Improving Security of Web-Based Application Using ModSecurity and Reverse Proxy in Web Application Firewall

Rizki Agung Muzaki
Cyber Security Engineering,
Department of Cyber Security
Politeknik Siber dan Sandi Negara
Bogor, Indonesia
rizki.agung@student.poltekssn.ac.id

Obrina Candra Briliyant
Cyber Security Engineering,
Department of Cyber Security
Politeknik Siber dan Sandi Negara
Bogor, Indonesia
obrina@poltekssn.ac.id

Maulana Andika Hasditama
Computer Engineering, Faculty of
Electrical Engineering and Informatics
Budapest University of Technology
and Economics (BME)
Budapest, Hungary
m.hasditama@edu.bme.hu

Hamzah Ritchi
Department of Accounting, Faculty of
Economics and Business
Universitas Padjadjaran
Bandung, Indonesia
hamzah.ritchi@unpad.ac.id

***Abstract*— The use of web applications has been undergoing rapid increase. Many individuals, groups, organizations or governments use web applications as a means to exchange information or support business-related tasks. Despite the increased adoption, web applications use is however directly associated with comparable threats and attacks. With the increasing threats and attacks on web applications, organizations require a more effective concept of web application security. Web Application Firewall (WAF) is a security concept that can be used to prevent various threats and attacks on web applications. WAF has the ability to filter packets, block dangerous HTTP requests, and also do logging. This paper demonstrates and proposes the implementation of WAF on a web-based application using ModSecurity and the Reverse Proxy method. From the tests carried out e.g. cross-site scripting, SQL injection and unauthorized vulnerability web scanning, all threats were successfully thwarted by ModSecurity and reverse proxy method implemented in the WAF.**

Keywords—Web Application Security, Web Application Firewall, ModSecurity, Reverse Proxy Method

I. PRELIMINARY

A. Background

The development and use of information technology are currently increasing so rapidly. Virtually all fields of work are facilitated by the internet. One of its uses is the use of web applications. The ease of getting through the internet is an important factor in the increasing use of web applications. Web applications typically consist of web servers and web pages. Based on a survey conducted by Netcraft until January 2018, the most widely used web server application is Apache [1]. Web pages are documents that are usually written in HTML, PHP, and other web programming formats that are accessed through the HTTP protocol [2].

Statistical data obtained by Positive Technologies, shows that the six main targets of attacks on web applications in 2018 are attacks on institutions in the financial sector, transportation companies, information technology, entertainment, health, and government institutions [3]. In Indonesia, government sites are the most frequent targeted objects of cyber-attacks [4]. The National Cyber Security Operations Center (Pusopskamsinas) of the National Cyber and Crypto Agency (BSSN) recorded 884,142 website incidents that occurred from January 1 to April 12, 2020 [4]. Based on the above reported data, it is important to conduct security improvement to prevent, reduce and overcome attacks from occurring. Some of the causes of attacks that easily occur in a web application are inadequate infrastructure used, and in its manufacture, there are still deficiencies in the concept of security [5].

One solution that can be used to prevent threats to web applications is to implement a Web Application Firewall (WAF) in a web application. WAF has the ability to filter packets, block HTTP traffic and also logging. ModSecurity is one of the most stable and effective Open Source-based application security modules that can be utilized for WAF implementation [6]. There are rules in ModSecurity that can be configured to prevent threats that occur in web applications. Because of its Open Source nature, ModSecurity can be an alternative solution at a low cost to strengthen the security of web applications.

In this research the WAF implementation was implemented on a web application. The WAF used in this study uses the ModSecurity application security module using the Reverse Proxy method. The ModSecurity module is installed on a Reverse Proxy device. Added configuration rules from OWASP, namely OWASP Core Rule Set so that the performance of ModSecurity becomes more complex. The implementation of WAF that has been done is tested by conducting an attack test on a web application. The attacks used for attack tests are *Cross-Site Scripting (XSS), SQL Injection,* and *Unauthorized Vulnerability Web Scanning*. The three types of attacks used to test attacks are based on a list of the most dangerous vulnerabilities in web applications according to the OWASP *Top Ten Web Application Security Risks* [7] .

