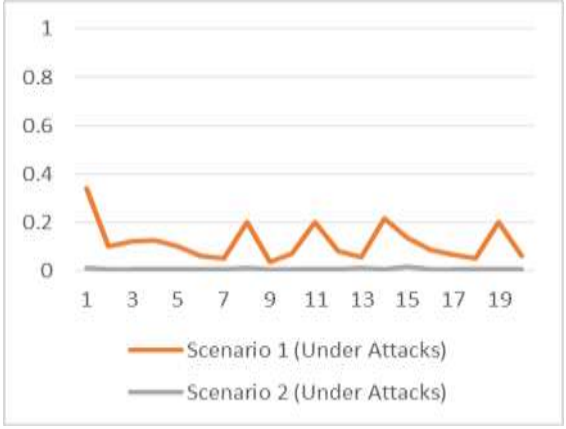

Figure 14: Result of the 'XML element count attack' request

5.2.6 XML entity expansion attack

As shown in Figure 15, Scenario 2 whichused the proposeddefense method has outperformed Scenario 1 for the twenty 'XML entity expansion attack' requests with an average response time of 0.067 second compared with Scenario 1 of 0.126 second.

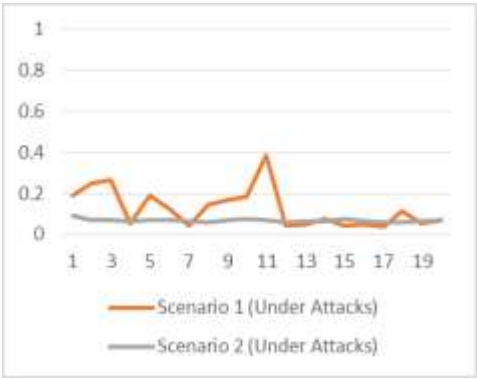

Figure 15: Result of the 'XML entity expansion attack' request

5.2.7 XML External Entity Attack

As for the 'XML external entity attack' requests, the results of both scenarios are depicted in Figure 16.Scenario 2 has again outperformed Scenario 1 for the twenty 'XML external entity attack' requests, with an average of 0.067 second response time compared with Scenario 1, which is 0.081 second.

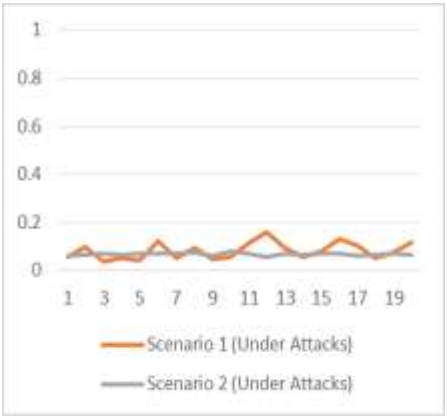

Figure 16: Result of the 'XML External Entity attack'request

5.2.8 XML Overlong Names Attack

Finally, for the 'overlong names' requests, the results of both scenarios are depicted in Figure17.Scenario 2 has also outperformed Scenario 1 for the twenty 'XML external entity attack' requests. On average, Scenario 1 has 0.085 second response time compared with 0.070 second in Scenario 2.

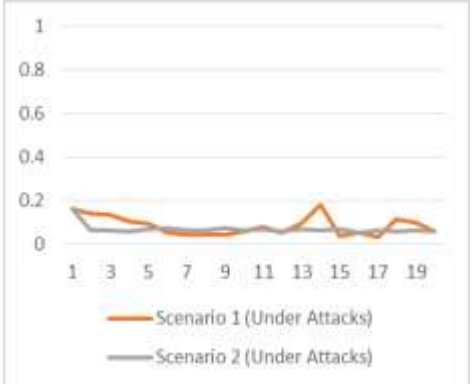

Figure 17: Result of the 'XML External Entity attack'request

Table 5describes the summary of the comparisons of the seven-XDoS attacks set on Scenario 1 (which did not use the proposed defence) and Scenario 2 (which usedthe proposeddefence) in average.

Table 5: Response time for the eight types of SOAP requests

Type	Average response time (in millisecond)						
	Oversized payload	Deep nested payload	XML attribute count	XML element count	XML entity expansion	XML external entity	XML overlong name
Scenario 1	459.75	171.6	130.2	118.2	126.1	81.6	85.2
Scenario 2	6.7	7.1	7.6	6.6	67.5	67.3	70.6

As shown in Table 5, generally the results have demonstrated a significant enhancement achieved by the proposed method. We can see that Scenario 2 which uses the proposed method has a significant difference when compared to Scenario 1 which uses the default security settings.

