Capítulo

5

Introdução à Web Application Firewalls (WAFs): Teoria e Prática

Felipe Melchior, Diego Kreutz, Maurício Fiorenza, Fernando Flora (Unipampa), Isadora Ferrão (USP), Rafael Fernandes, Thiago Escarrone (Unipampa), Douglas Macedo (UFSC)

Estudos recentes apontam que cerca de 50% das aplicações Web disponíveis na Internet possuem pelo menos uma vulnerabilidade de alta criticidade, como *SQL Injection* [Acunetix 2019]. Se for levado em consideração o nível de risco médio, cerca de 90% das aplicações podem ser consideradas vulneráveis. Ainda segundo relatórios especializados, a vulnerabilidade de *Cross-Site Scripting* (XSS) é uma das mais comuns e mais exploradas (representando cerca de 30%) em aplicações Web [HackerOne 2019]. Além de ser frequente, em alguns casos, a exploração da vulnerabilidade XSS permite ao atacante acessar recursos e dados privados de empresas [Cimpanu 2019].

Como uma forma de contribuir para melhorar o cenário crítico das aplicações Web disponíveis na Internet, especialistas em segurança da informação criaram a fundação OWASP (*The Open Web Application Security Project*)[1]. A entidade tem como principal objetivo disseminar conhecimento sobre segurança de aplicações Web disponíveis na Internet. Além disso, a OWASP também mantém um ranking tri-anual das 10 vulnerabilidades mais recorrentes em sistemas Web.

Segundo a classificação da OWASP de 2017[2], as vulnerabilidades de injeção de falhas (e.g., *SQL Injection*, *NoSQL Injection* e *LDAP Injection*) estão no topo do ranking. Outra vulnerabilidade recorrente em aplicações Web, que aparece na sétima posição, é XSS. A XSS também é um tipo de ataque de injeção de código malicioso nas páginas web. Ao explorar uma vulnerabilidade XSS, o atacante injeta códigos (ou scripts) que executam ações maliciosas como copiar *cookies* e roubar *tokens* de sessão ou quaisquer outros dados do usuário que estão no navegador.

Uma das formas de mitigar o impacto dessas vulnerabilidades antigas e recorrentes, como XSS e *SQL Injection*, é através da utilização de *frameworks* de desenvolvimento

[1] https://www.owasp.org
[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

de software como o Laravel[3], que implementa e oferece recursos de segurança que protegem, de forma nativa e transparente, as aplicações Web PHP contra falhas de código. Como exemplo, um *framework* como o Laravel é capaz de, por si só, mitigar a exploração de até 60% das vulnerabilidades mais recorrentes em aplicações PHP [Ferrão et al. 2018].

Outro tipo de ferramenta que tem sido utilizada na proteção de aplicações Web são os *Web Application Firewalls* (WAFs). 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). A função do WAF é interceptar e processar as requisições entre o cliente e a aplicação. A partir de um conjunto de regras, ele 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[4], na configuração padrão (isto é, sem nenhuma otimização), associado a um *framework* PHP como o Laravel, é capaz de mitigar em 70% a exploração de vulnerabilidades recorrentes em sistemas Web [Ferrão et al. 2018].

Contra-intuitivamente, a adição de um WAF pode também piorar a segurança de um sistema Web. Pesquisas recentes apresentam casos onde a adição de um WAF ocasiona uma queda (ao invés de um aumento) no nível de proteção da aplicação Web. Este é o caso do cenário utilizando o *framework* PHP Symfony[3] e os WAFs Naxsi[4] e ShadowDaemon[4]. Enquanto o Symfony mitiga em até 60% a exploração das vulnerabilidades mais recorrentes em aplicações Web, a adição dos WAFs Naxsi e ShadowDaemon reduz a mitigação para 50% [Ferrão 2018]. Isto significa que houve uma queda de 10% na segurança da aplicação Web ao adicionar os WAFs.

Os dois principais tipos de WAFs disponíveis no mercado são os que operam como sistemas *standalone* e os SaaS (*Security-as-a-Service*). Os *standalone* são, geralmente, instalados junto aos servidores Web, em que as aplicações estão sendo executadas. Janusec Application Gateway[4], Tempesta FW[4], ModSecurity, OpenWAF[4] e Naxsi são exemplos WAFs *standalone* gratuitos, enquanto que o Imperva[4] e o Citrix[4] são comerciais. Os WAFs SaaS são oferecidos como serviços de segurança sob demanda por terceiros, sendo essencialmente online, oferecidos por empresas como a CloudFlare[5] e a X Labs Security[5]. Neste modelo, o WAF atua como um *proxy* de redirecionamento, onde a requisição é analisada e processada antes de ser encaminhada para o servidor de aplicação propriamente dito.

Apesar de ser um tema relativamente antigo, a importância dos WAFs tem crescido rapidamente no contexto atual, onde ciber ataques, que exploram as vulnerabilidades mais recorrentes de aplicações Web, vêm causando uma quantidade crescente de graves incidentes de segurança [Machado et al. 2019]. Quase que diariamente, incidentes como o comprometimento de sistemas online e vazamento de dados são reportados mundo a fora, levando a prejuízos socioeconômico cada vez maiores [Romani 2016, Convergência Digital 2016, Portinari 2018, Machado et al. 2019]. Em alguns casos, empresas tem

[3] https://laravel.com, https://symfony.com, https://codeigniter.com/, https://phalcon.io/.

[4] https://modsecurity.org, https://github.com/nbs-system/naxsi, https://shadowd.zecure.org, https://www.imperva.com, https://www.janusec.com/, https://github.com/titansec/OpenWAF, https://github.com/tempesta-tech/tempesta, https://www.citrix.com.br.

