CURSO DE SISTEMAS DE INFORMAÇÃO

IDENTIFICAÇÃO E ANÁLISE DE FLOODING TCP SYN EM REDES IEEE 802.3 UTILIZANDO LÓGICA FUZZY

FILIPE NUNES ABRANTES

Palmas - TO

2018

CURSO DE SISTEMAS DE INFORMAÇÃO

IDENTIFICAÇÃO E ANÁLISE DE FLOODING TCP SYN EM REDES IEEE 802.3 UTILIZANDO LÓGICA FUZZY

FILIPE NUNES ABRANTES

Trabalho apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob a orientação do professor Me. Douglas Chagas da Silva.

Palmas - TO

2018

CURSO DE SISTEMAS DE INFORMAÇÃO

IDENTIFICAÇÃO E ANÁLISE DE FLOODING TCP SYN EM REDES IEEE 802.3 UTILIZANDO LÓGICA FUZZY

FILIPE NUNES ABRANTES

Trabalho apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob a orientação do professor Me. Douglas Chagas da Silva.

Prof. Me. Douglas Chagas da Silva
Orientador

Prof. Me. Marco Antonio Firmino Sousa
Convidado 1

Prof.ª Me. Stéphany Moraes Martins
Convidado 2

Palmas - TO
2018

Aos meus pais e meus irmãos, pelo apoio e incentivo.

Agradecimentos

Agradeço primeiramente à Deus, que tem me sustentado com sua perfeita graça e benevolência, desde o nascimento até o presente momento, no qual finalizo a primeira etapa da grande jornada em Sistemas de Informação.

Agradeço aos meus pais e meu irmão Jonathas, que sempre me deram apoio e em especial, minha irmã Andréa, a qual tem me ajudado em todos os âmbitos durante minha graduação na cidade de Palmas.

Agradeço à minha namorada e futura esposa Laura, pela compreensão nos momentos difíceis, pelas palavras de incentivo quando o desânimo ganhava força, pelas ausências e por todo apoio proporcionado durante o curso.

Agradeço aos amigos que conheci nesta jornada, embora não os veja constantemente, firmamos uma amizade duradoura, em especial meu amigo Lucas Góis, que sempre abriu as portas da sua casa nos momentos que precisei e Apolyane Farias, pela companhia diária e insistência para que eu terminasse este projeto.

Agradeço ao Mestre Douglas Chagas, pela disponibilidade na orientação deste trabalho e em não hesitar em repassar seu amplo conhecimento.

À todos que contribuíram direta ou indiretamente para minha formação: muito obrigado!

"Aqueles que se sentem satisfeitos sentam-se e nada fazem.
Os insatisfeitos são os únicos benfeitores do mundo."
(Walter S. Landor)

Resumo

A grande difusão de serviços tem despertado a atenção para assuntos de segurança, pois é neste ramo que um administrador de redes ou gestor de TI deve adotar e prover políticas para que acessos não autorizados, roubo de informações ou negação de serviços sejam mitigados, a fim de manter sempre disponível aquilo que deseja. Este trabalho consiste em identificar e classificar o ataque distribuído de negação de serviço, o famoso *DDoS* em redes *IEEE* 802.3, *ethernet*, tendo enfoque no ataque *flood TCP SYN*, provendo um módulo genérico e adaptável para que mecanismos de defesa sejam implementados agregando esta classificação. Para tal, utiliza-se um cenário específico, onde o agente malicioso está dentro da rede local e ataca os serviços da sua própria instituição. Para que seja possível classificar se há ou não ataque, é feita uma análise de todo tráfego da rede e após filtragem, são geradas entradas para o sistema baseado em lógica *fuzzy*.

Palavras-chaves: Análise de tráfego; Ataque *TCP SYN*; Lógica *Fuzzy*.

Abstract

The wide spread of services has raised the attention for security matters, as it is in this field that a network administrator or IT manager must adopt and provide policies so that unauthorized access, information theft or denial of services are mitigated in order to always keep what you want. This work consists of identifying and classifying the distributed denial-of-service attack, the famous DDoS on IEEE 802.3, ethernet networks, focusing on the TCP flood attack, providing a generic and adaptive module for defense mechanisms to be implemented by aggregating this classification. To do this, a specific scenario is used, where the malicious agent is inside the local network and attacks the services of its own institution. In order to be able to classify whether or not there is an attack, an analysis is made of all network traffic and after filtering, entries are generated for the system based on fuzzy logic.

Key-words: *Traffic Analysis; TCP SYN Attack; Fuzzy Logic.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACK - *Acknowledge* (Reconhecimento)

CPU - *Central Processing Unit* (Unidade Central de Processamento)

CSV - *Comma-Separated Values* (Valores Separados Por Virgula)

DDoS - *Distributed Denial of Service* (Ataque Distribuído de Negação de Serviço)

DMZ - *Demilitarized Zone* (Zona Desmilitarizada)

DNS - *Domain Name System* (Sistema de Nomes de Domínio)

DoS - *Denial of Service* (Negação de Serviço)

ICMP - *Internet Control Message Protocol*(Protocolo de Mensagens de Controle da Internet)

IEEE - *Institute of Electrical and Electronic Engineers* (Instituto de Engenheiros Elétricos e Eletrônicos)

IP - *Internet Protocol* (Protocolo de Internet)

IPFW - *IP Firewall*

ISO - *International Organization for Standardization* (Organização Internacional de Padronização)

LAN - *Local Area Network* (Rede Local)

OSI - *Open System Interconnection* (Sistema Aberto de Interconexões)

POM - Problema de Otimização Multiobjetivo.

RX - Recepção de pacotes

SMLI - *Statefull Multi-Layer Inspection* (Inspeção multicamada com estado)

SO - Sistema Operacional

SYN - *Syncronize* (Sincronizar)

TCP - *Transmission Control Protocol* (Protocolo de Controle de Transmissão)

TX - Transmissão de pacotes

Sumário

1 Introdução

No campo de redes, a área de segurança consiste em adotar e prover políticas para prevenir e monitorar o acesso não autorizado, uso incorreto, modificação ou negação de serviços em que estão associadas. Cabe ao gestor da rede monitorar as atividades, estruturá-la e fornecer mecanismos de prevenção e combate à qualquer anomalia, desde um gargalo ocasionado por *loop* de roteamento (condição em que um pacote é transmitido por vários roteadores sem conseguir alcançar sua rede ou hospedeiro de destino, causando a queda da mesma) ou ataques que comprometam todo o serviço (MCNAB, 2017). A identificação e análise de *flooding* de rede é feita através do monitoramento contínuo e em tempo de execução, proporcionando maior segurança na tomada de decisão.

