Web Application Firewalls (WAFs): o impacto do número de regras na latência das requisições Web (versão estendida)*

Felipe Melchior[1,3], **Diego Kreutz**[1,2,3], **Mauricio Fiorenza**[2,3]

[1]Laboratório de Estudos Avançados (LEA)
[2] Mestrado Profissional em Engenharia de Software (MPES)
[3] Universidade Federal do Pampa (UNIPAMPA)
fehmel@gmail.com,{Nome.Sobrenome}@unipampa.edu.br

Abstract. *Specialized reports indicate that nearly 90% of applications available on the Internet are vulnerable, i.e., have some sort of security flaw. Software development frameworks and WAFs (Web Application Firewalls) are examples of tools that can be used to avoid the exploitation of many vulnerabilities of Web applications. Recent findings show that such tools can help to detect and prevent the exploitation of over 70% of the most common vulnerabilities found in Web applications. In this work, we provide an overview of existing research and development on WAFs and we empirically evaluate the impact of the number of active rules on the latency of Web requests. Our findings show that the number of rules of a WAF has a significant impact on the latency of web requests, increasing over 2000% for 100k rules.*

Resumo. *Relatórios especializados indicam que aproximadamente 90% das aplicações disponibilizadas na Internet são vulneráveis, isto é, apresentam alguma falha de segurança. A exploração de muitas das vulnerabilidades presentes nas aplicações Web pode ser evitada através de ferramentas como frameworks de desenvolvimento de software e WAFs (Web Application Firewalls). Estudos recentes demonstram que estas ferramentas podem ajudar a detectar e impedir a exploração de mais de 70% das vulnerabilidades mais recorrentes em aplicações Web. Este trabalho tem como objetivo realizar um levantamento do estado da arte de WAFs e uma avaliação empírica do impacto da quantidade de regras ativas na latência das requisições Web. Os resultados experimentais demonstram que o número de regras de um WAF tem um impacto significativo na latência das requisições Web, ultrapassando 2000% para 100 mil regras.*

1. Introdução

O número de ataques e incidentes de segurança não para de crescer. Grande parte dos ataques explora vulnerabilidades de sistemas utilizados por múltiplas instituições. Por exemplo, recentemente (em julho de 2019), mais de 60 universidades e colégios dos Estados Unidos da América (EUA) foram comprometidos devido a um conjunto de vulnerabilidades existentes num único sistema [Muncaster 2019, Machado et al. 2019]. Outro exemplo recente, em junho de 2019, foram descobertas, por técnicos de uma prefeitura

*Este artigo é uma versão estendida do paper [Melchior et al. 2019a], publicado no WRSeg 2019 (https://errc.sbc.org.br/2019/wrseg/) e selecionado entre os melhores trabalhos do evento para publicação em revista.

do RS, diferentes vulnerabilidades críticas em um sistema Web utilizados por várias prefeituras municipais do estado[1].

Um dos principais problemas que tem levado a essa onda crescente de incidentes de segurança é a falta de formação e conhecimento na área. Por exemplo, estudos recentes apontam que ferramentas de segurança, como os *Web Application Firewalls* (WAFs), podem contribuir significativamente na proteção de sistemas contra a exploração das vulnerabilidades mais recorrentes em aplicações Web. Um WAF é um serviço de segurança implementado entre o cliente (e.g., navegador/*browser*) e a aplicação (e.g., sistema PHP rodando num servidor Web Apache) [Melchior et al. 2019b]. A função do WAF é interceptar e processar as requisições entre o cliente e a aplicação. A partir de um conjunto de regras, o WAF classifica as requisições em maliciosas, que são geralmente bloqueadas, e não-maliciosas, isto é, que são encaminhadas até a aplicação. Um WAF como o ModSecurity, na configuração padrão, ou seja, sem nenhuma otimização, associado a um *framework* PHP (*Hypertext Preprocessor*) como o Laravel, é capaz de mitigar em 70% a exploração de vulnerabilidades recorrentes em sistemas Web [Ferrão 2018].

