Redundant Firewalls for Web Applications

Dauto Jeichande[1] and Hugo Miranda[2,1]

[1] Faculdade de Ciências da Universidade de Lisboa
fc47011@alunos.fc.ul.pt
[2] LaSIGE
hamiranda@ciencias.ulisboa.pt

Abstract. Web Application Firewall (WAF) is a security technology that filters communications using the HTTP protocol to ensure the respect of the logic expected by the application layer. WAFs acceptance is predicted to grow significantly, much thanks to its support by two of the major HTTP content distribution platforms (Nginx and Apache).
The paper studies the contribution of diversity and redundancy in WAFs architecture, orchestrated in order to provide greater resistance to attacks that exploit new vulnerabilities. The architecture is designed so that the operating system and the web server of each component share the fewest possible vulnerabilities. This study compares the impact on performance and resilience of our architecture with several other configurations, with and without WAFs.

Keywords: Web Application Security, Web Application Firewall, Reverse Proxy, Diversity, Load Balancing

1 Introduction

Web business applications are very easy to access through the ubiquitous web browser and a great competitive advantage in the business world. Unfortunately, web access is also available for adversaries, which use it as an attack surface. According to Symantec, from 2012 to 2013, the proportion of scanned websites with vulnerabilities increased from 53% to 77%.[3]

A Web Application Firewall (WAF) is an appliance, server plugin, or filter that imposes a set of rules to Hyper-Text Transfer Protocol (HTTP) conversations. The rules protect the web application from the most common attacks such as injection, broken authentication and session management, Cross-Site Scripting (XSS), insecure direct object references, Cross-Site Request Forgery (CSRF) and invalidated redirects and forwards. These characteristics make of WAF a good candidate to complement both conventional network firewalls and Intrusion Prevention Systems (IPS) with an additional layer that analyzes the logic

[3] Internet Security Threat Report. Retrieved November 17, 2015, from http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf

of the application and apply strict security checks on the decoded request content. Correctly configured WAFs can detect known attacks and even new types of attacks by analyzing the pattern of the traffic [3]. This is in contrast with both conventional network firewalls and IPS, which address the intrusion problem from a link, network and transport layer perspective. IPS has no knowledge of the web application layer constructs, the data structure, and encoding. For this reason, IPSs either fail to prevent many attacks or produce false positives, depending on the security policies.

The concept of security through diversity is a topic of interest these days in the design of intrusion tolerant architectures. This paper aims at contributing to understand the practical impact of redundancy and diversity of WAFs in a web application. Redundancy is experimented as a tool to provide protection with high availability for the web business applications. However, as it is often argued that redundancy without diversity is useless against organized and systematic attacks, the test-bed architecture was built to equally experiment diversity in the key components, namely, the Operating System, Web Server and WAF technology. With this solution, if the adversary succeeds to break one node, there is a low probabilityof compromising the remaining redundant nodes [1].

The test-bed used Redmine as the target web application and was configured to prevent the top 10 web attacks identified by OWASP.[4] Redmine is an open source project management web application, written using the Ruby on Rails framework. In addition, the paper reports on comparative performance tests between the two WAFs experimented, namely Naxsi and Modsecurity.

2 Related Work

A generic architecture where redundant proxies filter client requests to a redundant group of diversified application servers was proposed to improve performance [5]. The case study was based on a travel agency web infrastructure. The level of redundancy depends on the current alert level. When the attack density increases, the number of web servers that process each client request will also increase and consequently the performance will degrade.

A study of different intrusion-tolerant architectures for web servers based on intelligent adaptive reconfiguration was performed [4]. The objective is to help to build a more secure and resilient server system.

Similar to our work, both authors designed solutions to protect a web application. Both proposals use Snort[5] Intrusion Detection System (IDS). Snort is installed on the proxies, which have limitations in analyzing the application layer. In contrast, this project focuses on providing availability and better performance in the prevention of web attacks by implementing redundancy and diversity in WAF proxies.

