Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall

Jatesh Jagraj Singh
jlnu1@student.concordia.ab.ca

Hamman Samuel
hamman.samuel@concordia.ab.ca

Pavol Zavarsky
pavol.zavarsky@concordia.ab.ca

Department of Information Systems Security and Assurance Management
Concordia University of Edmonton, Canada

***Abstract*—Organizations use various approaches to safeguard their web applications. One such approach is to deploy a web application firewall. These firewalls work when configured with appropriate rules. Optimal selection of rules ensure that the firewall will properly identify attacks and hence block them or take appropriate actions. Our study analyzes various bypass attack vectors against the popular ModSecurity web application firewall with the open source Core Rule Set (CRS) version 3. The attack vectors focus on the OWASP Top 10 risks, and are tested against different settings of the paranoia level in ModSecurity. Our aim is to assist in better transparency about the default configuration of the CRS with ModSecurity and hence help administrators in taking informed decisions about web applications security for different paranoia levels configurations.**

I. Introduction

The security of web applications is a serious concern for most of the organizations as web applications are used for conducting online businesses. According to Acunetix , nearly 55% of web applications contain high risk vulnerabilities for Cross-Site Scripting (XSS) and SQL injections. Moroever, 84% of web applications were susceptible to medium severity vulnerabilities such as Cross-Site Request Forgery (CRSF) [1]. To address the problem of vulnerable web applications, companies have explored dynamic approaches to secure their web applications using multi-layer defense network firewalls, Intrusion Detection and Prevention Systems (IDPS), Multi-Factor Authentication (MFA), security policy enforcement, deploying Web Application Firewalls (WAF), among others.

The study intends to evaluate the effectiveness of the popular ModSecurity WAF [2]. Specifically, we explore how the effectiveness of ModSecurity can be optimized via the Paranoia Level configuration. ModSecurity's Paranoia Mode allows additional strict checks to be performed while checking web traffic for potential attacks. Paranoia Mode was introduced in the Core Rule Set (CRS) version 3 in order to reduce false positive alerts, and paranoia levels allow system administrators to configure the WAF by balancing false and true positives [3]. Setting the Paranoia Mode to Level 1 ensures that false positives rarely occur. However, this could also lead to security breaches using more sophisticated attack vectors.

While setting a higher paranoia level will ensure a stricter security policy, it may also create unwanted false positives.

The main objective of our study is to conduct an experimental evaluation on how to balance paranoia levels with security needs. The results from the experimental analysis could potentially provide new insights for professionals on the detection capabilities of ModSecurity with CRS 3 for different paranoia level configurations.

II. Methodology

We experimented on ModSecurity with paranoia levels ranging from 1 to 4. At each level, we attempted to bypass the WAF security via various attack vectors from the OWASP Top 10 risks [4]. For our experimental setup, we installed three virtual machines named *attacker*, *firewall* and *server*. An overview of the setup is shown in Figure 1.

On the server, we set up a Deliberately Vulnerable Web Application (DVWA) acting as our test application and having vulnerable code that can be exploited using several types of attacks. Our DVWA was Mutillidae II [5].

On the firewall, we installed ModSecurity with CRS 3 running in reverse proxy mode, whereby the attacker would not have direct access to the server, and all incoming traffic would be redirected through the the firewall first. Essentially, the reverse proxy mode allows a firewall to be set up independent of the server.

On attacker's machine, we used OWASP ZAP [6] and Burp Suite [7] to preform attacks. In addition, some of the attacks were curated and performed manually via web browser [8]. The attack vectors are summarized in Table I.

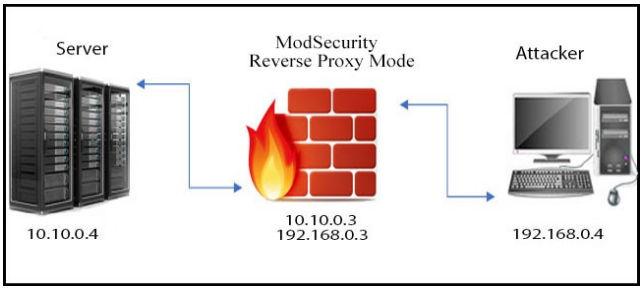

Figure 1. Overview of Experimental Setup

978-1-5386-5762-1/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDIS.2018.00030