Há diferentes desafios e oportunidades de pesquisa no contexto de WAFs, como algoritmos para detectar ataques que objetivam explorar vulnerabilidades específicas (e.g., *cross-site request forgery* (CSRF) e *cross-site scripting* (XSS)) [Srokosz et al. 2018, Rao et al. 2016], mecanismos para aumentar o desempenho de processamento de requisições em cenários com grandes volumes de requisições (e.g., milhares de requisições por segundo) [Moosa and Alsaffar 2008] e análises empíricas de funcionalidades e WAFs disponíveis no mercado [Clincy and Shahriar 2018, Razzaq et al. 2013].

Os objetivos deste trabalho são revisar o estado da arte e avaliar empiricamente o impacto de WAFs na latência das requisições Web. Para atingir os objetivos, foram investigados trabalhos relacionados existentes na literatura e WAFs disponíveis gratuitamente no mercado. Resumidamente, as contribuições deste trabalho são: (a) uma síntese do estado da arte; e (b) uma avaliação do impacto da quantidade de regras dos WAFs ModSecurity[2], Naxsi[3], ShadowDaemon[4] e xWAF[5] na latência das requisições Web. Os resultados dos testes experimentais indicam que há um impacto significativo na latência das requisições Web, que aumenta de acordo com o número de regras do WAF, podendo ultrapassar 2000% para 100k regras.

O restante do trabalho está organizado como segue. A Seção 2 apresenta uma revisão do estado da arte. Nas Seções 3, 4 e 5 são apresentados o desenvolvimento do trabalho, os resultados e as considerações finais, respectivamente.

2. Trabalhos Relacionados

Existem essencialmente dois tipos de WAFs, os *standalone*, que são instalados entre o cliente e a aplicação, e os SaaS, que utilizam redirecionamento DNS para que o tráfego

[1]Como os defeitos dos sistemas ainda não foram corrigidos pela respetiva empresa, por questões de segurança, os detalhes (e.g., referência adicional, informações técnicas, nome da prefeitura, nome da empresa) não estão sendo aqui divulgados.

[2]`https://modsecurity.org`

[3]`https://github.com/nbs-system/naxsi`

[4]`https://shadowd.zecure.org`

[5]`https://github.com/Alemalakra/xWAF`

seja encaminhado e processado na infraestrutura da empresa proprietária do WAF antes de ser encaminhado ao servidor do sistema Web.

WAFs *standalone* apresentam diferentes desafios técnicos, que requerem recursos humanos minimamente qualificados, como: (i) instalação, configuração e manutenção contínua; (ii) criação e gerenciamento de regras; (iii) otimização de regras; e (iv) suporte a grandes volumes de tráfego. O foco das discussões e análises deste trabalho são os WAFs *standalone*.

A Tabela 1 resume as principais contribuições, oportunidades de pesquisa e evidências empíricas de trabalhos relacionados a WAFs. Com relação à **Principal Contribuição**, há diferentes propostas de novos algoritmos para de detecção de ataques a vulnerabilidades específicas (e.g., CSRF e XSS). Por exemplo, ataques que visam explorar vulnerabilidades XSS podem ser bloqueados através da análise e identificação de padrões (e.g., caracteres "<" e ":") utilizados em requisições maliciosas aos sistemas Web [Rao et al. 2016].

