International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:12, 2012

Moving Towards Positive Security Model For Web Application Firewall

Asrul H. Yaacob, Nazrul M. Ahmad, Nurul N. Ahmad and Mardeni Roslee

***Abstract*— The proliferation of web application and the pervasiveness of mobile technology make web-based attacks even more attractive and even easier to launch. Web Application Firewall (WAF) is an intermediate tool between web server and users that provides comprehensive protection for web application. WAF is a negative security model where the detection and prevention mechanisms are based on predefined or user-defined attack signatures and patterns. However, WAF alone is not adequate to offer best defensive system against web vulnerabilities that are increasing in number and complexity daily. This paper presents a methodology to automatically design a positive security based model which identifies and allows only legitimate web queries. The paper shows a true positive rate of more than 90% can be achieved.

***Keywords*— Intrusion Detection System, Positive Security Model, Web application Firewall

I. Introduction

WITH the increased number of Internet's users, many applications have been designed to provide more and more services. Applications ranging from simple web site to a complex word editing application are now available in Internet. With the proliferation of the applications, everyone can create their own web site and web application with a few clicks. Several applications are more vulnerable to attack than the others. This is due to the fact that the applications are developed by community which includes both highly experienced and new programmers. The increased number of vulnerabilities is consistent with the number of applications deployed [1]. By exploiting vulnerability in the application, the attacker is able to do things, which by default are not authorised such as modifying the content of web pages, deleting tables in database or even distributing malwares. Thus, it is very important to protect the web application. The efforts taken by several organisation such as SANS and OWASP contributes to the decrease of incidents [1].

As reported in [2], [3], the top vulnerabilities in web application are SQL injection (SQLi) and cross-site scripting (XSS). These two types of vulnerabilities are caused by a common mistake in application development: improper sanitisation of user-supplied input [1]. As users control the input, the developers need to check before using it. In SQLi, an attacker uses the input to send SQL command to directly interact with the backend database. This makes possible for the attacker to retrieve sensitive information or even modify the database. In XSS, an attacker sends or stores a malicious script. Contrary to SQLi, the victim of the attack is not the application but the users of that application. Using this mechanism, attacker may distribute malwares to the visitor of the infected web. One of the ways to avoid the attacks, web developer needs to be more careful [4].

One approach to mitigate the problem is by using Web Application Firewall (WAF). It is considered as a subset of Intrusion Detection System (IDS), focusing mainly on the detection of attacks for web application. WAF has capability to perform the detection in two techniques: signature-based and anomaly-based. A signature-based WAF detects the attack by comparing the request to a database of known attack signature. As the latter offers the possibility to detect known and unknown attacks, researchers focus more on this technique compared to the former one. Various techniques have been investigated [5]–[10].

A signature-based WAF is a set of rules to identify the attacks, either known or unknown. The process of identifying the attacks is known as negative security or black listing model [11]. Most of the researches focus on designing WAF based on this model. The model needs to be updated when a new attack or a variant of old attack surfaces. Alternatively, in positive security or white listing model, a set of normal requests is created. Using this model, the list is updated if and only if the web application is updated. As the positive security model recognises only the normal request, the negative security model is also need to identify correctly the attack. Thus, the combination of both models should give the best result for WAF in terms of accuracy and speed of detection.

The objectives of this study are to investigate the possibility of constructing an automated positive security model and to measure the performance in identifying legitimate request. In this paper, the positive security model is constructed by using the log of our official web server. It is then used to classify the requests into either legitimate or attacks. The main contribution of this paper is the methodology to automatically construct a positive security model that can be used in WAF.

The remainder of this paper is structured as follow: Section II focuses on Hyper Text Transfer Protocol (HTTP) and the details of SQLi. Section III presents the background of WAF and both of the models used; positive and negative models. The description of our basic model and the methodology in building an automated positive security model is presented in section IV. The experiments and results are described in section V. The last section contains the conclusion and future work.

Asrul H. Yaacob and Nazrul M. Muhaimin are with Faculty of Information Science and Technology (FIST), Multimedia University (MMU), Jalan Ayer Keroh Lama, 75450 Melaka, Malaysia. (phone: +606-2523680; email: {asrulhadi.yaacob, nazrul.muhaimin}@mmu.edu.my)

Nurul N. Ahmad and Mardeni Roslee are with Faculty of Engineering (FoE), Multimedia University (MMU), Jalan Multimedia, 63100 Cyberjaya, Selangor, Malaysia. (email: {nurulnadia.ahmad, mardeni.roslee}@mmu.edu.my)

II. The Protocol & The Threats

A. The Protocol

Hypertext transfer protocol (HTTP) is the main protocol to access web application [12]. It is based on client-server architecture. A web client or browser sends a request to a web server. The server receives the request and replies to the client accordingly. HTTP request format and example are depicted in Fig. 1 and Fig. 2 respectively. On the other hand, the format and example of HTTP reply are depicted in Fig. 3 and Fig. 4 respectively.

