EXPERIMENTAL STUDY OF MODSECURITY WEB APPLICATION FIREWALLS

Co-authored by Timilehin David Sobola

Pavol Zavarsky

Sergey Butakov

Project report

Submitted to the Faculty of Graduate Studies,

Concordia University of Edmonton

in Partial Fulfillment of the

Requirements for the

Final Research Project for the Degree

MASTER OF INFORMATION SYSTEMS SECURITY MANAGEMENT

Concordia University of Edmonton

FACULTY OF GRADUATE STUDIES

Edmonton, Alberta

April 2020

EXPERIMENTAL STUDY OF MODSECURITY WEB APPLICATION FIREWALLS

Timilehin David Sobola

Approved:

Pavol Zavarsky [Approval on File]

Pavol Zavarsky Date: April 6, 2020

Primary Supervisor

Edgar Schmidt [Approval on File]

Edgar Schmidt, DSocSci Date: April 17, 2020

Dean, Faculty of Graduate Studies

Experimental Study of ModSecurity Web Application Firewalls

Timilehin David Sobola
Information Systems Security Management
Concordia University of Edmonton
Edmonton, Canada
tsobola@student.concordia.ab.ca

Pavol Zavarsky
Information Systems Security Management
Concordia University of Edmonton
Edmonton, Canada
pavol.zavarsky@concordia.ab.ca

Sergey Butakov
Information Systems Security Management
Concordia University of Edmonton
Edmonton, Canada
sergey.butakov@concordia.ab.ca

***Abstract*— Risks related to web security are too important to be ignored. The Open Web Application Security Project (OWASP) document maintains a rating of the top 10 common threats. Although not an official standard, is widely acknowledged in the classification of vulnerabilities. This paper evaluates the effectiveness of ModSecurity web application firewall with OWASP Core Rule Set (CRS) version 3.2 released in September 2019 to detect known web security risks. This paper proposes to provide insight on detection capability of ModSecurity with CRS v.3.2 at default level, how well it can protect web server against Denial of Service (DoS) attacks, and performance on web server in terms of Throughput (the average amount of bytes transmitted every second), Transaction rates (the amount of hits), Concurrency (the average number of parallel connections and increases as server efficiency declines). In addition, provides recommendation on areas of improvement and future research areas.**

Keywords—web application firewall (waf), OWASP, ModSecurity, OWASP Core Rule Set, throughput, transaction, concurrency

I. INTRODUCTION

Web application security is an information security division concerned with web security, web services and web applications. Protection for web applications is based on security concepts for applications but is applied directly to the web and network systems [2]. Most web application vulnerabilities are caused by cross-site scripting (XSS) and SQL injection attempts, typically made possible by faulty coding and failure to sanitise software inputs and outputs.

Akamai in its State of Internet / Security Volume 6 report, mentioned that, unique DDoS targets accounted for more than 40% against the conventional login (username and password) financial services industry, accounting for the plurality (74%) of device and server entry methods between November 2017 and October 2019 [9]. They experienced a record - attack against a financial company. The identified attacks contained 55,141,782 malicious login attempts registered on 7 August 2019. One of these attacks is the credential stuffing. Akamai [9] reported 662,556,776 attacks against the financial services industry on web apps and 7,957,307,672 attacks on all verticals. SQL Injection (SQLi) contributed to more than 72% of all attacks across all verticals. Local File Inclusion (LFI) was the leading form of attack against the financial services sector with 47% of activity recorded. DDoS attacks are not only a successful way of gaining targets ' attention, but they could also mask certain forms of threats, such as SQLi and LFI [9]. Hackers may pursue various methods to attack until they have effectively made significant progress, which is why a solid, multi-layer security strategy is required to prevent these attacks.

Web application firewalls (WAFs) identifies, monitors, and prevents HTTP traffic to and from a web server. Application vulnerabilities such as SQL injection, cross-site scripting (XSS), file inclusion and protection bugs can be prevented by controlling HTTP traffic. Many vendors provide firewall solutions for web applications. According to Gartner Magic quadrant [4], some of the top solutions include Cloudflare, ModSecurity, Fortinet, Barracuda Networks, Imperva, Qualys. Each WAF mentioned has its own unique features, but its performance is highly influenced by the policies configured.