Este trabalho objetiva realizar a identificação e análise de anomalias em redes *ethernet* (IEEE 802.3), no que tange a verificação de possíveis ataques distribuídos de negação de serviço, o famoso *DDoS*, com enfoque no *flood TCP SYN*, dentro de um cenário específico de rede. Para tal, serão utilizadas ferramentas e aplicações para obtenção das métricas de uso de recursos computacionais e de rede, para verificação de incidência de ataques. Utiliza-se ainda analisadores de tráfego de rede como coletor de informações mais detalhadas dos pacotes, as quais são fornecidas como entrada de um modelo de avaliação, com o intuito de auxiliar o processo de tomada de decisão e verificação de ocorrência de atividades relacionadas à ataques que objetivam a indisponibilidade dos serviços. Utiliza-se ainda de técnicas baseadas em lógica *fuzzy* (difusa, nebulosa), que admite pertinências variantes entre 0 e 1 (RIOS, 2012), de maneira a propiciar uma análise que considera incertezas.

O trabalho está organizado da seguinte forma: no capítulo 1 tem-se as abordagens iniciais, justificativas e objetivos desta pesquisa. No capítulo 2 é apresentada uma explanação sobre as técnicas de análise de rede, explicando como funciona o *DDoS* - (*Distributed Denial* of *Service*) e apresentação da lógica *fuzzy* com breve demonstração das suas funções. No capítulo 3 está descrita a metodologia deste trabalho, cenário e como todos os testes de coleta de dados, tomada de decisão e aplicação da solução foram realizados. Em seguida, tem-se as discussão dos resultados obtidos no capítulo 4; E por fim, no capítulo 5 tem-se as considerações finais e perspectivas de trabalhos futuros.

1.1 Justificativa

A constante evolução tecnológica, bem como a mobilidade e convergência dos mais variados tipos de dispositivos para acesso à internet, tem levantado questões relevantes em relação à segurança desses sistemas, algo diariamente explorado pelos veículos de comunicação.

É sabido que não há solução definitiva contra o ataque do tipo *DDoS*, pois ainda que haja inúmeros mecanismos de defesa, uma das dificuldades da reação a ele é o bloqueio de acessos legítimos aos serviços da rede. Ter um bom monitoramento é, sem dúvida, essencial para a proteção de ataques como estes, mas cada solução de prevenção e combate ao ataque varia a cada cenário de infraestrutura e organização da rede e servidores (SILVA, 2016).

Normalmente lê-se na literatura sobre *DDoS* de forma externa, na qual o ataque vem de fora e atinge uma organização ou instituição. Este trabalho aborda um contexto mais específico, onde o atacante está na própria instituição, podendo ser um aluno mal intencionado ou um funcionário descontente, com o intuito de prejudicar a empresa. Assim, tendo como foco a segurança de redes, especificamente sobre ataques de negação de serviço (*Denial of Service*, *DoS*), que este trabalho visa demonstrar a experimentação e junção de técnicas já existentes no intuito de identificar e analisar possíveis ataques de negação de serviço, provendo classificação adequada para que mecanismos de defesa, como *firewall* de camada 3 e 4, possam agir neste cenário específico.

1.2 Objetivos do Trabalho

1.2.1 Objetivo Geral

Identificar e analisar o *flooding* de rede (ataque *TCP SYN*) utilizando técnicas baseadas em lógica *fuzzy*, de maneira a auxiliar no processo de tomada de decisão, provendo a classificação adequada, propiciando a agregação de mecanismos de defesa dos sistemas.

1.2.2 Objetivos Específicos

- Analisar o tráfego da rede;
- Identificar a ocorrência de *flooding TCP SYN* em um cenário específico de rede;
- Avaliar os possíveis ataques, distinguindo-os de acessos legítimos;
- Realizar experimentos utilizando um ambiente de testes, simulando situações reais de acesso.

2 Referencial Teórico

2.1 Gerenciamento de Redes

O grande consultor e palestrante, largamente conhecido por seus trabalhos de gestão, William Edwards Deming, tem uma frase que diz assim: “Não se gerencia o que não se mede, não se mede o que não se define, não se define o que não se entende, não há sucesso no que não se gerencia” (COUTO, 2012). Ao tratar-se de redes, não é diferente. É necessário haver forte gestão para que ações preventivas e/ou corretivas sejam tomadas.

A medida que a Tecnologia da Informação foi evoluindo, ações de integralização tornaram-se imprescindíveis, a fim de criar modelos e padrões para interconectar os diversos tipos de sistemas e possibilitar a troca de informações entre dispositivos (MCNAB, 2017). Então, para suprir essa necessidade, a *ISO*[1] - *International Organization* for *Standardization* - organização não governamental conhecida por criar padrões e normalização, desenvolveu o modelo de Interconexão de Sistemas Abertos - *OSI* - *Open Systems Interconnection* - com a proposta de organizar as redes de computadores em sete camadas (KUROSE JIM. ROSS, 2013).

Figura 1 – Modelo de referência *OSI*. (KUROSE JIM. ROSS, 2013)

A Figura 1 demonstra o modelo de referência *OSI*, desenvolvido pela *ISO*, com as sete camadas específicas, que proporcionam a comunicação entre diversos dispositivos distintos, definindo diretivas genéricas.

Há ainda cinco áreas de padronização na gerência de redes, sendo elas: Gerência de Falhas, Gerência de Contabilização, Gerência de Configuração, Gerência de Desempenho e Gerência de Segurança (KUROSE JIM. ROSS, 2013). De forma resumida, estas são áreas que compreendem características nas quais o usuário final busca em um sistema, que é ter segurança na rede, interface amigável, que sempre esteja disponível e seja de baixo custo. Além disso,

[1] https://www.iso.org/about-us.html

essas áreas permitem a verificação, manutenção e provimento de qualidade de rede, buscando a correção contínua de eventuais problemas (KUROSE JIM. ROSS, 2013).

2.1.1 Análise de Tráfego de Rede

A análise de tráfego da rede possibilita a verificação de diversos fatores a fim de encontrar a causa de possíveis problemas na rede. Dentre estas possibilidades, estão (KUROSE JIM. ROSS, 2013): Encontrar pontos de bloqueio na rede; Detectar anomalias na rede; Identificar equipamentos e cabeamento defeituosos e observar *logs* de erro e outras mensagens não mostradas pelas aplicações.

Para analisar uma rede, é necessário prévio conhecimento a respeito do modelo de referência *OSI* e seus protocolos. Além de todos os tipos de gerências citadas, há ainda o monitoramento, imprescindível para verificar o andamento de todas as atividades da rede, fluxo de pacotes, desempenho, velocidade de transmissão de *links*, dentre outros (BULLOCK JESSEY. PARKER, 2017).

O monitoramento dentro do contexto de gerenciamento de redes é classificado como ativo e passivo. No passivo, não há interferência direta na rede, ocorre apenas a observação do fluxo de dados e coleta de informações para análise. No ativo, ocorre a manipulação de pacotes, ou seja, há um teste controlado, com quantificação da carga de dados necessária a fim de propiciar análise do desempenho da rede (BULLOCK JESSEY. PARKER, 2017).

