MÉTODOS PARA PREVENÇÃO E DEFESA DE ATAQUES DDoS

Fábio de Bittencourt[1]

Gustavo dos Santos de Lucca[2]

Resumo: Os ataques *DDoS* compreendem atualmente uma das principais e mais conhecidas formas de ataques a aplicações *on-line*. Por ter a sua execução considerada simples e aproveitando-se do trafego "normal" da internet, que também é utilizado pelos usuários comuns, o ataque *DDoS* representa uma das maiores ameaças à um serviço online. Isso se justifica, já que a sua execução não pode ser prevista, e qualquer aplicação conectada a Internet pode estar suscetível a esse tipo de ataque. Esses ataques possuem como foco a indisponibilidade do serviço, ou seja, paralisar, parcialmente ou completamente, a aplicação, afetando uma das propriedades básicas de segurança da informação. Qualquer infraestrutura que suporte uma aplicação *on-line* deve possuir uma estratégia de defesa contra esse tipo de ataque. Essa pesquisa apresenta um referencial bibliográfico sobre a defesa de ataques *DDoS*, a mesma se baseou em diversos trabalhos de autores sobre o assunto. Nele serão apresentadas estratégias que podem ser utilizadas para atenuar e proteger os serviços *on-line* de um ataque *DDoS*. Para um melhor entendimento sobre a melhor estratégia de defesa, dividiu-se o trabalho em dois mecanismos: prevenção e defesa. O primeiro, apresenta detalhes sobre como um serviço pode monitorar a rede e acessos para identificar um potencial ataque. O segundo, apresenta detalhes de como mitigar os efeitos de um ataque eminente ou que esteja em curso. Portanto, o principal objetivo dessa pesquisa é de apresentar estratégias para a prevenção e defesa de ataques *DDoS*, analisando o funcionamento, benefícios e as formas que devem ser aplicados para que o servidor de aplicação possa implementar para manter o ambiente mais seguro.

Palavras-chave: Servidor, Segurança, Ataques, *DDoS*, *DoS*.

1 INTRODUÇÃO

Em uma aplicação *on-line,* para que o usuário acesse um site, um aplicativo ou até um serviço de requisições (como, por exemplo, o serviço de emissão de nota fiscal eletrônicas (SEFAZ), da receita federal), os navegadores, aparelhos celulares ou dispositivos *IoT (Internet of Things)* conectam-se à um servidor, que por sua vez mantém a aplicação funcionando.

[1] Graduando em Engenharia da Computação Faculdade SATC. E-mail: fabiodebitt@hotmail.com

[2] Ms. em Engenharia, Professor da Faculdade SATC. E-mail: gustavo.lucca@satc.edu.br

Esses dispositivos, que se conectam à um servidor, são denominados clientes e, assim como qualquer outro recurso computacional, a capacidade de um servidor receber conexões de clientes é limitada.

A lógica por trás dessa quantificação de limitação possui diversos fatores, sejam por sua capacidade computacional de processamento ou em outros casos por deficiência da infraestrutura da rede do local onde o servidor se encontra fisicamente. Além disso falhas na projeção do software da aplicação, assim como técnicas de programação aplicadas erroneamente podem contribuir para a diminuição da capacidade de um servidor receber conexões de seus clientes.

Contudo, mesmo que todas as boas práticas sejam seguidas e os melhores recursos sejam alocados, é possível afirmar que nenhum servidor possui uma capacidade de conexão ilimitada e que esses fatores são apenas variáveis que podem contribuir positivamente ou negativamente para a quantidade de conexões que o servidor poderá receber.

Se por um lado a quantidade de conexões que o servidor pode receber é possível de se quantificar, é praticamente infinita a quantidade de requisições que podem ser recebidas de clientes externos, já que qualquer dispositivo conectado à internet pode ser considerado um potencial cliente da aplicação.

Dessa forma, caso um servidor exceda a sua capacidade de receber requisições de clientes, em função de algum motivo específico, seja ele legítimo ou não, a resposta à essas requisições começam a ser prejudicadas, já que o servidor não consegue processar todas as requisições recebidas. Portanto, se novas requisições serem recebidas pelo servidor, o serviço poderá apresentar lentidão, travamentos ou até indisponibilidade total.

Baseando-se nessa deficiência presente em todos os servidores, surgiu no início dos anos 2000, os ataques conhecidos como *DDoS* (*Distributed Denial of Service*), sigla em inglês, para ataques de negação de serviço distribuídos (STALLINGS, 2008).

Desde então, os ataques *DDoS* tem sido uma grande preocupação para profissionais de TI em todo o mundo por dois fatores. O primeiro se deve ao fato de que, cada vez mais as aplicações online serem o centro comercial de toda a humanidade. Segundo, é que atualmente, não é possível que esse tipo de ataque seja completamente extinguido da rede.

Sendo assim, um plano de defesa contra esse tipo de ataque deve ser formado no ambiente da aplicação, principalmente para casos de aplicações que necessitam estar *on-line* durante a maior parte do tempo, como aplicações bancárias, educacionais e governamentais.

Esse artigo reúne pesquisas de diversos autores e possui como objetivo apresentar os resultados dessas pesquisas, discuti-las e demonstrar ao usuário estratégias de prevenção e defesa que podem ser aplicadas a um ambiente que presta um serviço pela Internet. Dentre essas estratégias, também serão, quando possível, indicados quais os tipos de aplicação em que a estratégia melhor se encaixa.

Como parte de um estudo de caso, este artigo visa também, apresentar todo o contexto da problemática, incluindo um histórico sobre aplicações web e sobre a criação dos ataques *DDoS*.

Desde a popularização da Internet, os casos de *DDoS* tem crescido ano a ano, sendo que entre os primeiros casos de relevância desse tipo de ataque, podemos citar um de fevereiro de 2000, onde o Yahoo®, então um dos maiores domínios da internet, foi alvo de *DDoS*, que acabou custando a perda de contratos de publicidade por causa das 2 horas de inatividade dos seus serviços (SPECHT; LEE, 2004).

Como exemplo de periculosidade desse ataque, pode-se citar, no ano de 2016, o que foi considerado o maior ataque *DDoS* da história. O ataque aos servidores da DYN DNS, uma das maiores empresas do ramo de *hosting* do mundo. O ataque fez com que os serviços de Internet por ela mantidos, como o Twitter®, Spotify® e Netflix®, tivessem instabilidade ou ficassem indisponíveis. O impacto desse ataque foi tão abrangente, que a própria velocidade da internet, em todo o mundo, teve, em algum momento, uma instabilidade registrada (PAYÃO, 2016).

Além disso, também no ano de 2016, foi registrado um outro recorde. Durante um ataque à empresa francesa de *hosting* OVH, a taxa de velocidade do ataque chegou ao pico de 1.1 *Terabits* por segundo, atingindo mais de 145 mil câmeras de vigilância e outros dispositivos *IoT* em toda a Europa (GOODIN, 2016).

Esse novo recorde, segundo Dan Goodin (2016), editor de segurança da *ArsTechnica.com*, com taxas de velocidade antes impensáveis, tende a ser o novo normal quando se fala sobre ataques *DDoS* no futuro.

Com base nesses dados e na preocupação para que os ambientes de aplicações *on-line* estejam devidamente preparados, este artigo irá apresentar um plano teórico de contenção para ataques *DDoS.*

As duas seções seguintes deste artigo (2 e 3) são destinadas a informações técnicas contextualizando o *DDoS*, exemplificando o funcionamento que ocorre em uma rede antes e durante um ataque.

A seção 4 apresenta três diferentes estratégias de prevenção de ataques. A quinta (5) seção descreve ações que podem ser tomadas durante um ataque e os trabalhos futuros que serão realizados, por fim apresenta-se a conclusão desse trabalho.