Tabela 1. Quadro Resumo dos Trabalhos Relacionados

	Principal Contribuição	**Oportunidades de Pesquisa**	**Evidências Empíricas**
[Funk et al. 2018]	Aumento na taxa de detecção de ataques	Comparar com outros WAFs	Taxa de detecção 60% maior que o ModSecurity
[Srokosz et al. 2018]	Detecção de ataques *CSRF*	Implementar e avaliar os algoritmos	
[Rao et al. 2016]	Detecção de ataques *XSS*	Implementar e avaliar os algoritmos	
[Moosa and Alsaffar 2008]	WAF híbrido, usando heurísticas	Evoluir o WAF com novas heurísticas	WAF suporta um grande volume de requisições
[Razzaq et al. 2013]	Comparação analítica de quinze WAFs tradicionais	Comparar empiricamente os WAFs	
[Clincy and Shahriar 2018]	Boas práticas para criação de regras	Avaliar configurações padrão de WAFs	
[Rietz et al. 2016]	Visão de WAF no estado atual da Internet	Avaliar WAFs em cenários controlados	
[Singh et al. 2018]	Análise dos níveis de Paranoia do ModSecurity	Avaliar diferentes configurações	Maior taxa de detecção e de falsos positivos
[Ferrão 2018]	Avaliação de três WAFs em cenários controlados	Analisar outros WAFs	ModSecurity mitiga até 70% das vulnerabilidades

Outro exemplo são as boas práticas na criação de novas regras para um WAF. O primeiro passo é entender o objetivo da regra que está sendo criada [Clincy and Shahriar 2018]. Isto é crucial para o correto funcionamento e bom desempenho de um WAF. Por exemplo, uma regra estática, que bloqueia ataques de *SQL Injection* que utilizam a expressão "' OR 1=1– #", não irá bloquear um ataque que utilize a expressão "' OR 2=2– #".

A maioria dos estudos indica de forma explícita ou implícita **Oportunidades de Pesquisa** como implementar novas heurísticas para melhorar a taxa de detecção de WAFs

e avaliar a eficácia e o desempenho dos WAFs através de estudos empíricos. Um dos principais objetivos deste trabalho é justamente avaliar empiricamente diferentes WAFs com relação ao impacto do número de regras na latência das requisições Web.

As **Evidências Empíricas** são dados concretos que permitem melhor avaliar e validar uma pesquisa [Explorable 2009]. Por exemplo, o ModSecurity, em combinação com *frameworks* de desenvolvimento, é capaz de mitigar até 70% dos ataques que exploram as vulnerabilidades recorrentes em aplicações Web [Ferrão 2018]. Indo um pouco além, resultados recentes de pesquisa indicam que implementações específicas de WAFs podem atingir uma eficácia de detecção até 60% maior que a do ModSecurity [Funk et al. 2018]. Entretanto, como pode ser observado na tabela, a maioria dos trabalhos não apresenta evidências empíricas sobre os algoritmos ou mecanismos propostos.

3. Desenvolvimento

O desenvolvimento do trabalho pode ser dividido nas seguintes etapas:

Etapa 1: Seleção dos WAFs ModSecurity, Naxsi, ShadowDaemon e xWAF, utilizando os seguintes parâmetros: (a) WAFs *standalone*, (b) gratuitos e (c) de código aberto.

Etapa 2: Preparação e instalação de máquinas virtuais, uma para cada WAF, com 1 vCPU, 2GbB de RAM e a distribuição Linux Ubuntu Server 16.04. A máquina hospedeira possui processador i5 7300-HQ *quad-core* de 2.5GHz, 8GB de memória RAM, disco rígido *Western Digital*, modelo WD10SPZX, controladora HM170/QM170 Chipset SATA de 6.0GHz, com 5400 rotações por minuto e cache de 128MB, executando a distribuição Linux Manjaro versão 18.0.4 e o VirtualBox na versão 6.0.6.

Etapa 3: Instalação dos WAFs seguindo a documentação de cada ferramenta. O ModSecurity foi instalado através do gerenciador de pacotes do Ubuntu, enquanto que o Naxsi e o ShadowDaemon foram instalados a partir do código fonte, disponível no site oficial dos respectivos WAFs. Já no caso do xWAF, que é implementado em PHP, foi necessário adicionar o arquivo da ferramenta ao parâmetro `auto_prepend_file` do arquivo de configuração do PHP (`php.ini`) do servidor Apache. Isto faz com que o código do xWAF seja automaticamente incluído no início de cada arquivo PHP da aplicação.