2.1.2 Ferramentas de Análise de Tráfego

De acordo com Bullock Jessey. Parker (2017), um analisador de pacotes é um *software* que tem por finalidade monitorar, interceptar e capturar os dados trafegados na rede, a fim de possibilitar uma análise mais detalhada do seu conteúdo através da decodificação do mesmo.

Este tipo de ferramenta atua a partir da camada de Enlace (2) do modelo *OSI* e vai até à camada de Aplicação (7). Há ainda analisadores que atuam na camada Física, mas são componentes de *hardware*, como testadores de cabos, entre outros (BULLOCK JESSEY. PARKER, 2017).

Existem diversas ferramentas de análise de tráfego, algumas gráficas como o *Wireshark*[2] e outras em linha de comando, como o *mtr*[3] e *tshark*[4]. O *tcpdump*[5] entra também na lista de analisadores poderosos, pois apresenta as mesmas funcionalidades do *Wireshark*, porém não dispõe de uma interface gráfica nativa. Ele é mais indicado para implantação em servidores Unix, preparados para análise do tráfego da rede, com finalidades específicas. De acordo com o próprio site da ferramenta, o *mtr* é uma aplicação para o Sistema Operacional (SO) *Windows* e combina

[2] https://www.wireshark.org/
[3] http://winmtr.net/
[4] http://www.tshark.org/
[5] https://www.tcpdump.org/manpages/tcpdump.1.html

funcionalidades de *ping*[6] - utilitário que faz uso do protocolo *ICMP* com o objetivo de testar a conectividade de dispositivos - e *traceroute*[6] - utilitário que informa o caminho percorrido pelo pacote da origem até seu destino (MELO, 2017). Existem ainda mais duas ferramentas que não poderiam passar despercebidas: *ethtool*[7] e *iptraf-ng*[8]. O *ethtool* é uma ferramenta de linha de comando usada para fazer ajustes específicos nas interfaces de rede, além de demonstrar todas as informações de cada uma. O *iptraf-ng* é um *fork* do *iptraf*. Também é uma ferramenta baseada em console, onde exibe as informações e estatísticas de monitoramento, tráfego, falhas e demais dados sobre os protocolos.

O *software Wireshark* é um analisador de protocolo, de código aberto, que permite a captura do tráfego de uma rede em tempo real. Isto é feito através de uma das interfaces de rede que é selecionada no momento da captura. Um diferencial deste *software* é a facilidade na criação de filtros para os pacotes capturados, podendo ser expressões regulares ou seleção de opções disponíveis em sua interface (BULLOCK JESSEY. PARKER, 2017).

Este tipo de *software*, também chamado de *Sniffer* (farejador), que analisa possíveis anomalias nas redes, detecção de gargalos, verificação de mensagens trocadas, confirmação se as senhas utilizadas estão sendo realmente criptografadas e dentre outras funções realizadas que auxiliam o administrador de redes na tomada de decisões quanto à segurança da rede (BULLOCK JESSEY. PARKER, 2017).

2.2 Distributed Denial of Service (DDoS)

Abordando o contexto histórico do *DDoS* - ataque distribuído de negação de serviço, têm-se os primeiros registros datados em 1999, no entanto, tornou-se de fato uma ameaça cibernética em meados de 2000, quando *hackers* atacaram os sites do EBay, Amazon, CNN e Yahoo, deixando-os inoperantes por um período de tempo. Logo após, já no Brasil, estes mesmos ataques de negação de serviço foram registrados contra os sites da UOL, Globo e IG (SOLHA; TEIXEIRA; PICCOLINI, 2000).

2.2.1 Definição

Antes de definir como consiste o ataque, é necessário fazer uma correlação com Segurança da Informação, que visa preservar dados de algo ou alguém. Esta, tem cinco pilares fundamentais: confidencialidade, autenticidade, integridade, disponibilidade e não repúdio (*ISO* 27000). No cenário específico em que há *DDoS*, a integridade e disponibilidade são fatores comprometidos (SILVA, 2016).

[6] https://linux.die.net/man/8/ping
[6] https://linux.die.net/man/8/traceroute
[7] https://linux.die.net/man/8/ethtool
[8] http://iptraf.seul.org/about.html

O *DDoS* consiste no envio indiscriminado de requisições a um computador alvo, visando causar a indisponibilidade dos serviços (sistemas, páginas *web*, etc.), impedindo usuários legítimos acessarem os mesmos. Ressalta-se ainda que, no *DDoS*, não existe a tentativa de roubo de informações, apenas interromper o serviço. O *DDoS* é um ataque coordenado, disseminado usando uma grande quantidade de *hosts* comprometidos, chamados de zumbis, através de código malicioso (MCNAB, 2017).

Em uma fase inicial, o invasor identifica as vulnerabilidades em uma ou mais redes para a instalação de programas de *malware* em várias máquinas para controlá-las remotamente. Posteriormente, o atacante explora esses hospedeiros comprometidos para enviar pacotes de requisição para o alvo, podendo ser um servidor *web*, um site e quaisquer sistemas que estejam ao alcance da rede externa de computadores (MCNAB, 2017).

2.2.2 O Ataque

Para entender como funciona o ataque *DDoS*, é necessário demonstrar os agentes envolvidos em todo contexto, bem como as nomenclaturas utilizadas para identificá-los. *Cracker* atacante, é quem efetivamente cria e coordena o ataque; os *masters* atua como *controllers* (controladores) que recebem os parâmetros para o ataque e comanda os hospedeiros remotamente; adiante, têm-se os *hosts* zumbis, que são máquinas "sequestradas" através de um programa ou código malicioso que fora instalado sem a percepção do usuário e por fim, o alvo do ataque, podendo ser um site, um servidor *web*, entre outros (SADOK D. FEITOSA, 2008).

A partir do acesso às máquinas, é criada uma *botnet*, que é uma rede de *bots*[7] interligados com a finalidade de executar ações, que neste caso, são maliciosas, concretizando o *DDoS* (SILVA, 2016). Estas ações são controladas pelo *master*.

A Figura 2 ilustra o cenário do ataque *DDoS*, onde existe o atacante, que controla os *masters*, que são controladores, facilitando a distribuição do código malicioso para os *bots* e os *hosts* zumbis, que formam a chamada *botnet*, uma rede infectada, controlada remotamente pelo *cracker*, a fim de executar o ataque no alvo, também ilustrado.