2 APLICAÇÕES CLIENTE-SERVIDOR

A história da chamada arquitetura cliente-servidor confunde-se com a história da própria internet, pois enquanto cientistas da computação do Instituto de Pesquisa de Stanford (*Standford Research Institute*) desenvolviam essa arquitetura entre os anos de 1960 e 1970, estes mesmos cientistas participavam da construção da ARPANET (do inglês *Advanced Research Projects Agency Network* ou Agência de Pesquisa em Projetos Avançados, em Português), uma rede de comunicações utilizada pelo governo dos EUA na guerra fria contra a União Soviética e, conhecida hoje como uma das principais percursoras da internet (WEBER, 2013).

No projeto da ARPANET, os cientistas usaram os termos *server-host* e *user-host* que, vieram a aparecer nas RFC 4 e 5 e, continuaram a ser utilizados no Xerox PARC, laboratório de pesquisas da Xerox, no meio dos anos de 1970.

Inicialmente, uma rede cliente-servidor possuía o seu processamento centralizado, em um modelo onde vários clientes acessavam à um servidor, geralmente um mainframe, que processava todas as requisições e retornavam aos clientes as respostas. Este modelo foi predominantemente usado a partir dos anos 70 e, quase que exclusivamente, utilizados por redes locais de grandes universidades americanas (CHUNG, 2000).

Com a evolução da internet como um todo, a capacidade de processamento de apenas um servidor físico para processar aplicações e as requisições de seus clientes tornou-se pequena. Então, o processo teve que ser atualizado e, atualmente, para ambientes que possuem um nível alto de requisições, o processamento dos servidores são distribuídos em várias máquinas, em modo de processamento chamado de cooperativo. Este modo caracteriza-se pelas máquinas que compõem o servidor

compartilharem seus recursos entre si, dividindo a carga total de processamento (CHUNG, 2000).

Com essas informações Chung (2000, p. 2) define que: “O cliente de uma aplicação é quem requisita pelo serviço, enquanto que um servidor é quem disponibiliza o serviço, sendo que o “serviço” pode ser qualquer tipo de recurso”.

Na arquitetura cliente-servidor, pode-se explicar de forma resumida que, para que seja formada uma conexão entre as partes, o cliente deve inicialmente realizar uma requisição ao servidor solicitando um acesso, o servidor então devolve uma espécie de chave ao cliente que, com a chave em mãos, pode acessar a aplicação. Este processo de conexão entre cliente e servidor será detalhado mais tecnicamente quando for explicado o funcionamento de um ataque *DDoS*.

Com a popularização da Internet um problema na arquitetura cliente-servidor se apresentou durante os anos 2000, o chamando *C10K Problem*, que buscava uma forma de que servidores de aplicação pudessem suportar dez mil conexões ao mesmo tempo.

Para os padrões atuais, segundo Robert Graham, CEO da Errata Security®, o problema mudou e devemos chama-lo de *C10Mi Problem*, ou seja, como um serviço poderá suportar dez milhões de conexões ao mesmo tempo (HOFF, 2013).

Grandes corporações como o, Facebook® e Google® já possuem um tráfego muito maior que isso, porém como a sua estrutura é muito grande, pode-se considerá-los casos a parte e não se deve compará-los com qualquer serviço online que utiliza a arquitetura cliente-servidor.

Apesar de que, pode-se dizer que, a principal forma de aumento da capacidade de um servidor receber um número maior de conexões seja a alocação de um número maior de máquinas processando as requisições, alguns fatores podem otimizar esse processo e permitir um número maior de conexões à um servidor.

Todd Hoff (2013), em um artigo para o site *HighScalability.com*, lista diversos fatores que podem auxiliar servidores à conseguirem o número de dez milhões de conexões simultâneas. Dentre esses fatores pode-se destacar a escolha do sistema operacional, a codificação para permitir uma escalabilidade do servidor, a escolha do *CPU,* a sua capacidade de *multi-thread* e a quantidade de memória alocada e escalável para o processamento das requisições do servidor.

3 DDoS - NEGAÇÃO DE SERVIÇO DISTRIBUÍDO

O *DDoS*, sigla para *Distributed Denial of Service* (em tradução livre, Negação de Serviço Distribuído) é um ataque coordenado focado na indisponibilidade de serviços que é lançado através de sistemas de computadores comprometidos (SPECHT; LEE, 2004).

Esse tipo de ataque não visa o roubo de dados, inclusão de software malicioso no servidor ou qualquer outro impacto direto na integridade dos dados da aplicação ou dos seus clientes. O seu principal objetivo é retirar, seja parcialmente ou totalmente, o funcionamento das aplicações alvo, o que em determinados casos podem significar um pesado prejuízo para o mantenedor do serviço. Por exemplo, como um *e-commerce*, que deixa de receber acessos e compras de seus clientes em função da indisponibilidade do serviço.

Para atingir esse alcance, durante um ataque *DDoS*, computadores espalhados pela rede e máquinas virtuais (chamados de zumbis) que estão sob o controle de hackers realizam múltiplas requisições à uma aplicação ao mesmo momento e por um determinado período de tempo a fim de interromper o seu funcionamento (ABLIZ; ZNATI, 2015).

Este tipo de ataque evoluiu a partir do *DoS* (*Denial of Service*, em tradução livre, Negação de Serviço), que possuía a mesma forma de ataque, porém partia apenas de um cliente, o que acabou, com o avanço das tecnologias de segurança, se tornando um método ultrapassado de ataque e mais facilmente defensável para uma rede. Em um ataque *DDoS,* vários hosts atacam um mesmo servidor, o que dificulta as chances de defesa e resposta do servidor. A figura 1 exemplifica a diferença entre os ataques *DoS* e os ataques *DDoS*.

Stallings (2008) cita que o principal desafio ao lidar com ataques *DDoS* é o grande número de maneiras como eles podem operar e também as constantes formas novas que surgem. Assim, as contramedidas para *DDoS* precisam evoluir no mesmo ritmo que novas formas de ataques surgem.

Em um ataque *DDoS* pode-se classificar os serviços que estão sobre ataque como "*vítimas primarias*". Essa classificação se deve ao fato de que apesar de serem o principal alvo do ataque, esses serviços não são as únicas "*vítimas*", pois pode-se, ainda sim, classificar os computadores comprometidos e que são fonte das requisições utilizadas nos ataques, como "*vítimas secundárias*".

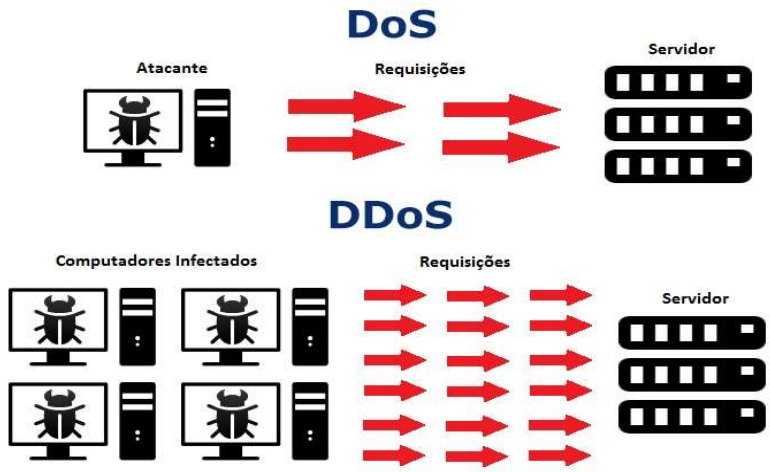


Figura 1: Funcionamento DoS e DDoS.
Fonte: Adaptado de Scudlayer (2015).

Specht e Lee (2004) classificam a arquitetura dos ataques DDoS em duas: O modelo agente-manipulador (Agent-Handler) e o modelo baseado em IRC (*Internet Relay Chat*), sendo que em ambos os modelos se tem uma estrutura mais ou menos definida em cliente, manipulador e agente, possuindo diferenças no modo de comunicação entre o cliente e os seus agentes.