The OWASP Core Rule Set is a project maintained also by the OWASP team as a set of generic rules used with ModSecurity aiming to protect against wide range of attacks including the OWASP Top Ten web risks [7].

This paper presents test results of ModSecurity with CRS v.3.2 against OWASP Top 10 risks to evaluate its effectiveness in detecting attacks and performance on web server. ModSecurity gives the ability to customize rules to industry specific needs. ModSecurity is commonly used for tracking, logging and identity management of web applications in real-time. The overall objective of this paper includes:

- Evaluating the performance of ModSecurity with CRS v.3.2 at default install (i.e. paranoia level 1).
- Study reasons for existing limitations of ModSecurity with CRS v.3.2 in detecting some attacks.
- Present some recommendations on how improvements can be made.

The next parts of this research paper are structured as follows. Section II outlines some related works with findings and limitations in their papers. Section III discusses some of the standards developed by the Web Application Consortium (WASC) to test web application firewalls. Section IV outlines the steps and tools used to perform the objectives of the research paper successfully. Section V provides detailed information on findings based on experiments conducted. Section VI describes the limitations and recommendations from the experiments conducted which are followed by conclusion in Section VII.

II. RELATED WORKS

There are not many papers out there that focus extensively on the effectiveness of ModSecurity firewall with CRS. Some

either place focus on detection capabilities or impact of the paranoia levels in ModSecurity firewall.

A study by Kim [10] evaluated the effectiveness of ModSecurity in detecting XSS and SQLi attacks without investigating a more comprehensive web attack set.

Jatesh Singh study [11], enumerated more comprehensive attacks without limitations to XSS and SQLi as some research papers do. The paper evaluated the effectiveness of paranoia levels in ModSecurity firewall with CRS v3.0 to detect vide range of web attacks but did not consider firewall performance on web server and firewall ability to detect or prevent DoS or DDoS attacks.

This paper evaluates a more comprehensive area of effectiveness of ModSecurity with CRS v3.2 in terms of detection capabilities (keeping in scope DoS attacks) and performance on web server by load testing.

III. Evaluation Criteria

The following assessment requirements established by the Web Application Security Consortium (WASC) are used to test ModSecurity with CRS v.3.2:

A. *Deployment Architecture*

It illustrates the key issues that decide the feasibility of applying a firewall to a web application in an environment. ModSecurity is configured in reverse proxy mode in this paper where traffic is redirected to pass by the WAF through DNS configuration changes or network-level traffic redirection. It can also be configured in embedded mode where it is installed on a server as a plug-in.

B. *Detection Techniques*

Web application firewalls must be able to identify manipulation intents and turn the data into a structured sequence to make rules and signatures usable [5]. Some normalization techniques used by ModSecurity are:

- URL-decoding (e.g. %XX)
- Self-referencing paths (i.e. usage of /./ equivalent and encoded alternatives).
- Eliminating comments (e.g. transform DELETE/**/FROM to DELETE FROM).

C. *Performance*

Performance is a complex issue. It is especially difficult to measure the performance of a WAF on the network level. This paper covers HTTP-level performance with the scope of:

- Maximum throughput.
- Maximum transaction rates.
- Request latency.
- Management under high attack load.

IV. Test Methodology

To conduct the tests, various tools that had the functionality needed to achieve the objectives effectively were chosen. Table I provides a list of the tools used to perform our experiments.