2.2.3 Tipos de ataque

Após a tomada das máquinas, conforme ilustrado na Figura 2, cada zumbi é controlado pelo *master* para fazer um ataque *DoS* (*Denial of Service*) negação de serviço simples, que é configurado como o mesmo tipo de ataque, só que deste modo, estará distribuído e o impacto será bem maior (BARDAL, 2014). A Tabela 1, extraída de (BARDAL, 2014), demonstra alguns tipos de ataques de negação de serviço:

[7] Saiba mais sobre bots em: <https://www.cnet.com/how-to/what-is-a-bot/>

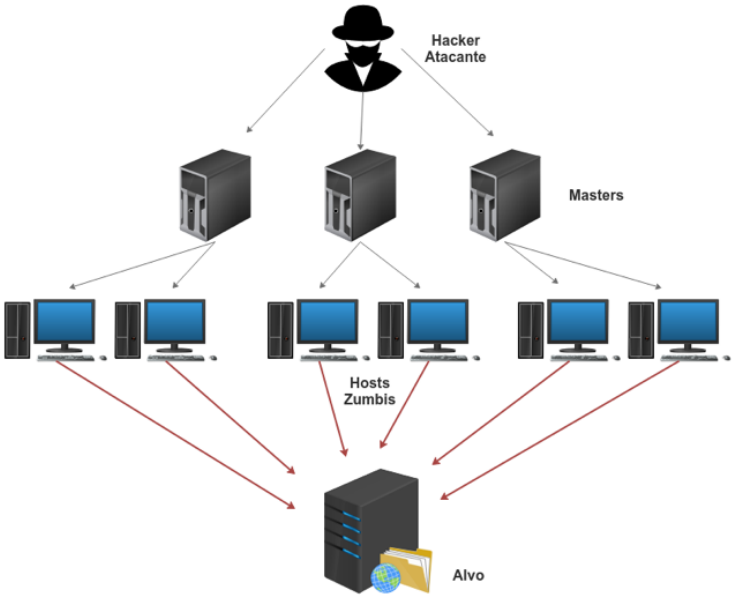


Figura 2 – Esquema de um ataque *DDoS*. (SADOK D. FEITOSA, 2008)

Tabela 1 – Tipos de ataque de negação de serviço. Liska e Stowe (2016)

TIPO DE ATAQUE	CARACTERÍSTICA (IMPACTO)
Baseado em conexões	Este objetiva consumir todas as conexões TCP do alvo, podendo atingir uma página *web*, um sistema ou qualquer outro serviço.
Volumétrico	Este tipo de ataque consome todo o tráfego disponível do alvo, ocasionando no isolamento do serviço.
Fragmentação	Um grande número de pacotes fragmentados são enviados à vítima e esgotando seu processamento, como memória, processador, disco. Sem poder para remontar os pacotes, o alvo fica indisponível

Há ainda uma técnica que amplifica um ataque. Esta utiliza servidores *DNS* com pouca restrição de uso. O *cracker* atacante frauda o endereço *IP* da vítima e envia uma pequena solicitação ao servidor. Este por sua vez, irá responder com um pacote muito grande direcionado ao alvo. Fazendo isto de forma distribuída, é impossível uma vítima resistir ao excessivo volume de dados (BARDAL, 2014).

2.2.4 Ataque *TCP SYN*

O *SYN flood* ou *TCP SYN*, é um ataque que explora uma conexão segura, como o TCP, que tem uma conexão chamada *three-way handshake*, que é um aperto de mão triplo ou em três etapas. Quando se inicia uma conexão TCP, o cliente envia uma mensagem do tipo *SYN* (*Syncronize*) inicial, dizendo que quer se conectar (BHATTACHARYYA; KALITA, 2016). O servidor, estando pronto, envia uma mensagem *SYN-ACK* (*acknowledge*), que é reconhecimento.

Por fim o cliente responde com uma mensagem *ACK* e a conexão está feita, pronto para transmitir dados (BHATTACHARYYA; KALITA, 2016). A Figura 3 ilustra a conexão, em que há um número de sequência que sempre é acrescido quando a mensagem é enviada.

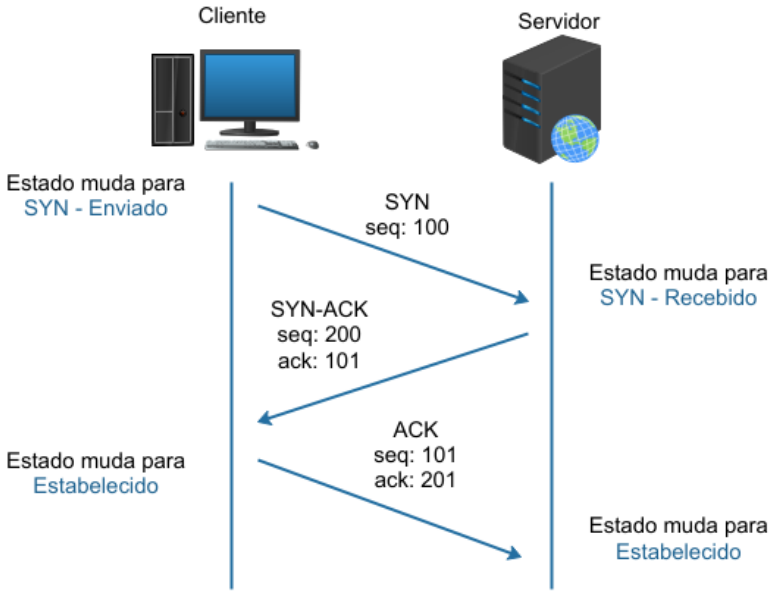


Figura 3 – Conexão TCP. Adaptado de (IBRAHIM, 2011)

.

Diante do exposto, o ataque *TCP SYN* ocorre quando um agente malicioso implementa de forma incompleta o protocolo TCP, a fim de que não envie a última mensagem ACK, necessária para completar o *three-way handshake*. Deste modo, o servidor ficará aguardando a mensagem, pois entende-se que pode ter havido um congestionamento real (BHATTACHARYYA; KALITA, 2016).

Ao inundar o servidor com dezenas de milhares de pacotes, fica inviável manter o serviço estável, causando imediatamente a negação de serviço. Esta técnica consome recursos de memória e processamento (MCNAB, 2017). O ataque ao processamento dá-se pelo simples fato do atacante gerar mais pacotes que o alvo pode processar. Quanto à memória, sua inundação ocorre pelo fato do alvo armazenar algumas informações sobre a conexão *TCP*, como *IP*, porta, estado de conexão, dentre outros. Para isso, é necessário alocar um espaço na memória. A quantidade excessiva de pacotes disparados causa o *flood* e a negação de serviço (BHATTACHARYYA; KALITA, 2016).

A Figura 4 ilustra o ataque *TCP SYN*, onde *cracker* atacante envia diversas requisições ao servidor, que responde com um *SYN-ACK* mas nunca recebe o *ACK* de volta. Quando um acesso cliente legítimo tenta acessar, ocorre a negação de serviço.