II. THEORETICAL BASIS

A. Web Application Security

Security in web applications is an issue that needs attention. There are several solutions that can be done to implement security services on web applications. Even though this is not completely perfect, it is a preventive measure from unwanted things. One solution that can be

978-1-7281-9098-3/20//$31.00©

applied is to implement a Web Application Firewall (WAF). The WAF serves to provide a look into the packet data traffic from a web application and can also function to block several attacks on web applications. Based on the OWASP Top Ten list, the top three attacks are Injection, XSS (Cross-Site Scripting), and Broken Authentication and Session Management [8] . These three attacks as it is widely known are attacks against targets against web servers.

B. Web Application Firewall

Web Application Firewall (WAF) is a security method in web applications. WAF is likened to a security shield for applications that are accessed using HTTP [9]. The location of the WAF on a network topology is in front or as a barrier between external and internal networks. The WAF seeks to prevent threats from attackers. In order to work, additional configuration needs to be made to the web server, but there is no need to make changes to the application builder script, so that the WAF can be applied to applications that are already running. Like firewalls in general, WAF filters incoming and outgoing data and can stop or block network traffic that is considered dangerous in accordance with the rules that have been applied.

C. ModSecurity

ModSecurity is a Web Application Firewall (WAF) module that is Open Source. ModSecurity works as a security application module for web servers. The function of ModSecurity is to detect and prevent attacks on web applications. Modsecurity provides configuration rules called SecRules to monitor HTTP traffic in real-time, logging, and filtering HTTP traffic based on the rules that have been applied [10]. The rules of ModSecurity are flexible so they can be configured according to the security needs of the web application.

D. Reverse Proxy Method

The Reverse Proxy method is a security method used to hide web servers. In the Reverse Proxy method, the client is not aware that the request sent does not go directly to the web server but instead is directed to the proxy server, so the client only knows the address of the proxy server [11]. The web server which is topologically covered by Reverse Proxy location is unknown to the client.

Reverse Proxy can also be used as a Web Application Firewall (WAF) deployment method for a web application. WAF which will be implemented in web applications on devices that have Reverse Proxy configured, so that the scope of WAF in increasing security becomes wider. The attacker also cannot see the actual location of the web server because it is topologically closed by the [9].

III. Research Methodology

The research method used in this study is an experimental research method. The experimental research method is part of a quantitative research method where the research is carried out in a laboratory with the aim of finding the effect of certain treatments on the object of research under controlled conditions [12]. The experimental research method is used because of its characteristics, namely the variables in the study can be chosen by researchers. Other variables that can affect the experimental process can also be tightly controlled.

A. Research Stages

The research stages followed the process of preparation, design, implementation, testing, and analysis for drawing conclusions.

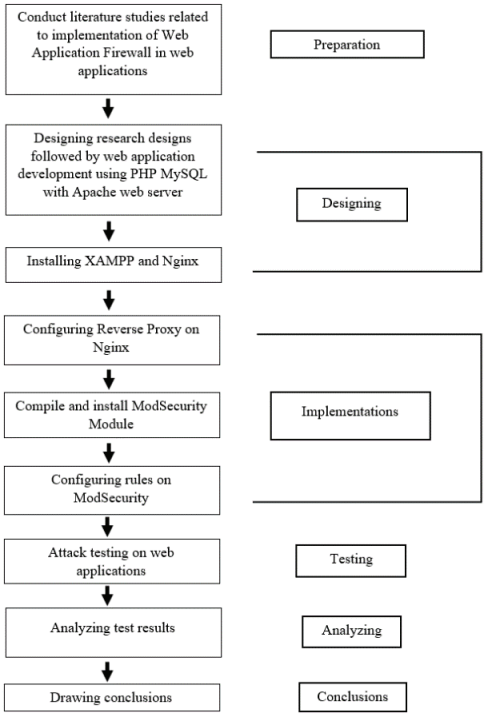

Fig 1. Research Stages

B. Preparation Process

The preparation process is the initial stage before conducting research. In this process, a literature study and consultation were conducted on matters that will be carried out in the research. The literature being studied were selected to address topics related to Web Application Firewall (WAF), the importance of WAF to improve security functions in web applications, how to implement WAF in web applications, the ModSecurity module, the configuration of Reverse Proxies on web servers, most common attacks on web applications, and the tools used in implementation and testing for the experiment.