[5] https://www.cloudflare.com/waf/, https://www.xlabs.com.br/waf/.

entrado com pedido de proteção contra falência devido a incidentes de segurança que levaram a vazamentos de dados, como foi o caso recente da empresa AMCA, nos EUA.

Estudos vêm identificando diferentes desafios e oportunidades de pesquisa no contexto de *Web Application Firewalls*, como algoritmos para detectar ataques que objetivam explorar vulnerabilidades específicas (e.g., *Cross-Site Request Forgery* (CSRF) e 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 revisão minuciosa do estado da arte, procurando comparar e entender o funcionamento de diferentes WAFs [Clincy and Shahriar 2018, Razzaq et al. 2013, Melchior et al. 2019].

O principal objetivo deste minicurso é apresentar um panorama geral da teoria e da prática de *Web Application Firewalls* na proteção de sistemas Web. Alguns dos conceitos básicos e um resumo do estado da arte são apresentados na Seção 5.1. Na sequência, Seção 5.2, são discutidos detalhes técnicos de instalação e configuração de quatro WAFs gratuitos, o ModSecurity, o Naxsi, o ShadowDaemon, o xWAF e o OpenWAF. Nesta seção, são exemplificadas configurações do ModSecurity e do xWAF, bem como formas de verificar na prática o funcionamento de um WAF. Como forma de exemplificar, através de números reais, o impacto de um WAF na latência das requisições Web, na Seção 5.3 são apresentadas e discutidas questões de desempenho dos quatro WAFs gratuitos selecionados anteriormente. Como poderá ser observado, há um impacto significativo na latência das requisições Web, que aumenta de acordo com o número de regras ativas. Além disso, existem diferenças de desempenho, igualmente significativas, entre os WAFs. Finalmente, nas Seções 5.4 e 5.5, são apresentadas uma análise da eficácia de WAFs e *frameworks* de desenvolvimento de software na proteção de aplicações Web e as considerações finais, respectivamente.

5.1. Conceitos Básicos e Estado da Arte

5.1.1. WAFs *standalone* e SaaS

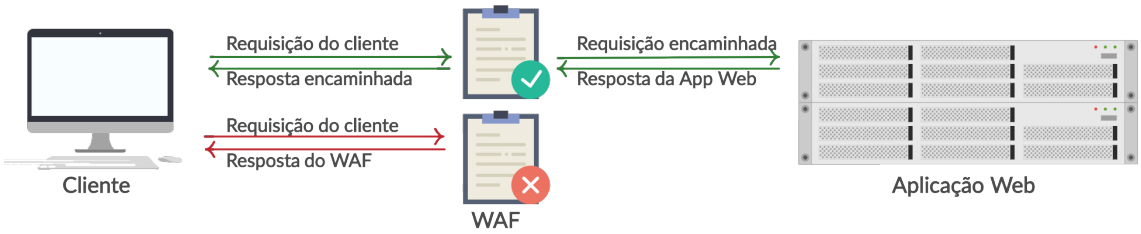
Existem WAFs *standalone*, que são instalados entre o cliente e a aplicação, e WAFs SaaS, que utilizam redirecionamento DNS[6] para que o tráfego seja encaminhado e processado na infraestrutura da empresa proprietária do WAF antes de ser encaminhado ao servidor do sistema Web.

A Figura 5.1 ilustra a utilização e o funcionamento de um WAF *standalone*. Geralmente, este tipo de WAF é instalado no mesmo servidor (i.e., mesma máquina virtual ou física) da aplicação Web que ele irá proteger, reduzindo a latência de comunicação (entre o WAF e o serviço Web) e aumentando a personalização das regras que irão proteger a aplicação. Como será discutido na sequência, criar regras personalizadas, específicas às aplicações Web que serão protegidas pelo WAF, é um dos processo importantes na instalação e manutenção dessas ferramentas de segurança.

Como ilustrado na figura, o cliente realiza uma requisição ao serviço Web (e.g., `links https://unihacker.club`). Esta requisição é recebida e processada pelo WAF, que decidirá se ela segue adiante, até a aplicação Web (e.g., servidor Web Apache2

[6] https://www.pcmag.com/encyclopedia/term/41620/dns

Figura 5.1: Visão geral de um WAF *standalone*



e aplicação PHP), ou se é rejeitada. Caso nenhuma das regras do WAF detecte algo de anormal na requisição, a aplicação Web processa a requisição e envia uma resposta ao cliente. Este processo é ilustrado pelas setas verde da figura. Entretanto, caso alguma das regras do WAF detecte algo suspeito na requisição (e.g., *SQL Injection*), ela será imediatamente negada, isto é, o cliente irá receber uma mensagem de erro do WAF, como ilustrado pelas setas vermelhas na imagem.