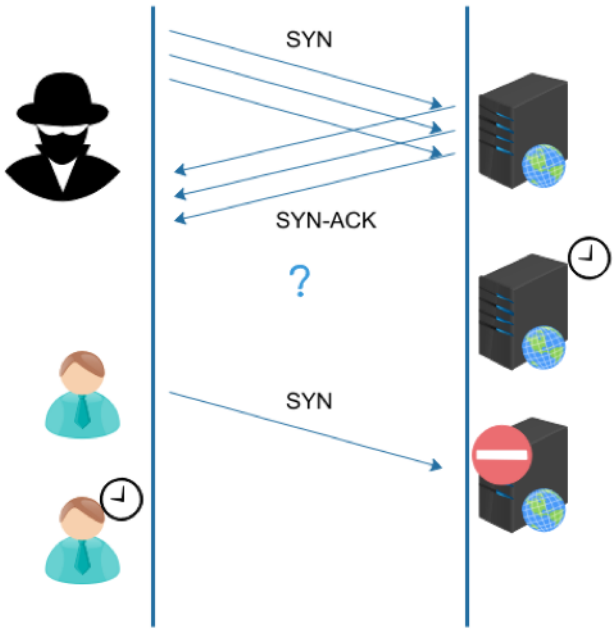


Figura 4 – Ataque *TCP SYN*. Adaptado de (IBRAHIM, 2011)

2.3 Lógica Fuzzy

A lógica *fuzzy* ou lógica difusa, foi incorporada em 1965 por Lofti Asker Zadeh, professor de Ciência da Computação da Universidade da Califórnia, publicando seu artigo (*Fuzzy Sets*[8], 1965). Após este primeiro passo, Zadeh e outros continuaram com o desenvolvimento da lógica *fuzzy*, entretanto a ideia não foi bem aceita dentro dos círculos acadêmicos, uma vez que algumas das ferramentas matemáticas subjacentes não foram exploradas na pesquisa (MARRO ALESSANDRO ASSI, 2010).

Apesar da não aceitação impedir o desenvolvimento da lógica *fuzzy* pela academia científica, posteriormente ocorreu um estouro no Japão e foi largamente implementada. A partir disso, continuou difundindo-se pelo resto do mundo e hoje é amplamente utilizada em diversos trabalhos, nos mais variados segmentos acadêmicos (MARRO ALESSANDRO ASSI, 2010).

2.3.1 Definição

É uma lógica diferente da clássica, booleana, que admite valores lógicos verdadeiro ou falso (0 ou 1). A lógica *fuzzy* trata valores variantes e subjetivos, entre 0 e 1, podendo assim ter uma pertinência com valor 0.1, 0.5 e 0.9 sendo quase falso, meio verdadeiro e quase verdadeiro, respectivamente. Este conceito surgiu devido à necessidade de lidar com situações complexas e tratar incertezas (RIOS, 2012);(RIGNEL D. G. DE S., 2011).

Para exemplificar, utilizando a lógica tradicional, uma pessoa com 34 anos de idade só iria pertencer ao grupo de meia-idade, considerado de 35 a 55 anos, se efetivamente completasse

[8] https://www.sciencedirect.com/science/article/pii/S001999586590241X

o 35º aniversário. Neste caso, as pessoas com 34 e 56 anos estariam fora deste grupo. A Figura 5 ilustra o exemplo, pressupondo a lógica tradicional (RIGNEL D. G. DE S., 2011).

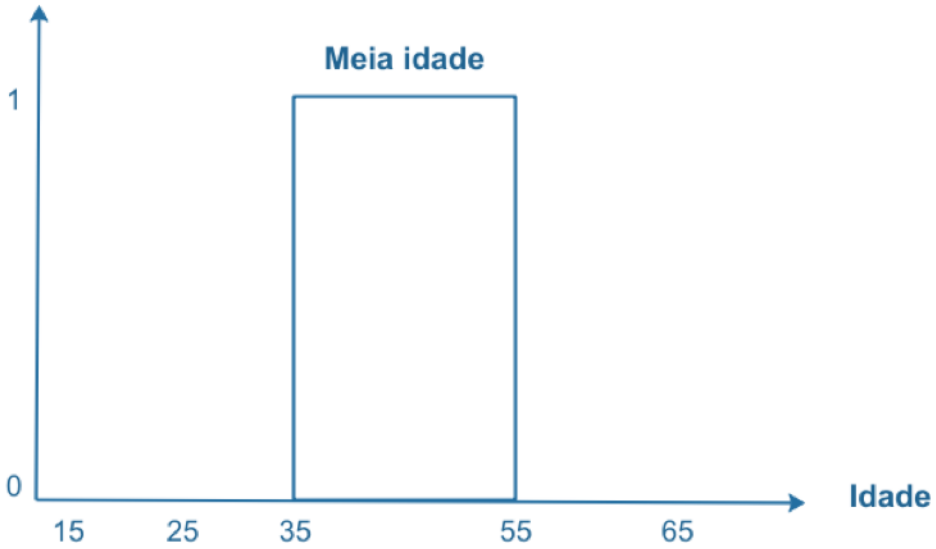


Figura 5 – Definição de meia-idade em conjuntos convencionais. (RIGNEL D. G. DE S., 2011)

Na Figura 6, é demonstrada a definição de meia-idade baseando na teoria *fuzzy*. O grau de pertinência neste caso inicia com uma pessoa de 25 anos, que começa a pertencer ao grupo, mas não totalmente, enquanto uma de 45 anos está ao auge da sua meia-idade e conforme vai aumentando a pertinência, mais a curva vai tendendo a baixar e sair.

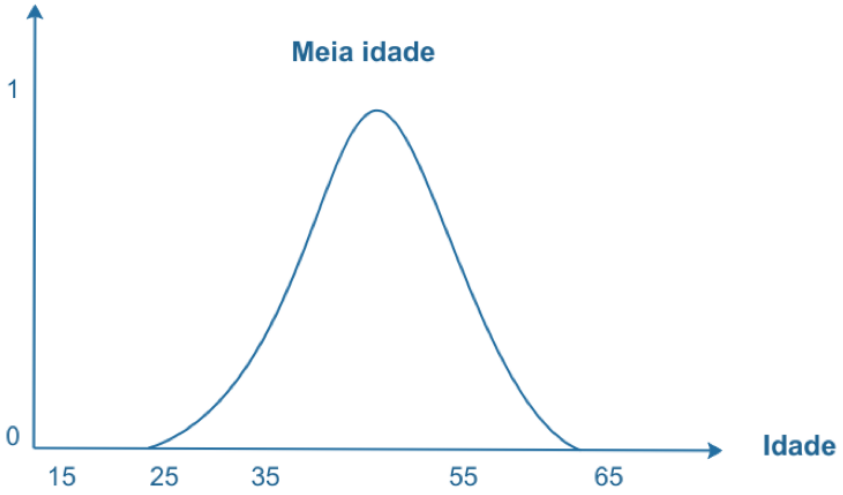


Figura 6 – Definição de meia-idade baseando-se na lógica *fuzzy*. (RIGNEL D. G. DE S., 2011)

2.3.2 Arquitetura de um sistema utilizando lógica fuzzy

Após conceituar e exemplificar alguns elementos da lógica *fuzzy*, serão explanadas as quatro partes básicas que compõem um sistema *fuzzy* (RIOS, 2012), (SANTOS, 2009) e (SOUSA, 2005): Entrada: são os dados coletados a partir da análise de um cenário específico,

sendo numéricos, nos quais o sistema irá se basear para a tomada de decisão; Fuzzificação: processo de transformação dos dados em informações *fuzzy*; Sistema *Fuzzy*: composto da base de conhecimento, devidamente treinada e tomador lógico de decisões; Defuzzificação: transforma de volta os dados processados na parte anterior em informações numéricas e inteligíveis. A Figura 7 ilustra a arquitetura.

