MinimalisticWAF: Um Web Application Firewall baseado em ganchos de APIs I/O

Muryllo Pimenta de Oliveira[1], Carlo Marcelo Revoredo da Silva[2]

Universidade de Pernambuco (UPE) – Garanhuns, PE – Brasil

{muryllo.pimenta[1], marcelo.revoredo[2]}@upe.br

Abstract. *Many web servers are protected only by proxy-based firewalls and are vulnerable to attacks directed at application-layer protocols. This article discusses a standalone firewall model that is resident in the application layer requiring as few changes as possible in the implementation. Results showed that the technology contained several attacks on the actual server address from botnets looking for vulnerable database services, applications with free access to upload files and routes with broken authentications and shell code interpreters. This protection was achieved through hook techniques of write and read functions in the network socket and comparisons with the stack of discretionary access lists and request filters present in the firewall. It was therefore inferred that Minimalistic WAF provides assisting protection for applications hosted on node.js servers that suffer direct attacks on the actual address, making the application doubly protected.*

Resumo. *Muitos servidores na web são protegidos somente por firewalls baseados em proxy e são vulneráveis à ataques direcionados à protocolos da camada de aplicação. Esse artigo aborda um modelo de firewall standalone que fica residente na camada de aplicação exigindo o mínimo de alterações possível na implementação. Resultados mostraram que a tecnologia conteve diversos ataques no endereço real do servidor oriundos de botnets procurando por serviços de bancos de dados vulneráveis, aplicações com livre acesso a upload de arquivos e rotas com autenticações e interpretadores de código shell quebrados. Essa proteção foi obtida por meio de técnicas de gancho de funções de escrita e leitura no soquete de rede e comparações com a pilha de listas discricionárias de acesso e filtros de requisições presentes no firewall. Inferiu-se, portanto, que o Minimalistic WAF fornece proteção auxiliar às aplicações hospedadas em servidores node.js que sofrem ataques diretos ao endereço real, tornando a aplicação duplamente protegida.*

1. Introdução

De acordo com o relatório da WhiteHat, 32% dos ataques e vulnerabilidades exploradas são em aplicações *web* específicas, tornando a maior categoria dentre as ameaças existentes (WhiteHat, 2019). Como a maioria das aplicações vem sendo desenvolvidas para trabalhar nas nuvens desde então, o número de aplicações vulneráveis à roubo de dados e invasão de bancos de dados vem aumentando drasticamente. O último levantamento da OWASP manteve 10 vulnerabilidades mais relevantes nas aplicações atuais, dentre elas a injeção de código remoto, execução de scripts entre sites e autenticações quebradas (OWASP, 2020).

Diante do cenário hostil e desprotegido da web, faz-se necessário o uso de aplicações que diminuam as despesas com cibercrimes, previstos para atingirem até $ 6 trilhões de dólares para o ano de 2021 (MORGAN, Steve, 2019). Tais ferramentas, como os *firewalls* de aplicação são eficazes na proteção de servidores, atuando como uma camada de proteção extra que atuam quando alguma brecha da aplicação é explorada. Os WAFs *(Web Application Firewalls)* surgiram no contexto dos *firewalls* de 3° geração e foram descritos pela primeira vez por Gene Spafford, Bill Cheswick e Marcus Ranum como um *software* que atua na camada de aplicação e consegue entender os protocolos de comunicação HTTP (*Hypertext Transfer Protocol*), FTP (*File Transfer Protocol*), SSH (*Secure Shell*), entre outros, realizando bloqueios nas requisições e tornando o processo de filtragem das requisições HTTP e conexões TCP reconhecíveis e interpretáveis pelos *firewalls* em um nível mais abstrato e próximo das regras de negócio da aplicação (ROMANOFSKI, Ernest, 2002, p. 5-6).

Diante disso, se faz interesse abordar sobre a adoção dessas soluções no cenário atual, em que tantas falhas de segurança são exploradas na infraestrutura das aplicações Web. O presente estudo propõe um firewall baseado em ganchos de API como proposta para minimizar ataques em servidores Web, intitulado como *Minimalistic WAF*[1]. A estrutura do manuscrito segue com as seguintes seções: na Seção 2 são descritos os trabalhos correlatos a proposta. Na Seção 3 é apresentado uma visão geral da proposta, como detalhamento das funcionalidades e decisões arquiteturais. Na Seção 4 descreve uma prova de conceito[2] (PoC) sobre os resultados até o momento da proposta. Por fim, na Seção 5 são descritas as conclusões e destacados os trabalhos futuros.