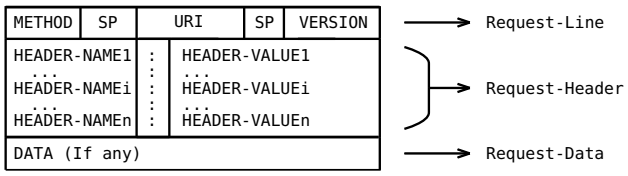

Fig. 1 HTTP Request Format

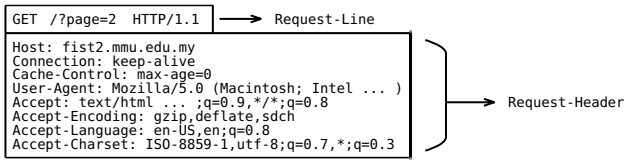

Fig. 2 HTTP Request Example

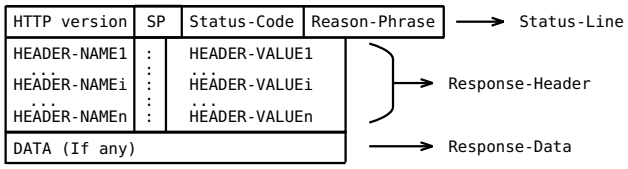

Fig. 3 HTTP Reply Format

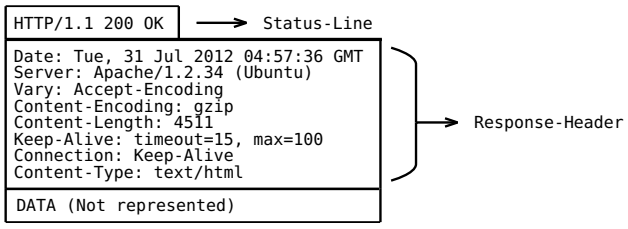

Fig. 4 HTTP Reply Example

Based on the Fig. 2, the method used to obtain the web page is GET. This is the most common and the default request method. This method also is used when users click on a link of a web page. With this request method, the DATA field is generally empty as there is no user-supplied data. If there is a need to transmit data from users, two request methods can be used; GET and POST. In the former request method, the data is encoded in the Uniform Resource Identifier (URI) part or the request. In the later request method, the data from users in encoded in the DATA field.

The information in the HEADER part is generated by browser, as shown in the Fig. 2, where the request is from Mozilla browser. It contains the information related to the request and the session such as cookies and the capabilities of the browser or server. This information is used by browser and server to communicate better and to identify the user. Generally, users do not need to modify or know the content of the header. The same goes for the header part of the reply message from the server.

To identify an attack towards web application, it is important to know the information that can be modified by the attacker. The easiest way to transmit the attack is with the URI. The information in the URI includes the location, the name as well as the parameters for a resource as depicted in Fig. 5. The parameters, if any, are normally generated by web application at the server side. In some cases, the user input will form the parameters of the URI. Based on the parameters submitted, a different result is send by the web application.

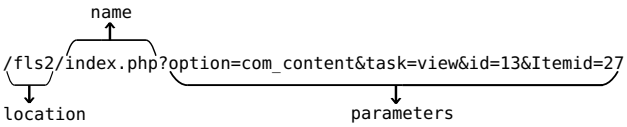

Fig. 5 URI Example

Using a modified and often malicious value of parameters, a vulnerable web application can be attacked. The attacker can directly types the URI that contains a malicious parameters in the browser that resulted a GET request method. Specialised tools can also be used to submit a more sophisticated attack using POST. In this case, the parameters are normally in DATA part of the request. Detecting an attack submitted using GET method is easier than detecting an attack using POST method. This is because URI is normally recorded in a log file while the DATA part of the request is not stored.

B. The Threats

The submitted parameters are used by the web application at server side. The respond from the server may vary depending on the value of the parameters. Developers are expected to check the input before using it. Since the query can be modified by attackers, there is a high probability that the parameters contains malicious data. Two major attacks related to this are SQLi and XSS.