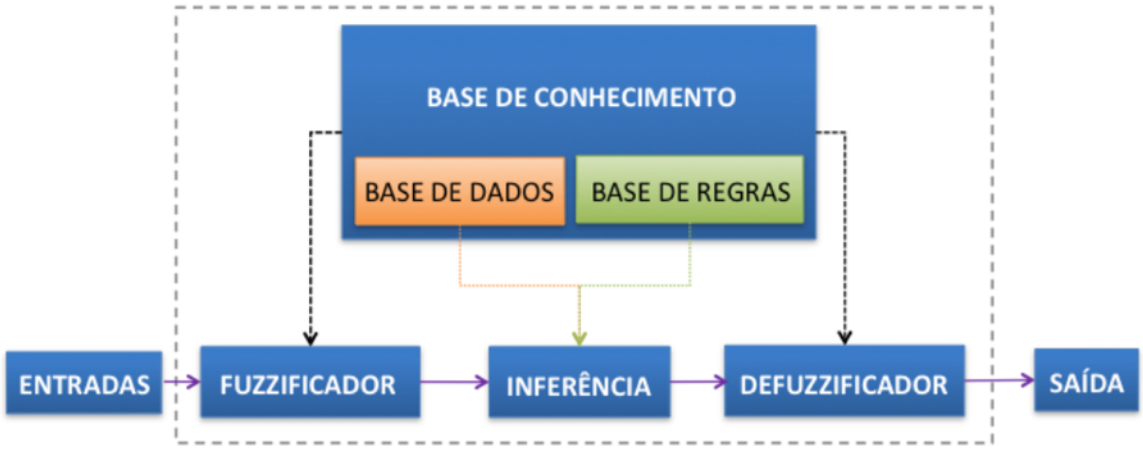


Figura 7 – Representação de um sistema *fuzzy* (RIOS, 2012)

2.3.3 Inferência Nebulosa

Após introduzir os dados a serem tratados com a lógica *fuzzy*, dentro do raciocínio lógico, ocorre a inferência nebulosa, que também é a passagem pelo fuzificador. Esta fase visa a aplicação de regras no modelo *fuzzy*, que consiste em *IF* e *THEN* (SE isso ENTÃO isso), sendo que todo o conjunto da sentença é uma informação difusa (PEDRYCZ W. GOMIDE, 2007). Há ainda a agregação, que calcula a importância de uma determinada regra para a situação corrente e a composição, que mede a influência de cada regra nas variáveis de saída. A partir disso, torna-se mais viável a implementação de fluxos que façam interação com o conhecimento humano, como por exemplo:

- SE a quantidade de requisições ao servidor for maior que 5000 e for classificado como um ataque ENTÃO ative os mecanismos de defesa.

É importante destacar que o exemplo acima denota o uso da lógica *fuzzy* com uma regra inteligível, que poderia ser usada em um sistema comum, mas o que muda é que os conjuntos *fuzzy* permitem graus de pertinência, entre 0 e 1, podendo ser um falso-positivo ou um falso-negativo (PEDRYCZ W. GOMIDE, 2007).

Desta forma, o sistema possui um conjunto de regras que podem aparentar validade simultânea, mesmo que sejam contraditórias, então a solução é obtida através da agregação dos

resultados, alcançados por um modelo matemático, utilizando uma das funções de pertinência que serão demonstradas na seção 2.3.6

Após determinar as regras utilizadas no sistema *fuzzy*, precisa-se adotar um método para o processo de decisão. Dentre os existentes, há quatro principais que se destacam na literatura: Mamdani, Larsen, Tsukamoto e Takagi e Sugeno (RIOS, 2012).

A Figura 8 apresenta o método Mamdani, que é o mais utilizado. Esté método baseia-se em uma estrutura simples com operações de MIN-MAX, seguindo a seguinte regra de inferência:

Se x é A e y é B Então z é C

Observa-se que o método Mamdani seleciona valores mínimos MIN dos antecedentes e MAX dos consequentes, obtendo saídas similares (RIOS, 2012).

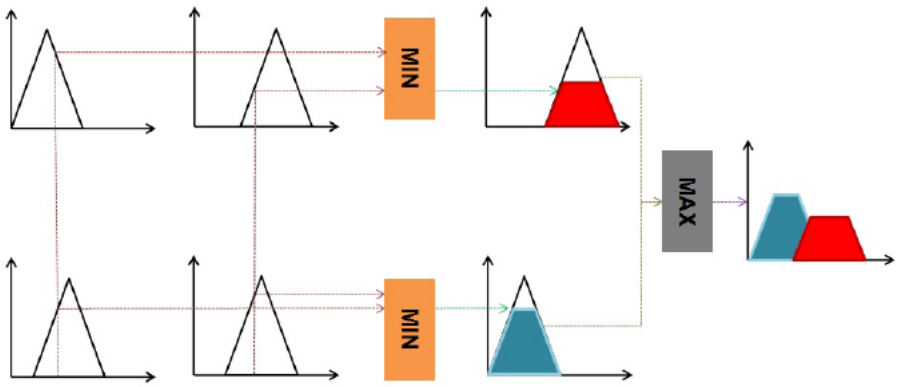


Figura 8 – Representação do método Mamdani. (RIOS, 2012)

2.3.4 Conjuntos Fuzzy

Nos conjuntos convencionais, os elementos só pertencem à ele se, e somente se, estiverem de acordo com a condição pré-estabelecida (SANTOS, 2009). Para exemplificar, a temperatura de um determinado ambiente pode ser considerada baixa, média ou alta de acordo com a temperatura em graus. A Tabela 2 ilustra o exemplo:

Tabela 2 – Ilustração da temperatura no conjunto convencional

GRAU (c)	**CONDIÇÃO DA TEMPERATURA**
25º	BAIXA
50º	MÉDIA
75º	ALTA

Como visto na Tabela 2, a temperatura só se encaixa em uma condição quando está no grau determinado, ou seja, 49,9ºc ainda seria baixa. Um conjunto de dados *fuzzy* é caracterizado por uma classe de objetos com um grau contínuo de pertinência. Este pode ser interpretado como uma ponte que liga o conceito impreciso à sua modelagem numérica, na qual atribui-se a pertinência entre 0 e 1 (VAZ, 2006).

De acordo com Sousa (2005), a função de pertinência *fuzzy* no universo *uF: U => [0,1]* associa cada elemento x que pertence ao universo U com um intervalo de 0,1, caracterizando o seu grau de pertinência, o qual representa pontos intermediários dentro do contexto.