Etapa 4: Instalação da aplicação Web PHP que implementa as dez vulnerabilidades mais recorrentes em sistemas Web segundo a OWASP [Ferrao et al. 2018]. Para executar a aplicação, foi utilizado o PHP versão 7.0.3, o MySQL versão 5.7.25 e o servidor Web Apache (versão 2.4.18), exceto no caso do WAF Naxsi, que é compatível apenas com o servidor Web Nginx (foi utilizada a versão 1.13.1).

Etapa 5: Teste de funcionamento dos WAFs através da criação de uma regra que bloqueia requisições específicas provenientes do comando `curl` do Linux. Utilizando como exemplo o ModSecurity, a regra utiliza o arquivo `curl.txt`, que contém uma lista dos `User-Agents` (e.g., *curl/7.64.1*), um por linha, do comando `curl`. O WAF bloqueia (`deny` no algoritmo) e registra (`log`) todas as requisições cujo cabeçalho contém um `User-Agent` listado no arquivo `curl.txt`. Ao bloquear uma requisição, o WAF registra o motivo do bloqueio (`msg`) e os detalhes da solicitação, como URL da requisição e endereço IP do cliente, por fim, retorna como resposta ao cliente o código HTTP 403 (`status`), que significa que a solicitação foi entendida pelo servidor, porém não será atendida.

```
SecRuleEngine On
SecRule REQUEST\_HEADERS:User-Agent "@pmFromFile curl.txt"
 "id:12345,deny,log,status:403,msg:'cURL tentando enviar
    requests'"
```

4. Resultados

Ao instalar um WAF, o administrador do sistema deve analisar as necessidades da aplicação e configurar o WAF de acordo, incluindo regras próprias para ataques específicos ou aumentando o número de regras ativas, por exemplo [Rao et al. 2016, Clincy and Shahriar 2018]. Enquanto que, por um lado, um número maior de regras ativas pode levar a uma maior capacidade de detecção, por outro lado, isto pode impactar a latência de processamento das requisições Web.

Com o objetivo de identificar o impacto do número de regras na latência das requisições, em diferentes WAFs, foi implementado um programa em Python utilizando a biblioteca `Requests` [6]. O programa simula dois tipos de usuários, um não malicioso e outro malicioso. Enquanto que as requisições do primeiro não são bloqueadas (**Pass**), as do segundo são bloqueadas (**Match**) pelo WAF. Os WAFs e o sistema Web (cenário controlado) foram virtualizados, enquanto que o programa Python foi executado a partir da máquina hospedeira.

4.1. Impacto na latência das requisições Web

Na Tabela 2, a latência de uma requisição (em milissegundos) representa a média aritmética de mil requisições. Vale ressaltar que, inicialmente, os WAFs foram configurados com 200 regras ativas. Posteriormente, foram adicionadas novas regras ao conjunto.