O cliente é primeira ponta de um ataque *DDoS* e se caracteriza como sendo de onde o atacante se comunica com o resto do seu sistema de ataque. Já os manipuladores são softwares localizados pela internet que o cliente usa para se comunicar com os seus agentes. Geralmente esses softwares são colocados em roteadores ou ambientes de redes comprometidos, sendo que os atacantes têm preferência por colocar esses softwares em redes com um número alto de tráfego. Sendo assim quanto maior o tráfego, menor é a chance do seu software malicioso ser identificado. E, por fim, os agentes são dispositivos na rede que estão comprometidos por um vírus ou qualquer outro programa malicioso que dá ao cliente o controle para executar o ataque a partir dessas máquinas.

Um cliente controla muitos manipuladores, que por sua vez controlam outros muitos agentes, e são os agentes que de fato irão realizar as requisições aos serviços que se deseja atacar, o que possibilita o ataque distribuído e de alta intensidade.

No fluxo do modelo agente-manipulador, o cliente se comunica com os seus manipuladores espalhados pela internet, que, por sua vez, faz contato com os agentes que estão sendo controlados para que o ataque seja feito. A diferença desse processo para o modelo baseado em *IRC*, é que neste outro modelo os manipuladores estão em um canal *IRC*, que provém benefícios como o uso de portas legitimas para o contato

com os agentes. Como servidores *IRC* possuem um largo volume de dados, a identificação de uma comunicação de ataque *DDoS* se torna ainda mais difícil.

Em ambos os modelos apresentados os agentes são considerados também vítimas do ataque (ou vítimas secundárias, como vimos anteriormente) pois, geralmente, os usuários desses dispositivos não possuem conhecimento que o seu dispositivo está infectado ou participando de um ataque. Esses agentes também podem ser referidos, em outras literaturas, como “zumbis” ou “bots".

Para que um ataque ocorra é necessário que em algum momento os manipuladores infectem os agentes com seus softwares maliciosos. Sendo que em relação às formas de instalação desses softwares, que controlam o ataque *DDoS* e que infectam os agentes, temos duas formas de instalação que são classificadas em: passiva ou ativa (SPECHT; LEE, 2004).

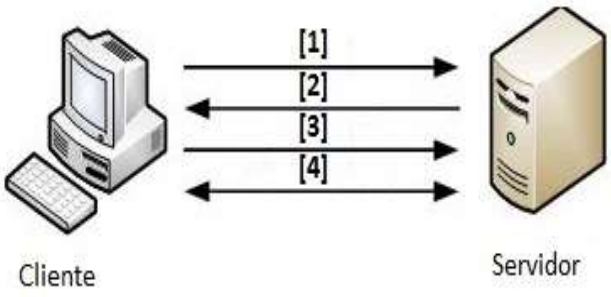
Na forma de instalação passiva, a instalação geralmente ocorre através de compartilhamentos de arquivos infectados por parte do atacante, ou a construção de web sites para que o usuário seja infectado através de falhas no navegador local da vítima secundária.

Já uma instalação ativa, geralmente vem acompanhada de um escaneamento, por parte do atacante, de redes externas, onde o seu objetivo é encontrar redes com vulnerabilidades. Uma vez encontradas, essas redes sofrem ataques via script ou alguma outra forma que conceda ao atacante o controle do serviço, máquina ou rede, abrindo a partir daí um ambiente onde o atacante pode instalar o seu programa que facilitará o seu ataque futuro.

Com os seus agentes preparados e os manipuladores postos na rede de forma que o cliente consiga a comunicação com eles, a infraestrutura de um ataque está montada. Dentre as muitas formas de ataque *DDoS* existentes podemos citar o chamado ataque de esgotamento de recurso, onde o agente envia para o servidor vários pacotes seguidos para apenas “bagunçar” a rede do servidor, deixando essa rede sem espaço para requisições de clientes legítimos.

Specht e Lee (2004) citam um exemplo dessa forma de ataque, o TCP SYN, que atuam em cima do processo *three-way handshake* (aperto de mão em três vias, em tradução livre) comum na arquitetura cliente-servidor e explicado na seção II. Essa forma de ataque também está exemplificada no trabalho de Juels e Brainard (1999) em que é apresentado uma forma de defesa para esse tipo de ataque e que veremos na próxima seção deste artigo.

O *three-way handshake* funciona, de forma resumida, em uma conexão TCP/IP conforme apresentado na Figura 2.



[1] O host (cliente) inicia a conexão e envia um pacote com uma *flag* SYN ativado (no header do TCP).
[2] O servidor, então ao receber, envia um segundo pacote com as *flags* SYN e ACK, que será utilizado pelo cliente para confirmar a conexão com o servidor.
[3] Por último ao receber o pacote do servidor, o cliente repassa ao servidor mais um pacote, dessa vez apenas com a *flag* ACK ativada.
[4] Terminando o processo inicial está estabelecida a conexão com o servidor.

Figura 2: Funcionamento do *three-way handshake*.
Fonte: Adaptado de Kamal (2016).

Em uma das formas de esgotamento de recurso, os agentes enviam pacotes para o servidor com a *flag* SYN ativada com um endereço IP de origem falso (em inglês *spoofed*), ou seja, o IP que é enviado ao servidor não é o do computador de origem. Sendo assim, o servidor retorna o segundo pacote com as *flags* para um IP inexistente na rede, então o pacote se perde. Além disso, o agente continua disparando novos pacotes para o servidor e o servidor continua enviado pacotes ao limbo.

Com um grande volume de pacotes SYN sendo enviados e nenhum pacote SYN e ACK respondido por um cliente, o servidor vai ficando sem recursos (memória ou processador) e, em algum momento, ficará incapacitado de responder requisições legítimas.

Além da comunicação pelo protocolo TCP, Specht e Lee (2004), declaram que ataques de esgotamento de recursos também podem ser feitos em comunicações através de protocolos UDP e/ou ICMP.

4 ESTRATÉGIAS DE PREVENÇÃO

Conforme citado na seção I, qualquer servidor de aplicação conectado à internet está aberto para ataques externos, em razão disso uma estratégia de prevenção deve ser traçada para diminuir as chances de um ataque ser bem-sucedido, facilitar a identificação do ataque e bloquear requisições maliciosas ao servidor.

Inicialmente, parte-se do ponto que é praticamente impossível prever quando um ataque irá ocorrer, portanto as estratégias visam fortalecer o servidor de uma aplicação contra a eminência de um ataque. Contudo é preciso considerar que, apenas em conjunto com uma gerência ativa e recorrente da rede poderá ser dito que o servidor possui uma defesa proativa contra o *DDoS*.

As estratégias apresentadas aqui são uma das formas utilizadas na prevenção de ataques *DDoS*, sendo que existem várias outras que podem ser utilizadas, levando em consideração que certas estratégias são mais adequadas para um tipo de rede, equipamentos e protocolos.

A) Enigmas para Requisições

Uma das estratégias que podem ser aplicadas em um servidor para que ataques *DDoS* sejam evitados é a utilização de enigmas na troca de requisições entre o cliente e o servidor.

Enigmas (em inglês *Puzzles*) é um termo utilizado para um cálculo ou uma função que o cliente deve processar em seu ambiente para provar a veracidade de sua requisição. O ideal é que o cliente despenda nesse cálculo um número significativo de processamento (inclusive em cálculo de funções criptografadas como multiplicação modular).

Essas funções não devem ser tão simples que o cliente não tenha quase nenhum trabalho no seu processamento e nem tão complexa a ponto de comprometer a velocidade da conexão do cliente com o servidor (ABLIZ; ZNATI, 2015).

