jShield: Uma Solução Open Source para Segurança de Aplicações Web

Márcio A. Macêdo[2], Ricardo G. Queiroz[1], Julio C. Damasceno[2]

[1]Centro de Ensino Unificado de Teresina (CEUT)
Av. dos Expedicionários, 790 – 64.046-700 – Teresina – PI – Brasil

[2]Centro de Informática (Cin) - Universidade Federal de Pernambuco (UFPE)
Cidade Universitária – 50.740-540 – Recife – PE – Brasil

mam5@cin.ufpe.br, ricardoqueiroz@ieee.org, jcd@cin.ufpe.br

Abstract. *The purpose of this article is describe a solution for web application security based on an application firewall which is implemented using a model called reverse proxy. The application firewall use as much the model of negative filter as the model of positive filter for, apart from providing applications with more security, avoiding the occurrence of false positive what could filter applications authentic packets. This way, the firewall aggregates both the easy configuration of the negative security model with the better security provided by positive model.*

Resumo. *O propósito deste artigo é descrever uma solução para segurança de aplicações web baseada em um firewall de aplicações que é implementado usando o modelo de proxy reverso. O firewall de aplicações proposto emprega tanto o modelo de filtragem negativo, quanto o modelo de filtragem positivo, para além de prover as aplicações com mais segurança, evitar a ocorrência de falsos-positivos, o que poderia filtrar pacotes autênticos das aplicações. Dessa forma, o firewall agrega de forma eficiente a facilidade de configuração do modelo de segurança negativo com a melhor segurança provida pelo modelo positivo.*

1. Introdução

A proteção de servidores de aplicações *web*, permitindo acesso às aplicações somente por pessoas autorizadas, é uma necessidade global. Isso se deve a explosão do desenvolvimento de aplicações *web*, pois programadores, empresas e corporações têm observado que a Internet é atualmente um dos meios mais importantes para se fazer comércio e isso provocou o desaparecimento de fronteiras para os negócios. Contudo, devido ao crescimento exponencial desse mercado, empresas têm contratado mão de obra inexperiente, e têm recorrido a soluções de desenvolvimento fornecidas por diversos *frameworks*, que muitas vezes escondem vários problemas de segurança.

Um dos grandes problemas que estas empresas vêm enfrentando é o crescimento dos ataques que exploram vulnerabilidades presentes em aplicações *web*. Essas vulnerabilidades em questão são geradas durante o processo de desenvolvimento do software e colocam as aplicações e conseqüentemente as empresas em risco, podendo levar ao vazamento de informações privilegiadas, o que pode acarretar no comprometimento parcial ou total do modelo de negócio.

Apesar das dificuldades no tratamento de falhas de aplicações, devido ao surgimento de novas vulnerabilidades e da diversificação das técnicas de ataques a camada de aplicação [CERT.BR 2009], algumas empresas têm adotado possíveis soluções para esse problema como: o investimento em treinamento de desenvolvimento de código seguro para suas equipes, a utilização de um processo de software confiável e baseado em testes de segurança e a utilização de IDS (*Intrusion Detection Systems*) [Bace and Mell 2001]. Porém, nenhuma dessas medidas ataca o problema de forma totalmente eficaz ou pelo menos dá ao desenvolvedor a garantia de proteção da sua aplicação a médio/longo prazo.

A solução apresentada neste artigo, é implementada através de um *firewall* de aplicações que trabalha diretamente na camada de aplicação, como um agente detector de padrões de ataques que pode encaminhar o tráfego suspeito para *honeypots*, além de funcionar como ponto único de verificação através de um *proxy* reverso e filtrar os possíveis pacotes que possam apresentar ameaças à aplicação, criando uma camada de filtragem de forma segura e confiável.

2. Web Application Firewall

De acordo com [WAFEC 2006], *Web Application Firewall* (WAF) é uma nova tecnologia de segurança que tem o papel de proteger aplicações *web* de ataques. As soluções de um WAF são capazes de prevenir ou até mesmo neutralizar um ataque a uma aplicação, conseguindo filtrar de forma eficiente o que um IDS (*Intrusion Detection Systems*) e *firewalls* de rede não conseguem. Isso se deve ao fato de um WAF agir diretamente na camada de aplicação, filtrando os dados e principalmente parâmetros utilizados em uma transação HTTP [Jones, Bejtlich and Rose 2006].