2. Contextualização

Diante de todas as tecnologias criadas para a contenção de ataques às redes, a popularização da internet culminou no que mais tarde ficou conhecido como "*Boom da Internet*" nos anos 90, trazendo consigo diversas vulnerabilidades na camada de aplicações do modelo OSI (*Open System Interconnection*) o que consequentemente atrai a atenção dos *hackers* e traz a necessidade de novas medidas de proteção (ROMANOFSKI, Ernest, 2002, p. 3-5). As gerações de *firewalls* anteriores não levaram em consideração as vulnerabilidades e brechas que uma aplicação em específico podem ter, o que na prática, significa que o modelo de filtro de pacotes e de inspeção de estado de conexão, por exemplo, não são suficientes para conter injeções de SQL (*Structured Query Language*) à bancos de dados, injeções remotas de código, entre outras vulnerabilidades sobre o protocolo HTTP (ROMANOFSKI, Ernest, 2002, p. 3).

O *firewall* proposto neste trabalho utiliza técnicas de gancho de funções no soquete principal do Node. Os ganchos de APIs (*Application Programming Interface*) são desvios feitos em funções de um sistema com a finalidade de alterar seu comportamento de forma global ou local e notificar serviços de terceiros quando eventos específicos ocorrem. O conceito também conhecido por *API Hooking* tem sido utilizado há anos por programadores experientes na área de *Endpoint Protection* e desenvolvimento de *softwares* antivírus para os sistemas operacionais Windows (Hsu, Fu-Hau et al. 2012, p. 1-5).

[1] https://github.com/MurylloEx/Mini-WAF
[2] https://github.com/MurylloEx/Mini-WAF-Proof-Of-Concept

2.1. Conceitos sobre Firewall

Os *firewalls* são responsáveis por aplicar políticas de segurança em uma rede de computadores e prevenir o acesso não autorizado. Seu principal papel na rede é filtrar portas, restringir tráfego de dados, prevenir exploração de serviços e bloquear tentativas de acesso não autorizado por injeções de código. Assim como os antivírus, firewalls tem seus métodos de detecção e proteção em parte baseados nos ganchos de funções de entrada e saída. Embora indispensável, um *firewall* sozinho não garante a segurança de uma infraestrutura se a mesma não adota um conjunto de boas práticas de segurança (MORENO, Daniel, 2017, p. 396-428), contudo, a presença do *firewall* garante minimizar possíveis incidentes.

2.2. Trabalhos Relacionados

Os estudos [Clincy, V., & Shahriar, H., 2018] reforçam a necessidade de WAFs bem configurados para evitar falsa assimilação de segurança. Neste trabalho foram feitas simplificações e melhorias no processo de criação de regras dos WAFs e que diminuem o número de erros de regras proposta em [Ahmad, A. *et al*, 2012] nas ações disruptivas e condições dos filtros. A iniciativa de expandir a influência do CRS ModSecurity para o ambiente Node.js foi fortemente influenciada pelo trabalho de [YARI, Imrana A. *et al*, 2019] que utilizou uma abordagem de testes semelhante com o DVWA e portou o ModSecurity para o ambiente do Apache Tomcat.

Houve uma iniciativa no ano de 2016 para se criar um WAF, voltado para aplicações que utilizam o *framework express,* pelo desenvolvedor de *software* Daniel Ruf. O projeto consistia em um *firewall* acoplado ao servidor. Mais tarde, foi descontinuado sem receber mais atualizações do desenvolvedor, embora ainda possa ser acessado ao procurar pelo seu nome *node-waf*[3] no repositório do NPM (*Node Package Manager*).

Na mesma linha, no ano de 2017 o projeto *express-waf* também foi publicado tendo apresentado um caráter inovador ao sugerir a utilização do MongoDB como unidade de armazenamento das regras e filtros além de permitir a criação de módulos de terceiros com sua própria API. Em 2019 um projeto chamado *waf-js* foi publicado no repositório oficial do NPM, também como um *firewall* embora muito inferior em termos de detecção que o seu sucessor por utilizar apenas listas de palavras e expressões regulares na busca de ameaças.