Nessa estratégia, os enigmas são desenvolvidos para que a verificação do seu resultado seja feita com o mínimo de processamento por parte do servidor, sendo assim o servidor não fique proporcionalmente tão sobrecarregado quanto o cliente que está realizando a requisição.

Com o processamento dessas funções, além de provar a veracidade em acessar o recurso disponibilizado pelo servidor, há uma estratégia de defesa atrás do uso dos enigmas em requisições. Como o cliente (aqui cliente da aplicação ou agente de um ataque *DDoS*) possui um certo processamento para ter acesso ao servidor, o próprio processamento do cliente limita a sua participação em um ataque *DoS* ou *DDoS*.

Em um acesso comum em um ambiente cliente-servidor, conforme já apresentado na seção II, temos o processo em três vias. Com a utilização de enigmas nas requisições de acesso, o processo descrito anteriormente ganha mais uma etapa, sendo que as outras também recebem modificações.

Quando o cliente realiza uma requisição ao servidor (C1), o servidor responde ao cliente passando um enigma para ser solucionado (S1), esta função então é processada pelo cliente e o seu resultado enviado ao servidor (C2) e, então de acordo com a resposta do cliente, o servidor concede (S2) ou não o acesso ao cliente (ABLIZ; ZNATI, 2015). A figura 3 ilustra esse processo.

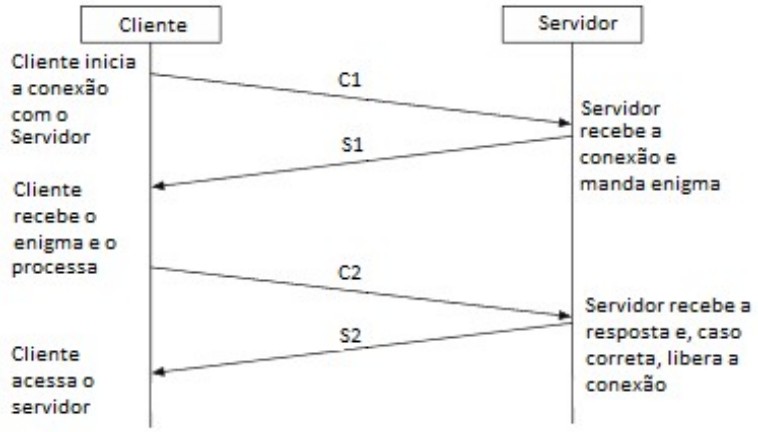


Figura 3: Interação do Cliente-Servidor com a utilização de enigmas.
Fonte: Adaptado de Abliz e Znati (2015).

Conforme Abliz e Znati (2015), esta proposta foi inicialmente defendida por Dwork e Noar em 1992, porém foi apenas com Juels e Brainard em 1999, na mesma época em que os ataques *DoS* começaram a ocorrer nos EUA contra sites famosos como o Bing® e o The New York Times®, que um modelo de utilização de enigmas em requisições foi realmente desenvolvido.

Em seu trabalho inicial, Juels e Brainard (1999) defendiam a ideia que o enigmas nas requisições seriam valiosos contra os ataques de esgotamento de serviços e já adiantavam que trabalhos futuros poderiam melhorar essa forma de defesa, procurando deixa-la mais compacta e eficiente de ser utilizada.

Levando em consideração isso, Abliz e Znati (2015) levantam em seu trabalho que esse processo possui alguns problemas a serem resolvidos e propuseram uma nova forma de utilização dos enigmas.

Um desses problemas citados como estarem pendentes de resolução é relacionado a literatura existente sobre enigmas, que não dá uma ideia clara de como

determinar a dificuldade dos enigmas de acordo com a aplicação do servidor e tão pouco sobre quando determinar que a utilização desses enigmas deve ser “ligada” ou “desligada”.

Um exemplo que ilustra a primeira parte desse problema pode ser feito comparando uma API que acessa um serviço de processamento maciço de dados, e um acesso à um site estático de uma empresa ou entidade governamental. Esses dois servidores possuem necessidades e tipos de clientes diferentes, então a dificuldade dos seus enigmas deveria ser proporcional à sua função, pois caso contrário, a dificuldade de se atacar ambos os serviços seria a mesma.

O segundo problema citado pelos autores (2015), pode ser interpretado como um desperdício do processamento que o cliente realizou em relação à utilização desse processamento pelo servidor. Nessa estratégia o servidor sempre realiza o cálculo do enigma que está sendo recebido do cliente.

B) Enigmas Produtivos para Requisições

Abliz e Znati (2015) defendem a utilização de enigmas produtivos (em inglês, *productive puzzles*) como uma evolução da utilização dos enigmas nas requisições, que pretende além de outras coisas resolver os problemas apresentados, principalmente em relação ao desperdício do processamento do servidor.

Os métodos existentes de construção de enigmas focam em criarem funções F, sendo que solucionar y = F(x) é necessário pouco poder de processamento, porém solucionar o seu inverso (x = F^(-1)(y)) consumo um poder maior de processamento (ABLIZ; ZNATI, 2015).

Com isso, o cliente ao receber uma função inversa possui um determinado conjunto de trabalho (tempo ou recurso) para processar a função enquanto que o servidor possui um trabalho ínfimo para verificar a sua resposta, solucionando por sua vez a função normal.

Para requisições com enigmas produtivos, o sistema não passa apenas uma função, mas sim uma série de funções, que o usuário deve calcular e retornar a resposta de todas elas corretamente para obter acesso ao servidor.

Em um ambiente assim, o servidor possui algo parecido como uma biblioteca de funções que são armazenadas por ele, e que serão utilizadas para requisições em algum momento. Dentro dessa biblioteca, o servidor já possui processado a resposta

para uma parte dessas funções armazenadas, sendo que para a outra parte, o servidor não conhece a solução. Uma tarefa que o servidor já possua a sua resposta é chamada de tarefa conhecida, enquanto que uma tarefa que o servidor não tenha calculado a sua resposta é chamada de tarefa desconhecida.

As tarefas conhecidas, quando são utilizadas nas requisições, possuem o seu resultado armazenado comparado com o valor enviado pelo cliente para identificar se o processamento foi feito corretamente e se trata de uma conexão legítima. Já as tarefas desconhecidas não possuem nenhum tipo de validação, sendo consideradas corretas para qualquer valor que seja passada pelo cliente.

Os autores (2015) deixam claro que, já neste ponto da requisição, é possível que um cliente tente burlar a verificação de segurança e mandar qualquer resposta ao servidor, pressupondo que todas as tarefas enviadas à ele pelo servidor sejam tarefas desconhecidas. Porém, devido a aleatoriedade das tarefas enviadas ao cliente isso pode apresentar a intenção de ataque por parte do cliente caso o servidor receba uma tarefa conhecida com o resultado incorreto e, consequentemente, o corte da sua comunicação com o servidor.

Essa questão das tarefas conhecidas e desconhecidas acabam por minimizar, o que na visão dos autores (2015), era um dos problemas dos enigmas em requisições. Como o servidor já sabe a resposta das tarefas em um local previamente armazenado, elas não precisam ser reprocessadas, sobrando ao servidor apenas comparar o resultado vindo do cliente com o seu valor armazenado.

Em seu trabalho, Abliz e Znati (2015) chegam a explicar, matematicamente, a probabilidade de um cliente trapacear a verificação, sendo que o seu cálculo explica que quanto maior o número de tarefas (tanto conhecidas como não conhecidas), a probabilidade de qualquer chance de trapaça pelo cliente diminui exponencialmente.

O protocolo de comunicação do enigma produtivo é bem similar ao apresentado no método anterior, onde há 4 "momentos" de comunicação entre o cliente o servidor.