C. Design and Implementation

At this stage the design of the research environment was conducted by creating a network topology followed by the development of the design environment. After that, the research implementation was then carried out.

- Creating a Network Topology
  The design of the research environment began by creating a network topology as the basis for the development of the research environment. Figure 2 illustrate the network topology designed.

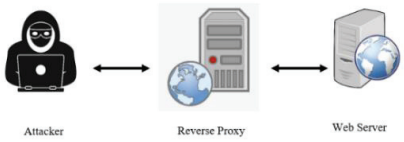

Fig 2. Network Topology

It can be seen that the network topology consists of 3 objects, namely the attacker, the Reverse Proxy device, and the web server.

- Building the research environment
  The research environment was established according to the network topology that had been created. The following are the steps to build the research environment:
  1. Configure XAMPP as an Apache web server.
  2. Configure Reverse Proxy on Nginx as a proxy for the Apache web server.
  3. Compile and Install ModSecurity as a WAF security module in the Nginx proxy server.
  4. Configure rules on ModSecurity using the Open Web Application Security Project Core Rule Set (OWASP CRS).

D. Testing

The current research posits that the implementation of WAF in a web application may improve security functions, so as to prevent damage from attacks that may occur against web application. Testing conducted in this research were determined to be performed under two conditions. The first was attacks on a web application by not applying ModSecurity and Reverse Proxy and the second was an attack test on a web application applying ModSecurity and Reverse Proxy. Several variables are used at the testing stage, they are:

- Independent variable : Two conditions of applying ModSecurity and Reverse Proxy to a web application during an attack, namely active and inactive.
- Bound variable : The security level of the web application after an attack test, which is bound to two conditions of applying ModSecurity and Reverse Proxy.
- Control variable : There were three different simulated attacks that targeted the same object in which two treatments were implemented, namely with and without applying ModSecurity and Reverse Proxy. The attacks used were *Cross-Site Scripting (XSS), SQL Injection,* and *Unauthorized Vulnerability Web Scanning*.

E. Analysis

After testing, the next step was the analysis to see whether if there were different result between the first and second conditions. The following are the indicators of test success:

- If in the first condition three attacks are successfully carried out, while in the second condition all three attacks can be prevented by ModSecurity and Reverse Proxy, then the implementation is effective in improving the security of the web application.
- If in the first condition three attacks were successfully carried out, while in the second condition there were one or two attacks that successfully penetrated the web application, then the result of the implementation was less effective in improving the security of the web application.
- If in the first and second conditions all three attacks are successfully carried out and can penetrated the web application, then the implementation of ModSecurity and Reverse Proxy are not effective in improving the security of the web application.

The results of the analysis are then used to draw conclusions.

IV. Discussion

Test attacks against the web applications were employed under two conditions. The first, the attacks were done to web application with no ModSecurity and Reverse Proxy applied. In the second condition, attacks were done to web application with ModSecurity and Reverse Proxy applied. There were three type of test attacks, namely *Cross-Site Scripting* (XSS), *SQL Injection*, and *Unauthorized Vulnerability Web Scanning*.

A. Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attack test is called *stored XSS*. The technical attack of *stored XSS* is by inserting malicious scripts in the webpage of the web application, then the scripts will be stored permanently on the server. The attack was carried out to the registration page of the web application.

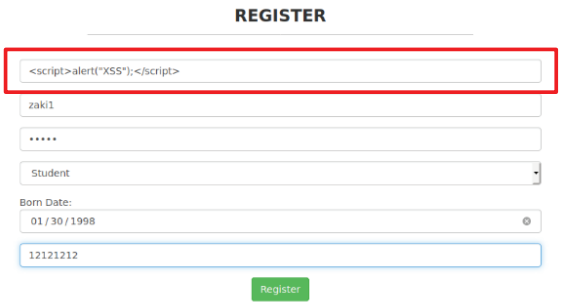

Fig 3. *Stored XSS* Attack Script

1) ModSecurity and Reverse Proxy not applied.

The result of the *stored XSS* attack against the web application without applying the ModSecurity and Reverse Proxy is “successful”, indicated by the "XSS" alert appears on the webpage that displays the input results from the register page, as shown in Figure 4.