A ferramenta apresentada baseia-se no modelo de regras amplamente adotado para o ModSecurity, conhecido como *Core Rule Set* (CRS) que define regras padrões para vetores de ataques na *web*. O CRS está em desenvolvimento ativo e conta com o apoio da comunidade OWASP desde 2010. O ModSecurity é um dos *firewalls* mais conceituados e que serviu como literatura base para criação das regras do *Minimalistic WAF*.

3. Visão geral do sistema

Por ter uma abordagem atuante na infraestrutura e que intercepta requisições de forma intermediaria, como um *middleware*, o *Minimalistic WAF* propõe atenuar lacunas existentes em certas soluções baseadas em nuvem. Soluções dessa natureza, a exemplo do CloudFlare, são suscetíveis a exploração através de leitura automatizada de histórico de DNS, com isso, o atacante consegue burlar esse tipo de mecanismo. A Figura 1

[3] https://www.npmjs.com/package/node-waf

ilustra o esquema de um servidor que implementa um *firewall* de aplicação como um serviço de *middleware* responsável por interceptar todo fluxo de dados de entrada e saída.

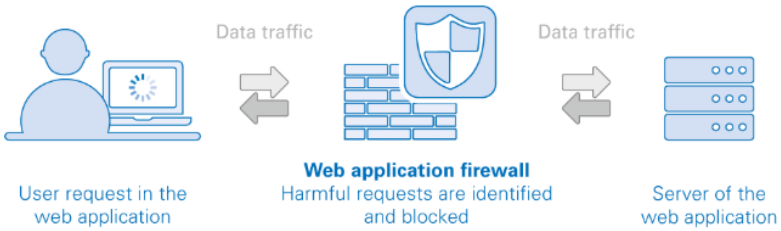

Figura 1. Representação de um esquema de Firewall de Aplicação Web entre cliente e servidor (Adaptado de WAF Service © TÜV Rheinland 2020)

Utilizando um banco de regras de acesso, é possível tomar a decisão de bloquear uma requisição HTTP(S) de acordo com o sentido do fluxo, a informação em tráfego, a origem e o método usado suportado no protocolo HTTP.

3.1. Os ganchos de APIs no Minimalistic WAF

Na linguagem de programação JavaScript, os ganchos podem ser divididos quanto à forma de desviar uma função. De acordo com a Tabela 1, os ganchos assumem 3 formas distintas de desvio baseadas em: desreferenciamento de ponteiro, salto incondicional e substituição de protótipo ou construtor de classe, respectivamente.

Tabela 1. Classificação dos ganchos de APIs utilizados na ferramenta (Fonte: Autor).

Classificação	Detalhe de implementação
Desreferenciamento de ponteiro	Substitui a referência (ponteiro) de uma função por uma *stub* contendo a função de desvio interna que é executada primeiro e retorna à execução à função original encapsulada.
Salto incondicional (função trampolim)	Substitui os primeiros 5 bytes de uma função na memória pela instrução de salto incondicional *jmp* e um ponteiro de 32 bits com o deslocamento relativo para a função que substituirá a execução da original.
Substituição de protótipo ou construtor	Um gancho simples que substitui a propriedade *prototype* de uma classe no JavaScript e permite a utilização externa do polimorfismo dinâmico.

A maioria dos ganchos aplicados são de desreferenciamento de ponteiro, visto que são simples e versáteis (KORET, Joxean & BACHAALANY, Elias, 2015, p. 174). Diferente do primeiro, há os ganchos de salto incondicional, que também tendem a apresentar problemas de condição de corrida, uma vez que para retomar a execução da função original deve-se restaurar os 5 primeiros bytes usados para o desvio (MalwareTech, 2015).

O *firewall* adota o uso de métodos *stub* que substituem as 7 principais funções de escrita no soquete do Node, permitindo que o interpretador de regras performe a varredura dos dados em tráfego e procure por tentativas de exploração detectáveis pelos filtros. O cenário ideal para reproduzir um firewall baseado em ganchos de I/O (*Input and Output*) pode ser obtido no interpretador de JavaScript assíncrono e orientado a eventos (Node.js) por facilitar a escrita dos ganchos de desreferenciamento e prototipação.