2.3.5 Variáveis Linguísticas

De acordo com Trillas e Eciolaza (2015), as variáveis linguísticas são termos adotados por conjuntos *fuzzy*. Um exemplo básico, é a temperatura de um determinado processo, em que as variáveis linguísticas assumem valores baixa, média, alta, como ilustra a Figura 9. Estes são descritos pelos conjuntos *fuzzy*.

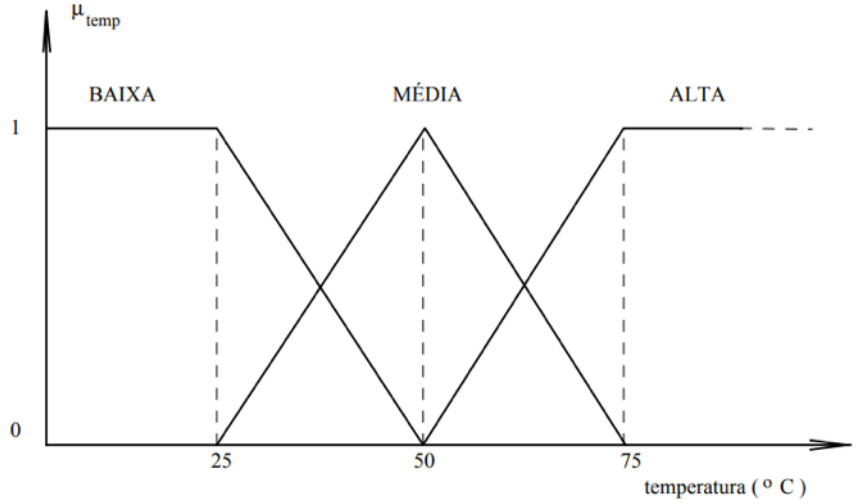


Figura 9 – Variável Linguística Temperatura. (TRILLAS; ECIOLAZA, 2015)

> A principal função das variáveis linguísticas é fornecer uma maneira sistemática para uma caracterização aproximada de fenômenos complexos ou mal definidos. Em essência, a utilização do tipo de descrição linguística empregada por seres humanos, e não de variáveis quantificadas, permite o tratamento de sistemas que são muito complexos para serem analisados através de mecanismos matemáticos convencionais (TRILLAS; ECIOLAZA, 2015, p.6) .

Há ainda uma ilustração de valores diferenciados, que podem ser expressos como alta, não alta, muito alta, bastante alta, não muito alta, alta mas não muito alta. Estes valores compostos são formados a partir de negações, conectivos e modificadores. Acompanhe a Tabela 3 (TRILLAS; ECIOLAZA, 2015), a seguir com alguns exemplos:

2.3.6 Funções de Pertinência

As funções de pertinências são modelos matemáticos responsáveis por refletir o conhecimento que se tem em relação à intensidade que um objeto pertence ao conjunto *fuzzy* (MARRO ALESSANDRO ASSI, 2010). A definição da função de pertinência é subjetiva, pois pode alterar

Tabela 3 – Termos modificadores *fuzzy*

CATEGORIA	TERMO
Termos Primários	Nomes de conjunto fuzzy especificados dentro de um universo (alto, baixo, pequeno, médio, grande).
Conectivos lógicos	Negativa: Não Conectivos: E ou OU, Conectivos implícitos: mas, porém.
Modificadores	Muito, pouco, levemente, extremamente, bastante, razoavelmente.
Delimitadores	Parênteses ( )

os resultados de acordo com sua definição. Estas funções podem assumir várias formas, sendo a trapezoidal ou a triangular, as mais utilizadas (VAZ, 2006).

A seguir serão apresentadas algumas funções de pertinência, especificamente a função triangular e gaussiana (MARRO ALESSANDRO ASSI, 2010).

2.3.6.1 Triangular

A função triangular é caracterizada por indicar um único registro, no universo de discurso, o valor máximo da pertinência (SOUSA, 2005). A triangular consiste em uma função de x com três parâmetros a, b, c, onde a e c determinam o intervalo, para que valores diferentes de zero sejam assumidos e b é o único ponto onde ocorre o grau de pertinência máximo. Essa função tem a seguinte equação (SANTOS, 2009), (RIOS, 2012):

$$(x;a,b,c) = max(min(\frac{x-a}{b-a}, \frac{c-x}{c-b})0) \tag{2.1}$$

A Figura 10 ilustra a função triangular, onde estão representados no eixo horizontal e vertical, o universo de discurso e o grau de pertinência, respectivamente. No ponto a [3] e c [8] é delimitado o intervalo onde a função irá assumir valores maiores que zero. No ponto c é onde ocorre o maior valor de pertinência.

2.3.6.2 Gaussiana

A função de pertinência gaussiana ou distribuição normal tem características por sua média e desvio padrão. Apresenta-se de forma contínua e possui decaimento suave, com valores maiores que zero em todo universo de discurso estudado (SANTOS, 2009).

A função Gaussiana tem a seguinte equação:

$$\mu gaus(x,s,c) = exp\{\frac{-(x-c)}{s}\}2 \tag{2.2}$$

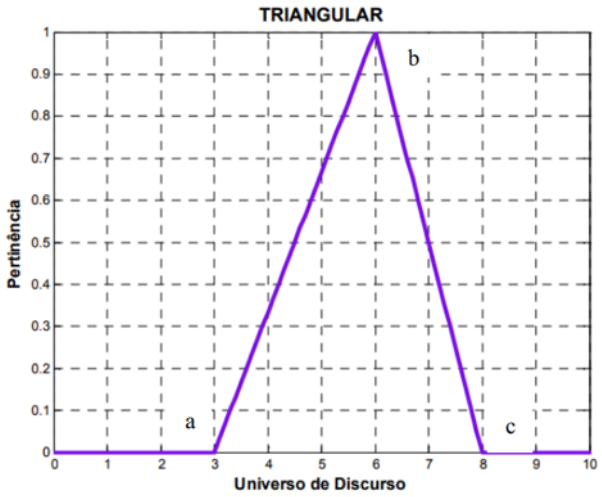


Figura 10 – Função de pertinência triangular (NETO et al., 2006)

.

A Figura 11 ilustra a curva gaussiana, com acentuação suave, atingindo o grau de pertinência máximo na média de [5]. No eixo x, a variável é contínua e concentra valores em torno de um valor central, havendo simetria entre eles.