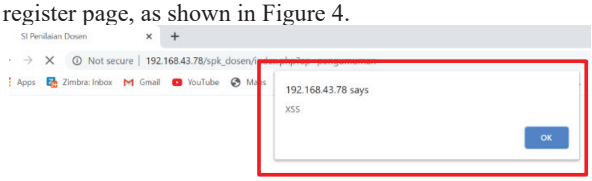

Fig 4. *Stored XSS* Attack Succeed

2) ModSecurity and Reverse Proxy applied

The result of the *stored XSS* attack on the web application with ModSecurity and Reverse Proxy applied was “unsuccessful”, indicated by the web application responded with the 403 / Forbidden code, that prevent the malicious script stored permanently on the server database, as shown in Figure 5.

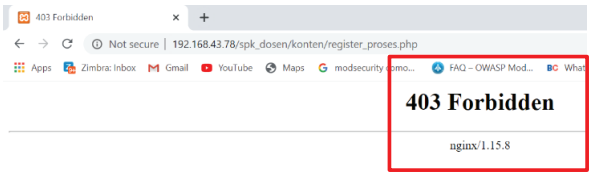

Fig 5. *Stored XSS* Attack Failed

ModSecurity has detected a *stored XSS* attack and prevent attack access. Following is the log of the *stored XSS* attack attempt:

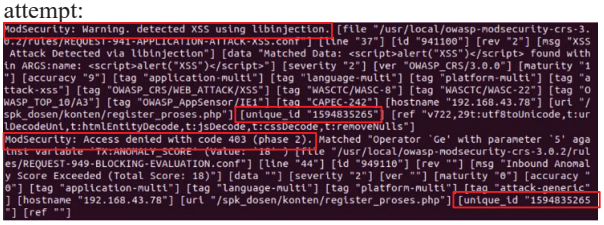

Fig 6. *Stored XSS* Attack Log

B. *SQL Injection*

SQL Injection attack test conducted by entering the *SQL Injection* script on the login page. The purpose of the *SQL Injection* attack is so that the attacker can access the web application as a legitimate user. The attack was done by entering malicious SQL scripts in the username and password fields.

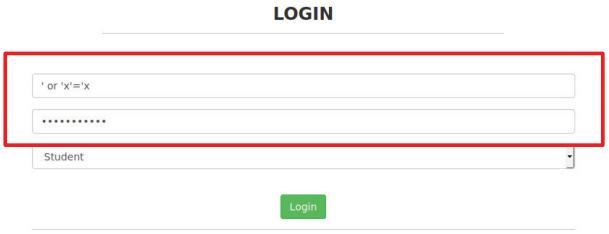

Fig 7. *SQL Injection* Attack Script

1) *ModSecurity and Reverse Proxy not applied*

The results of the *SQL Injection* attack on the web application without applying the ModSecurity was successful, indicated by unauthorized access to the web application login page, as shown in Figure 8.

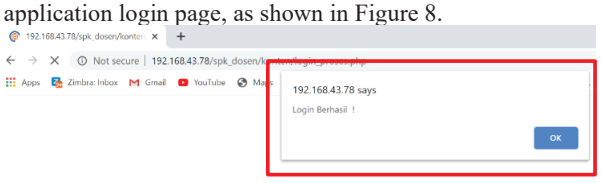

Fig 8. *SQL Injection* Attack Succeed

2) *ModSecurity and Reverse Proxy applied*

The result of the *SQL Injection* attack on the web application with ModSecurity and Reverse Proxy applied were unsuccessful, indicated by the web application responded with the 403 / Forbidden code, that prevent access granted on the login attempt, as shown in Figure 9.

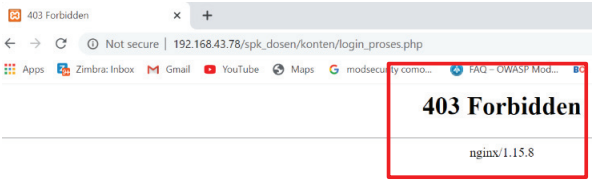

Fig 9. *SQL Injeciton* Attack Failed

ModSecurity has detected an *SQL Injection* attack and prevent attack access. Following is the log of the *SQL Injection* attack attempt:

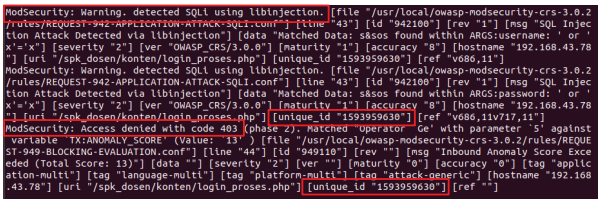

Fig 10. *SQL Injection* Attack Log

C. *Unauthorized Vulnerability Web Scanning*

Unauthorized Vulnerability Web Scanning attack were conducted using Arachni 1.5 tools. The purpose of the Unauthorized Vulnerability Web Scanning attack is to find out the vulnerabilities that exist in the web application without consent from the application owner/provider.

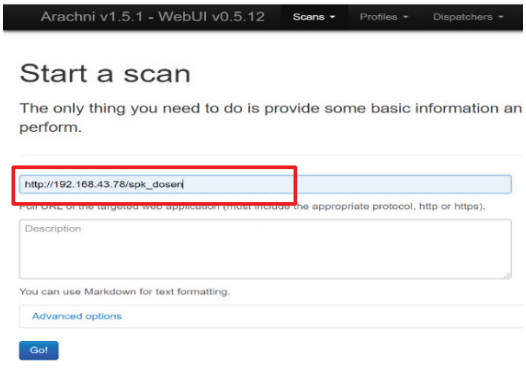

Fig 11. Unauthorized Vulnerability Scan Using Arachni 1.5.1

1) *ModSecurity and Reverse Proxy not applied*

The result of the Unauthorized Vulnerability Web Scanning attack on the web application without ModSecurity and Reverse Proxy enabled were successful, indicated by vulnerabilities found by Arachni 1.5.1 with a total of 38 vulnerabilities. The details of the 38 issues obtained are 8 Medium level issues, 25 Low level issues, and 5 Informational level issues, with the time required by Arachni 1.5.1 to perform a scan that is 28 minutes 4 seconds. The scan results are shown in Figure 12

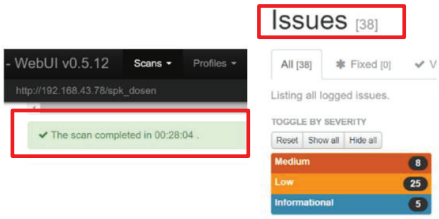

Fig 12. *Unauthorized Vulnerability Web Scanning* without ModSecurity and Reverse Proxy enabled

2) ModSecurity and Reverse Proxy applied

The results the Unauthorized Vulnerability Web Scanning attack on the web application with ModSecurity and Reverse Proxy enabled were less successful, indicated by only 2 vulnerabilities found. The details of the 2 issues were 1 low level issue, and 1 informational issue, with the time required by Arachni 1.5.1 to scan is 42 seconds. The scan results are shown in Figure 13.

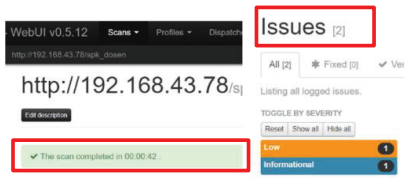

Fig 13. *Unauthorized Vulnerability Web Scanning* with ModSecurity and Reverse Proxy enabled

ModSecurity successfully detects a scan carried out by an attacker and prevent scan access. Following is the log of the *Unauthorized Vulnerability Web Scanning* attack attempt:

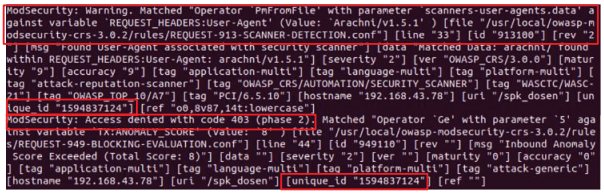

Fig 14. *Unauthorized Vulnerability Web Scanning* Attack Log

D. Analysis

Based from the results of the attack tests that have been conducted, we can analyzed the research variables using a table of analysis that summarize the testing conditions and the testing results.