2.1. Implementações de um WAF

Os WAFs podem ser implementados de três formas diferentes, cada uma possuindo vantagens e desvantagens. Segundo [WAFEC 2006], as formas de implementação de um WAF são:

- No nível da camada de rede.
- Através de um *proxy* reverso.
- Diretamente no servidor web.

2.2. Técnicas de detecção de ataques de um WAF

De acordo com [WAFEC 2006], os *firewalls* de aplicações Web podem ser configurados para detectar ataques de duas diferentes formas:

- Modelo de segurança negativo.
- Modelo de segurança positivo.

2.2.1. Modelo de segurança negativo

O modelo de segurança negativo é simples de ser configurado e tem por padrão permitir o tráfego de todos os pacotes de solicitação, filtrando somente os que obedecem a alguma assinatura ou regra do WAF (*black list*). O sucesso dessa implementação é determinado pela eficiência com que o *firewall* de aplicação Web consegue detectar solicitações nocivas, de acordo com sua base de regras e assinaturas.

O problema em implementar o modelo de segurança negativo, reside no risco do banco de dados de regras filtrar um grande número de falso-positivos (quando a regra filtra pacotes autênticos da aplicação) e por esse modelo ser mais suscetível a técnicas de evasão.

A vantagem de se usar o modelo de segurança negativo, se deve a facilidade de configuração que o mesmo proporciona, pois serão criadas regras e assinaturas de ataques que serão aplicadas a todas as requisições.

2.2.2. Modelo de segurança positivo

O modelo de segurança positivo é complexo considerando sua configuração e implementação. Por padrão ele bloqueia todo e qualquer tráfego e permite somente os pacotes de solicitação que respeitem a algumas regras que garantem que a solicitação é segura para a aplicação (*white list*). Essa forma de configuração é mais segura e eficiente, pois necessita de menos regras de segurança para filtragem.

O problema em configurar um *firewall* de aplicação com esse modelo se deve a grande necessidade de conhecimento sobre a aplicação e, a partir desse conhecimento, julgar o que é nocivo ou não, e assim, criar uma regra totalmente personalizada para proteger a aplicação de ataques.

3. Ataques a Aplicações Web

De acordo com [Ceron *et al.* 2008], os ataques contra aplicações *web* representam uma parcela considerável dos incidentes de segurança ocorridos nos últimos anos. Isso vem ocorrendo pela falta da devida preocupação com requisitos de segurança nessas aplicações.

Esse é um dos principais motivos que explicam o fato da Internet ter se tornado um ambiente repleto de vulnerabilidades e meio para frequentes ataques. O problema se agrava mais ainda com a utilização dos mecanismos de busca como uma poderosa ferramenta para localizar sites e sistemas vulneráveis.

Segundo o instituto de pesquisa [OWASP 2010] as 10 (dez) vulnerabilidades mais críticas em aplicações *web* em ordem são:

- Falhas de Injeção (SQL *Injection*, PHP *Injection*, etc);
- *Cross* Site *Scripting* (XSS Refletido/Persistido);
- Quebra de Autenticação e Gerenciamento de Sessão;
- Referência Insegura Direta a Objetos;
- *Cross Site Request Forgery* (CSRF);
- Má Configuração de Segurança;
- Armazenamento Criptográfico Inseguro;
- Falha de Restrição de Acesso à URL.
- Proteção Insuficiente a Camada de Transporte;
- Redirecionamentos e Encaminhamentos Inválidos;

Das vulnerabilidades citadas acima, as mais graves (as 5 primeiras) ocorrem por erros na etapa de desenvolvimento da aplicação. Atualmente uma das soluções adotadas para defesa contra esses ataques, são a revisão de código e a adoção de metodologias e

práticas de desenvolvimento de código seguro como, por exemplo, a centralização da verificação dos dados de entrada. O problema dessa solução se encontra no alto custo de manutenção e treinamento para capacitar a equipe em desenvolvimento de códigos seguros, além da necessidade do gerenciamento constante da segurança da aplicação.