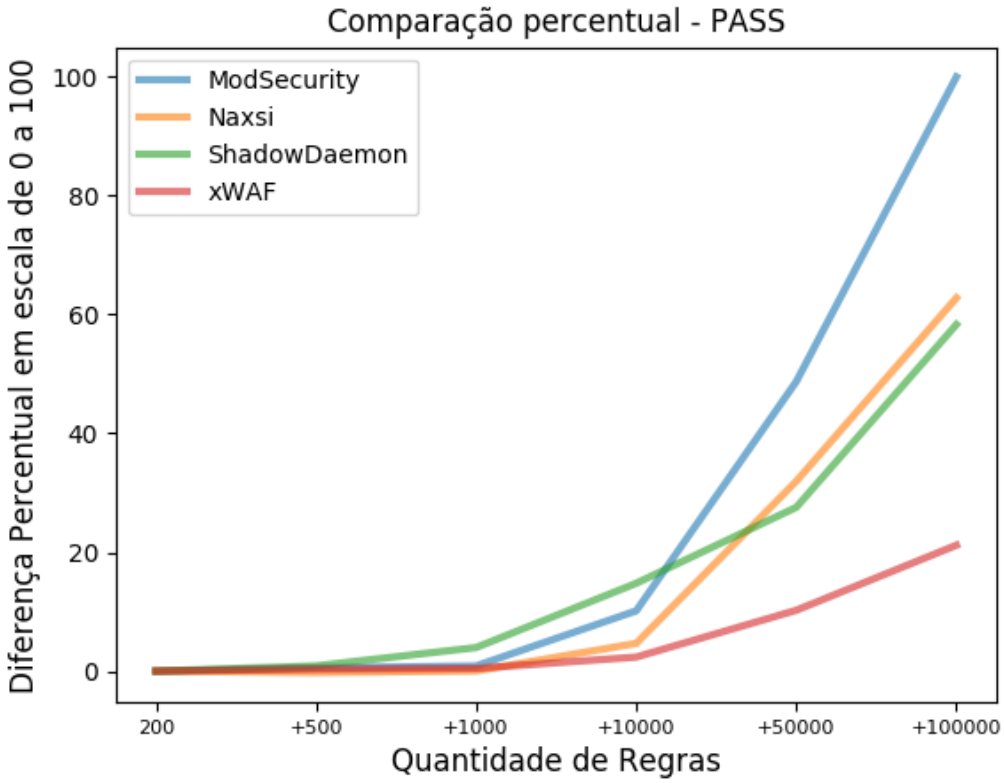
Tabela 2. Tempo de acesso de acordo com a quantidade de regras

	ModSecurity		**Naxsi**		**ShadowD.**		**xWAF**	
	Pass	**Match**	**Pass**	**Match**	**Pass**	**Match**	**Pass**	**Match**
200	1,66ms	1,39ms	1,71ms	1,21ms	1,64ms	1,40ms	1,62ms	1,53ms
+ 500	1,87ms	1,40ms	1,61ms	1,25ms	1,98ms	1,74ms	1,75ms	1,68ms
+ 1000	2,00ms	1,42ms	1,72ms	1,31ms	3,17ms	1,99ms	1,79ms	1,64ms
+ 10000	5,62ms	1,91ms	3,59ms	2,89ms	7,34ms	4,12ms	2,52ms	1,75ms
+ 50000	20,68ms	4,35ms	14,56ms	12,72ms	12,27ms	9,44ms	5,52ms	2,57ms
+ 100000	40,69ms	7,77ms	26,96ms	23,75ms	24,13ms	16,38ms	9,69ms	3,71ms

Os resultados mostram que, por via de regra, o usuário não malicioso acaba sendo o mais prejudicado com o aumento do número de regras ativas. Isto ocorre devido ao fato de uma requisição normal (**Pass**) ser analisada e processada por todas as regras ativas. Por outro lado, uma solicitação maliciosa é bloqueada na primeira regra que identificar o ataque (i.e., primeiro **Match**).

[6]Os scripts utilizados nos testes estão disponíveis em `https://bit.ly/2LcaF8m`

Figura 1. Aumento percentual do usuário normal (PASS)