TABLE 1. Attack Test Results Table

Attack	Web Application Conditions	
	First (without WAF)	*Second (with WAF)*
SQL Injection	Succeeded	Failed
Cross-Site Scripting	Succeeded	Failed
Unauthorized Vulnerability Web Scanning	Succeeded	Failed

Based on the results of the three attack tests that have been carried out. It shows that the attacks on the web application were successful in the condition where no ModSecurity and Reverse Proxy were applied, and unsuccessful or failed in the condition where ModSecurity and Reverse Proxy were applied. From the testing results, all three attacks can be prevented by ModSecurity and Reverse Proxy, then the implementation is effective in improving the security of the web application.

The following is an explanation of the results of the attack test analysis that has been carried out:

1) Results of Cross-Site Scripting Attack Test Analysis

The test results of stored XSS attack in the first condition are successful. An "XSS" alert appears on the page which was previously entered via a script on the Register page. The entered script is successfully stored in the web application database. Then when user open the page where the page displays data stored in the database, including the stored XSS script. The request sent by the attacker is sent through the Reverse Proxy without checking from the WAF because ModSecurity is disabled. From the Reverse Proxy, it is immediately forwarded to the web server for a response. Without WAF, the web application cannot detect that the request sent by an attacker is a malicious request that has a stored XSS script in it.

The results of the attack test in the second condition failed because ModSecurity was successful in warding off the attack, which was indicated by the web application giving a 403 or Forbidden response code. The request sent by the attacker through the Reverse Proxy is checked first by ModSecurity before proceeding to the web server. If the request is detected as dangerous, then the request will be rejected, so the request does not reach the web server. From the attack log in Figure 6, requests that contain XSS stored scripts have been detected successfully by ModSecurity via Libinjection contained in OWASP CRS rules, to be precise in the configuration of rules "REQUEST-941-APPLICATION-ATTACK-XSS.conf". Code id 941100 is a code of rules to perform XSS malicious script checks on packets from received requests. The script that ModSecurity detected successfully through Libinjection, namely <script> alert (“XSS”); </script>. Then the request is marked as a malicious request by ModSecurity by providing unique_id 1594835265. After the stored XSS attack is successfully detected, ModSecurity then rejects the request from the attacker using the configuration rules "REQUEST-949-BLOCKING-EVALUATION". Code id 949110 is a rules code for rejecting malicious requests based on the previously created unique_id, namely unique_id 1594835265.

2) Results of SQL Injection Attack Test Analysis

The results of the SQL Injection attack test in the first condition were successfully carried out. Login page was successfully compromised using the entered script, allowing attackers to access the service as a legitimate user. The request sent by the attacker is sent through the Reverse Proxy without checking from the WAF because ModSecurity is disabled. From the Reverse Proxy, it is immediately forwarded to the web server for a response. Without WAF, the web application cannot detect that the request sent by the attacker is a malicious request with a SQL Injection script in it.

The second condition attack test results failed because ModSecurity was successful in warding off the attack, which was indicated by the server giving a response code 403 or Forbidden. The request sent by the attacker through the Reverse Proxy is checked first by ModSecurity before proceeding to the web server. If the request is detected as dangerous, then the request will be rejected, so the request does not reach the web server. From the attack log in Figure

10, ModSecurity detected a request that contained a SQL Injection script through Libinjection contained in the OWASP CRS rules, to be precise in the configuration rules "REQUEST-942-APPLICATION-ATTACK-SQLI.conf". Code id 942100 is code from rules to perform malicious SQL script check on packet from received request. The script that ModSecurity detected successfully through Libinjection, namely 'or' x '=' x. Then the request is marked as a malicious request by ModSecurity by providing unique_id 1593959630. After the SQL Injection attack is successfully detected, ModSecurity rejects the request from the attacker using the configuration rules "REQUEST-949-BLOCKING-EVALUATION". Code id 949110 is a rules code for rejecting malicious requests based on the previously created unique_id, namely unique_id 1593959630.