4. Trabalhos Relacionados

4.1. ModSecurity

O WAF ModSecurity [ModSecurity 2009] é o mais conhecido e utilizado na Internet. Ele é um projeto *Open Source* e é mantido atualmente pela Breach Security, uma empresa que também vende soluções comerciais. "O ModSecurity é um *firewall* de aplicações *web* que funciona interligado ao *container* Apache como um módulo acoplado, e trabalha de duas formas, diretamente no servidor ou como um *proxy* reverso. Ele fornece proteção contra uma série de ataques a aplicações *web* e permite a monitoração do tráfego HTTP, registro e análise em tempo real" [ModSecurity 2009].

O ModSecurity geralmente usa o modelo de segurança negativo e, por ser o mais conhecido e também o primeiro *firewall* de aplicações *Open Source*, o mesmo possui vários outros projetos relacionados, também *Open Source*, que ajudam a melhorar essa solução. Dentre eles podemos citar:

- ***ModSecurity Core Rules*** - são um conjunto de regras para o modelo negativo que irão detectar os ataques *web* mais comuns e conhecidos. Essas regras são um excelente ponto de partida para os iniciantes do ModSecurity.
- ***ModSecurity Console*** - cria um controle centralizado para as instâncias do ModSecurity em uma rede de forma individual para geração dos relatórios de *logs* e alertas.
- ***ModSecurity Profiler*** - analisa o tráfego de aplicações *web* e cria perfis da aplicação de forma semântica. Esses perfis geram regras, que poderão ser usadas para facilitar a implementação de um modelo de segurança positivo.

4.2. UrlScan

Outra solução de WAF não comercial existente é o URLScan [Microsoft 2009] para o *container* de aplicações *web* IIS (*Internet Information Server*) da Microsoft. O "UrlScan v3.1 é um filtro ISAPI – *Internet Server Application Programming Interface* – que lê a configuração de um arquivo chamado Urlscan.ini que possuirá regras para impedir que certos tipos de solicitações (enumeradas no Urlscan.ini) sejam executadas pelo IIS" [IISteam 2009].

O URLScan é uma solução que os usuários de Microsoft IIS devem considerar fortemente, mesmo que possuam outro *firewall* de aplicações *web* instalado, porque, primeiramente, é gratuito e, segundo, a aplicação roda no servidor *web* em si e, portanto, pode ser usado como mais uma camada de segurança e proteção.

4.3. WebKnight

WebKnight [Aqtronix 2009] é um *firewall* de aplicação para o container IIS da Microsoft e outros servidores *web* que dão suporte a ISAPI. Ele é uma ferramenta *Open Source* e é liberado sob a licença GNU *General Public License*. Em outras palavras, o

WebKnight é um filtro ISAPI que protege o servidor *web*, bloqueando algumas solicitações consideradas nocivas. Se um alerta é acionado, o WebKnight irá assumir e proteger o servidor *web*.

Ele analisa todas as requisições e toma decisões baseando-se em suas regras de filtragem, que são definidas pelo administrador. Estas regras não são baseadas em um banco de dados de assinaturas de ataque que precisam de atualizações constantes. Ao invés disso, ele utiliza filtros contra *buffer overflow*, injeção de SQL, *directory transversal*, codificação de caracteres e outros ataques. Desta forma, o WebKnight irá proteger o servidor contra os ataques conhecidos e desconhecidos.

Por ser um filtro ISAPI, ele tem a vantagem de trabalhar em estreita colaboração com o servidor *web*, desta forma ele pode fazer mais do que outros *firewalls* de aplicação *web* (WAF) e sistemas de detecção de intrusão (IDS), como por exemplo, verificação de tráfego criptografado.

5. jShield – *Web Application Firewall*

O jShield é um *firewall* de aplicações desenvolvido na linguagem de programação Java. Um dos motivos para escolha desta linguagem foi pelo ótimo suporte que Java oferece ao gerenciamento do protocolo HTTP, além da necessidade de um *proxy* reverso robusto e estável para os *containers* que rodam aplicações desenvolvidas nessa linguagem como, por exemplo, o Tomcat da *Apache Software Foundation*.

Ele funciona através de um *proxy* reverso, podendo utilizar tanto o modelo de filtragem negativo baseado em regras de *black list* (configuração padrão), como também o modelo de filtragem positivo utilizando regras específicas para cada tipo de requisição, baseando-se em regras de *white list* (deve ser configurado). O jShield trabalha com um ponto único de verificação, que tem como função receber todas as solicitações HTTP direcionadas aos servidores *web* e filtrar pacotes potencialmente nocivos às aplicações, redirecionando esse tráfego de pacotes suspeitos para os *honeypots* a fim de documentar o ataque.