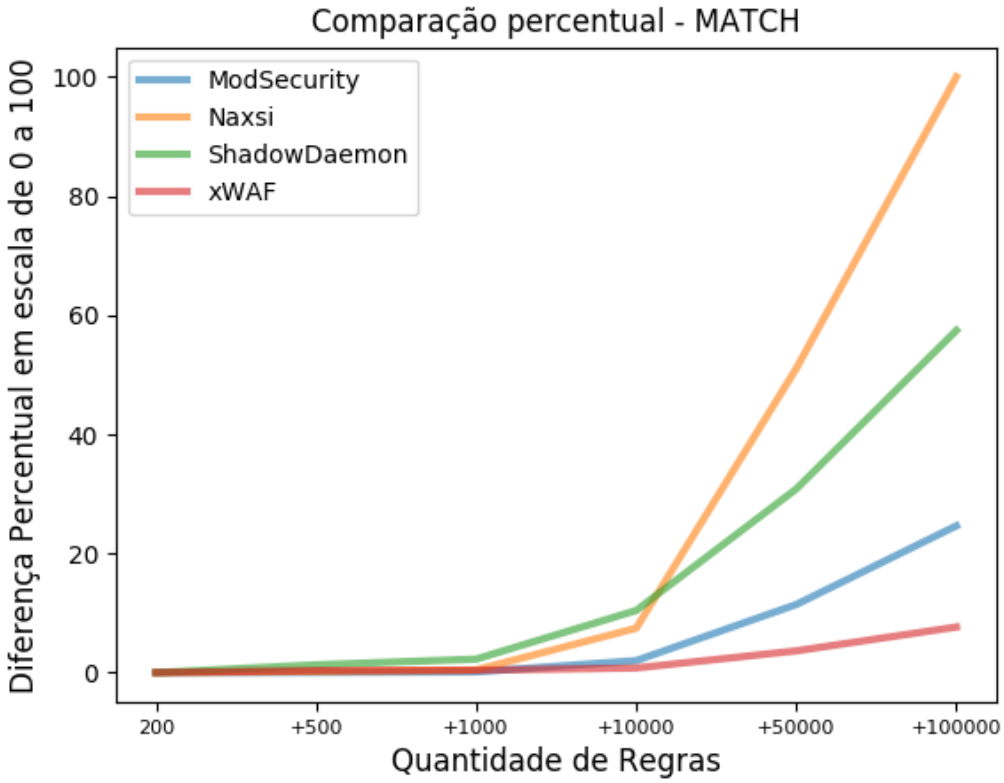


Com o aumento progressivo no número de regras, o ModSecurity passou de 1.66ms (200 regras) para cerca de 40ms de latência (100k regras), o que representa um aumento percentual aproximado de 2.350%. Este aumento, somado ao tempo de latência da rede de um usuário, pode impactar o desempenho do sistema e a experiência do usuário. Como pode ser facilmente observado na Figura 1, que ilustra o aumento percentual dos tempos com relação ao aumento gradativo das regras, o ModSecurity é o WAF que mais impactou na latência das requisições Web. Já o xWAF resultou no menor aumento de latência até 100k regras, atingindo apenas 10ms, o que representa um aumento de cerca de 500%. Entretanto, mesmo no caso do xWAF, o aumento na latência das requisições Web é significativo. Vale também ressaltar que o xWAF funciona apenas com sistemas PHP.

Nos experimentos com requisições bloqueadas pelos WAFs (**Match**), a ferramenta Naxsi chegou a uma latência aproximada de 24ms na detecção dos ataques. Adicionalmente, este WAF atingiu também o maior aumento percentual no caso do **Match**, chegando a 1.800%. Enquanto isso, o xWAF aumentou percentualmente a latência das requisições em apenas 150%. Ambos os resultados podem ser observados na Figura 2.

Tanto no caso do usuário não malicioso, quanto no caso do agente malicioso, os menores tempos de latência (10ms e 4ms) e menores percentuais de aumento (500% para o **Pass** e 150% para o **Match**, respectivamente), de acordo com o número de regras, foram do xWAF. Esta latência significativamente menor do xWAF, para grandes números

Figura 2. Aumento percentual do usuário malicioso (MATCH)



de regras, pode ser explicada pelo fato de o código do xWAF ser mais simples e por ele ser incluído diretamente no código fonte das aplicações PHP.