O principal desafio no processo de adoção dos WAFs *standalone* é a disponibilidade de recursos humanos qualificados, que são necessários para lidar com os seguintes desafios técnicos:

(dt_1) instalação, configuração e manutenção contínua;

(dt_2) criação e gerenciamento de regras;

(dt_3) otimização de regras; e

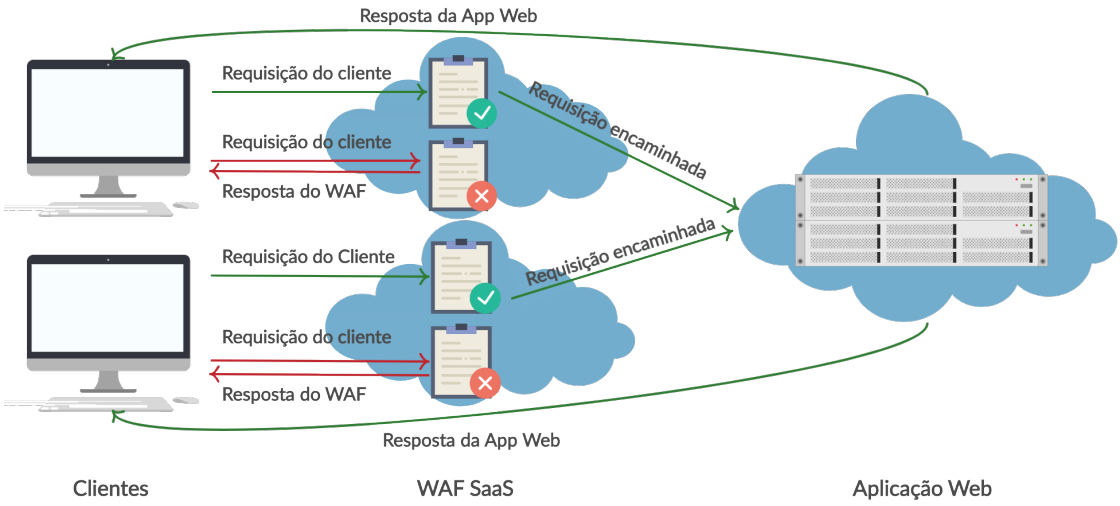
(dt_4) suporte a grandes volumes de tráfego.

Os WAFs SaaS surgem como uma forma de resolver o problema da falta (ou custo) de mão-de-obra especializada, necessária no caso dos WAFs *standalone*. Um WAF SaaS é um serviço oferecido sob demanda e por terceiros, geralmente empresas especializadas em segurança de sistemas. Entretanto, utilizar um WAF SaaS implica em firmar um contrato de confiança entre o contratante e a contratada, pois o tráfego da empresa contratante irá passar pelos sistemas da contratada. Em outras palavras, a confidencialidade e a privacidade dos dados entra em discussão uma vez que fica fácil para qualquer pessoa com acesso à infraestrutura da contratada realizar interceptação de tráfego.

A Figura 5.2 ilustra um WAF SaaS. Enquanto que o controle das requisições é essencialmente idêntico a um WAF *standalone* (i.e., deixa passar ou bloqueia a requisição), há pelo menos dois diferenciais, importantes, que merecem atenção. Primeiro, o tráfego do cliente é redirecionado para a infraestrutura do WAF. Na prática, tomando como exemplo o domínio `unihacker.club` ou `www.xlabs.com.br`, ao invés de o tráfego ir diretamente para a infraestrutura que mantém o domínio, a aplicação Web, o tráfego passa

primeiramente pelo domínio (e.g., `web.armor.zone` da XLabs Security) da empresa que fornece o WAF SaaS. Neste caso, o redirecionamento é realizado diretamente no serviço de DNS do domínio que está contratando o serviço de WAF (e.g., apontar o domínio `unihacker.club` para `web.armor.zone`).

Figura 5.2: Ilustração de um WAF SaaS



O serviço de WAF SaaS é escalável e capaz de atender picos de tráfego, bem como um grande ou variável número de clientes. Para tanto, um WAF SaaS é geralmente construído sobre uma rede CDN (*Content Delivery Network*) ou uma plataforma de computação em nuvem com múltiplos *datacenters* espalhados pelas regiões de atuação ou interesse da empresa do WAF. Como ilustrado na Figura 5.2, há dois clientes utilizando instâncias distintas do WAF. Estas instâncias podem estar na mesma localização geográfica (e.g., mesmo país) ou em localizações geográficas completamente distintas.

5.1.2. Visão Geral do Estado da Arte

A Tabela 5.1 apresenta de forma resumida as principais **contribuições**, **oportunidades** de pesquisa e **evidências** empíricas encontradas em trabalhos relacionados a *Web Application Firewalls*.

Contribuição. Há pesquisas e desenvolvimentos que propõe 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 ":") frequentemente utilizados em requisições maliciosas aos sistemas Web [Rao et al. 2016].

Outro assunto importante são as boas práticas na criação de novas regras para o WAF. O primeiro passo é entender o objetivo da regra que está sendo criada [Clincy

Tabela 5.1: Resumo do estado da arte

	Contribuição	**Oportunidades**	**Evidências**
[**Funk et al. 2018**]	WAF com objetivo de melhorar a taxa de detecção de ataques	Comparar com outros WAFs, além do ModSecurity	Taxa de detecção cerca de 60% maior em relação ao detectado pelo 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 para aplicações governamentais	Evoluir o WAF com novas heurísticas	WAF lida bem com um volume grande de requisições
[**Razzaq et al. 2013**]	Comparação analítica de quinze WAFs tradicionais	Comparar de forma empírica os WAFs e adicionar soluções *SaaS*	
[**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	Níveis Paranoia aumentam a taxa de detecção e de falsos positivos
[**Ferrão 2018**]	Avaliação de três WAFs em cenários controlados	Analisar outros WAFs e melhorar a taxa de detecção de ataques	ModSecurity mitiga 70% das vulnerabilidades do Top Ten da OWASP

and Shahriar 2018]. Isto é crucial para o bom funcionamento e 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- #"`.

Oportunidades. A maioria dos estudos indica de forma explicita ou implícita oportunidades de pesquisa e desenvolvimento, como implementar novas heurísticas para melhorar a taxa de detecção de WAFs, avaliar e comparar a qualidade dos WAFs através de estudos empíricos, avaliar soluções SaaS e avaliar empiricamente as di-

ferentes configurações dos WAFs. Neste minicurso, mais precisamente nas Seções 5.3 e 5.4, são apresentados e discutidos dados sobre o impacto de WAFs na latência de requisições Web e na segurança de aplicações Web.