Além disso, o jShield também faz a verificação dos dados enviados da aplicação para o cliente, onde ele filtra os dados da transação para evitar que a aplicação ataque o cliente. Um exemplo que podemos citar são os ataques *cross-site scripting* persistidos.

Primeiramente, o *firewall* de rede recebe todas as solicitações HTTP provenientes da internet que foram endereçadas para os servidores *web,* e redireciona essas solicitações para o jShield.

Após receber os pacotes, o jShield normaliza os dados da transação a fim de evitar tentativas de evasão, ao mesmo tempo que verifica os parâmetros e métodos da solicitação HTTP em busca de assinaturas ou padrões de ataques.

Depois de receber e analisar cada pacote do tráfego comum repassado pelo *firewall* de rede, o jShield decide se o pacote é nocivo ou não para a aplicação *web*. Caso a solicitação não seja nociva, o jShield encaminha os pacotes para os servidores *web* de forma totalmente transparente para a aplicação.

Caso a solicitação seja nociva, o jShield pode simplesmente descartar o pacote sem encaminhar a solicitação do atacante para nenhum local abortando a conexão, outra possibilidade, seria redirecioná-lo para uma página de acesso negado ou encaminhar as

solicitações do atacante para um aplicativo *honeypot*, que irá receber e documentar toda a tentativa de ataque, para podermos posteriormente analisar e gerar novas regras e assinaturas de ataques com base nas inferências realizadas nesses *logs*.

Outra característica interessante do jShield é que o mesmo implementa um algoritmo que ativa subclasses de regras referente às grandes categorias de ataques baseadas no top 10 de ataques a aplicações web [OWASP 2010], aumentando assim a capacidade da ferramenta em detectar e prevenir ataques nessas categorias. A figura abaixo (Figura 1) demonstra o funcionamento do jShield:

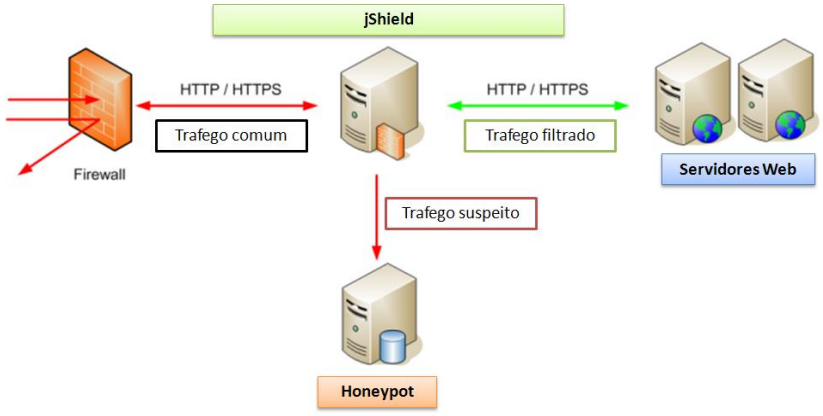

Figura 1. Esquema de gerenciamento de tráfego e filtragem do jShield.

5.1. Configuração do jshield

O jShield possui dois arquivos de configuração, no qual em um deles é configurado o *proxy* reverso e no outro o firewall de aplicações *web*. A seguir será definido como configurar cada uma dessas instâncias.

5.1.1. Configuração do *proxy* reverso

A configuração do *proxy* reverso do jShield é feita de forma simples e prática através de arquivos XML. A ferramenta possui várias funcionalidades que serão listadas e especificadas a seguir. O arquivo principal de configuração das funcionalidades é o data.xml, nesse arquivo o usuário especifica as funcionalidades que prefere ativar no *proxy* reverso. A imagem abaixo (Figura 2) demonstra o arquivo data.xml:

```
1<?xml version="1.0" encoding="UTF-8"?>
2
3<config>
4    <server
5    className="br.com.jshield.servers.BaseServer"
6    domainName="www.ceut.com.br"
7    isRewriting="true">
8        <rule className="br.com.jshield.rules.AcceptEverythingRule" />
9    </server>
10</config>
```

Figura 2. Exemplo de configuração do data.xml

No arquivo de configuração, o usuário define a classe de configuração do servidor de *proxy* reverso que será responsável pelo encaminhamento dos dados, o

endereço IP ou nome do *website* que deverá receber as solicitações encaminhadas e a classe de uma regra específica para o contexto.