[4] Top 10 2013. Retrieved June 15, 2016, from `https://www.owasp.org/index.php/Top_10_2013-Top_10`

[5] `https://www.snort.org`

OS Pairs	*Driver*	*Kernel*	*Sys.Soft.*	*Total*	OS Pairs	*Driver*	*Kernel*	*Sys.Soft.*	*Total*
Win2000-Win2003	0	42	43	85	Win2003-Win2008	0	18	22	40
OpenBSD-FreeBSD	1	14	18	33	NetBSD-FreeBSD	2	13	10	25
OpenBSD-NetBSD	1	8	8	17	Win2000-Win2008	0	8	8	16
Debian-Red Hat	0	5	8	13	NetBSD-Solaris	0	4	6	10
FreeBSD-Solaris	0	5	3	8	OpenSolaris-Solaris	0	3	3	6
OpenBSD-Solaris	0	5	1	6	Solaris-Red Hat	0	3	3	6
NetBSD-Red Hat	0	0	6	6	FreeBSD-Red Hat	0	1	4	5
OpenBSD-Red Hat	0	1	3	4	NetBSD-Debian	0	0	4	4
FreeBSD-Win2000	1	3	0	4	OpenBSD-Win2000	0	3	0	3
NetBSD-Win2000	1	2	0	3	Solaris-Win2000	0	3	0	3
OpenBSD-Win2003	0	2	0	2	FreeBSD-Win2003	0	2	0	2
Solaris-Debian	0	1	1	2	Debian-Ubuntu	0	0	2	2
NetBSD-Win2003	0	1	0	1	NetBSD-Win2008	0	1	0	1
FreeBSD-Debian	0	0	1	1	FreeBSD-Win2008	0	1	0	1
Debian-Win2000	0	0	1	1	Ubuntu-Red Hat	0	0	1	1
Ubuntu-Win2000	0	0	1	1	Red Hat-Win2000	0	0	1	1
OpenBSD-Win2008	0	1	0	1	Solaris-Win2003	0	1	0	1
Debian-Win2003	0	0	1	1	Ubuntu-Win2003	0	0	1	1
Red Hat-Win2003	0	0	1	1					

Table 1. Common vulnerabilities on isolated thin servers

2.1 Implementing Diversity

The work described in this paper implements diversity at the operating system, web server and WAF technology levels. This section compares and discusses the existing alternatives at each of these levels.

Operating System Table 1 shows the vulnerabilities shared by each pair of operating systems between 1994 and 2011 as reported by the National Vulnerability Database [2]. The table shows that Ubuntu and Red Hat have only one common vulnerability in system software making them good candidates for comparison in a diversity experiment. Our work elected Ubuntu and the CentOS distribution as the operating systems. CentOS was selected considering that it is built from much of the Red Hat Enterprise Linux code base.

Web server The web servers addressed in our work are Apache and Nginx, widely considered the two most popular web servers. As of November 17th, 2015, Nginx 1.9.5 has no reported vulnerabilities, reducing the probability of it sharing the vulnerabilities reported for the Apache 2.4.7, the versions used in this study.[6]

Web Application Firewall This work compared two of the most popular WAF services, namely Modsecurity[7] and Naxsi[8] which integrate very well with,

[6] Current CVSS Score Distribution For All Vulnerabilities. Retrieved November 17, 2015, from `https://www.cvedetails.com`

[7] ModSecurity: Open Source Web Application Firewall. Retrieved October 28, 2015, from `https://www.modsecurity.org`

[8] Nbs-system/naxsi. Systems, network and cloud services. Retrieved June 06, 2016, from `https://github.com/nbs-system/naxsi`

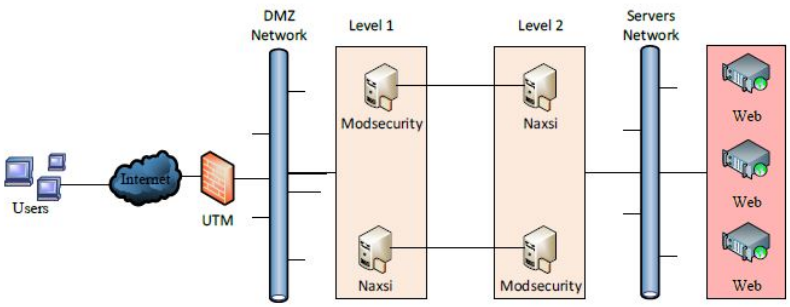

Fig. 1. Target architecture

respectively, Apache and Nginx. As of November 17th, 2015, neither Naxsi 0.54 or Modsecurity 2.7.7, the two versions used in this work, had any reported vulnerabilities.[9] It is expected that using other WAF solutions and combine them to have more diversified nodes will improve the web security. The number of diversified nodes will depend on the security requirements.