6. Conclusion

Theavailability of web services is an important factor for business continuity. Therefore, the XDoS attacks pose a real risk for the web services. The XDoS attacks consume a lot of the computational resources, such as the CPU or memory, rendering the system unavailable to the legitimate users.The XDoS attacks that were tested in this paper have demonstrated that they would stopup a web server witha lower amount of resources.
To solve this problem, a middleware tool is proposed and developed. The tests conducted on the tool demonstrate that the tool is efficient in detecting and preventing these attacks. Moreover, the middleware toolprovidesclose to real-time detection and prevention of XDoS and flooding attacks.Furthermore, the middleware tool provides close to 100% service availability for normal requests.

References

[1] Tiwari, S. and P. Singh. Survey of potential attacks on web services and web service compositions. in Electronics Computer Technology (ICECT), 2011 3rd International Conference on. 2011. IEEE.
[2] Jensen, M., N. Gruschka, and R. Herkenhöner, A survey of attacks on web services. Computer Science-Research and Development, 2009. **24**(4): p. 185-197.
[3] Gupta, A.N. and P.S. Thilagam, Attacks on web services need to secure xml on web. Computer Science & Engineering, 2013. **3**(5): p. 1.
[4] Jan, S., C.D. Nguyen, and L.C. Briand. Automated and effective testing of web services for XML injection attacks. in Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016. ACM.
[5] Mainka, C., J. Somorovsky, and J. Schwenk. Penetration testing tool for web services security. in Services (SERVICES), 2012 IEEE Eighth World Congress on. 2012. IEEE.
[6] Shahriar, H., V. Clincy, and W. Bond, Classification of Web-Service-Based Attacks and Mitigation Techniques, in Security and Privacy Management, Techniques, and Protocols. 2018, IGI Global. p. 360-378.
[7] Jan, S., C.D. Nguyen, and L. Briand. Known xml vulnerabilities are still a threat to popular parsers and open source systems. in Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on. 2015. IEEE.

[8] OWASP. OWASP Top 10 Application Security Risks - 2017. 2017; Available from: https://www.owasp.org/index.php/Top_10_2017-Top_10.

[9] Gupta, A. and R. Verma, Securities Perspective in ESB-Like XML-Based Attacks: Interface Abstraction, Data Privacy, and Integrity, in Exploring Enterprise Service Bus in the Service-Oriented Architecture Paradigm. 2017, IGI Global. p. 97-115.

[10] Späth, C., et al. SoK: XML Parser Vulnerabilities. in WOOT. 2016.

[11] Chan, G.-Y., F.-F. Chua, and C.-S. Lee, Intrusion detection and prevention of web service attacks for software as a service: Fuzzy association rules vs fuzzy associative patterns. Journal of Intelligent & Fuzzy Systems, 2016. **31**(2): p. 749-764.

[12] Vissers, T., et al., DDoS defense system for web services in a cloud environment. Future Generation Computer Systems, 2014. **37**: p. 37-45.

[13] Rajaram, A.K. and B.C. Babu. API based security solutions for communication among web services. in Advanced Computing (ICoAC), 2013 Fifth International Conference on. 2013. IEEE.

[14] Sindhu, S. and R. Kanchana. Security solutions for web service attacks in a dynamic composition scenario. in Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on. 2014. IEEE.

[15] Falkenberg, A., et al. A new approach towards DoS penetration testing on web services. in Web Services (ICWS), 2013 IEEE 20th International Conference on. 2013. IEEE.

[16] Utsai, S. and R.B. Joshi, DOS Attack Reduction by using Web Service Filter. International Journal of Computer Applications, 2014. **105**(14).

[17] Anitha, E. and S. Malliga. A packet marking approach to protect cloud environment against DDoS attacks. in Information Communication and Embedded Systems (ICICES), 2013 International Conference on. 2013. IEEE.

[18] Karnwal, T., T. Sivakumar, and G. Aghila. A comber approach to protect cloud computing against XML DDoS and HTTP DDoS attack. in Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on. 2012. IEEE.

[19] Xu, H., A. Reddyreddy, and D.F. Fitch, Defending Against XML-Based Attacks Using State-Based XML Firewall. JCP, 2011. **6**(11): p. 2395-2407.

[20] Chonka, A., W. Zhou, and Y. Xiang. Defending grid web services from xdos attacks by sota. in Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on. 2009. IEEE.

[21] Masdari, M. and M. Jalali, A survey and taxonomy of DoS attacks in cloud computing. Security and Communication Networks, 2016. **9**(16): p. 3724-3751.

[22] Altmeier, C., et al. AdIDoS–Adaptive and Intelligent Fully-Automatic Detection of Denial-of-Service Weaknesses in Web Services. in International Workshop on Data Privacy Management. 2015. Springer.