Evidências empíricas são dados concretos que permitem melhor avaliar e validar uma pesquisa. Por exemplo, o WAF ModSecurity, na configuração padrão, utilizando um cenário controlado, é capaz de mitigar até 70% dos ataques que objetivam mitigar as vulnerabilidades mais recorrentes em aplicações Web, conforme dados apresentados na Seção 5.4. Indo um pouco além, resultados de pesquisa indicam que implementações específicas de WAFs podem atingir uma eficácia de detecção de cerca de 60% maior que o ModSecurity [Funk et al. 2018].

Analisando o estado da arte, é possível identificar alguns dos principais desafios encontrados na utilização de *Web Application Firewalls*, como detectar ataques específicos (e.g., CSRF), compreender os aspectos de desempenho de um WAF em diferentes cenários, dominar a ferramenta a ponto de criar regras eficazes e otimizar a utilização dos recursos disponíveis.

Detecção de Ataques. Alguns ataques, como os que exploram CSRF e *Server Side Request Forgery* (SSRF), são difíceis de serem mitigados [Srokosz et al. 2018]. *Tokens* podem ser utilizados nas requisições como uma forma de mitigar este tipo de ataque. Entretanto, um *token* mal formulado pode ser burlado pelo atacante. Para resolver o problema, o WAF pode utilizar autenticação adicional ao detectar uma anomalia (um desvio de padrão) entre a requisição atual e o histórico de requisições do usuário.

Desempenho. Em alguns cenários, o desempenho de WAFs é crítico devido ao grande número de requisições a serem processadas [Moosa and Alsaffar 2008]. Arquiteturas híbridas, ancoradas em heurísticas e redes neurais, podem ser utilizadas para melhorar o desempenho dos WAFs nesse tipo de cenário.

Domínio da ferramenta. Explorar as diferentes configurações de um WAF é importante para que se possa extrair o máximo de eficiência da ferramenta. Tomando como exemplo o ModSecurity, a solução possui os chamados níveis de Paranoia, que são tipos de regras ativadas apenas quando necessário, como durante uma sequência de ataques, por exemplo [Singh et al. 2018]. No nível de Paranoia baixo, o número de falsos positivos é baixo, porém a taxa de detecção também é menor. Ao ativar o nível de Paranoia alto, todas as regras presentes na configuração são ativadas ao mesmo tempo, detectando um número maior de ataques. Entretanto, a taxa de falsos positivos também pode aumentar.

5.2. Instalação, Configuração e Verificação do Funcionamento de WAFs

Nesta seção são apresentadas informações de instalação, configuração e verificação do funcionamento dos WAFs: ModSecurity, Naxsi, ShadowDaemon, xWAF e OpenWAF. A instalação dos WAFs seguiu a respectiva documentação de cada ferramenta. Ferramentas

como ModSecurity, Naxsi e ShadowDaemon seguem uma linha de instalação bem semelhante, podendo mudar alguns detalhes específicos de configuração. O xWAF possui um processo de instalação diferentes dos demais WAFs visto que trata-se de um código PHP que necessita ser incluído no código da aplicação Web. Já o OpenWAF requer a instalação de pacotes e compilação a partir do código fonte.

O ModSecurity será utilizado para exemplificar a instalação de um WAF na forma tradicional. Seus arquivos estão disponíveis nos repositórios de pacotes da maioria das distribuições Linux. Consequentemente, basta executar o comando de instalação de acordo com o gerenciador de pacotes do sistema em questão. Como foram utilizadas máquinas virtuais rodando Linux Ubuntu Server 16.04, o ModSecurity foi instalado através do APT, utilizando o seguinte comando: `apt install libapache2-modsecurity`.

Após a instalação do pacote, o primeiro passo consiste em habilitar o ModSecurity, o que pode ser realizado através da adição da linha `SecRuleEngine On` no arquivo `/etc/modsecurity/modsecurity.conf`. Na sequência, devem ser adicionadas as regras. Seguindo a documentação da ferramenta, é recomendado o uso do pacote de regras CRS (*Core Rule Set*)[7], que consiste em um conjunto de regras pré-definidas. Estas regras, segundo a documentação, mitigam as dez vulnerabilidades mais recorrentes na Web conforme classificação da OWASP.

O passo seguinte consiste em configurar o arquivo `security2.conf`, disponível no diretório `/etc/apache2/mods-enabled/`, que representa a extensão do ModSecurity no Apache2. No arquivo é necessário indicar o diretório (e.g., `/var/cache/modsecurity`) onde serão armazenados os dados (e.g., dados de endereços IP, dados de sessões) persistentes do ModSecurity e a localização do conjunto de regras (e.g., `/usr/share/modsecurity-crs/`) que serão carregadas e utilizadas pelo ModSecurity. O conteúdo final do arquivo é ilustrado a seguir.

```
<IfModule security2_module>
  # Diretório para armazenar os dados persistentes
  SecDataDir /var/cache/modsecurity
  # Inclusão de todos os arquivos de regras do diretório
  IncludeOptional /usr/share/modsecurity-crs/*.conf
</IfModule>
```

Enquanto que o processo de configuração dos WAFs Naxsi e ShadowShadow é similar ao ModSecurity, a configuração do xWAF é diferente. O xWAF é um código PHP que implementa uma classe de funcionalidades e necessita ser instanciada em cada arquivo PHP da aplicação Web, como ilustrado a seguir.

```
<?php
    require('xwaf.php');
    $xWAF = new xWAF();
    $xWAF->start();
```