3 Test-bed

Figure 1 depicts the target architecture used. All the components are deployed between a Unified Threat Management (UTM) and the web servers. The UTM includes network firewalling and anti-spam functions. All HTTP traffic is flowing through the WAFs. Incoming requests are addressed to the WAFs, who are responsible for forwarding them to the back-end web servers on behalf of the originating client. The UTM equally plays the role of load-balancer, distributing the load by the two "Level 1" WAFs. Each of the "Level 1" WAFs is connected to a single "Level 2" WAF of the opposing model, thus enforcing all the requests to be evaluated by the two WAF models. The "Level 2" WAFs send each of the web requests to be processed by the web application. It should be noted that WAFs use the reverse proxy mode, what permits to terminate connections if an attack is detected. As will be discussed in the tests and performance evaluation section, this option has the drawback of notably increasing latency.

3.1 Configuration

Each WAF is run on a virtual server emulating an Intel Xeon 2.4GHz with 2Gb of RAM and a 30Gb hard drive. VirtualBox[10] was used as the hyper-visor. Servers

[9] Current CVSS Score Distribution For All Vulnerabilities. Retrieved November 17, 2015, from `https://www.cvedetails.com`

[10] Welcome to VirtualBox.org! Retrieved June 15, 2016, from `https://www.virtualbox.org/`.

were arranged in two combinations, to guarantee that each HTTP request will necessarily traverse two distinct Operating Systems, Web Servers and WAFs. The **Naxsi** servers used CentOS v. 7.1.1503 as the operating system, on top of which Nginx 1.9.5 with the Naxsi 0.54 served HTTP requests. **Modsecurity** servers putted together an Ubuntu 14.04.3 LTS operating system, Apache 2.4.7 and Modsecurity 2.7.7. All the software versions were selected for being the most recent at the 30th of November of 2015, date of the beginning of the tests.

Nginx – Naxsi

Naxsi was configured with the core rules suggested at the framework web site[11] and complemented with the doxi rules.[12] Naxsi default rules aim at preventing SQL injections, remote file inclusion, directory traversal, cross-site scripting, evasion and file uploads. Doxi rules, which are available as an independent Github repository aim to provide additional protection against this kind of attacks.

Naxsi was initially configured in "LearningMode". With this setup, the HTTP requests that raise suspicious are recorded in the error log file but are not blocked. "LearningMode" facilitates the creation of a whitelist given that log records can be applied to the Naxsi rule-sets using the `nxapi` tool.

Whitelists play a fundamental role in the reduction of false positives. To create the whitelist, users are required to use the web application, traversing the Naxsi WAF exclusively with legitimate traffic. This was accomplished by using the firewall to restrict access to the application from the IP address producing the legitimate learning traffic.

Apache – Modsecurity

Modsecurity using the OWASP Modsecurity Core Rule Sets (CRS)[13] was enabled as an Apache2 module. According to the OWASP web site, Modsecurity CRS provides protections for the following attack/threat categories:

HTTP protection detecting violations of the HTTP protocol and a locally defined usage policy;
Real-time blacklist look-ups which utilizes 3rd party IP reputation;
HTTP denial of service protections defense against HTTP flooding and slow HTTP DoS attacks;
Common web attacks protection detecting common web application security attacks such as SQL injections, XSS and CSRF;
Automation detection detecting bots, crawlers, scanners and other surface malicious activity;

[11] `https://github.com/nbs-system/naxsi`
[12] Doxi-rules. Retrieved June 06, 2016, from `https://bitbucket.org/lazy_dogtown/doxi-rules`
[13] Category:OWASP ModSecurity Core Rule Set Project. Retrieved June 15, 2016, from `https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project`