Porém, o conteúdo dos pacotes se difere, tanto na quantidade de tarefas, por exemplo, além de identificações do IP do cliente e de um tempo de expiração da resposta da tarefa, que após ter sido encerrado, o servidor passa a não mais reconhecer aquele processo de conexão cliente-servidor. A figura 4 ilustra o conteúdo dos pacotes na comunicação entre cliente e servidor em um ambiente de enigmas produtivos.

O outro ponto citado pelos autores (2015) como falho no modelo anterior seria em relação à dificuldade das funções que são enviadas para o cliente solucionar. Para os enigmas produtivos, é proposto um modelo matemático para definir a dificuldade dos enigmas presentes no servidor, levando em consideração algumas variáveis do ambiente computacional. Dentre essas variáveis presentes no cálculo, pode-se citar, por exemplo, a quantidade média de clientes ativos no servidor, o poder de processamento médio dos clientes e a taxa de velocidade dessas requisições.

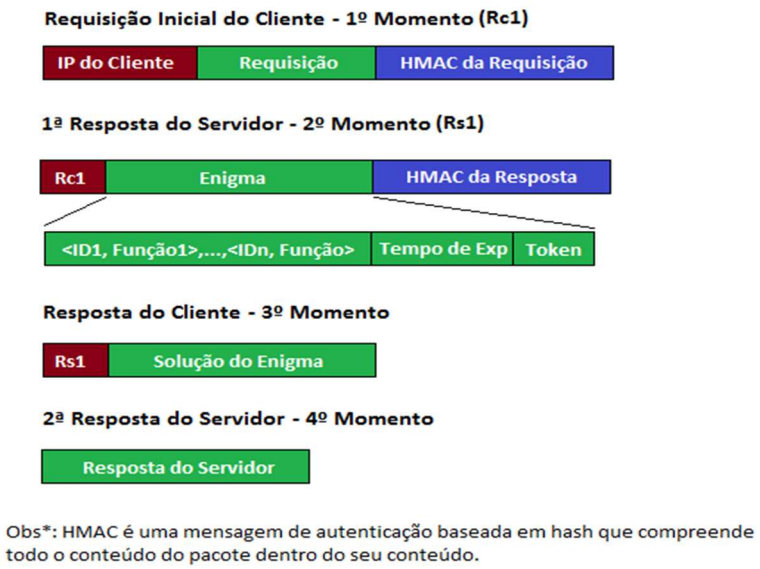


Figura 4: Comunicação Cliente-Servidor em um Ambiente com Enigmas Produtivos.
Fonte: Adaptado de Abliz e Znati (2015).

Além disso, Abliz e Znati (2015) definem um outro cálculo matemático, desta vez para definir a dificuldade de um enigma para um determinado cliente. Nesse cálculo, os autores utilizam entre outras variáveis, o tempo médio de resolução do enigma e uma variável, chamado no seu trabalho de Fator de Correção *(Correction Factor,* em inglês), que é utilizada para ajustar o valor da dificuldade em certos casos em que o servidor se encontra.

Contudo, ainda que todo o processo já descrito seja respeitado, o cliente ainda pode tentar realizar um ataque de esgotamento de serviço, enviando ao servidor, múltiplas respostas dentro do tempo de expiração do enigma (no 3º momento da comunicação).

Porém, para cuidar dessa questão, é proposto pelos autores (2015) uma estrutura de dados chamada de *Auto-Expire Cache* (Cache de expiração automática, em tradução livre.), que além de controlar o tempo de expiração das chamadas, possui uma outra função que impede o ataque de esgotamento de serviço.

Essa estrutura é formada por duas tabelas *hash* de espaço fixo, onde a memória disponibilizada para as requisições ao servidor é controlada pelo tamanho dessa estrutura e liberadas após o seu tempo ter terminado.

Ao receber uma nova requisição, o servidor aloca no *Auto-Expire Cache* as informações dessa requisição. Como no pacote enviado para o cliente tem-se um id único, o sistema consegue identificar a requisição que é relacionada a resposta que está recebendo. Então ao receber essa requisição, o servidor além de verificar se o tempo já expirou, o sistema verifica se a resposta já foi retornada, caso não tenha sido, o servidor segue o seu fluxo, caso contrário, o servidor ignora a resposta.

Para apresentarem esse novo modelo, Abliz e Znati (2015) realizaram um teste prático, implementando esse ambiente. Segundo o seu trabalho, os autores chegaram à conclusão, após o seu experimento, de que este novo método pode ajustar dinamicamente a dificuldade de seus enigmas, baseando-se na carga atual do servidor e no tempo de resposta das requisições, assim como eles haviam proposto em teoria.

C) IPS – Sistema de Prevenção de Instrução

Além dos ataques *DDoS*, são variadas as formas que um servidor pode sofrer um ataque, ataques com o objetivo de acessar remotamente o controle do servidor ou ataques contra a estrutura física do servidor são alguns exemplos.

Ao considerar esse fato, é importante que o método de prevenção de ataques esteja preparado para identificar diversos tipos de ataques. Um método que pode ser utilizado nesses casos é o chamado *IPS*.

Os Sistemas de Prevenção de Intrusão, *IPS* (*Intrusion Protection System*s, em inglês) podem ser definidos como uma ferramenta de defesa proativa contra-ataques à computadores e redes (KUMAR; SINGH; JAYANTHI, 2016). O *IPS* faz parte, junto com o *IDS*, do *IDPS*, sigla em inglês para *Intrusion Detection and Protection System*s, (Sistemas de Detecção e Proteção de Intrusão, em tradução livre).

Essa ferramenta de rede, monitora atividades da própria rede ou de um sistema afim de encontrar comportamentos indesejados ou maliciosos ao sistema. A ferramenta pode prevenir ataques antes que eles acessem a rede através do exame no histórico de dados, sendo que quando um ataque é identificado, o *IPS* pode bloquear e registrar o pacote ofensivo ao sistema.

Kumar, Singh e Jayanthi (2016) citam que se pode classificar os sistemas *IPS* em duas categorias: Os sistemas baseados na rede (*Network-based*) e os sistemas baseados no *host* (*Host-based*).

Um sistema baseado na rede pode ser um computador ou plataforma especial de hardware, com um software de detecção instalado. Esse hardware deve ser colocado em um ponto estratégico na rede (como um *gateway* ou uma sub-rede) para analisar todo o tráfego que irá passar naquele determinado nó da rede.

Kavan, Škodová e Klíma (2014) citam, porém, que a decisão do local de onde o sensor deve ser colocado é muito importante. Eles citam o caso da posição do sensor em relação ao NAT (sigla para *Network Address Translation*. Em tradução livre, Tradução de Endereço de Rede.). Se o NAT é acionado no ponto de entrada, colocar o sensor antes desse ponto pode retirar do *IPS* a visualização de determinados pacotes internos. Sendo que por um outro lado, colocar o sensor após o NAT retirará do *IPS* a informação do tráfego que já fora bloqueado no gateway.

Além disso, os autores (2014) citam também que a quantidade de informação pode influenciar negativamente no processo de monitoramento da rede. Sendo que, por exemplo, caso o sensor receba uma quantidade de dados excessiva, e o próprio sensor não consiga processar aquela informação em tempo real, algumas alterações na rede podem ser sentidas. Dentre essas alterações, podem ser citadas um aumento excessivo na latência da rede, a perda de pacotes e a passagem de pacotes não checados para os seus destinos.

Um sistema baseando no host é instalado e projetado para bloquear ataques contra o sistema do próprio host. O *HIPS* (Sigla para *Host-Based IPS*), como é chamado esse tipo de sistema, é pré-configurado para determinar as regras de proteção baseando-se em assinaturas de ataques ou intrusão.

Levando em consideração essas regras, o *HIPS* irá capturar as atividades suspeitas no sistema e em então irá bloquear ou permitir que elas ocorram.

A aplicação do *HIPS* é de grande utilidade no monitoramento, sendo que suas atividades podem monitorar aplicações, requisições de dados, tentativas de conexão de rede, entre outras.