[7] https://github.com/SpiderLabs/owasp-modsecurity-crs

```
        ... código da aplicação ...
?>
```

As três linhas adicionadas no início do arquivo PHP da aplicação importam o código do xWAF, disponível no GitHub[8], instanciam um novo objeto da classe `xWAF` e invocam o método `start()` do objeto criado, inicializando o WAF propriamente dito. Entretanto, não é viável realizar manualmente a inserção e manutenção das três linhas de código em todos os arquivos do sistema a ser protegido. Para automatizar o processo, basta configurar o arquivo `php.ini` do PHP, adicionando o parâmetro `auto_prepend_file`, como ilustrado a seguir. Este parâmetro irá fazer com que o xWAF seja automaticamente adicionado a todos os arquivos PHP disponíveis no respectivo servidor Web.

```
...
; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file = /var/www/html/waf.php
...
```

Diferentemente do ModSecurity e Naxsi, a solução xWAF não necessita de configuração inicial das regras uma vez que possui um conjunto de regras embutido e ativo em seu código fonte. Caso desejado, outras regras podem ser adicionadas através da edição do código do WAF (e.g., arquivo `waf.php`). Já a ferramenta ShadowDaemon necessita que as regras sejam adicionadas manualmente após a instalação, através de uma interface Web própria.

O OpenWAF foi projetado para ser utilizado com conjunto com o servidor Web Nginx[9] e é divido em dois módulos. O primeiro é baseado em ModSecurity e realiza a verificação e aplicação das regras. O segundo módulo realiza a análise comportamental do trafego com o objetivo de identificar ataques realizados por malwares e violação de cookies, por exemplo.

A instalação do OpenWAF é realizada através de download e compilação de três pacotes pacotes (`pcre-8.42.tar.gz`[10], `openssl-1.1.0h.tar.gz`[11] e `openresty-1.13.6.2.tar.gz`[12]) e do código fonte do WAF[13], conforme detalhado na documentação oficial. A adição e o gerenciamento de regras é realizado através da edição dos arquivos disponíveis no diretório `/opt/OpenWAF/conf`, que é o repositório padrão de regras da ferramenta.

Finalmente, após instalados e configurados, os WAFs precisam ser testados. O teste de funcionamento dos WAFs pode ser realizado através da criação de uma simples regra que bloqueia requisições especificas provenientes do comando `curl`. Utilizando

[8] `https://github.com/Alemalakra/xWAF`
[9] https://www.nginx.com/
[10] https://ftp.pcre.org/pub/pcre/pcre-8.42.tar.gz
[11] https://www.openssl.org/source/openssl-1.1.0h.tar.gz
[12] https://openresty.org/download/openresty-1.13.6.2.tar.gz
[13] `git clone https://github.com/titansec/OpenWAF.git`

como exemplo o ModSecurity, a implementação da regra é apresentada no Algoritmo 1. 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.

Algorithm 1 Bloqueia requisições do `curl`

```
1: SecRuleEngine On
2: SecRule REQUEST_HEADERS:User-Agent "@pmFromFile curl.txt
   id:12345,deny,log,status:403,msg:'curl tentando enviar
   requests'"
```

Caso o usuário envie uma requisição ao servidor (e.g., `curl 192.168.1.1/index.html`), utilizando uma versão do comando `curl` listada no arquivo `curl.txt`, a solicitação será automaticamente bloqueada. O cliente recebe como resposta o código HTTP 403 (`status`)[14], que significa que a solicitação foi entendida pelo servidor, porém não será atendida.

5.3. Impacto dos WAFs na Latência das Requisições Web

Ao instalar um *Web Application Firewall*, o administrador do sistema deve analisar as necessidades da aplicação e configurar o WAF de acordo, incluindo regras específicas 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, pode levar a um impacto na latência das requisições Web.

Com o objetivo de tentar identificar o impacto do número de regras, em diferentes WAFs, na latência das requisições, foi implementado um programa em Python utilizando a biblioteca `Requests`. 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.

Para os testes de desempenho, foram instaladas quatro máquinas virtuais, uma para cada WAF (ModSecurity, Naxsi, ShadowDaemon, xWAF), com 1 vCPU, 2Gb de RAM e a distribuições Linux Ubuntu Server 16.04. A máquina hospedeira possui processador i5 7300-HQ quad-core de 2.5Ghz, 8Gb de memória RAM, placa de vídeo GTX 1050, disco rígido Western Digital, modelo WD10SPZX de um terabyte, 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 o VirtualBox na versão 6.0.6. Os WAFs foram virtualizados, enquanto que o programa Python foi executado a partir da máquina hospedeira.

A aplicação Web PHP, que implementa as dez vulnerabilidades presentes no ranking da OWASP de 2013[2], conforme proposto por trabalhos recentes [Ferrão 2018], tam-

[14] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

bém foi virtualizada junto aos WAFs. 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).

A Tabela 5.2 resume os tempos médios de latência de uma requisição (em milissegundos). Os valores correspondem a média de um mil requisições. Vale ressaltar que as regras que bloqueiam as requisições dos agentes maliciosos foram inseridas no início do conjunto de regras de cada WAFs. Portanto, os números podem variar caso as regras sejam inseridas em diferentes posições dentro do conjunto de regras do WAF.