Integration with anti-virus scanning for file uploads detects malicious files uploaded through the web application;
Tracking sensitive data tracks credit card usage and blocks leakages
Trojan protection detecting access to Trojans horses;
Identification of application defects alerts on application, misconfigurations;
Error detection and hiding disguising error messages sent by the server;

Modsecurity was configured in "self-contained mode", where the triggering of any rule will result in the execution of any disruptive/logging actions specified on the current rule. It should be noted that "self-contained mode" is in contrast with Naxsi mode as the latter follows a scheme where each matched rule adds to the request score and requests are dropped if their score is above a predefined threshold. In addition, Naxsi redirects dropped requests to "/RequestDenied" page while Modsecurity returns a page with "Forbidden" message (code 403).

Load balancer

The load balancer service was configured with a HTTP monitor, which constantly sent fake requests to the "Level 1" WAFs to check their availability.

To ensure service availability in case of "Level 2" WAF failures, "Level 1" servers voluntarily terminate their HTTP server if they find their counterpart to be unavailable. Monitoring is performed by sending HTTP requests every 30s using the monit tool.[14]

4 Tests and Performance Evaluation

The WAFs were evaluated in separate and combined. Naxsi, Modsecurity and combinations of both were evaluated according to their security effectiveness, throughput, stability, reliability and usability. In this paper, security effectiveness measures their capability to detect and repel specific attacks. To evaluate the overhead introduced by WAFs, response times were compared to a setting without WAFs. The stability and reliability measure the WAFs capability to distinguish legitimate from malicious traffic.

4.1 Security Effectiveness

The effectiveness of the security provided by the redundant WAF architecture was evaluated using the OWASP Zed Attack Proxy (ZAP),[15] the Open Vulnerability Assessment System (OpenVAS)[16] and the burp[17] suites. In addition, we

[14] Easy, proactive monitoring of Unix from `https://mmonit.com`
[15] OWASP Zed Attack Proxy Project. Retrieved October 28, 2015, from `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`
[16] Open Vulnerability Assessment System. Retrieved June 06, 2016, from `http://www.openvas.org`
[17] Burp Suite. Retrieved June 15, 2016, from `https://portswigger.net/burp`

also evaluated security by running customized security tests taking in consideration the vulnerabilities reported up to the 12th of March of 2016 in Redmine Security Advisories.[18]

Automatic Tests In the tests without WAF protection, the OWASP Zed Attack Proxy exposed an XSS vulnerability, considered to be of high-risk level. In addition, several low risk level vulnerabilities (incomplete or no cache-control pragma HTTP header set, absence of anti-CSRF Tokens and password autocomplete in the browser) were found.

When the Naxsi WAF was used, the test identified two low-level risks vulnerabilities: password auto-completes in the browser and the absence of anti-CSRF tokens. A third low-level risk was reported by the scanner: the incomplete or no cache-control and pragma HTTP header set. However, the vulnerability to this risk is incorrectly diagnosed by ZAP, which checks if cache-control HTTP header is set with no-cache, no-store, must-revalidate, private and that the pragma HTTP header is set with no-cache. If any of them is missing, it is considered incomplete and thus triggers the alert. The redmine application has a customized page (404.html) which is returned in case of error code 404 and it is not delivered with the recommended settings mentioned. It was also verified that the server in some cases may return a response with two cache-control headers. This problem was reported to OWASP ZAP support team and raised an issue on Github,[19] in order to fix the cache control scanner so that it can accommodate multiple headers.

The scan results using the Modsecurity WAF highlighted two low risks level vulnerabilities, namely: password auto-completes in the browser and the absence of anti-CSRF tokens. Contrary to Naxsi, With Modsecurity the incomplete or no cache-control and pragma HTTP header set vulnerability is not detected. The error page sent by the server with incorrect parameters is disguised. It should be noted that, OWASP ZAP requires disabling the rules with id 960015 and 960021 to be able to scan the application. These rules respectively check if an accepted header is present and if an accepted header exists, but is empty. The problem emerges from Modsecurity generating the warning about the detection of CSRF while OWASP ZAP expects it to be created by the application. This is an interesting proof of concept as Modsecurity uses content injection to inject a bit of JavaScript to set the CSRF token on any URLs or forms for any HTML pages requested. When the subsequent request comes in after rule 981144, Modsecurity compares the CSRF and rejects any requests without a valid token.