Como pode ser observado nos resultados da Tabela 2, ao se aproximar de 100k regras adicionais no Naxi, a latência média das requisições fica próxima de 27ms. A título de comparação, considerando uma rede cabeada de fibra óptica e oito saltos, a latência de acesso ICMP ao servidor de resolução de nomes (DNS) do Google (endereço IP 8.8.8.8) é de cerca de 24ms. Neste exemplo, o WAF está simplesmente dobrando o tempo de acesso a um serviço similar ao DNS da Google.

```
   $ traceroute to 8.8.8.8 (8.8.8.8), 30 hops max,
    60 byte packets
 1  _gateway (192.168.0.1)  2.275 ms  2.353 ms  2.427 ms
 2  Dinamico-199-198.redeconesul.com.br (186.251.199.198)
    4.895 ms  4.997 ms  4.979 ms
 3  Dinamico-199-197.redeconesul.com.br (186.251.199.197)
    4.958 ms  4.936 ms  4.999 ms
 4  xe-1-3-1.3933.ar4.gru1.gblx.net (64.214.128.61)
    11.088 ms  11.380 ms  11.360 ms
 5  64.209.11.190 (64.209.11.190)  95.183 ms
    ae0-120G.ar4.GRU1.gblx.net (67.16.148.6)
    25.781 ms 64.209.11.190 (64.209.11.190)  94.825 ms
 6  72.14.212.213 (72.14.212.213)
    26.745 ms  23.466 ms  24.700 ms
 7  * * *
```

```
8  google-public-dns-a.google.com (8.8.8.8)
   24.003 ms  24.904 ms  25.148 ms
```

4.2. Consumo de recursos computacionais

Durante os testes de latência, foi investigado também o consumo de recursos computacionais. A Tabela 3 resume os resultados para 200, 50k e 100k regras. As colunas P e M indicam a utilização de CPU e memória RAM, respectivamente. Estes dados foram obtidos através do programa `HTOP`[7] que foi utilizado para monitorar o uso de memória RAM (em *Megabytes*) e a porcentagem de consumo de CPU.

Tabela 3. Resumo da utilização de recursos computacionais

	ModSecurity		**Naxsi**		**ShadowD.**		**xWAF**	
	P	**M**	**P**	**M**	**P**	**M**	**P**	**M**
200	50%	200	46%	240	41%	253	37%	259
50k	80%	411	87%	280	84%	303	70%	270
100k	94%	750	91%	330	89%	315	84%	297

Novamente, como esperado e de acordo com os resultados apresentados na Seção 4.1, os WAFs ModSecurity e Naxsi apresentaram as maiores percentagens de consumo médio de CPU, ambas acima de 90%. Este aumento significativo (de 50% para 95% no caso do ModSecurity, por exemplo) na utilização do processador pode ajudar a explicar o aumento no tempo de resposta dessas duas soluções. Além disso, como pode ser observado na Tabela 3, mais uma vez de acordo com os resultados apresentados na Tabela 2, a ferramenta xWAF foi a que resultou no menor consumo de CPU para 100k regras.

Com relação ao consumo de memória RAM, todas as soluções, com excessão do ModSecurity, mantiveram um consumo inferir a 350 MB. Além disso, a variação no consumo de memória, entre 200 e 100k regras, foi pequena. O único caso destoante foi o ModSecurity, que passou de 200 MB para cerca de 750 MB, representando um aumento de 275% no uso de memória disponível no servidor. Para 100k regras, o xWAF foi o que menos consumiu memória RAM. Além disso, o xWAF foi o WAF que percentualmente menos aumentou o consumo de memória, passando de 259 MB (com 200 regras) a 297 MB (com 100k regras), o que representa um aumento de apenas 15%.

5. Conclusão

Neste trabalho foi realizado um levantamento do estado da arte e uma avaliação empírica dos WAFs ModSecurity, Naxsi, ShadowDaemon e xWAF. Os resultados mostram que há um impacto significativo na latência das requisições Web com o aumento do número de regras no WAF. Segundo os experimentos realizados, o melhor desempenho foi do xWAF, mantido e criado pela comunidade, com cerca de 10ms de latência para 100k regras. Entretanto, o xWAF é limitado a sistemas PHP e precisa ser adicionado ao código fonte. Já os WAFs ModSecurity, Naxsi e ShadowDaemon são de uso geral, porém, atingiram tempos de latência de aproximadamente 40ms, 27ms e 24ms, respectivamente.