Tabela 5.2: Tempo de acesso de acordo com a quantidade de regras

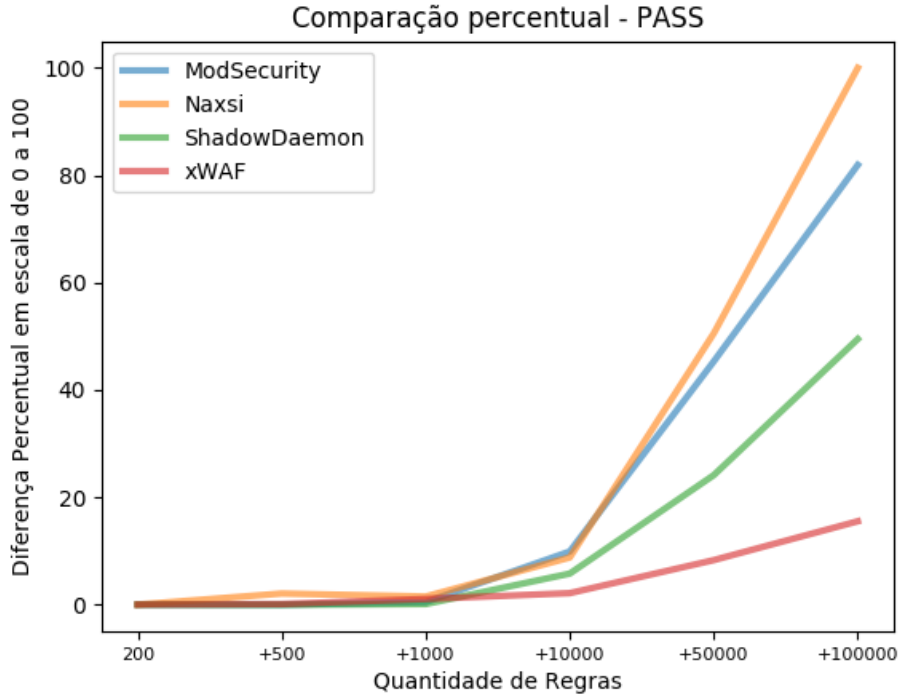
	ModSecurity		**Naxsi**		**ShadowD.**		**xWAF**	
	Pass	**Match**	**Pass**	**Match**	**Pass**	**Match**	**Pass**	**Match**
Padrão	2.70	2.54	1.31	1.01	2.91	2.70	1.31	1.42
+ 500	2.69	2.49	1.90	1.10	2.89	2.74	1.34	1.38
+ 1000	2.98	2.38	1.74	1.23	2.97	2.84	1.61	1.54
+ 10000	8.53	3.39	3.82	3.19	6.58	4.57	1.92	1.77
+ 50000	29.40	6.11	15.76	13.20	18.23	13.04	3.68	2.88
+ 100000	50.97	9.64	29.90	27.39	34.34	26.23	5.75	4.27

Como pode ser observado na Tabela 5.2, existe uma diferença significativa na latência entre os WAFs. Considerando que todos os WAFs foram executados na configuração padrão (linha “padrão” na tabela), a diferença pode ser explicada parcialmente pela quantidade inicial de regras em cada WAF. Por exemplo, o Naxsi inicia com cerca de 70 regras, enquanto que o ModSecurity com aproximadamente 150 regras. Isto, por si só, praticamente já explica a diferença de 2.70ms e 1.31ms (no **Pass**) entre os dois WAFs. Outro motivo provável são os próprios mecanismos de processamento das regras que o WAF implementa.

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**).

Com o aumento progressivo no número de regras, o ModSecurity passou de 2.70 (170 regras) para cerca de 50ms de latência (100k regras), o que representa um aumento percentual aproximado de 1780%. Entretanto, percentualmente, a solução Naxsi teve um desempenho pior com relação ao aumento do número de regras, variando de 1.31ms (70 regras) para 29.90ms (100k regras), o que representa um aumento percentual de latência na ordem de 2180%, como pode ser melhor observado na 5.3. A Figura 5.4 apresenta o aumento percentual para os casos de **Match**.

Figura 5.3: Aumento percentual do usuário normal (PASS)