Geralmente, um sistema *IPS* é formado pelos seguintes componentes:

- Módulo de Coleta de Dados (*Data Collector*);
- Gerador de Característica (*Feature Generator*);
- Detector de Incidente (*Incident Detector*);

- Gerador de Modelo (*Model Creator*);
- Gerência de Resposta (*Response Manger*).

A forma de relacionamento entre os componentes do IPS é apresentada na figura 5.

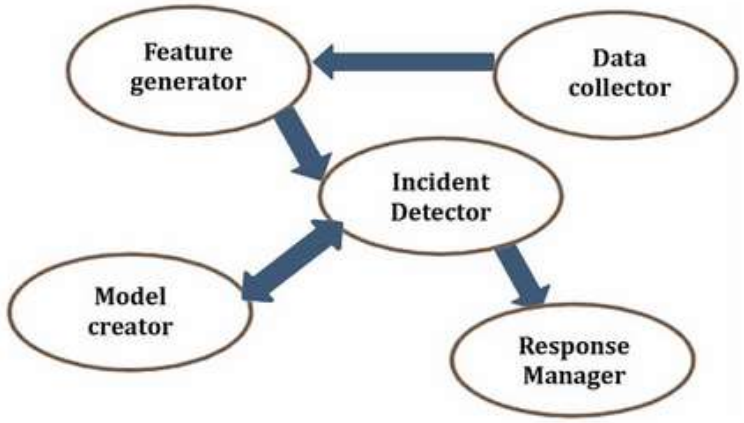


Figura 5: Componentes de um Sistema IPS.
Fonte: Kumar, Singh e Jayanthi (2016).

Além de serem utilizados na defesa contra os ataques *DDoS*, o *IPS* também pode ser configurado para ser eficaz contra outros tipos de ataques como por exemplo o *Zero-Day Attack* (Ataque do Dia Zero, em tradução livre).

Um *Zero-Day Attack* é um ataque até então inédito, baseado em uma vulnerabilidade desconhecida, ou em um novo tipo de ataque em vulnerabilidade existente. Este termo *Zero-Day* vem do número de dias entre o anúncio da vulnerabilidade e a data em que ocorre o seu primeiro ataque (KUMAR; SINGH; JAYANTHI, 2016).

Para a detecção de um ataque, o sistema IPS possui duas metodologias, que são chamadas pelos autores (2016) de “comuns”. São elas: “Detecção por Assinatura ou Má Utilização” e o “Sistema Baseado na Detecção de Anomalia”.

Uma assinatura é um conjunto de bytes que identifica um tipo de ataque, sendo que o IPS funciona como uma biblioteca de assinaturas para realizar as suas detecções. A detecção de assinaturas envolve comparar comportamentos de usuários com o de intrusos, ou procurar, no tráfego da rede, um conjunto de bytes ou pacotes que são maliciosos.

Essa forma de detecção pode ser efetiva contra ameaças conhecidas, porém é bem frágil contra os *Zero-Day Attacks*, que não possuem uma assinatura formada já que são ataques sem nenhum histórico existente para o *IPS* comparar.

Já o sistema baseado na detecção de anomalia é mais simples. Nesse sistema um intruso pode ser observado através da comparação de seu comportamento em relação à de um usuário normal, ou do comportamento esperado pelo sistema.

Esse tipo de sistema, ao contrário do anterior, pode ser bastante eficaz contra o *Zero-Day Attacks*, porém segundo os autores (2016), esse método tem um alto índice de alarmes falso-positivo, justamente por essa comparação entre os comportamentos esperado e realizado não ser tão dinâmica quanto é necessária.

Em um protótipo de sistema baseado no método de prevenção *IPS*, Kavan, Škodová e Klíma (2014), citam que é possível que esse método seja configurado para defender preventivamente a rede de ataques *DoS* e, consequentemente, *DDoS*.

Essa defesa ao *DDoS* pode ser observada em ambas as metodologias apresentadas, seja pela assinatura que os agentes estão utilizando nesse ataque e que o sistema pode possuí-la e identifica-la como maliciosa. E também, no caso da detecção de anomalia, pelo comportamento fora dos padrões normais, que um agente apresenta durante o ataque.

5 DEFESA DE UM ATAQUE DDoS

Com os ataques *DDoS* se tornando uma das principais ameaças para serviços online, apenas ter um ambiente com proteções preventivas para esse tipo de ataque podem, ainda assim, não ser o suficiente.

Como o trabalho de Bhardwaj et al. (2016) cita, por mais que uma forma de defesa ou prevenção contra os ataques *DDoS* seja boa, é praticamente impossível que ela possa proteger o serviço de todas as maneiras em que um ataque *DDoS* pode ser executado.

Em razão disso, os autores (2016) citam que o ambiente em que o servidor se encontra, principalmente se tratando de um ambiente na nuvem, precisa ser cercado de várias formas de defesa.

Sendo assim, será apresentado aqui, duas formas de defesa dentre as disponíveis e que podem ser aplicadas à um servidor de aplicação.

A) Arquitetura de Rede Multicamada para Atenuação de *DDoS*

A arquitetura de rede multicamada para atenuação de *DDoS* (em inglês, *Multi-Tiered Network Architecture DDoS Mitigation*) possui um tipo de aplicação específica, para as quais essa forma de defesa é mais indicada. Segundo Bhardwaj et al. (2016), aplicações na nuvem, mais especificamente os *SaaS* (*Software as a Service*), que possuem serviços financeiros críticos ou aplicações governamentais devem utilizar esse tipo de defesa em seus serviços.

A implementação dessa arquitetura, apresentada pelos autores (2016), deve ser feita no ambiente de nuvem híbrida, compostas por três nuvens separadas. Sendo que duas dessas nuvens devem ser públicas, essas nuvens irão fazer a defesa e o filtro do tráfego que irá acessar à aplicação. A terceira nuvem, deve ser uma nuvem privada, mais segura, e que irá receber a aplicação e os seus dados críticos.

As duas primeiras nuvens são as defesas contra o *DDoS*, sendo que apenas tráfego legítimo possui a permissão de acessar a terceira camada da aplicação. Mais especificamente, a primeira camada faz a defesa da rede. Nessa nuvem deve ser configurado que apenas tráfego de entrada à acessará. Sendo que, segundo Kumar, Singh e Jayanthi (2016), esse nível pode atenuar entre 80% e 90% do ataque *DDoS*. Isso é possível através da configuração de outras estratégias nesse ponto, como o *IPS*.

Após ser checado no primeiro nível, o pacote passa para a segunda nuvem da aplicação. Neste nível, a aplicação filtra os pacotes verificando se dentre eles existe algum pacote danoso à aplicação como: *Key loggers*, *Malware*, *Spyware* e etc.

Como esse fluxo ocorre em nuvens públicas, a escalabilidade e o processamento não são um problema, sendo assim é possível comportar um ataque de esgotamento de serviço por exemplo.

Após passarem pelas duas nuvens públicas, o pacote pode finalmente acessar a aplicação na nuvem privada. Uma vez processado, a nuvem privada devolve o resultado para a nuvem pública na segunda camada que devolve ao usuário pela internet. A figura 6 ilustra o caminho das requisições nessa arquitetura de defesa.