A classe BaseServer configura o *proxy* reverso do jShield para simplesmente receber e encaminhar as solicitações dos usuários para o *website* destino configurado. Por outro lado, o *proxy* reverso do jShield possui varias classes de regras para configurar a ferramenta, dentre elas podemos citar:

- **AcceptEverythingRule:** com essa regra o servidor irá aceitar e encaminhar todas as transações que receber.
- **CompositeRule:** essa classe é usada para combinar duas ou mais regras para serem aplicadas no servidor de *proxy* reverso.
- **DirectoryRule:** essa regra é usada para especificar um diretório de acesso único, no qual as transações só serão aceitas caso sejam encaminhadas para esse diretório. Essa regra também pode ser usada para mapear outros *websites* nesses diretórios, podendo dessa forma, mapear vários *websites* para proteção.
- **IPRule:** com essa regra o servidor só aceitará transações especificas de uma lista de endereços IP pré-estabelecida.
- **TimeRule:** essa regra configura o servidor para aceitar transações somente em um determinado intervalo de tempo (Ex: das 06:00h às 18:00h).

5.1.2. Configuração do web application firewall

A configuração do *firewall* de aplicações também é feita através de arquivos XML, o arquivo que contém as regras e a configuração do *firewall* de aplicações é o rules.xml. Nesse arquivo, o usuário irá configurar o modo de filtragem que o *firewall* de aplicações deverá funcionar, e também especificar as regras de filtragem. A Figura 3 demonstra um exemplo de configuração com modo negativo para o rules.xml e a Figura 4 demonstra um exemplo de configuração com o modo positivo.

```
1<?xml version="1.0" encoding="UTF-8"?>
2
3<jShieldConfig mode="negative">
4    <rule filter=".*select.+from.*" exception="Tentativa de ataque detectada: Sql Injection" />
5    <rule filter=".*union.+all.*" exception="Tentativa de ataque detectada: Sql Injection" />
6    <rule filter=".*union.+select.*" exception="Tentativa de ataque detectada: Sql Injection" />
7    <rule filter=".*having.+1=1.*" exception="Tentativa de ataque detectada: Sql Injection" />
8    <rule filter=".*drop.+database.*" exception="Tentativa de ataque detectada: Sql Injection" />
9</jShieldConfig>
10
```

Figura 3. Exemplo de configuração do rules.xml para o modo negativo

```
1<?xml version="1.0" encoding="UTF-8"?>
2<jShieldConfig mode="positive">
3    <rule id="1" page="noticia.php">
4        <parameter name="id" regex="[0-9]+" />
5    </rule>
6    <rule id="2" page="artigo.php">
7        <parameter name="id" regex="[0-9]+" />
8    </rule>
9</jShieldConfig>
10
```

Figura 4. Exemplo de configuração do rules.xml para o modo positivo

5.2. Vantagens e desvantagens de se utilizar o jShield

Por ser uma aplicação composta de um esquema de *proxy* reverso e *firewall* de aplicações, o jShield possui várias vantagens frente a outros *firewalls* de aplicação, dentre essas vantagens podemos citar:

- Funciona em qualquer container HTTP que forneça suporte a Java, como por exemplo, o *container* Tomcat da *Apache Software Foundation*.
- Além de esconder toda a estrutura interna dos servidores *web* por funcionar através de um *proxy* reverso, o jShield proporciona a possibilidade da utilização de balanceamento de cargas e *clustering* de servidores de forma simples e escalável caso seja instalado no Tomcat, pois o mesmo pode utilizar todas as funcionalidades do *container* em que foi instalado.
- Simples configuração através de arquivos XML.
- Auto-reconfiguração de regras e proteção da aplicação, utilizando um algoritmo sensível ao contexto que ativa temporariamente subcategorias de filtros para melhorar a defesa do *firewall* de aplicações frente à variedade de ataques e diminuir gargalos nas transações efetivadas na aplicação.
- Minimiza a necessidade da equipe de desenvolvimento ter que se preocupar exaustivamente com a segurança da aplicação e, assim, poder manter o foco somente no modelo de negócio, pois o jShield se encarrega de manter a aplicação segura.