It is common to have a database connected to the server and the parameters are used to form an SQL query. In Fig. 5, parameter *id* or *Itemid* may be used in a program as shown in Algorithm 1. The value of *id* generated by the web application is valid thus forming the expected SQL query. In case of SQLi, the attacker provides an *id* containing malicious SQL query. One of the possible value for *id* is '*1; DROP TABLE content_table*'. If the web application is vulnerable and the code is executed, the table *content_table* will be deleted. Other examples of SQLi can be found in [13].

The parameters can also be used to display a dynamic message to the visitor. If the parameters are not verified, it is possible to submit a malicious message to another visitor.

Algorithm 1 Example of vulnerable SQL query

```
SELECT contents
FROM content_table
WHERE id=$_GET['id'];
```

An example of PHP code vulnerable to XSS is presented in Algorithm 2. If the parameter *name* is set correctly to a person name, e.g. 'code.php?name=guest', then everything is run as expected. However, if the attacker launches an XSS attack using 'code.php?name=guest<script>alert('attacked')</script>', then a popup windows will be executed.

Algorithm 2 Example of code vulnerable to XSS

```
<?php
  $name = $_GET['name'];
  echo "Welcome $name<br>";
  echo "Go to <a href="http://yourhome.com/">
      Home</a>";
?>
```

Both of the attack describe previously sent the malicious content to the web server. The target of SQLi attacks is the web server whereas the target of XSS is another user. A successful SQLi attack leads to server compromise. In order to prevent the attack, malicious codes need to be block before reaching the server. In XSS, the server will act as an intermediate system in an attack towards the end user. The malicious codes need to be block before relaying it to the end users.

III. Web Application Firewall

A. Definition

Web application becomes one of the main platforms for the attackers to gain access to the system. In order to protect web application, the administrator can deploy WAF. The main functionality of WAF is to protect the web application from attacks or intrusions. WAF inspects both incoming and outgoing traffic to web server [14]. The concept of intrusion detection is widely covered under IDS and IPS researches. Thus, most of the researches apply the same concepts in developing WAF. The researches in WAF are mainly on the detection technique based on anomaly and negative security model.

In order to inspect web traffic, WAF operates in one of the following modes: reverse proxy, transparent proxy and host-based. In reverse proxy mode, the WAF is place between the web client and web server. The clients use the address of the WAF as the web server and all the requests are forwarded to WAF. This enables the WAF to inspect the requests and based on the result, WAF will forwards if it is legitimate or will drop if it is an attack. In transparent proxy mode, the functionality is the same as reverse proxy except that the web server address is used directly and WAF is transparent to the users. Both reverse and transparent proxy may require a separate device. In host-based mode, WAF is part of the web server and normally function as a module. All the requests are first analysed by WAF module and then followed by the web server module.

Two approaches are used to detect the attacks; signature-based and anomaly-based. The former is used to identify a known attacks and a regular updates of signatures is required whereas the later is used to identify unknown or new attacks which will be the deviation of the model constructed in the initial attacks-free learning phase. Both of the approaches will identify the attack instead of legitimate web requests. This model is known as negative security model where the focus is on the attacks. Another security model is positive security model where the focus is on legitimate web requests.

B. Negative Security Model – Blacklisting

A negative security model focuses in the detection of attacks. In signature-based detection approaches, it defines the signatures of attacks. This list is also known as blacklist. Construction of blacklist requires an expert domain to analyse a known attack. Generally, the patterns in the blacklist are generic and with some customisation can be used for all web application. Every web request needs to be compared to all the patterns in the blacklist before it can be considered attack-free. The process of verification depends on the size of the blacklist and often it is time consuming [15].

Defining the patterns in negative security model can be a very complex task. Even though the basic model is quite generic, web administrator still needs to customise the model accordingly. Applying the generic model to a web application without customisation normally will result into a high false positive rate [7]. The customisation also needs a domain expert, as there is no automated process that could help. A common way to define the model is to use regular expression but such move to define the attack imposes some risks [16].

The blacklist also needs to be updated regularly and the frequency of updates depends on the vendors. Thus, the accuracy of detection depends on the updates rate [15]. Even though the blacklist is updated on time, it is still not a complete list of all possible attacks. Attackers can used various techniques in order to evade the detection [7]. The evasion techniques like white space diversity, fragmentation, segmentation and encodings can be used in SQLi [17].