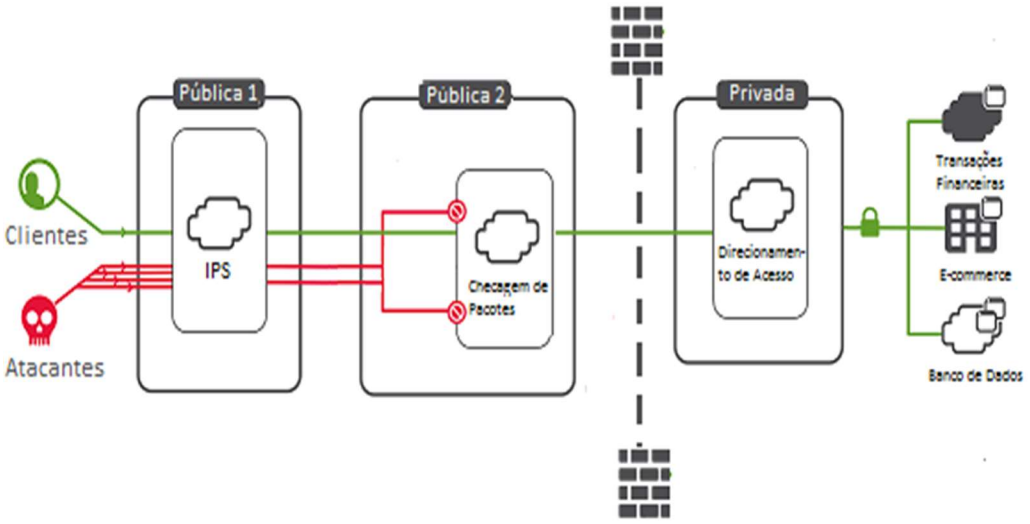


Figura 6: Arquitetura de Rede Multicamada para Atenuação de *DDoS*.
Fonte: Adaptado de F5 Networks (2016).

Essa estratégia possui o objetivo claro de restrição do dado que está sendo acessado, portanto pode não ser indicada à serviços mais comuns. Para esse tipo de caso será apresenta-se como uma outra alternativa de defesa, o *IDS*.

B) IDS – Sistemas de Detecção de Intrusão

Como sugerido pelo nome, o *IDS* ou Sistemas de Detecção de Intrusos (em inglês *Intrusion Detection System*), possui como objetivo a detecção de ameaças e subsequentemente a criação de relatórios ou o envio de aviso para os administradores da rede (HOCK; KORTIŁ, 2015).

Em termos mais técnicos, KUmar, Singh e Jayanthi (2016) definem o *IDS* da seguinte forma: “A detecção de intrusos é o processo de monitoramento e análise de eventos que ocorrem em um computador ou em uma rede para detectar comportamento de usuários que conflitam com o comportamento esperado pelo sistema.”.

Garg e Maheshwari (2016) explicam em seu trabalho, que o *IDS* prove três principais fenômenos em relação à segurança de uma aplicação. São eles: Monitoramento, Detecção e Resposta.

O *IDS* pode ser comparado com um firewall, porém como Hock e Kortiš (2015) explicam, a principal diferença entre os dois é a camada OSI que cada sistema examina. Em um firewall, o exame é feito nas camadas de rede ou transporte, já em um *IDS* a camada de aplicação também pode ser analisada. Em relação à esse fato,

não estão aqui considerados o chamado *Next Generation Firewall*. Essa nova geração também possui a possibilidade de análise da camada de aplicação, contudo, isso é apenas possível pelo fato de que esses firewalls já possuírem o sistema *IDS* em suas aplicações.

Como dito anteriormente na seção IV, o *IDS* faz parte, junto com o *IPS*, do *IDPS*. Sendo que nesse sistema, o *IPS* possui a habilidade de detectar ataques antes que eles ocorram, enquanto em um sistema *IDS*, os pacotes são analisados durante o seu fluxo na rede e a detecção. Caso aconteça, a ação será feita em tempo real. Além disso, com a criação do relatório de ataque pelo *IDS*, o *IPS* pode futuramente utilizar essa nova informação para impedir um novo ataque ou ameaça. A figura 7 exibe, de forma simples, a topologia de ambos os sistemas do *IDPS*.

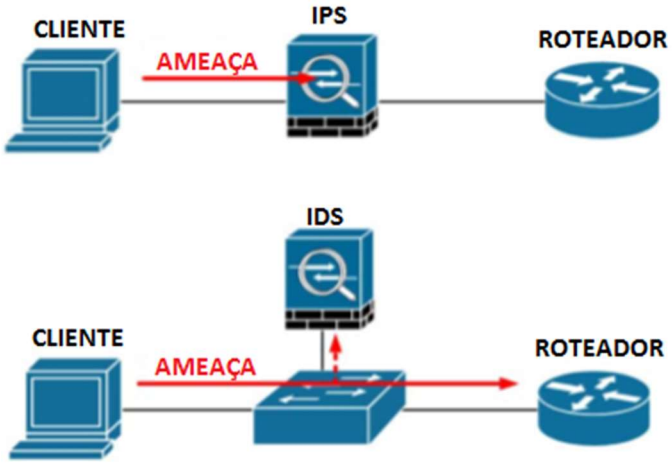


Figura 8: Topologias *IPS-IDS*.
Fonte: Adaptado de Hock e Kortiš (2015).

Um fato bastante comum é que os termos IPS e IDS estejam interligados, um exemplo é a citação em diversas literaturas de um software utilizado para monitorimento de rede, o Snort, mantido pela *Cisco Software*®, como sendo um software para *IDS* (HOCK; KORTIŁ, 2015, GARG; MAHESHWARI, 2016). Contudo, segundo seu próprio site oficial, o *Snort*® é classificado como um software tanto para *IPS* como para *IDS* (SNORT, 2016).

Assim como acontece no *IPS*, a posição onde o software de *IDS* é colocado em relação ao tráfego da rede é extremamente importante, sendo que segundo A Aryachandra, Arif e Anggis (2016), esta posição é chave para o seu sucesso como mecanismo de defesa.

Além da sua eficácia, a posição pode influenciar em outros fatores do servidor segundo os autores (2016), como por exemplo em um ambiente na nuvem, é citado

que o posicionamento dentro ou fora da nuvem do sistema *IDS*, possui uma influência forte no consumo de memória *RAM* e processamento do servidor de aplicação.

Garg e Maheshwari (2016) explicam que uma implementação *IDS* ideal na rede deve possuir as seguintes características:

- **Curto prazo de resposta:** Garantir que qualquer comportamento anormal seja detectado em um tempo previamente estipulado;
- **Alta probabilidade de detecção:** Identificar o máximo de comportamento anormal na rede;
- **Baixo índice de falsos alarmes:** Possuir em seu funcionamento um baixo número de falsos alarmes;
- **Especificação:** Uma vez que um ataque é identificado, informações suficientes sobre este ataque devem estar disponíveis para se obter uma melhor resposta;
- **Escalabilidade:** Possuir a capacidade de expandir seus recursos para não prejudicar o monitoramento da rede.

Seguindo essas especificações, os autores (2016) garantem uma série de benefícios para a rede, sendo que se pode elencar, dentre os mais importantes, os seguintes benefícios:

- Detecção de ataques e várias outras violações de segurança;
- Observar e analisar as atividades de um servidor ou de uma rede como um todo;
- Estimar atividades irregulares;
- Avaliar a integridade do sistema e de seus dados.

Além dessas duas formas de defesa aqui apresentadas, existem disponíveis no mercado muitas outras, cada uma com uma determinada especificação, cabendo ao administrador da rede então, escolher a forma de defesa que mais se aplica à sua rede ou aplicação.

6 TRABALHOS FUTUROS

Após esse trabalho, a ideia futura é de continuar as pesquisas sobre o alcance e o impacto que um ataque *DDoS* pode atingir em um servidor de aplicação na rede.

Um dos pontos que deve nortear trabalhos futuros será um estudo sobre o impacto desses ataques, especificamente em redes brasileiras, assim como a identificação dos verdadeiros autores.

Porém, ao contrário deste artigo que se resumiu apenas à teoria, a expectativa para o próximo trabalho é que algum experimento prático seja realizado para testar as medidas propostas e analisar os seus resultados.