Nas próximas seções deste trabalho, métodos de proteção contra ataques de negação de serviço e XSS (*Cross Site Scripting*) utilizarão ganchos como recurso pelo interpretador de regras do *firewall*. A Figura 2 ilustra no código-fonte, um total de 7 ganchos criados nas funções de resposta a uma requisição e armazenados em uma pilha que, posteriormente, é acessível por um índice e pode ser desfeito.

```
//I/O hooks in firewall Middleware
Hooks.push(wafutils.Hook('send',   SendStub,   res));
Hooks.push(wafutils.Hook('end',    EndStub,    res));
Hooks.push(wafutils.Hook('set',    SetStub,    res));
Hooks.push(wafutils.Hook('header', HeaderStub, res));
Hooks.push(wafutils.Hook('json',   JsonStub,   res));
Hooks.push(wafutils.Hook('jsonp',  JsonpStub,  res));
Hooks.push(wafutils.Hook('write',  WriteStub,  res));
```

Figura 2. Ganchos de I/O interceptando as principais funções de escrita do framework Express.js.

Com os métodos *stub* devidamente configurados nas funções de saída, todas as operações de escrita são validadas por filtros contendo centenas de expressões regulares, muito presentes nos bancos de dados de regras de *firewall*. Diante desse cenário, o *firewall* faz uma chamada regular em cada método *stub* para uma função anônima, utilizada no desvio e contida no *middleware* que realiza a inspeção do objeto *arguments*. O objeto em questão é uma variável local e associativa, de índice base 0, utilizada para listar os argumentos passados na função I/O com a finalidade de aplicar as mesmas técnicas de inspeção por meio de listas de palavras-chave e expressões regulares utilizado nas regras de entrada de dados.

3.2. As Listas Discricionárias de Controle de Acesso (DACLs) implementadas nos ganchos de I/O

As DACLs são a primeira camada de proteção que compõe uma regra de *firewall* no Mini-WAF pois bloqueiam ou permitem um acesso à API conforme algumas informações presentes na requisição HTTP. Uma DACL é composta por pelo menos 9 campos, são eles: a camada de rede usada; o tipo de correspondência esperada; a ação de gerenciamento; a direção da DACL; o método HTTP procurado; o endereço Ipv4 ou Ipv6; o cabeçalho de agente de usuário e finalmente a descrição da DACL.

Seu funcionamento é semelhante, em termos conceituais, às DACLs dos sistemas operacionais baseados no NT Kernel 6.0 ou superior, utilizando-se dos métodos, endereços de IP e agentes de usuário para autorizar ou negar um acesso à API, tornando-os, assim, os grupos alvos da lista discricionária de controle de acesso. A camada de rede é usada para especificar o protocolo de IP utilizado, caso Ipv4 ou Ipv6, conforme a necessidade do administrador do *firewall*. Em alguns servidores, determinados métodos são perigosos e costumam ser bloqueados, tais como o método DELETE utilizado em várias aplicações para excluir um recurso.

Em certos casos, o uso de filtros é dispensável caso a necessidade do administrador do *firewall* seja somente bloquear o acesso de uma faixa de IPs de um determinado país ou um método específico, além de também poder bloquear cabeçalhos de agente de usuário possivelmente gerados por *botnets* (redes de computadores utilizadas para atacar outros servidores disponíveis na web).

3.3. Os filtros implementados nos ganchos de I/O

Diferentemente dos filtros tradicionais empregados nos primeiros modelos de *firewall*, como o filtro de pacotes mencionado anteriormente, os filtros no nível de aplicação visam analisar o tráfego de dados em alto nível para conter possíveis combinações maliciosas. A exemplo do filtro de carga-útil ou *payload* de uma requisição HTTP, utiliza-se expressões regulares na tentativa de detectar injeções de SQL, semelhante à sanitização de entradas oferecidas por bibliotecas JavaScript como o lodash. O princípio de funcionamento do filtro da ferramenta é descrito nas seguintes etapas:

1. Os filtros são carregados para a memória a partir de um arquivo em formato JSON (*JavaScript Object Notation*);
2. O *firewall* define identificadores exclusivos para cada regra;
3. O *firewall* inicializa os ganchos nas funções de escrita e instala o *Middleware* nas funções de entrada;
4. Quando interceptada a requisição, as regras são avaliadas primeiro pelas DACLs e depois pelos filtros;
5. Se a requisição for interceptada e bloqueada por alguma DACL ou filtro, não há resposta, somente o código 403 (*Forbbiden*);
6. Caso a requisição seja aceita implícita ou explicitamente pelo *firewall*, as operações de escrita são validadas pelos ganchos de I/O. Em caso de inadequações, a resposta é bloqueada, todo o *buffer* de saída é liberado e o código retornado é 403 (*Forbbiden*).

4. Prova de Conceito (*Proof of Concept* - PoC)

Como prova de conceito, é proposto um ambiente de testes[4] para simular uma aplicação *web* vulnerável à diversos ataques utilizando o conhecido DVWA[5]. O *firewall* foi instalado em uma aplicação pequena escutando a porta padrão do protocolo HTTP, servindo como um proxy na rede interna para o serviço do Apache que escuta na porta 8081 conforme mostra o esquema ilustrado na Figura 3.

Figura 3. Esquema de atuação do Mini-WAF na proteção da aplicação DVWA.

A realização dos testes é feita no sistema operacional pré-configurado e disponível em formato de imagem de máquina virtual para o *software* Oracle VM VirtualBox diretamente no repositório onde consta o manual de instalação junto com capturas de telas indicando o fluxo de configuração do ambiente. O sistema operacional utilizado (Lubuntu LXQt 20.04) vem com um *script* de linha de comando chamado *poc-setup.sh* que realiza o *download* e a instalação da ferramenta e todas as suas dependências em suas versões mais atualizadas. O ambiente de testes em execução necessita apenas de um computador

[4] https://github.com/MurylloEx/Mini-WAF-Proof-Of-Concept#readme
[5] http://www.dvwa.co.uk/

com acesso à *internet*, memória RAM de no mínimo 1 GB livres e o Oracle VM VirtualBox instalado para a importação do arquivo OVF (imagem do sistema).

4.1 Manual de instalação e licença da ferramenta

A ferramenta pode ser obtida por meio da instalação num projeto do node.js e está presente no repositório oficial do NPM com o nome de pacote mini-waf[6]. Com um simples guia de aplicação na página do projeto é possível aplica-lo em aplicações web com menos de 50 linhas de código. O projeto utiliza a licença MIT e é disponibilizado como um *software* livre e de código aberto, permitindo qualquer desenvolvedor realizar alterações. Atualmente, o manual de instalação com a licença e o ambiente de testes estão disponíveis no GitHub sob o nome de repositório "Mini WAF Proof Of Concept".

4.2 Resultados em ambiente real

O Mini-WAF foi implantado em um servidor mantido em execução por 3 meses e recebeu mais de 30 atualizações ao longo do tempo de funcionamento. Diante do cenário, o servidor foi atacado diariamente, sendo a maioria deles originados por *botnets* russas, chinesas, alemãs e estadunidenses. A maior quantidade dos ataques sofridos veio dos endereços de IPs russos: 5.101.0.209 e 195.54.160.135. A maior parcela dos ataques sofridos no servidor foi originada pela *botnet* ilustrada na Figura 4.

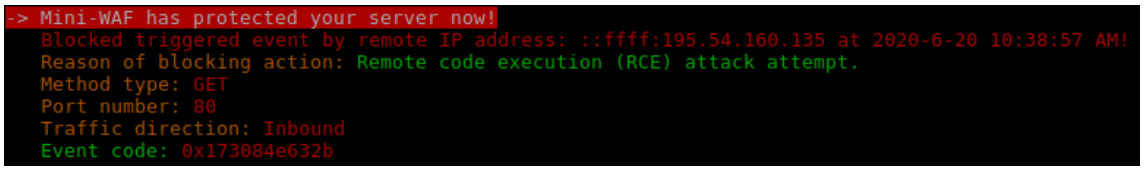

Figura 4. Tentativa de execução remota de código bloqueada pelo Mini-WAF.