Since the signature-based detection cannot detect unknown attacks until it is updated, many of the researchers focus on the anomaly-based detection technique. This technique consists on modelling a normal behaviour and detects the deviation as an attack [18]. Thus, new attacks can be identified if it is significantly deviates from normal behaviour. Nevertheless, this technique still needs to be used together with signature-based technique due to lower detection rate and throughput [19].

C. Positive Security Model – Whitelisting

Another model in WAF is positive security model. In this model, the focus is on the legitimate or valid requests. A whitelist is constructed based on the model of legitimate request. It is then used to classify the incoming request. A request that is not in the whitelist is considered as an attack.

The size of whitelist is smaller compared to blacklist as it only contains the model of legitimate request. Even though every requests needs to be analysed, the result is known at the first match again one of the signatures or patterns. Thus, one of the advantages of this model compared to the negative security model is that legitimate requests will be processed faster. Since the majority of the requests are legitimate [20], the time spends in analysis is lesser.

In order to create the whitelist, the same approaches as previous could be used. The signature of legitimate requests can be constructed manually as mentioned in [7]. In this case, the whitelist contains all the valid resources in the forms of strings or regular expressions. It is also possible to use an advanced technique such as in [18] to create a model of legitimate requests. The whitelist only needs to be updated if the web application changes significantly, which will occurs less frequently compared to negative security model. For a large-scale web application, it is simply not possible to manually define the whitelist containing every single page and possible parameters, as it is too complicated. Thus, in order to use positive security model, it is desirable to have an automated ways to create the whitelist.

IV. Automated whitelist model

A. Basic Model

Assuming that the web server is already functional, it is possible to automatically create a model based on the information in the log file. A standard log file contains the required information to create the model. The more information we have in the log file, the more reliable model could be constructed. From the log, we extract the following information:

- IP address of the client
- Requested URI
- Referrer
- Response size
- Response status code

As the main source of information is the log file, the model created can be considered very basic. It is impossible to have a complete analysis, as the most of the information on the HEADER and the DATA part is not present. The model created will be able to identify a normal request mainly based on the URI. As illustrated in Fig. 6, the model together with identification module is used together in order to protect web server.

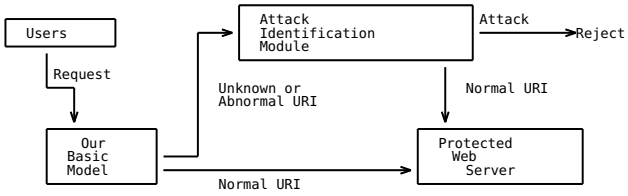

Fig. 6 Usage of Basic Model

From the log file, the information is extracted and saved into a database. The URI is separated into smaller part: location, name and parameters. The model in this paper consists of a list of valid URI extracted from the log. Any URI that is not in the list is considered as an unknown request. A further analysis is needed to determine if the request is malicious or not. We have identify three cases where the model will classify the request as unknown:

1) Low hit: A URI with a low request number will not satisfy the required we set even though it is a valid one.
2) New resources: The model identifies the URI as unknown when a new web page or resource added as in basic model only known URIs are included.
3) Attack scenario. As stated in A., attacker needs to modify the parameters and sent the modified URI in order to launch an attack. In this case, the model will be able to identify the request as unknown.

B. Building the Model

The model is built based on a URI that is considered normal. In this paper, we are building a model that contains a complete and valid URI obtained from log. A valid and normal URI is requested by normal and trusted users whereas a malicious URI is requested by the attackers. By identifying a trusted user, it is possible to determine either URI is valid or not. Since the information about the user is not in the log, we use the client's IP and referrer as an alternative. Also included in the model the URI for static resources.

1) Static Resource: A webpage contains dynamic and static resources. Dynamic resources changes based on the parameters given. A scripting or programming language is used to create dynamic resources. Example of dynamic resources includes pages generated with PHP. Whereas, static resources do not change and normally do not required any parameter. Example of static resources includes images, CSS file and javascript file. One of the characteristics of static resources is fixed size. Based on this criteria, we include in the model, the URI that always return the same response size.

2) Trusted Machine : As user information is not included in the log, there is a need to identify a valid and trusted user based on others information. In this case, the IP of the user can be used. If a trusted and valid user using the same computer whenever he want to connect to the server, then the computer is consider trusted. An example is the computer of the web administrator. If it is possible to differentiate the administrator's machine from others, then the URI send by that machine can be included in the model.