E por utilizar o esquema de *proxy* reverso o jShield possui as seguintes desvantagens:

- Leve aumento no gargalo da rede.
- Caso haja um único servidor jShield na rede, torna-se um ponto de falha que merece atenção especial, pois caso o jShield falhe, todas as aplicações *web* também deixarão de se comunicar via Internet.
- Necessidade de configuração do encaminhamento das conexões HTTP para o jShield.

6. Comparação com Outras Ferramentas

Para validar a nossa pesquisa e a nossa solução, foram feitas várias comparações de funcionalidades, requisitos e atributos entre o jShield e os trabalhos relacionados citados nesse trabalho, que propõem o mesmo objetivo: proteger as aplicações *web* de ataques. Para fazer o levantamento desses dados, utilizamos a documentação e os manuais de todas as ferramentas que foram utilizadas nessa comparação. A Tabela 1 demonstra a comparação executada entre as principais funcionalidades dessas ferramentas.

Após analisarmos a tabela de comparação das principais funcionalidades dessas soluções, chegamos a conclusão que o jShield possui várias vantagens sobre as outras ferramentas comparadas, destacando-se principalmente nos quesitos anti-evasão, caching e clustering.

Tabela 1. Tabela comparativa entre soluções WAF.

	Modo Positivo	Modo Negativo	Anti-Evasão	Caching	Clustering
jShield WAF	X	X	X	X	X
ModSecurity	X	X			X
WebKnight		X			
UrlScan		X			

6.1. Análise de desempenho do jShield

Além da comparação de funcionalidades com outras ferramentas, também foram feitos vários testes de carga e estresse no jShield. O objetivo desses testes foi medir o desempenho do jShield (vazão), o número de usuários simultâneos que o mesmo suporta (escalabilidade), e também a perda de velocidade que ocorre na transação que passa pelo jShield (latência). Para executar esses testes utilizamos a ferramenta Apache JMeter [JMeter 2009].

O Apache JMeter, é uma aplicação *desktop* desenvolvida na linguagem de programação Java, e é designada para a realização de testes de carga em aplicações cliente/servidor. Ele pode ser usado para simular cargas de trabalho em um servidor, rede, aplicações ou mesmo em objetos. O seu desenvolvedor original foi Stefanno Mazzochi, membro da *Apache Software Foundation*, mas hoje a ferramenta, que é *Open Source*, é resultado do trabalho de milhares de pessoas [JMeter 2009].

Para efetuar os testes de desempenho, utilizamos o *website* da faculdade CEUT (http://www.ceut.com.br/). Os testes foram divididos em três etapas, das quais o itinerário do teste era acessar a página e fazer a solicitação de uma notícia no banco de dados, sempre visando o pior caso. No primeiro caso, foram utilizados 10 usuários acessando a aplicação simultaneamente. No segundo caso, foram utilizados 20 usuários simultâneos. E por final, no terceiro caso, foram utilizados 40 usuários simultâneos acessando a aplicação. Cada etapa do teste foi repetida 1000 vezes, e com os resultados das mesmas foi feito uma média ponderada para chegar aos valores finais obtidos.

Os testes foram realizados em uma máquina com processador dual core, 2 Gb de RAM e uma conexão com a internet do tipo 3G com a velocidade de 128 Kbps. Na maquina foi instalado e configurado um Apache Tomcat com o jShield devidamente configurado para o modelo de filtragem negativo contendo 30 regras de filtragem em sua *black list*. Como já mencionado, foi configurado para o jShield encaminhar todo tráfego recebido para o *website* da faculdade CEUT. Os Gráficos 1 e 2 demonstram a interpretação dos dados obtidos dos testes realizados.

Com os resultados do primeiro teste (ver Gráfico 1), podemos observar que com 10 usuários simultâneos, o tempo de resposta através do jShield foi levemente superior ao tráfego comum (quando a solicitação não passa pelo jShield), pois a média por solicitações foi aproximadamente 5 segundos mais eficiente que a média por solicitações do tráfego comum. Porém, a vazão de dados do tráfego com jShield, obteve aproximadamente 4 operações a menos por minuto que o tráfego comum (Gráfico 2).