Not surprisingly, the tests sequentially composing Naxsi and Modsecurity exposed the two low-level risk vulnerabilities previously observed, namely: password auto-complete in the browser and the absence of anti-CSRF tokens. Al-

[18] Redmine Security Advisories. Retrieved June 06, 2016, from `https://www.redmine.org/projects/redmine/wiki/Security_Advisories`

[19] Incomplete or no cache-control and pragma HTTP header (False positive?) Issue #2405 zaproxy/zaproxy. Retrieved June 06, 2016, from `https://github.com/zaproxy/zaproxy/issues/2405`

Table 2. Customized tests results

Severity	**Description**	**No WAF**	**Naxsi**	**Modsecurity**	**Naxsi+ Modsecurity**
Moderate	Open Redirect	User redirection possible	Blocked	Not blocked	Blocked
High	Persistent XSS	XSS possible	Blocked	Blocked	Blocked
High	Bypass CSRF	Possible to create users with admin privileges	Not blocked	Not blocked	Not blocked

Table 3. Vulnerabilities identified in automatic and customized Tests

Test	**No WAF**	**Naxsi**	**Modsecurity**	**Naxsi+Modsecurity**
Automatic	4	3	2	2
Customized	3	1	2	1
Distinct	5	3	2	2

though these results suggest that there is no considerable security improvement when WAFs are combined, it should be noted that the Level 1 WAF can hide the presence of Level 2 WAF. This creates additional challenges to the adversary, that may show beneficial in customized attacks scenarios. The OpenVAS scan did not report any web vulnerability in all of the test cases.

Customized Tests Table 2 shows that using both WAFs secures Redmine application against attacks defended by any of the WAFs. The customized tests were designed taking into consideration the list of security vulnerabilities[20] that were fixed in Redmine releases prior to its version 1.3. An example of the customized "Bypass CSRF" attack which was not avoided by any of the WAFs can be found in Appendix A. This attack aims to create a user with admin privileges. The session must be from a user with permissions to create users.

Table 3 depicts a comparison between the number of vulnerabilities when not using a WAF protection, using naxsi or modsecurity separately and combining both WAFs. Results show that combination of both WAFs improves the protection of redmine application over the possible attacks. The last row counts the number of distinct vulnerabilities after remaining those that were found in both experiments.

[20] Redmine Security Advisories. Retrieved June 06, 2016, from `https://www.redmine.org/projects/redmine/wiki/Security_Advisories`

4.2 Performance

The performance evaluation aimed to determine the latency penalty imposed by the use of the WAFs. Tests consisted in simulating 25 concurrent users for a 600s period using the visual studio web performance and load tests tool.[21] The tests considered distinct activities, including documents download and upload, retrieval of web pages and database intensive operations, for example, web site content search. To force the requests to be delivered to servers, all content caching was disabled.

The average response time confirms the expectations that with some exceptions, the WAF adds some delay in the access to Redmine pages. However, it should be noted that most of the average response times lie within the $1.0s$ limit that is generally considered acceptable for web site navigation.[22] The results evidence two anomalies with the average response time of a login page (/login POST) and a webpage running a database query (/my/page) being consistently lower when naxsi and modsecurity are used. The results were observed repeatedly in the multiple experiments performed to identify the reason, although the root cause was not found.

Table 4 depicts the response times of the distinct operations in scenarios that make use of individual WAFs, combine them sequentially *Naxsi + Modsecurity* and put together the architecture presented in Fig. 1 *concurrent 2x Naxsi + Modsecurity*. The *Under Attack* column extends the later with a concurrent application of the OWASP ZAP tool.

A side-by-side comparison of Naxsi and Modsecurity shows no clear winner, with each of the WAFs presenting an improved performance on some of the web pages tested. Relevant for this study is the comparison of the composition of both WAFs with and without a load balancer, which evidence that the introduction of the later can absorb the negative performance penalty of the former by equally distributing the load by the two WAF circuits defined.