[7] `https://hisham.hm/htop/`

Os resultados evidenciam que é importante investigar o impacto das regras e configurações de WAFs nas requisições dos usuários. Os resultados permitiram verificar, também, que os usuário não maliciosos são os mais prejudicados com o aumento no número de regras. Isto ocorre devido ao fato de as requisições do agente malicioso serem rejeitadas no primeiro **Match**, enquanto que as requisições dos demais usuários precisam ser processadas por todas as regras.

Referências

Clincy, V. and Shahriar, H. (2018). Web Application Firewall: Network Security Models and Configuration. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 835–836.

Explorable (2009). Evidencia Empirica. `https://bit.ly/2UaKa7j`.

Ferrao, I. G., de Macedo, D. D. J., and Kreutz, D. (2018). Investigação o do impacto de frameworks de desenvolvimentode software na segurança de sistemas web. In *16a Escola Regional de Redes de Computadores (ERRC)*. `https://bit.ly/2WQAfZd`.

Ferrão, I. G. (2018). Análise black-box de ferramentas de segurança na web. Trabalho de conlusão de curso, Universidade Federal Do Pampa. `https://bit.ly/2XjhblU`.

Funk, R., Epp, N., and A., C. C. (2018). Anomaly-based Web Application Firewall using HTTP-specific features and One-Class SVM. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação (ReABTIC)*, 2(1).

Machado, R. B., Kreutz, D., Paz, G., and Rodrigues, G. (2019). Vazamentos de Dados: Histórico, Impacto Socioeconômico e as Novas Leis de Proteção de Dados. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. SBC. `http://errc.sbc.org.br/2019/wrseg/papers/machado2019vazamentos.pdf`.

Melchior, F., Kreutz, D., and Fiorenza, M. (2019a). Web Application Firewalls (WAFs): o impacto do número de regras na latência das requisições Web. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*, Alegrete-RS, Brasil. `http://errc.sbc.org.br/2019/wrseg/papers/melchior2019web.pdf`.

Melchior, F., Kreutz, D., Fiorenza, M., Flora, F., Ferrao, I., Fernandes, R., Escarrone, T., and Macedo, D. (2019b). Introdução a Web Application Firewalls (WAFs): Teoria e Pratica. In *17a Escola Regional de Redes de Computadores*, Alegrete-RS, Brasil. `http://errc.sbc.org.br/2019/mc/melchior2019wafs.pdf`.

Moosa, A. and Alsaffar, E. M. (2008). Proposing a Hybrid-intelligent Framework to Secure e-Government Web Applications. In *Proceedings of the 2nd International Conference on Theory and Practice of Electronic Governance*, pages 52–59. ACM.

Muncaster, P. (2019). Over 60 US Colleges Compromised by ERP Exploit. `https://bit.ly/2SC8zlm`.

Rao, G. R. K., Prasad, R. S., and Ramesh, M. (2016). Neutralizing Cross-Site Scripting Attacks Using Open Source Technologies. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, pages 24:1–24:6. ACM.

Razzaq, A., Hur, A., Shahbaz, S., Masood, M., and Ahmad, H. F. (2013). Critical analysis on web application firewall solutions. In *IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 1–6.

Rietz, R., König, H., Ullrich, S., and Stritter, B. (2016). Firewalls for the Web 2.0. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*.

Singh, J. J., Samuel, H., and Zavarsky, P. (2018). Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall. In *1st International Conference on Data Intelligence and Security (ICDIS)*, pages 141–144.

Srokosz, M., Rusinek, D., and Ksiezopolski, B. (2018). A New WAF-Based Architecture for Protecting Web Applications Against CSRF Attacks in Malicious Environment. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*.