No segundo resultado (ver Gráfico 1), observamos que com 20 usuários simultâneos, o tempo de resposta através do jShield foi muito superior ao tráfego

comum, pois a média por solicitações, foi aproximadamente 27 segundos mais eficiente que a média por solicitações do tráfego comum. Já a vazão dos dados foi aproximadamente 0,3 operação por minuto inferior ao tráfego comum (ver Gráfico 2), portanto, foram considerados tecnicamente iguais.

E por final, no terceiro caso (ver Gráfico 1), observamos que com 40 usuários simultâneos, o tempo de resposta através do jShield também foi considerado muito superior ao tráfego comum, pois a média por solicitações foi aproximadamente 25 segundos mais eficiente que a média do tráfego comum. No entanto, a vazão dos dados foi aproximadamente 0,4 operação por minuto inferior ao tráfego comum (ver Gráfico 2), portanto, mais uma vez, considerados tecnicamente iguais.

Então, com os resultados obtidos, podemos concluir que as solicitações sendo encaminhadas pelo jShield, além de serem mais seguras, são mais eficientes quando se trata de tempo de resposta, e não causam prejuízos consideráveis ao número de operações por minuto da aplicação que está sendo protegida. Isto se deve pelo jShield possuir um algoritmo de *cache*, o que ajudou bastante nos resultados dos testes.

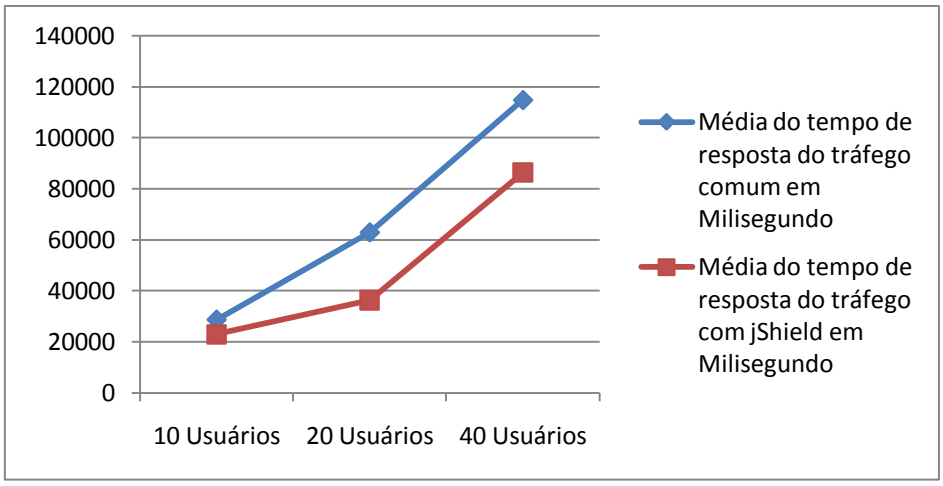

Gráfico 1. Representação gráfica do tempo de resposta.

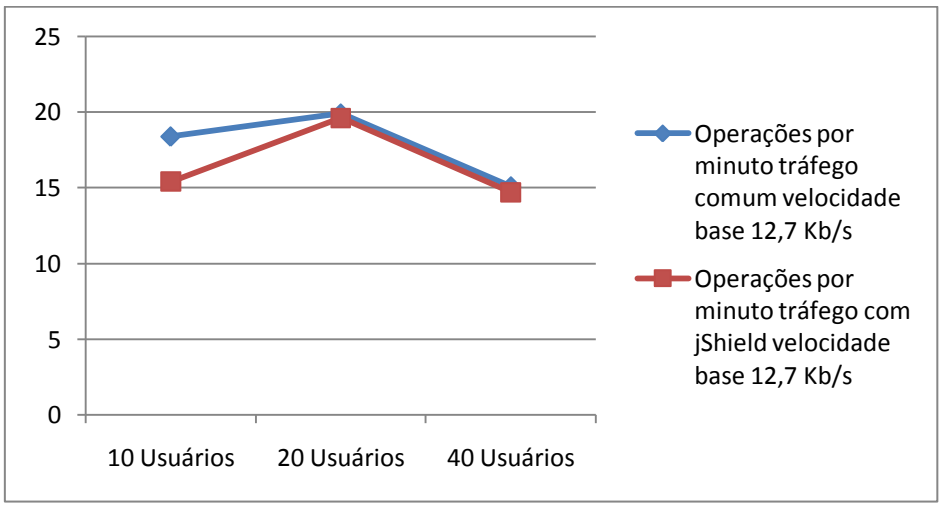

Gráfico 2. Representação gráfica de operações por minuto.