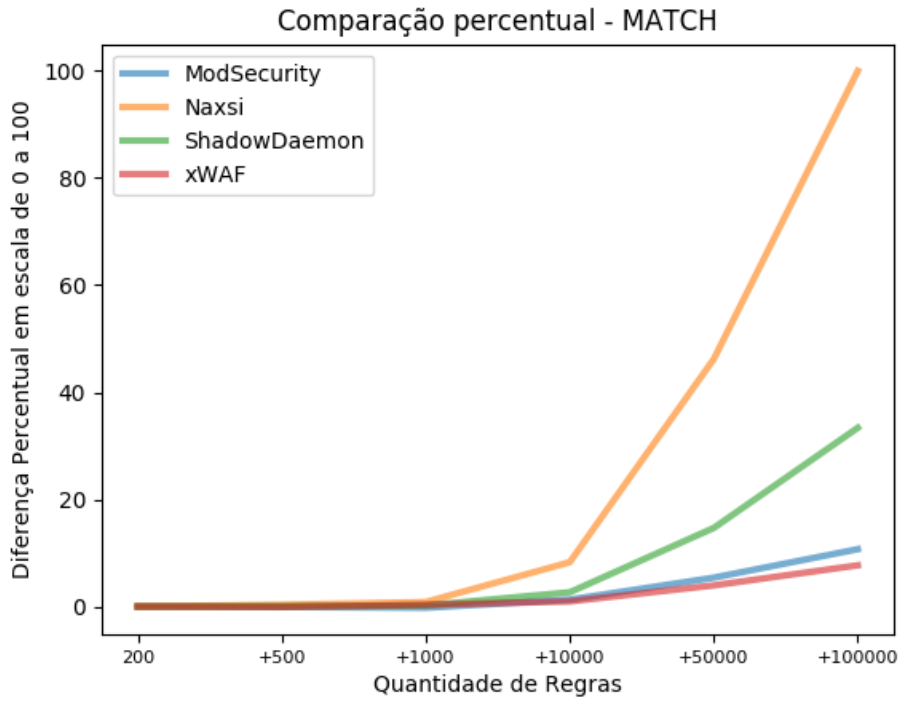


Nos experimentos com requisições bloqueadas pelo WAF (**Match**), a solução Naxsi chegou a uma latência aproximada de 27ms na detecção dos ataques. Adicionalmente, este WAF atingiu também o maior aumento percentual, chegando a 2600%, conforme mostra a Figura 5.4. Enquanto isso, o xWAF aumentou percentualmente a latência das requisições em apenas 200%. Os dados do gráfico nos permitem concluir que os WAFs mantiveram um aumento percentual similar até 10k regras. Entretanto, com mais de 10k regras, a diferença começou a ficar visualmente significativa. Enquanto que o Naxsi e o ShadowDaemon tiveram um aumento exponencial, podemos dizer que os outros dois mantiveram um aumento linear. Estes resultados podem estar relacionados aos mecanismos internos, de processamento da regras, que cada WAF utiliza.

Tanto no caso do usuário não malicioso, quanto no caso do agente malicioso, os menores tempos de latência (6ms) e menores percentuais de aumento (330%), de acordo com o número de regras, foram do xWAF. Esta latência significativamente menor do xWAF, para grandes números de regras, pode ser parcialmente explicado pelo fato de o código do xWAF ser incluído diretamente no código fonte das aplicações PHP. Novamente, temos aqui outro ponto interessante de investigação futura.

Como pode ser observado nos resultados da Tabela 5.2, a partir de cinquenta mil regras adicionais, a latência média das requisições fica próxima de 30ms utilizando o ModSecurity. A título de comparação, considerando uma rede cabeada de fibra óptica, cuja saída do `traceroute` é apresentada a seguir, a latência de acesso ICMP ao servidor

Figura 5.4: Aumento percentual do agente malicioso (MATCH)