Além disso, como ocorre em trabalhos totalmente teóricos, algumas correções poderão ser feitas no futuro, quando, com o resultado da prática em mãos, as definições teóricas forem confrontadas.

Um outro ponto, que se deseja analisar em trabalhos futuros, seria sobre planos de recuperação de ataques *DDoS*, especialmente com foco em continuidade de negócios. O plano de continuidade de negócios pode, inclusive, receber um trabalho focado em sua aplicação, já que a sua amplitude ultrapassa a recuperação de ataques *DDoS* e pode ser utilizada para qualquer fato que ocorra e que acabe retirando do ar a aplicação.

Por fim, um assunto pouco explorado, e que também poderia ser um tema de trabalhos futuros, seria o impacto monetário e no mercado como um todo, que um serviço pode sofrer após ser atacado. Esse assunto, na opinião dos autores, pode responder à questão mais importante quando se trata em ataques *DDoS*: “Por que uma aplicação é atacada? ”.

7 CONCLUSÃO

A principal conclusão que este trabalho pode apresentar é a de que nenhuma aplicação está livre de ataques *DDoS*, assim como é impossível prever que um ataque aconteça. Justamente por causa disso os métodos de prevenção devem ser cada vez mais adotados nos serviços *on-line*.

Pode-se concluir também que são extremamente grandes a quantidade de formas que um ataque *DDoS* pode apresentar, assim como é a grande quantidade de defesas propostas contra esse tipo de ataque.

Contudo, é possível dizer também que, mesmo uma rede com múltiplos conceitos de defesa estabelecidos, não necessariamente nos dá certeza de que eles serão eficazes.

Isso porque, segundo a grande maioria dos autores pesquisados, a grande chave para proteger a rede de uma aplicação, antes mesmo da instalação desses artifícios, é o conhecimento das fraquezas da rede, além do conhecimento sobre os pontos da rede que devem receber o mecanismo de defesa e os pontos da rede que

seja deseja fortalecer. Pois sem esse estudo preliminar, as ferramentas de defesas podem mais atrapalhar, inclusive umas às outras, a identificação de um ataque.

Assim como o estudo preliminar é importante, um acompanhamento da rede também se faz importante, principalmente em relação à prevenção dos ataques. Um exemplo importante sobre o monitoramento é apresentando nas técnicas de defesa do *IDPS*, onde a geração de alarmes é uma das funcionalidades básicas apresentadas.

Por fim, assim como novas formas de ataque são criadas a todo momento, o recurso humano envolvido no monitoramento, precisa acompanhar essa evolução.

Em um mundo totalmente depende das ações da internet, os ataques *DDoS* tendem a ser um aspecto exaustivamente estudado, pois devido a sua grande abrangência e facilidade de execução, devem fazer da segurança e da infraestrutura das redes, senão os mais importantes aspectos da tecnologia da informação, um dos mais sensíveis.

REFERENCIAS

A ARYACHANDRA, A; ARIF, y Fazmah; ANGGIS, S Novian. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY (ICOICT), 4., 2016, Bandung. **Intrusion Detection System (IDS) server placement analysis in cloud computing.** Bandung: IEEE, 2016. p. 1 - 5.

ABLIZ, Mehmud; ZNATI, Taieb F. In: 2015 INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS SECURITY AND PRIVACY (ICISSP), 1., 2015, Loire Valley. **Defeating DDoS using Productive Puzzles.** Loire Valley: IEEE, 2015. p. 1 - 10.

BHARDWAJ, Akashdeep et al. In: INTERNATIONAL CONFERENCE CLOUD SYSTEM AND BIG DATA ENGINEERING (CONFLUENCE), 6., 2016, Noida. **Solutions for DDoS attacks on cloud.** Noida: IEEE, 2016. p. 1 - 5.

CHUNG, Lawrence. **Client-Server Architecture.** 2000. Disponível em: <https://www.utdallas.edu/~chung/SA/2client.pdf>. Acesso em: 08 out. 2016.

F5 NETWORKS. **DDoS Protection:** Security Solutions. 2016. Disponível em: <https://f5.com/products/security/distributed-denial-of-service-ddos-protection>. Acesso em: 09 dez. 2016.

KUMAR, G. Dileep; SINGH, Manoj Kumar; JAYANTHI, M. K.. **Network Security Attacks and Countermeasures:** Advances in Information Security, Privacy, and Ethics. 2. ed. Igi Global, 2016. 357 p.

GARG, Akash; MAHESHWARI, Prachi. In: INTERNATIONAL CONFERENCE ON ADVANCED COMPUTING AND COMMUNICATION SYSTEMS (ICACCS), 3., 2016, Coimbatore. **Performance analysis of Snort-based Intrusion Detection System.** Coimbatore: IEEE, 2016. p. 1 - 5.

GOODIN, Dan. **Record-breaking DDoS reportedly delivered by >145k hacked cameras:** Once unthinkable, 1 terabit attacks may soon be the new normal. 2016. Disponível em: <http://arstechnica.com/security/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/>. Acesso em: 19 nov. 2016.

HOCK, Filip; KORTIŁ, Peter. In: INTERNATIONAL CONFERENCE ON EMERGING ELEARNING TECHNOLOGIES AND APPLICATIONS (ICETA), 13., 2015, Stary Smokovec. **Commercial and open-source based Intrusion Detection System and Intrusion Prevention System (IDS/IPS) design for an IP networks.** Stary Smokovec: Ieee, 2016. p. 1 - 4.

HOFF, Todd. **The Secret to 10 Million Concurrent Connections:** The Kernel is the Problem, Not the Solution. 2013. Disponível em: <http://highscalability.com/blog/2013/5/13/the-secret-to-10-million-concurrent-connections-the-kernel-i.html>. Acesso em: 08 out. 2016.

JUELS, Ari; BRAINARD, John. **Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks.** 1999. Disponível em: <http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/juels.pdf>. Acesso em: 05 nov. 2016.

KAMAL, Mohamed. **TCP Protocol:** Three-way Handshake. 2016. Disponível em: <http://www.networkers-online.com/blog/2016/05/tcp-protocol-three-way-handshake/>. Acesso em: 09 dez. 2016.

KAVAN, Daniel; l·KODOVÁ, Klára; KLÍMA, Martin. In: INTERNATIONAL CONFERENCE ON SECURITY AND CRYPTOGRAPHY (SECRYPT), 11., 2014, Viena. **Network-based intrusion prevention system prototype with multi-detection.** Viena: Ieee, 2014. p. 1 - 9.

PAYÃO, Felipe. **Quebrando a internet:** estamos sofrendo o maior ataque DDoS da história. 2016. Disponível em: <http://www.tecmundo.com.br/ataque-hacker/110842-grande-ataque-ddos-afeta-twitter-psn-spotify-outros-estragos.htm>. Acesso em: 19 nov. 2016.

SCUDLAYER. **The difference between DoS and DDoS.** 2015. Disponível em: <https://www.scudlayer.com/blog/en/2015/05/the-difference-between-dos-and-ddos>. Acesso em: 08 dez. 2016.

SNORT. **Snort:** Network Intrusion Detection & Prevention System. 2016. Disponível em: <https://www.snort.org/>. Acesso em: 19 nov. 2016.

SPECHT, Stephen M.; LEE, Ruby B.. **Distributed Denial of Service:** Taxonomies of Attacks, Tools and Countermeasures. In: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON SECURITY IN PARALLEL AND DISTRIBUTED SYSTEMS, 2004. p. 543 - 550.

STALLINGS, William. **Criptografia e Segurança de Redes:** Princípios e Práticas. 4. ed. São Paulo: Pearson Prentice Hall, 2008.

WEBER, Marc. **CHM Fellow Douglas C. Engelbart.** 2013. Disponível em: <http://www.computerhistory.org/atchm/chm-fellow-douglas-c-engelbart/>. Acesso em: 08 out. 2016.