3) Results of Analysis of Unauthorized Vulnerability Web Scanning Attack Test

The results of the Unauthorized Vulnerability Web Scanning attack on the first condition were successfully carried out. The Arachni 1.5.1 vulnerability scanner tool successfully scans the web application with 38 issues. The request sent by the attacker is sent through the Reverse Proxy without checking from the WAF because ModSecurity is disabled. From the Reverse Proxy, it is immediately forwarded to the web server for a response. In every request sent by arachni to look for vulnerabilities in the SIPDOSEN web application, there is a Request Header containing a User-Agent. The value of the User-Agent sent is Arachni / v1.5.1. In the first condition, the request sent by the attacker using Arachni 1.5.1 tools is not checked when the packet passes through the Reverse Proxy because ModSecurity is in an inactive state,

The results of the second condition attack test failed because ModSecurity was successful in warding off the attack, which was marked by the failure of Arachni 1.5.1 to perform a full scan and only received 2 issues. The request sent by the attacker through the Reverse Proxy is checked first by ModSecurity before proceeding to the web server. If the request is detected as dangerous, it will reject the request, so the request does not reach the web server. From the attack log in Figure 14, ModSecurity has successfully detected a vulnerability scanning attempt using the configuration rules “REQUEST-913-SCANNER-DETECTION.conf”. Code id 913100 is a code of rules to check User-Agent values of requests that are checked for ModSecurity. When an attacker sends a request using Arachni 1.5.1 through a Reverse Proxy, ModSecurity checks the contents of the packet. ModSecurity successfully detected a vulnerability scanning attempt by checking the User-Agent value on the packet being sent. The User-Agent value sent by the attacker contains “Arachni / v1.5.1”, where the value is contained in the scanners-user-agents.data file, so the request sent is included in the category of vulnerability scanning attempts. Then the request is marked as a malicious request by ModSecurity by providing unique_id 1594837124. After the vulnerability scanning attempt is detected, ModSecurity then rejects the request from the attacker using the configuration rules "REQUEST-949-BLOCKING-EVALUATION".

V. Conclussions and Recommendations

Based on the testing and analysis stage that has been done, it can be concluded that implementing WAF in a web-based application using ModSecurity and Reverse Proxy method can add to the security functions of web applications. From the attacks carried out e.g. *Cross-Site Scripting, SQL Injection* and *Unauthorized Vulnerability Web Scanning*, all threats were successfully thwarted by ModSecurity and reverse proxy method implemented in the WAF.

For further research, development can be carried out to provide *interface* of the attack log, so it can help more efficiently detects attacks. Other development can also be done to make an alert system so that the application’s administrator can be notified if there is an attack attempt.

References

[1] Netcraft, "Netcraft," 28 Februari 2019. [Online]. Available: https://news.netcraft.com/archives/2019/02/28/february-2019-web-server-survey.html.

[2] V. Clincy and H. Shahriar, "Web Application Firewall: Network Security Models and Configuration," in *42nd IEEE International Conference on Computer Software & Applications*, 2018.

[3] Positive Technology, "Attacks on web applications: 2018 in review," 26 Juni 2019. [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/web-application-attacks-2019/.

[4] Security Advisory, "Rekap Serangan Siber (Januari – April 2020)," Badan Siber dan Sandi Negara, 20 April 2020. [Online]. Available: https://bssn.go.id/rekap-serangan-siber-januari-april-2020/. [Accessed 30 08 2020].

[5] Anggrahito, R. Ibrahim, A. Fajri and E. Murniyanti, "Implementasi Web Application Firewall Menggunakan ReverseProxy dan ModSecurity Sebagai Alternatif Pengamanan Aplikasi Web Pada Sektor Pemerintah," *CITEE,* pp. 199-205, 2019.

[6] S. Prandl, M. Lazarescu and D.-S. Pham, "A Study of Web Application Firewall Solutions," in *ICISS*, Perth, 2015.

[7] OWASP, "OWASP Top Ten," OWASP, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 31 07 2020].

[8] B. Sullivan and V. Liu, Web Application Security A Beginner's Guide, New York: The McGraw-Hill Companies, 2012.

[9] J. Pubal, "Web Application Firewalls," SANS Institute Reading Room , 2015.

[10] Trustware SpiderLabs, "ModSecurity Open Source Web Applications Firewall," Trustwave Holdings, Inc., 2020. [Online]. Available: https://modsecurity.org/about.html. [Accessed 31 08 2020].

[11] The Apache Software Foundation, "Apache HTTP version 2.4," The Apache Software Foundation, 2020. [Online]. Available: http://httpd.apache.org/docs/current/mod/mod_proxy.html#page-header. [Accessed 12 07 2020].

[12] Sugiyono, Metode Penelitian Kuantitatif, Kualitatif, dan R&D, Bandung: Alfabeta.CV, 2017.