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
```

Em resumo, os resultados evidenciam a importância de investigar o impacto das regras e configurações de WAFs nas requisições dos usuários. Dependendo dos requisitos da aplicação, pode ser mais interessante um WAF de menor latência, por exemplo. Por outro lado, um WAF de maior latência, como é o caso do ModSecurity, pode apresentar uma maior acurácia em sua configuração padrão, isto é, sem incluir regras novas e específicas, como pode ser visto na Seção 5.4.

5.4. Eficácia dos WAFs na Segurança de Aplicações Web

Como apresentado e discutido em [Ferrão 2018], a Tabela 5.3 sumariza os resultados obtidos com os testes de três WAFs (ModSecurity, Naxsi e Shadow Daemon) adicionados a cenários controlados contendo uma versão da aplicação Web vulnerável (vide Seção 5.3) implementada nos *frameworks* de desenvolvimento Laravel[3], Synpony[3], Codeigniter[3] e Phalcon[3]. Na tabela constam os nomes dos WAFs, os nomes dos *frameworks*, as vulnerabilidades detectadas pelos respectivos *scanners*, os nomes dos *scanners* e a porcentagem de vulnerabilidades detectadas, respectivamente.

O WAF que apresentou o melhor desempenho foi o ModSecurity, chegando a evitar a detecção de 70% das vulnerabilidades nos *frameworks* Laravel e Symfony, isto é, os *scanners* detectaram apenas 30% das vulnerabilidades existentes na aplicação Web. Os WAFs Shadow Daemon e Naxsi contribuíram menos que o ModSecurity na mitigação da exploração das vulnerabilidades dos ambientes avaliados. No caso do Shadow Daemon, isto era esperado uma vez que este WAF promete mitigar essencialmente vulnerabilidades de injeção de código (como SQL, XML e comandos) e inclusão remota de arquivos. Já o Naxsi afirma mitigar apenas as vulnerabilidades de *SQL injection* e XSS. Portanto, a escolha do WAF pode fazer uma diferença significativa na hora de proteger uma aplicação Web.

Todos os WAFs, em todos os cenários avaliados, mitigaram a vulnerabilidade de injeção de código. No caso do ModSecurity, ele possui o `slr rules`, que é um diretório de arquivos de regras que oferecem proteção contra ataques de *SQL injection*. Os arquivos contêm regras específicas para diferentes assinaturas desta vulnerabilidade. Finalmente, vale ressaltar ainda que nenhum dos WAFs possui regras específicas para evitar as vulnerabilidades de utilização de componentes com vulnerabilidades conhecidas e *Cross Site Requesty Forgery*. Como consequência, estas duas vulnerabilidades não foram mitigadas.

5.5. Considerações Finais

Este minicurso apresentou (a) os conceitos básicos de WAFs *standalone* e SaaS, (b) o estado da arte, (c) aspectos práticos de instalação, configuração e verificação de funcionamento de WAFs *standalone*, (d) o impacto destas ferramentas na latência das requisições Web, e (e) a eficácia na proteção de aplicações Web. Nos itens (c), (d) e (e), foram utilizados como referência WAFs *standalone* gratuitos, como o ModSecurity, Naxsi, Shadow Daemon e xWAF. Enquanto que a instalação e operação destes WAF é relativamente similar, com excessão do xWAF, o impacto na latência das requisições Web varia de forma significativa. No que diz respeito a eficácia em termos de proteção das aplicações Web, novamente, há uma diferença significativa entre os WAFs. Portanto, fica evidente a neces-

Tabela 5.3: Vulnerabilidades detectadas nos cenários dos *frameworks* e WAFs

WAF	***Framework***	**Vulnerabilidades detectadas**	***Web scanners***	**% V.**
ModSecurity	Laravel	Má configuração de segurança Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Zed attack proxy] [Nessus]	30%
	Symfony	Exposição de dados sensíveis Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Skipfish] [Nessus]	30%
	Codeigniter	Má configuração de segurança Exposição de dados sensíveis Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Andiparos, Grabber, Paros, Skipfish] [Uniscan, Skipfish, Vega] [Nessus]	40%
	Phalcon	*Cross-site scripting*, Vulnerabilidades conhecidas, *Cross-site requesty forgery* Má configuração de segurança Exposição de dados sensíveis	[Nessus] [Paros] [Vega]	50%
Shadow D.	Laravel	Má configuração de segurança *Cross-site scripting* Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Zed attack proxy] [Nessus, Grabber, Vega, Zed attack proxy] [Nessus]	40%
	Symfony	*Cross-site scripting*, Vulnerabilidades conhecidas, *Cross-site requesty forgery* Má configuração de segurança Exposição de dados sensíveis	[Vega, Nessus] [Andiparos, Paros, Skipfish] [Skipfish]	50%
	Codeigniter	Má configuração de segurança *Cross-site scripting* Exposição de dados sensíveis Quebra de sessão Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Andiparos, Nessus, Paros] [Paros, Vega] [Uniscan, Skipfish, Vega] [Skipfish], Quebra de controle de acesso [Nessus]	70%
	Phalcon	Má configuração de segurança Exposição de dados sensíveis Quebra de sessão, Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Paros] [Vega] [Skipfish] [Nessus]	60%
Naxsi	Laravel	*Cross-site scripting* Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Nessus, Graber, Vega] [Skipfish] [Nessus]	40%
	Symfony	*Cross-site scripting* Má configuração de segurança Exposição de dados sensíveis Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Vega, Zed attack proxy] [Andiparos, Paros, Skipfish] [Skipfish] [Nessus]	50%
	Codeigniter	Má configuração de segurança Exposição de dados sensíveis Quebra de sessão, Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Andiparos, Grabberm Paros] [Uniscan, Skipfish, Vega] [Skipfish] [Nessus]	60%
	Phalcon	*Cross-site scripting* Má configuração de segurança Exposição de dados sensíveis Quebra de sessão Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Vega, Zed attack proxy] [Paros] [Vega] [Skipfish] [Skipfish] [Nessus]	70%

sidade de investigação e avaliação dos WAFs para verificar o atendimento aos requisitos específicos da aplicação Web. Enquanto um WAF como o xWAF pode ser adequado e interessante para uma aplicação Web PHP, devido ao seu baixo impacto na latência das requisições destinadas à aplicação, ele certamente não é um WAF adequado para as demais aplicações Web, desenvolvidas nas mais variadas linguagens de programação, como JavaScript, Java, Python, Perl, C e Go.

Referências

[Acunetix 2019] Acunetix (2019). Web Application Vulnerability Report. https://bit.ly/2wh62TH.

[Cimpanu 2019] Cimpanu, C. (2019). Security bug would have allowed hackers access to Google's internal network. https://zd.net/2F8Mju8.

[Clincy and Shahriar 2018] 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.

[Convergência Digital 2016] Convergência Digital (2016). Ataques hackers provocaram um prejuízo de R$ 30 bilhões no Brasil. http://bit.do/hackers-prejuizo.

[Ferrão 2018] Ferrão, I. G. (2018). Análise black-box de ferramentas de segurança na Web. Trabalho de conlusão de curso, Curso de Ciência da Computação, Universidade Federal Do Pampa. http://dspace.unipampa.edu.br:8080/jspui/handle/riu/3695.

[Ferrão et al. 2018] Ferrão, 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 *3o Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg)*. http://arxiv.kreutz.xyz/wrseg2018_scanners_frameworks.pdf.

[Funk et al. 2018] 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*, 2(1).

[HackerOne 2019] HackerOne (2019). The HackerOne Top 10 Most Impactful and Rewarded Vulnerability Types. https://bit.ly/2RiWbGu.

[Machado et al. 2019] Machado, R., 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*. http://errc.sbc.org.br/2019/wrseg/papers/machado2019vazamentos.pdf.

[Melchior et al. 2019] Melchior, F., Kreutz, D., and Fiorenza, M. (2019). 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*. http://errc.sbc.org.br/2019/wrseg/papers/melchior2019web.pdf.

[Moosa and Alsaffar 2008] 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*, ICEGOV '08, pages 52–59, New York, NY, USA. ACM.

[Portinari 2018] Portinari, N. (2018). Ataques hackers são mais temidos por empresas que inflação e austeridade. http://bit.do/ataque-hacker.

[Rao et al. 2016] 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*, ICTCS '16, pages 24:1–24:6, New York, NY, USA. ACM.

[Razzaq et al. 2013] Razzaq, A., Hur, A., Shahbaz, S., Masood, M., and Ahmad, H. F. (2013). Critical analysis on web application firewall solutions. In *2013 IEEE Eleventh*

International Symposium on Autonomous Decentralized Systems (ISADS), pages 1–6.

[Rietz et al. 2016] Rietz, R., König, H., Ullrich, S., and Stritter, B. (2016). Firewalls for the Web 2.0. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 242–253.

[Romani 2016] Romani, B. (2016). : 6 casos de ataque hacker. https://super.abril.com.br/tecnologia/6-casos-de-ataque-hacker/.

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

[Srokosz et al. 2018] 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)*, pages 391–395.