Em média com mais de 8 tentativas por dia, ela busca por endereços com as portas 80 e 443 disponíveis para tentar possíveis injeções de código em aplicações comprometidas do ThinkPHP (MORENO, Daniel, 2017, p. 316). É importante salientar que todas as tentativas de acesso não autorizado por esse endereço foram bloqueadas e como medida adicional o endereço de IP foi denunciado no banco de dados público AbuseIPDB[7].

4.3 Vantagens, desvantagens e limitações sobre soluções semelhantes

A ferramenta propõe facilidade de uso, versatilidade entre serviços Node.js e proteção adicional contra inclusão de arquivos no *express*, bem como travessia de diretório presentes em versões anteriores em que soluções similares como o waf-js e node-waf não são capazes de atuar, possuem baixa taxa de detecção ou uso complexo. O Mini-WAF possui um banco de regras em desenvolvimento, além de suporte oficial somente ao Node.js, inicialmente, tornando-o limitado a essa tecnologia. Embora seja extremamente eficaz, é de conhecimento público que os *firewalls* sozinhos são inócuos no ponto de vista da segurança quando não são acompanhados pelas melhores práticas de programação e desenvolvimento seguro de aplicações apontados pela OWASP.

[6] https://www.npmjs.com/package/mini-waf
[7] https://www.abuseipdb.com/

5. Conclusões e trabalhos futuros

Os *firewalls* de aplicação *web* baseados em *proxy* são eficientes na contenção de ataques contra servidores através do nome de domínio, à exemplo do CloudFlare, um dos maiores provedores de redes de distribuição de conteúdo e WAFs do mundo. Entretanto, esse tipo de proteção não é suficiente para proteger servidores integralmente. Resultados no ambiente de testes atuaram como prova de conceito da eficácia do *Minimalistic WAF* ao proteger um servidor que teve seu endereço de IP revelado por ferramentas que consultam os históricos DNS e fazem varreduras em nomes de domínio mal configurados em busca do endereço de IP real do servidor.

O *firewall* conteve diversos ataques em 3 meses de pleno funcionamento em cenário real num dispositivo IoT (Tinkerboard R/BR) hospedando um servidor Apache com uma aplicação PHP vulnerável à injeções de SQL, identificando e bloqueando diversas conexões de *botnets* russas, chinesas e alemãs que realizavam tentativa de injeções remotas de código e comandos SQL. Recomenda-se que os administradores de redes com servidores que utilizam serviços de WAF como *proxy* preocupem-se também em utilizar *firewalls* embarcados na aplicação para uma maior proteção de seus servidores de origem. Como trabalhos futuros, pretende-se estender o Mini-WAF para outros *frameworks* por meio de *drivers* do *SpringBoot* e FastAPI no Java e Python.

Referências

Ahmad, A, *et al* (2012). "Formal reasoning of web application Firewall rules through ontological modeling", July.

Clincy, V., & Shahriar, H. (2018). "Web Application Firewall: Network Security Models and Configuration". July.

Ernest Romanofski. (2002) "A Comparison of Packet Filtering vs Application Level Firewall Technology", SANS Institute, June, p. 1-6.

Hsu, F.-H. et al (2012) "Antivirus Software Shield Against Antivirus Terminators". IEEE Transactions on Information Forensics and Security, June, p. 1-5.

KORET, Joxean & BACHAALANY, Elias. (2015) "The Antivirus Hacker's Handbook", June, p. 174.

MalwareTech. (2015) "Inline Hooking for Programmers (Part 1: Introduction)", June, available at: https://www.malwaretech.com/2015/01/inline-hooking-for-programmers-part-1.html

MORENO, Daniel. (2017) "Pentest em Aplicações Web", 1th edition, June, p. 316-428.

OWASP Foundation, Inc. (2020) "OWASP Top Ten", June, available at: https://owasp.org/www-project-top-ten/.

Rheinland, TÜV. (2020) "Web Application Firewall (WAF) Service", June, available at: https://www.tuv.com/world/en/web-application-firewall.html

Steve Morgan. (2019) "2019 Official Annual Cybercrime Report", Herjavec Group, June.

WhiteHat Security. (2019) "Application Security Statistics Report", vol. 14, June, p. 2.

YARI, Imrana A. *et al*. (2019) "Towards a Framework of Configuring and Evaluating ModSecurity WAF on Tomcat and Apache Web Servers", ICECCO, July.