7. Trabalhos Futuros

Apesar de muito ter sido implementado e a idéia principal da solução estar funcionando e pronta para entrar em produção, existem várias outras tarefas e idéias que deixamos para segundo plano e podem ser implementadas para somar a solução, dentre elas podemos citar:

- Implementação de uma GUI (*Graphic User Interface*) para ferramenta na qual poderia facilitar ainda mais a configuração da ferramenta, como também, gerar relatórios estatísticos baseados nos *logs* de detecção do jShield.
- Inserir um algoritmo que envia SMS ou mensagens de e-mails para uma determinada lista de pessoas quando obedecer a uma determinada situação, por exemplo, enviar uma mensagem de alerta para a equipe de segurança quando uma sequencia muito grande de ataques for bloqueada em um curto espaço de tempo.
- Implementação de um algoritmo de *profiling*, que deverá aprender como funcionam as aplicações onde a ferramenta de *profiling* foi instalada, e então, gerar regras do modelo positivo para o jShield, o que facilitará bastante a implementação desse modelo.
- Implementação de um algoritmo para SSL *offloading*, o que ajudará o jShield a acelerar conexões SSL.
- Evolução do algoritmo de classificação de ataques do jShield usando Redes Neurais Artificiais.
- Evoluir o algoritmo de *caching* do jShield, o que poderá melhorar ainda mais os resultados já obtidos nos testes que foram executados na ferramenta.

8. Conclusões

O jShield é um firewall de aplicação que implementa algumas características diferenciada dos firewalls de aplicação tradicionais e, devido a sua arquitetura de aplicação, poderá facilmente evoluir da categoria de software adaptativo para a de software autonômico. As funcionalidades oferecidas pelo jShield tentam tirar, senão completamente, mas grande parte da responsabilidade com a análise de segurança das aplicações Web das mãos dos desenvolvedores. Assim, a sempre prioritária preocupação com as regras de negócios das aplicações pode continuar sendo o foco da atenção dos desenvolvedores.

Como demonstrado nos testes, o jShield além de tornar a aplicação mais segura, agiliza o tempo de resposta das solicitações graças ao algoritmo de *caching* implementado, ao mesmo tempo que não há perdas significantes na vazão dos dados que passam pela ferramenta, tornando-o uma ferramenta extremamente atrativa pelos seus resultados.

Referências

Aqtronix (2009). WebKnight - Application Firewall for Web Server. http://www.aqtronix.com/?PageID=99. Novembro.

Bace, R., Mell, P. (2001). Intrusion Detection Systems. NIST - National Institute of Standards and Technology. http://www.snort.org/docs/nist-ids.pdf.

Ceron, J, et al. (2008). Vulnerabilidades em Aplicações Web: uma Análise Baseada nos Dados Coletados nos honeypots. VIII Simposio Brasileiro de Segurança. http://sbseg2008.inf.ufrgs.br/proceedings/data/pdf/st06_02_resumo.pdf.

CERT.BR (2009). Estatísticas dos Incidentes Reportados ao CERT.br. http://www.cert.br/stats/incidentes/index.html, Novembro.

IISteam, (2009). Using URLScan. http://learn.iis.net/page.aspx/473/using-urlscan/, Novembro.

JMeter (2009). JMETER: Uma Aplicação desktop projetada para testes de carga e medidas de performance. http://jakarta.apache.org/jmeter/. Novembro.

Jones, K. J., Bejtlich, R., Rose, C. W. (2006). Real Digital Forensics – Computer Security and Incident Response, Addison-Wesley.

Microsoft (2009). UrlScan Security Tool. http://technet.microsoft.com/en-us/security/cc242650.aspx. Novembro.

ModSecurity (2009). ModSecurity: open source web application firewall. http://www.modsecurity.org. Novembro.

OWASP (2010). The ten most critical web application security vulnerabilities. Open Web Application Security Project - OWASP. http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf.

WAFEC (2006). Web Application Firewall Evaluation Criteria. Web Application Security Consortium. http://www.webappsec.org/projects/wafec/v1/wasc-wafec-v1.0.pdf.