TABLE I. List Tools Used

Tools	Description
Kali Linux	Linux-based operating system featuring pre-installed penetration testing tools.
Metasploitable2	Deliberately unsafe Linux virtual machine that can be used for security training, security tools testing, and conventional penetration testing.
Nmap scanner	Free and open source (licensed) network exploration and security auditing functionality.
Apache Bench	Single-threaded command line computer program designed to measure HTTP web server performance.
THC Hydra	Parallel password cracker which supports various attack protocols.
Weevely	Stealth PHP browser shell, simulating telnet link.
Siege	Multi-threaded benchmarking and http load testing tool.
SlowHTTPTest	Used to send partial HTTP requests to get a denial of service from target HTTP server

A. *Network Setup*

With the tools gathered, we created a testbed to simulate the experiment. Figure I below show the network diagram used for experimentation.

Fig. 1. Network architecture used for simulations.

Kali Linux is used to simulate the attacker machine. It has preinstalled penetration testing tools. ModSecurity is configured in reverse proxy mode on Ubuntu OS running Apache 2.2.8 server. Metasploitable2 is configured as the vulnerable webserver with two vulnerable web applications inbuilt (Damn Vulnerable Web Application (DVWA) and Mutillidae). We used DVWA for this experiment as it has a much-simplified layout than mutillidae. We configured CRS v.3.2 with ModSecurity and left all settings in the default install state.

Each experiment followed the steps shown in Table II to obtain reliable results. The main objective is to test for performance at default install, we did not focus on creating or editing rules in CRS v.3.2.

TABLE II. Test Steps

1. Configure ModSecurity with default CRS v.3.2 rule sets.	**Planning**
2. Perform tests for different paranoia level.	**Execution**
3. Check ModSecurity audit log for alerts if attack is detected or not. 4. Analyze result and clear log file to prepare for another attack process and run Step 2 again for another level.	**Analysis**

V. RESULTS AND DISCUSSION

The following attack vectors were used to the effectiveness of ModSecurity with CRS v.3.2 from the OWASP Top 10 risks. Table III below, shows a summary of the results from the simulated attacks done keeping in scope the four paranoia levels (PL) and not limiting to just the default level (PL 1).

TABLE III. TEST SUMMARY

Attacks	PL 1	PL 2	PL 3	PL 4
XSS Stored (file upload)	Failed	Failed	Failed	Failed
XSS Reflected	Pass	Pass	Pass	Pass
SQL Stored Injection	Failed	Pass	Pass	Pass
SQL Injection in URL (GET)	Pass	Pass	Pass	Pass
SQL Injection in Login forms (POST)	Pass	Pass	Pass	Pass
PHP code injection	Pass	Pass	Pass	Pass
Command Injection	Pass	Pass	Pass	Pass
Path Traversal	Pass	Pass	Pass	Pass
Local and Remote File inclusion	Pass	Pass	Pass	Pass
DoS Attacks				
Slow Headers (Slowloris)	Down	Down	Down	Down
Slow Body (R-U-Dead-Yet)	Down	Down	Down	Down
Range Attack (Apache Killer)	Up	Up	Up	Up

where,

Failed *– Successful attempt to bypass the firewall.*
Pass *– Failed attempt to bypass the firewall.*
Down *– DoS attempt rendered service, unavailable.*
Up *– DoS attempt but service still available.*

This asset/target and threat-based approach formed the baseline of our experiment in evaluating the effectiveness of ModSecurity with CRS v.3.2 in detecting web attacks.

A. *Performance of ModSecurity with CRS v.3.2 in detecting cross-site scripting (XSS) attacks*

The test summary of the XSS stored and reflected attacks are shown in Table III. To conduct our experiment on XSS stored attack, we uploaded a **.xhtml** file to the webserver backend which contained a malicious JavaScript code that logs the session cookie of the web application in the browser. With the approach, we were able to bypass the firewall at the different paranoia levels.

B. *Performance of ModSecurity with CRS v.3.2 in detecting stored SQL injection attack*

For this analysis, we tried different payloads to try retrieve a list of users from the database of the webserver. CRS v.3.2 was able to detect these payloads and logged them. We tried injecting "**1 exec sp_ (or exec xp_) AND 1=1**" payload and was able to retrieve a user from the database. CRS v.3.2 rule set was unable to detect this payload at the default paranoia level i.e. PL1 but was blocked at PL 2 to 4.