TABLE I. Attack Vectors

Vector ID	Description	Source	Implementation & Notes
1	SQL injection	OWASP T10-A1	0923ac83-8b50-4eda-ad81-f1aac6168c5c%' – 8ce743eee973471b9511d8fedc927335%" --
2	PHP code injection	OWASP T10-A1	http://<url>/?message=phpinfo()
3	XML injection	OWASP T10-A1	<script>+-+-1-+-+alert(1)</script>
4	SSI injection	OWASP T10-A1	<! --#exec cmd="pwd"- ->
5	XPath injection	OWASP T10-A1	' or '1'='1 <script>alert('XSS')</script>
6	Insecure login forms	OWASP T10-A2	<p><label for="login">Login:</label>tonystark <p><label for="password">Password:</label>I am Iron Man
7	Logout management	OWASP T10-A2	Even after successful logout, attacker can log back in by pressing back button
8	Weak passwords	OWASP T10-A2	Brute force and dictionary attacks
9	Administrative portal	OWASP T10-A2	http://<admin_url>/?admin=1
10	Session ID in url	OWASP T10-A2	http://<url>/?PHPSESSID=obalo252hrim97c29ds59n2hkb
11	Cookie attributes	OWASP T10-A2	Set-Cookie: PHPSESSID=umdlcd8i7188upcm3i661q5ern; path=/
12	Password transmitted over HTTP	OWASP T10-A2	Form item: "login" = "tonystark" Form item: "password" = "I am Iron Man"
13	Reflected XSS	OWASP T10-A3	<script>alert(1);</script> ¼script¾alert(¢xss¢)¼/script¾
14	Stored XSS	OWASP T10-A3	'"()&%1<ScRiPt >prompt(976827)</ScRiPt> ¼script¾alert(¢xss¢)¼/script¾
15	DOM-based XSS	OWASP T10-A3	<IFRAME SRC="javascript : alert('XXS');"></IFRAME>
16	JavaScript Injection	OWASP T10-A3	<ahref="j[785bytesof(
)]avascript:alert(1);">XSS
17	HTML injection (reflected) GET	OWASP T10-A3	<script>alert("1")</script>
18	HTML injection (reflected) POST	OWASP T10-A3	<body oninput=alert(1)><input autofocus>
19	HTML injection (reflected)	OWASP T10-A3	http://<url>/htmli_current_url.php
20	Directory transversal	OWASP T10-A4	c:/Windows/system.ini
21	Network misconfiguration	OWASP T10-A5	http://<url>/test.php
22	Sensitive information sent via unencrypted channel	OWASP T10-A6	Form item: "login" = "bee" Form item: "password" = "bug"
23	File include	OWASP T10-A7	<?php echo system_exec('ls -la'); ?> <?php echo system_exec($_GET['cmd']); ?>
24	Testing for CSRF	OWASP T10-A8	<form action="http://<url>/vulnerabilities/csrf/?" method="GET"> <input type="password" AUTOCOMPLETE="off" name="password_new" value="test"> <input type="password" AUTOCOMPLETE="off" name="password_conf" value="test"> <input type="submit" value="Click me!" name="Change"> </form>
25	Vulnerable applications present on web server	OWASP T10-A9	User credentials are sent in clear text, TRACE method is enabled, Apache server-status enabled
26	Client-side URL redirect	OWASP T10-A10	http://<url>/?url=http://www.google.com
27	Triple encoding and above	OWASP T10-A10	<b/%25%32%35%25%33%36%25%36%36%25%32%35%25%33%36%25%36%35mouseover=alert(1)>

III. RESULTS

Our findings are grouped by paranoia level. For our results tables, the *Success* column represents whether there was a successful breach of the firewall. The *Tools Used* column represents which software programs (or manual inputs) were used to perform attacks on the firewall.

TABLE II. Results for Paranoia Level 1

Vector ID	Paranoia Level	Success	Tools Used
1	1	YES	ZAP/ Burp Suite
2	1	NO	ZAP/ Burp Suite
3	1	NO	ZAP/ Burp Suite
4	1	NO	ZAP/ Burp Suite
5	1	YES	ZAP/ Burp Suite
6	1	NO	ZAP/ Burp Suite
7	1	YES	Manual Input
8	1	NO	Manual Input
9	1	NO	ZAP/ Burp Suite
10	1	NO	ZAP/ Burp Suite
11	1	NO	ZAP/ Burp Suite
12	1	NO	ZAP/ Burp Suite
13	1	YES	ZAP/ Burp Suite
14	1	NO	ZAP/ Burp Suite
15	1	NO	ZAP/ Burp Suite
16	1	YES	ZAP/ Burp Suite
17	1	NO	ZAP/ Burp Suite
18	1	NO	ZAP/ Burp Suite
19	1	NO	ZAP/ Burp Suite
20	1	YES	ZAP/ Burp Suite
21	1	NO	ZAP/ Burp Suite
22	1	NO	ZAP/ Burp Suite
23	1	YES	ZAP/ Burp Suite
24	1	NO	ZAP/ Burp Suite
25	1	NO	ZAP/ Burp Suite
26	1	YES	ZAP/ Burp Suite
27	1	YES	Manual Input

TABLE III. Results for Paranoia Level 2

Vector ID	Paranoia Level	Success	Tools Used
1	2	YES	ZAP/ Burp Suite
7	2	YES	Manual Input
20	2	YES	ZAP/ Burp Suite
23	2	YES	ZAP/ Burp Suite
26	2	YES	ZAP/ Burp Suite
27	2	YES	Manual Input

TABLE IV. Results for Paranoia Level 3

Vector ID	Paranoia Level	Success	Tools Used
1	3	MAYBE	ZAP/ Burp Suite
7	3	YES	Manual Input
20	3	YES	ZAP/ Burp Suite
26	3	YES	ZAP/ Burp Suite
27	3	YES	Manual Input

TABLE V. Results for Paranoia Level 4

Vector ID	Paranoia Level	Success	Tools Used
1	4	MAYBE	ZAP/ Burp Suite
7	4	YES	Manual Input
26	4	YES	ZAP/ Burp Suite
27	4	YES	Manual Input

Firstly, Table II shows our results for paranoia level set at 1 (default). The results for all attack vectors attempted are shown. For Tables III, IV, and V, only those attack vectors that led to a successful breach are shown. Those not shown were unable to bypass the WAF.