3) Trusted Referrer : It is trivial to include URI of static resource and URI from trusted machine in the model. A less trivial one is to include URI from a trusted referrer. A referrer is the URI of a resource. Most of the time, it is a web page's URI. Assuming a user is viewing a web page and he click on a link, both of the URI are normally transmitted to the server. The URI of the link is placed in the URI part of the request message, whereas the URI of the web page is transmitted in the HEADER part of the request. A normal web browser formats the request message with correct information. Based on referrer from the log, it is possible to identify the valid URI. Assuming the attackers may forge the referrer information, not all referrer may be trusted. A trusted referrer must be selected

in order to ensure the validity of the URI. It is important to identify the referrer before adding the URI in the model. A trusted referrer must have a high number of:

- resources if refer to: that means the referrer contains many links to the other resources in the same server. The links can be images, another web page or javascripts
- client having it as a referrer: that means many clients visited the page (referrer) and follow the links in the page

From this list of referrers, only the top-n of the referrer is considered as trusted referrer. It is then possible to added URI having the trusted referrer to the model. The referrer itself can be added to the model if it is from the same server.

All the URI to be included in the model can be automatically extracted from the web server's log, even if there are attacks recorded in the log. The model can be updated by the administrator as needed. Once the model is created, it can be used until a new web application is used.

V. Experiment and Results

To test the performance of our model, we used the log file for our faculty web server. The log contains 7008 records from two web applications. A manual verification and classification was done for all the records. This verification is important as it is used as a based for our model. The request is classified into normal request, attack and deadlink. A deadlink request is a request to a deleted resource. From the log, the number of distinct URI is 758. The distribution between normal request, attack and deadlink is shown in Table I and Table II.

TABLE I
Distribution of Overall Records

	Number	Percent
Normal	6940	99.03
Attack	61	0.87
Dead Link	7	0.10
Total	7008	100.00

TABLE II
Distribution of Distinct URI

	Number	Percent
Normal	758	94.75
Attack	36	4.50
Dead Link	6	0.75
Total	800	100.00

Our model is created using the log file based on the concepts describe in Section B.. Since the objective of positive security model is to identify a normal request, we classify the record to either manual or unknown. The result is then compared to the manual verification. Only the requests classified as normal are compared. As our focus is more on the trusted referrer, only the local machine, i.e. 127.0.0.1, is considered as trusted machine. Overall result is shown in Table III. The result shows that both static and trusted machine contribute a little in forming the model. The classification of URI is mostly based on the URI from trusted referrer. This could changes if we include more machine in the trusted machine.

The total number of requests identified as normal by our model is not simply the sum of the three techniques. Several URI may be identified as valid by more than one technique. Corollary, the total number of unknown is less than the sum of the three techniques. URI that is not identified by one techniques may be identified by others techniques. By combining the three techniques, our model is able to identify correctly 90.908% of the normal requests.

TABLE III
Overall Result

	Normal	Unknown	TP Rate (%)
Static resource	585	6423	8.429
Trusted machine	356	6652	5.130
Trusted referrer	5932	106	85.476
Total	6309	699	90.908

The result in Table III is obtained by limiting the number of trusted referrer, by resources and by client, to the top 20 referrers only. The actual number referrer is not exactly $2\times$N as there may be the same referrer in both lists. For top 20, it is about 12% of the total 238 referrers identified from the log file. Table IV show the result of identification, by trusted referrer and overall, when the number of trusted referrer is varied. The last row shows the maximum true positive rate for our model since we include every referrers in the trusted referrer.

TABLE IV
Trusted Referrer

Top-N	No of referrer	Trusted referrer		Overall	
		No	%	No	%
20	29	5932	85.476	6309	90.908
30	45	5967	85.980	6332	91.239
50	66	5984	86.225	6349	91.484
100	138	6072	87.493	6437	92.752
300	238	6128	88.300	6493	93.559

The overall result show the rate of identification by the model. Table V show the number of URI contains in the model. This URI need to be compared with the number of URI manually identified, Table II. Only trusted referrer is shown since it provide the majority of the identification. From this table, we know that the model covers only about 40% of the valid requests. Even though the model only covers less than half of the valid requests, the true positive rate is more than 90%. This is mainly because of the remaining URIs not included in the model are a low hit request or infrequently accessed. Another indication about the low hit resources is to look at the difference in the increase rate of the model and the overall identification. An increase of 10% in the completeness of the model only increases 2% in the overall identification.