C. *Performance of ModSecurity with CRS v.3.2 in detecting protocol attack*

This attack was performed using *Weevely* to perform a URL encoding abuse attack to breach protocol. We generated a shell file i.e. **<filename>.sh** on the attacker machine and then uploaded the file to the webserver. The firewall should be able to detect such file being uploaded but no detection. With the file successfully uploaded, we tried to use *Weevely* to connect remotely to the webserver using the shell file we uploaded but the firewall was able to detect and log that.

D. *Performance of ModSecurity with CRS v.3.2 in detecting file inclusion attack*

For this test, we tried manipulating application-level code to insert random, local and remote data into parameter field, but an alert was logged in the firewall log file of possible file inclusion attack. This attack was detected at PL 1 (paranoia level 1).

E. *Performance of ModSecurity with CRS v.3.2 in detecting DoS attacks*

Slow HTTP DoS attacks occur under the HTTP protocol, enabling the server to completely acknowledge them. If the HTTP request is not complete or the transmission rate is low, the server must wait for the rest of the data to keep its resources busy. If the system uses too many resources, this could lead to a denial of service. We used *SlowHTTPTest* tool to send partial HTTP requests to get a denial of service from target HTTP server with ModSecurity sitting in-between to prevent our attack.

TABLE IV. TEST PARAMETERS FOR DOS ATTACKS

Test Types	SLOWHEADERS, SLOWBODY, RANGE ATTACKS
Number of connections	1000
Verb	GET
Content-Length header value	4096
Data max length	52
Interval between follow up data	5 seconds
Connections per second	200
Timeout for probe connection	3 seconds
Target test duration	60 seconds
Using proxy	HTTP proxy at server.com:80

From Table IV, 1000 connections were used to test with *slowheaders* (slowloris attack), *slowbody* (r-u-dead-yet attack), and *range* attacks (Apache killer attack) respectively. The test ran with 5 seconds wait for data, 200 connections per seconds, using GET request method against the vulnerable web server address, maximum data length of 52 bytes, with a 3 second time out for a total of 60 seconds to conduct each attacks.

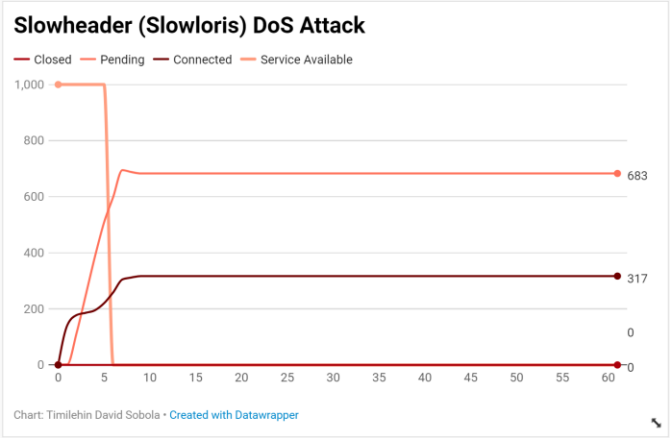

Fig. 2. Slowheader (slowloris) attack result chart.

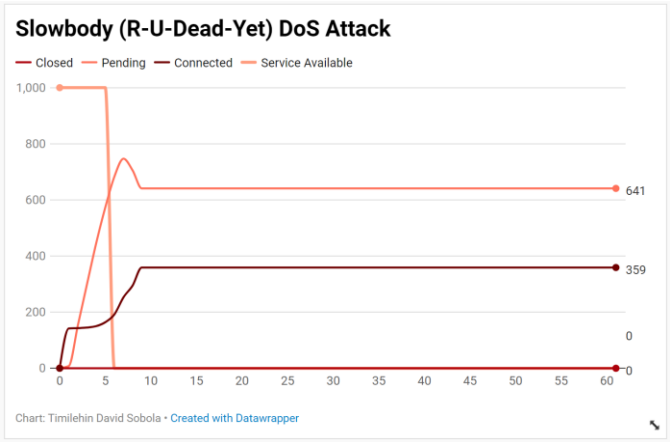

Fig. 3. Slowbody (R-U-Dead-Yet) attack result chart.