At Paranoia Level 4, ModSecurity was able to handle nearly all the attacks with the exception of a few. At this level, the number of false positives generated were high. For example, the web application was unable to register a new user, despite this being a legitimate action, as ModSecurity was treating registration as an attack.

We also summarize the success and failure results obtained from our study, grouped by paranoia levels, in Figure 2 for a comparative analysis of how the Paranoia Mode correlates with the effectiveness of ModSecurity to block various types of attack vectors.

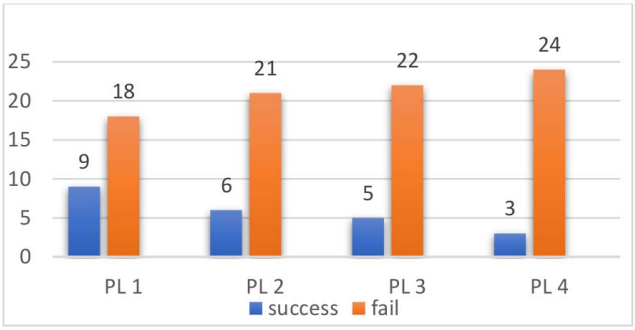

Figure 2. Results Summary

IV. DISCUSSION

From our results, we can observe that at the default paranoia level 1, the number of attacks that bypass the firewall are relatively high. The success rate drops as the paranoia levels are increased because more rules are activated. At the maximum paranoia level 4, most of the attacks are blocked by the firewall. At this level, all the rules present in CRS 3 are actived, including the experimental rules. This led to an observable increase in the number of false positive alerts being generated by ModSecurity. For instance, user registration was not possible, even though that was a legitimate function of the test web application. Another observation we made was that triple encoded attack vectors were generally not being detected by ModSecurity.

V. RELATED LITERATURE

Not many studies have empirically verified the usefulness of ModSecurity against enumerated attack vectors, and none have specifically looked into the CRS 3 and the Paranoia Mode. One study by [9] checked the effectiveness of ModSecurity in detecting XSS and SQLi attacks. Firstly, attacks were performed without configuring ModSecurity with CRS. Secondly, based on the of the initial results, appropriate rules within the CRS were activated.

Thirdly, the basic attacks were repeated with ModSecurity configured with selected rules. Some other unpublished works have also looked at specific XSS and SQLi attack vectors. Our study investigated a more comprehensive enumeration of attack vectors.

VI. Conclusion

In this paper, we tested the ModSecurity WAF enabled with CRS 3, and empirically analysed the effect of different paranoia levels on the detection abilities of ModSecurity, and the results can help professionals to understand the working of rules in different Paranoia Mode settings. Our observations show that increasing the paranoia level may have trade-offs with false positive alerts. In addition, we noted that triple encoded attack vectors were undetected even at the maximum paranoia level.

Further investigation will be very useful in the area of triple encoded attack vectors and whether they entail a zero-day vulnerability in ModSecurity or CRS 3. In addition, we intend to explore the relationship between paranoia levels and false positives. Other interesting topics to explore include evaluating the trade-offs involved with enabling all CRS rules at the maximum paranoia level against firewall performace impact.

References

[1] Acunetix Team, "Acunetix Web Application Vulnerability Report 2016", 2016. [Online] Available: https://www.acunetix.com/acunetix-web-application-vulnerability-report-2016 [Accessed: 1-Apr-2017].

[2] OWASP, "ModSecurity: Open Source Web Application Firewall", 2017. [Online] Available: https://www.modsecurity.org [Accessed: 19-Jan-2017].

[3] SpiderLabs, "OWASP ModSecurity Core Rule Set (CRS) Project", 2017. [Online] Available: https://www.github.com/SpiderLabs/owasp-modsecurity-crs [Accessed: 15-Mar-2017].

[4] OWASP, "OWASP Top Ten Cheat Sheet", 2017. [Online] Available: https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet [Accessed 12-Mar-2017].

[5] jdruin, "OWASP Mutillidae II", 2017. [Online] Available: https://www.sourceforge.net/projects/mutillidae [Accessed: 9-Feb-2017].

[6] OWASP, "OWASP Zed Attack Proxy (ZAP)", 2017. [Online] Available: https://www.github.com/zaproxy/zaproxy [Accessed 19-Jan-2017].

[7] PortSwigger Ltd., "Burp Suite", 2017. [Online] Available: https://www.portswigger.net/burp [Accessed 19-Jan-2017].

[8] OWASP, "XSS Filter Evasion Cheat Sheet", 2017. [Online] Available: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet [Accessed: 01-Oct-2017].

[9] 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 [Accessed: 29-Jan-2017].