TABLE V
Completeness of Model

Top-N	Trusted referrer	
	No	%
20	299	39.45
30	321	42.35
50	338	44.59
100	378	49.87
300	410	54.09

VI. Conclusions

In this paper we present a method to build an automatically a whitelist of valid HTTP request to be used in a positive security model in WAF. The previous log of web server is used to create the model. The result show that it is possible to have identify correctly the valid request with identification rate more than 90%. The model should help to reduce the work to identify an attack, as the number of unknown request will be less than 10%. Even though the model created only covers about 40% of valid request, the performance in identifying a normal request is satisfying.

Since the model is created based on the log, it is not able to identify a new resources added to the web application. Our future work will be to integrating a generic URI capable to identify a new resources based on the current URI pattern generated from web application. Another area of improvement includes the detection of low hit resources.

References

[1] T. Scholte, D. Balzarotti, and E. Kirda, "Have things changed now? An empirical study on input validation vulnerabilities in web applications," *Computers & Security*, vol. 31, no. 3, pp. 344–356, May 2012.

[2] OWASP, "OWASP Top 10 Application Security Risks - 2010," OWASP The Open Web Application Security Project, Tech. Rep., 2010.

[3] WhiteHat Security, "WhiteHat Website Security Statistic Report - Winter 2011," WhiteHat Security, Tech. Rep., 2011.

[4] Symantec Corp., "Symantec Internet Security Threat Report," Symantec Inc., Tech. Rep., 2011.

[5] H. T. Nguyen, C. Torrano-Gimenez, G. Alvarez, S. Petrović, and K. Franke, "Application of the Generic Feature Selection Measure in Detection of Web Attacks," in *Computational Intelligence in Security for Information Systems*, ser. Lecture Notes in Computer Science, vol. 6694. Springer, 2011, pp. 25–32.

[6] M. F. Abdollah, A. H. Yaacob, S. Shahib, I. Mohamad, and M. F. Iskandar, "Revealing the Influence of Feature Selection for Fast Attack Detection," *International Journal of Computer Science and Network Security*, vol. 8, no. 8, pp. 107–115, 2007.

[7] A. Moosa, "Artificial Neural Network based Web Application Firewall for SQL Injection," *World Academy of Science, Engineering and Technology*, no. 64, pp. 12–21, 2010.

[8] V. Alarcon-Aquino, C. A. Oropeza-Clavel, J. Rodriguez-Asomoza, O. Starostenko, and R. Rosas-Romero, *Intrusion Detection and Classification of Attacks in High-Level Network Protocols Using Recurrent Neural Networks*. Springer Netherlands, 2010, pp. 129–134.

[9] A. H. Yaacob, I. K. T. Tan, S. F. Chien, and H. K. Tan, "ARIMA Based Network Anomaly Detection," in *2010 Second International Conference on Communication Software and Networks*, no. 1. Ieee, 2010, pp. 205–209.

[10] A. Gulve, "Survey On Intrusion Detection System," *International Journal Of*, vol. 4, no. 1, pp. 7–13, 2011.

[11] A. Razzaq, A. Hur, M. Masood, K. Latif, H. F. Ahmad, and H. Takahashi, "Foundation of Semantic Rule Engine to Protect Web Application Attacks," in *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*. Ieee, 2011, pp. 95–102.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616, Hypertext Transfer Protocol – HTTP/1.1," 1999.

[13] F. S. Rietta and G. Way, "Application layer intrusion detection for SQL injection," in *Proceedings of the 44th annual southeast regional conference on ACMSE 44*. ACM Press, 2006, p. 531.

[14] S. Stankovic and D. Simic, "A Holistic Approach to Securing Web Applications," *Journal of Computing*, vol. 2, no. 1, pp. 16–20, Jan. 2010.

[15] R. Koch, "Towards Next-Generation Intrusion Detection," in *Cyber Conflict (ICCC), 2011 3rd International*, 2011, pp. 1–18.

[16] D. Bates, A. Barth, and C. Jackson, "Regular expressions considered harmful in client-side XSS filters," in *Proceedings of the 19th international conference on World wide web - WWW '10*. New York, New York, USA: ACM Press, Apr. 2010, p. 91.

[17] O. Maor and A. Shulman, "SQL Injection Signature Evasion Whitepaper," 2004.

[18] C. Torrano-Gimenez, A. Perez-Villegas, and G. Alvarez, "A Self-learning Anomaly-Based Web Application Firewall," in *Computational Intelligence in Security for Information Systems*, ser. Advances in Intelligent and Soft Computing, A. Herrero, P. Gastaldo, R. Zunino, and E. Corchado, Eds. Springer Berlin / Heidelberg, 2009, vol. 63, pp. 85–92.

[19] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, Feb. 2009.

[20] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.