The standard deviation of the response time metric, depicted in Fig. 2, clearly shows that the request type is the most influencing factor in the predictability of the response time. The exception is the Naxsi+Modsecurity composition when the load balancer is not used which consistently presents an above average standard deviation.

4.3 Reliability and Usability

The experience in implementing both WAFs clearly revealed the relevance of whitelists in the usability of the protected web sites as a number of false positives could easily be found. In this regard, the "LearningMode" of Naxsi and its nxtool are a competitive advantage over Modsecurity. The "LearningMode" permits to

[21] Performance Testing Guidance for Web Applications. Retrieved June 06, 2016, from `https://msdn.microsoft.com/en-us/library/bb924375.aspx`

[22] Response Time. Retrieved June 06, 2016, from `http://www.loadtestingtool.com/help/response-time.shtm`

Table 4. Average Response Time

	Average page response time (in seconds)					
Page	No WAF	Naxsi	Modsec	Naxsi+ Modsec	Concurrent 2x Naxsi + Modsec	Under Attack
https://moldtrack.ai-emea.com	1.93	2.47	2.23	3.30	1.54	2.01
/attachments/432/Report_August.pdf	0.41	0.42	0.59	1.00	0.68	1.17
/issues/show/1219	3.97	4.39	4.88	5.25	4.42	6.60
/login [POST]	1.26	1.18	1.78	2.21	1.35	2.39
/my/page	0.72	0.44	0.47	0.50	0.67	1.07
/projects	0.26	0.41	0.28	0.51	0.55	0.95
/projects/activity/antex-infra	0.56	1.88	3.20	3.26	0.47	1.44
/projects/antex-infra/documents	0.12	0.24	0.36	0.61	0.27	0.60
/projects/antex-infra/issues	0.76	1.01	1.07	1.52	1.03	1.74
/projects/antex-infra/issues/new GET	0.84	0.98	1.00	1.35	0.92	1.80
/projects/antex-infra/issues/new POST	0.94	1.05	1.23	1.44	1.60	2.86
/projects/settings/antex-infra	0.21	0.49	0.57	0.95	0.41	0.87
/projects/show/antex-infra	0.25	0.45	0.48	0.60	0.39	0.78
/search/index/antex-infra	0.39	0.57	0.42	0.72	0.63	1.12

define custom made rules in production sites, avoiding the risk of locking out legitimate users from the application.

Naxsi does not rely upon predefined signatures, so it should be capable of defeating complex, unknown, obfuscated attack patterns. Naxsi by default reads a small subset of simple (and readable) rules containing of known patterns involved in website vulnerabilities. For example, $<, |$ or drop expression are not supposed to be part of a URI. In terms of usability, the rules are very simple and easy to understand. In contrast, Modsecurity uses the Perl Compatible Regular Expressions (PCRE) library for pattern matching of the rules against requests. PCRE is a popular library, available for many operating systems and therefore, highly portable. Using regular expressions make rule creation more flexible, although more complex.

5 Conclusion and Future Work

The traditional firewall architectures are in most cases designed for availability and simplify administration tasks by implementing redundant nodes with the exact same characteristics and configuration. From a security point of view, diversification enables the servers to be more resilient to attacks because the probability of using the same exploit to compromise diverse systems is lower. This paper investigated a combination of Naxsi and Modsecurity, two popular open source WAFs, to protect a web infrastructure.

The security test results show that the combination of Naxsi and Modsecurity offer a better protection. This is a great advantage against high skilled adver-