Figures 2 and 3 above, indicates that for the first 5 seconds, the server was still available to respond to requests and then became unavailable in the 6th second. After running for 60 seconds, the number of pending requests were 683 and 641 left to be handled by the server with 317 and 359 successful connections before the server went down for the *slowheader* and *slowbody* attacks respectively and no closed connections.

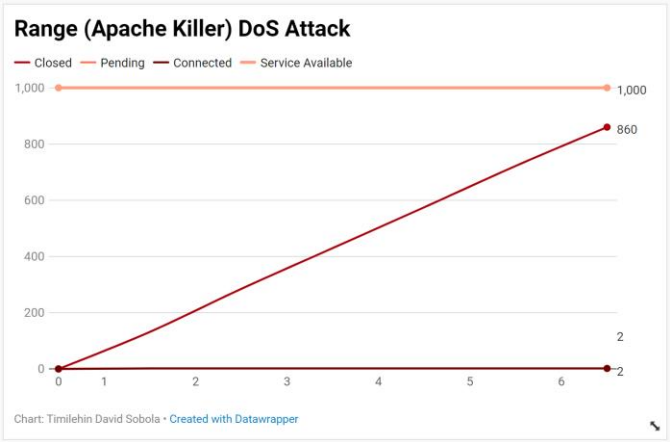

Fig. 4. Range (Apache killer) attack result chart.

Figure 4 above indicates that, within the 6 seconds, the server remained available and handled 860 successful connections, closed 2 connections and left 2 pending before the WAF shutdown the attack.

F. *Performance of ModSecurity with CRS v.3.2 against Nmap port scanner*

Nmap port scanner was used to seek response from IPs and open ports to search for vulnerabilities using a database of established resources. Expectation is for ModSecurity not to have any accessible ports outside the network. The result is outlined in Table IV.

TABLE V. PORT SCAN WITH NMAP

Ports Found	Ports Open
None	80 – standard traffic purpose

G. *Performance of ModSecurity with CRS v.3.2 based on Throughput, Transaction rate and Concurrency level*

There are constant trade-offs between performance and effectiveness, so it is right to judge a WAF's effectiveness within the context of its performance hence being able to avoid a lag in performance due to added features. To have an overview of the WAF performance, we ran *siege* which is a great benchmarking tool that helped simulate concurrent users requesting resources for a given period while increasing the number of concurrent users (***C***) and then comparing the result.

Table VI below, notice how *concurrency* level increases as the firewall's performance decreases but service remained available even with 100 concurrent users simulated. The *throughput* identifies the average amount of bytes transmitted every second to all imitated users. The amount decreases as the number of simulated users increases. Also, the *transaction* rate identifies the number of requests the firewall could handle per second. The transaction rate also diminishes with number of concurrent users.

TABLE VI. PERFORMANCE TEST

	C 5	C 10	C 20	C 30	C 100
Transactions	44221	40390	30010	23792	22051
Availability	100%	100%	100%	100%	100%
Elapsed time (s)	59.40	59.93	59.34	59.59	59.75
Date transferred (mb)	137.57	125.65	93.36	74.01	68.59
Response time (s)	0.01	0.01	0.04	0.07	0.27
Transaction rate (trans/sec)	744.46	673.95	505.73	399.26	369.05
Throughput (mb/sec)	2.32	2.10	1.57	1.24	1.15
Concurrency	4.90	9.89	19.87	29.89	99.57
Successful transactions	44221	40390	30010	23792	22051
Failed	0	0	0	0	0
Longest	0.10	0.39	0.49	0.64	1.03
Shortest	0.00	0.00	0.00	0.00	0.01

Where,

C = concurrent users

VI. Limitations and Recommendations

Although ModSecurity with CRS v.3.2 at default install was able to detect some of the attack vectors, some were not detected, and some were detected but not logged. A security risk could arise with insertion of attack payloads into file uploads which there is no policy in place for.