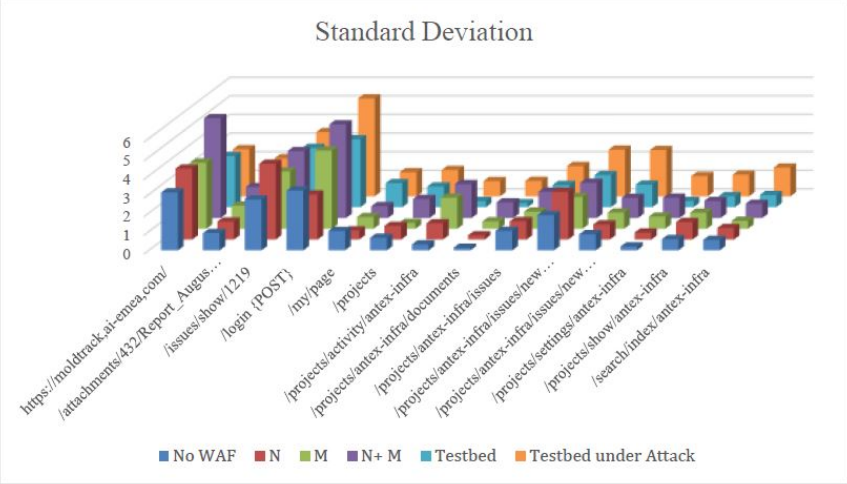

Fig. 2. Response Time Standard Deviation

saries, which combine multiple attack methodologies and tools in order to reach and compromise their target. Our solution trades off security by performance. As expected, our proposed architecture adds overhead to the response time of the requests due to the additional processing involved. However, the results showed that the user experience can still be considered acceptable, specially when load is balanced by two alternative threads.

As future work, authors plan to focus in improving the security of the proposed solution by implementing the rejuvenation principle and a combination of dynamic application security testing (DAST) [6]. The rejuvenation implies that the nodes are periodically restored to the last correct known image. The architecture and technology used in this study facilitates the implementation of this concept. A virtual machine can be easily restored from a snapshot and the redundant node ensure the service continuity while the other node is rejuvenated. The DAST is a process of security testing an application or software product in a running state. A DAST scanner (Burp, OWASP Zed Attack Proxy) generates a report that serves as an input for WAF signatures.

References

1. A. Bessani, A. Daidone, I. Gashi, R. Obelheiro, P. Sousa, and V. Stankovic. Enhancing fault/intrusion tolerance through design and configuration diversity. In *Proc. of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS)*, June 2009.

2. M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Software: Practice and Experience*, 44(6):735–770, 2014.
3. E. Kazanavicius, V. Kazanavicius, A. Venckauskas, and R. Paskevicius. Securing web application by embedded firewall. *Elektronika ir Elektrotechnika*, 119(3):65–68, 2012.
4. S. Raj and G. Varghese. Analysis of intrusion-tolerant architectures for web servers. In *Procs. of the 2011 International Conference In Emerging Trends in Electrical and Computer Technology (ICETECT)*, pages 998–1003. IEEE, March 2011.
5. A. Saidane, V. Nicomette, and Y. Deswarte. The design of a generic intrusion-tolerant architecture for web servers. *IEEE Transactions on Dependable and Secure Computing*, 6(1):45–58, 2009.
6. N. Tymoshyk and S. Breslavskyi. Web application firewalls: Next big thing in security. http://www.esecurityplanet.com/network-security/web-application-firewalls-next-big-thing-in-security.html, March 2015. Retrieved June 06, 2016.

A Script of CSRF attack

```
<html>
<body>
    <form method=POST action="https://moldtrack.ai-emea.com/users/add"\
       <input style="display: none" type="text" value="adversary"
         size="25" name="user[login]" id="user\_login"/>
       <input style="display: none" type="text" value="adversary"
         size="30" name="user[firstname]" id="user\_firstname"/>
       <input style="display: none" type="text" value="adversary"
         size="30" name="user[lastname]" id="user\_lastname"/>
       <input style="display: none" type="text" value="adversary@ai-emea.com"
         size="30" name="user[mail]" id="user\_mail"/>
       <input style="display: none" type="password"
         size="25" name="password" id="password" value="adversary"/>
       <input style="display: none" type="password"
         size="25" name="password\_confirmation" id="password\_confirmation"
         value="adversary"/>
       <input style="display: none" type="checkbox"
         value="1" name="user[admin]" id="user\_admin"/>
       <input style="display: none" type="hidden"
         value="1" name="user[admin]"/>
       <input style="display: none" type="submit"
         value="Create" id="commit" name="commit"/>
  </form>
  <script>document.getElementById("commit").click();</script>
</body>
</html>
```