A recommendation could be to inspect all contents of HTTP request headers and files but could impede on the firewall performance. This might be far fetched as ModSecurity performance is not as encouraging compared to other competitors although it boasts of flexibility over others. A use case as mentioned by Trustwave Holdings Inc. [6], is the introduction of strict profile checks using positive or negative security model. The positive security model allows access through specific rules where each rule added allows greater access while having no rules in place will block everything by default. This can severely limit the attack methods attackers can use to exploit a vulnerability. The downside to the positive model is the intense care needed in its implementation so as not to block out legitimate users. Negative security model on the other hand, works on the premise that attack methods used by hackers are known so exploits are blocked based on this knowledge and creating patches or updates for new vulnerabilities that occur. Very little work is needed in the negative model. This model has no way to prevent against zero-day attack since it relies on maintaining the WAF to stay up to date on exploits. A recommendation will be to bridge the gap between positive and negative security models while maintaining good performance measures.

VII. Conclusion

According to a post by the OWASP CRS team [8], a Denial of Service vulnerability was identified on ModSecurity 3.0.x releases caused by malformed cookie header which at that time had not come to our attention at the time of conducting the experiments in this paper. The experiments and findings indicated that ModSecurity with CRS v.3.2 still has loopholes and can be bypassed by putting time and efforts. This paper examined specifically, the top ten web security risks, the techniques, and new ways these attack vectors can be carried out to successful penetrate a firewall. The paper's main contribution is towards the understanding of the effectiveness of ModSecurity with CRS v.3.2 in terms of detection capabilities of web attacks and performance when subject to heavy traffic (DoS). It also contributes to the confidence areas (strong and weak) for further improvements on the Core Rule Sets. As discussed under limitations and recommendations, future research would involve an approach in balancing the positive and negative security models to help improve effectiveness and thereby maintaining good management overhead i.e. ensure proper enforcement, refinement and verification of policies. Also, an evaluation on the stability and reliability of ModSecurity with CRS in blocking during an extended attack, attempting to pass legitimate traffic under extended attack, and port detection using protocol fuzzing and mutations would be a good focus area.

References

[1] Walter Hop, OWASP ModSecurity Core Ruke Set Version 3.2.0," 2019. [Online]. Available: https://coreruleset.org/20190924/owasp-modsecurity-core-rule-set-version-3-2-0/.

[2] "Web Application Security Overview," 2010. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648636(v=pandp.10)?redirectedfrom=MSDN.

[3] Common Weakness Enumeration, "2019 CWE Top 25 Most Dangerous Software Errors," [Online]. Available:http://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html.

[4] T. SpiderLabs, "ModSecurity Rules," Trustwave SpiderLabs, [Online]. Available: https://modsecurity.org/rules.html.

[5] Web Application Security Consortium, "Web Application Firewall Evaluation," [Online]. Available: http://projects.webappsec.org/f/wasc-wafec-v1.0.pdf.

[6] SpiderLabs, Trustwave Holdings Inc., "ModSecurity Reference Manual", [Online]. Available: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x)#Attack_Prevention_and_Virtual_Patching

[7] OWASP ModSecurity Core Rule Set, [Online]. Available: https://coreruleset.org/

[8] Christian Folini, "CVE-2019-19886–HIGH–DoS against libModSecurity 3", January 18, 2020. [Online]. Available: https://coreruleset.org/20200118/cve-2019-19886-high-dos-against-libmodsecurity-3/

[9] Akamai, "State of the Internet / Security Volume 6, Issue 1", [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-financial-services-hostile-takeover-attempts-report-2020.pdf

[10] I. M. KIM, "Using Web Application Firewalls to Detect and Block Common Web Application Attacks", 2011. [Online] Available: https://www.sans.org/reading-room/whitepapers/webservers/web-application-firewall-detect-block-common-web-application-attacks-33831

[11] Jatesh Singh, "Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall", 2016, IEEE IDS 2018, Texas