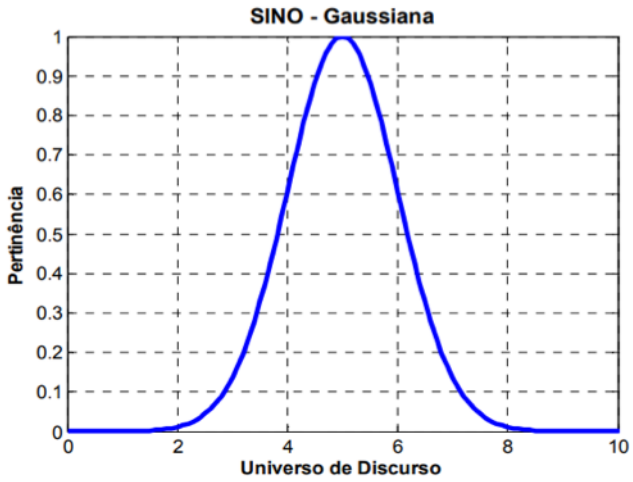


Figura 11 – Função de pertinência gaussiana (NETO et al., 2006)

.

2.3.7 Raciocínio Nebuloso

O raciocínio nebuloso implica em todo sistema ou processo lógico. Este está dividido em 5 (cinco) etapas: (1) Fuzzificação: ocorre a transformação das variáveis de entrada em valores nebulosos; (2) Aplicação dos operadores *fuzzy* como descrito no item anterior, utilizando as variáveis linguísticas. (3) Aplicação da inferência nebulosa. (4) Combinação de todas as saídas *fuzzy* e (5) Defuzzificação (TRILLAS; ECIOLAZA, 2015).

Na primeira etapa, tem-se a entrada de dados, que são associados à uma função de pertinência, viabilizando a obtenção do grau de verdade da proposição, limitando esse valor

entre 0 e 1. No segundo passo, aplica-se a inferência nebulosa, que consiste nos operadores *fuzzy*, que são o E e OU, utilizados para definir o grau máximo e mínimo da pertinência do conjunto (TRILLAS; ECIOLAZA, 2015).

Na terceira etapa, aplica-se a função de pertinência, que é o processo em si do conjunto nebuloso, cria-se implicações e as regras, como por exemplo: SE está frio ENTÃO desliga o ar condicionado. O quarto passo já traz as saídas e resultados correspondentes ao processo *fuzzy* (TRILLAS; ECIOLAZA, 2015).

Por último, no quinto passo ocorre a defuzzificação, que é a transformações dos valores dos resultados em formato inteligível numérico, partindo para tomada de decisões e/ou análises a partir de tais informações (TRILLAS; ECIOLAZA, 2015).

2.4 FIREWALL

2.4.1 Conceito de *firewall*

O *firewall* é um mecanismo de proteção na rede que pode atuar em diversos níveis, de acordo com sua implementação. Uma boa definição é dada por Kurose Jim. Ross (2013, p. 538):

> Um *firewall* é uma combinação de *hardware* e *software* que isola a rede interna de uma organização da internet em geral, permitindo que alguns pacotes passem e bloqueando outros. O *firewall* permite a um administrador de rede controlar o acesso entre o mundo externo e os recursos da rede que ele administra, gerenciando o fluxo de tráfego para esses recursos.

Em tradução literal, *firewall* significa “barreira de fogo”, corroborando sua funcionalidade como uma camada protetora, permitindo que apenas pacotes permitidos passem para o destino. Vale ressaltar que a existência de um *firewall* não é solução para todos os problemas de segurança e ainda, se mal implementado, o risco é maior do que se não houvesse um (BAQUI, 2012). Na próxima seção serão explanados os dois tipos de *firewall's* mais populares, bem como as arquiteturas clássicas de utilização dos mesmos.

2.4.2 *Firewall* filtro de pacotes

A implementação de um determinado *firewall* varia da necessidade dentro de uma topologia específica, além de critérios do desenvolvedor. O *firewall* do tipo filtro de pacotes é um dos tipos mais primitivos, que encabeçou novas arquiteturas (SOEIRA; SILVA; TABORDA, 2013).

O filtro de pacotes é metodologicamente simples, mas que cumpre bem o seu papel e oferece segurança para o cenário de rede. Este tipo de *firewall* possui uma tabela de regras predefinidas pelo gestor da rede. Todos os pacotes são inspecionados e com base nesta tabela, estes pacotes podem ou não serem descartados (SOEIRA; SILVA; TABORDA, 2013).

As filtragens ainda podem ser caracterizadas em estáticas ou dinâmicas. Na filtragem estática, o que acontece é o filtro baseado unicamente na tabela de regras, analisando cada pacote de forma independente. De acordo com necessidade dos serviços, esta regra pode ocasionar bloqueio em requisições que esperam respostas específicas. A segurança deste tipo de implementação depende de como o gestor criou sua base de regras, indicando assim o que está acessível (BAQUI, 2012). Na filtragem dinâmica, ocorre um filtro mais adaptativo ao cenário, pois o contexto topológico é considerado, evitando assim que bloqueios de pacotes essenciais à determinados serviços aconteçam (BAQUI, 2012).

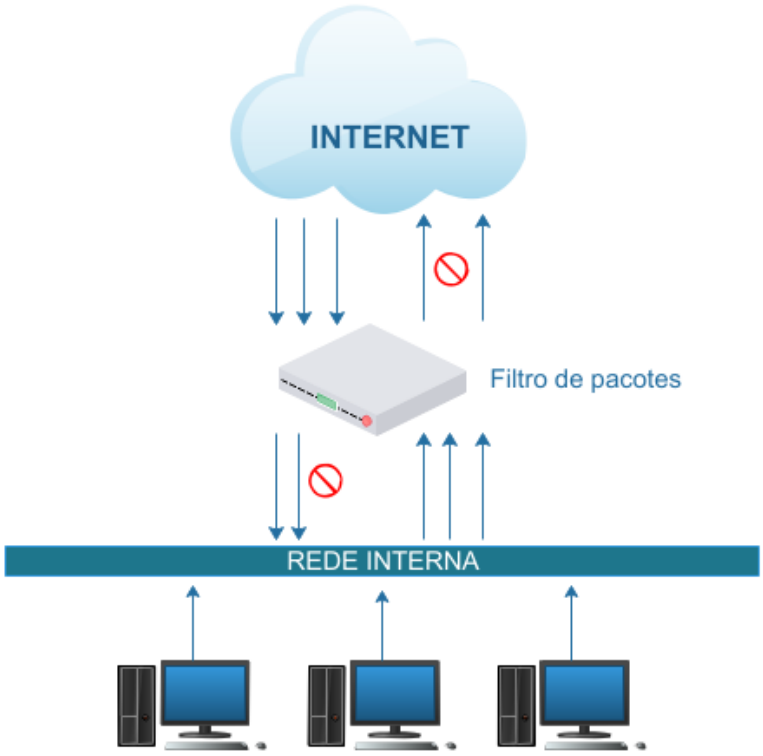


Figura 12 – *Firewall* filtro de pacotes

A Figura 12 ilustra o filtro de pacotes, onde apenas requisições permitidas são trafegadas.

2.4.3 *Firewall* baseado em estados

Este tipo de implementação, também chamado de *Statefull Firewall*, é considerado mais inteligente que o filtro de pacotes, pois analisa o cenário e já tem uma predefinição do que possa vir a acontecer (BAQUI, 2012). Além da base de regras, também há nesta arquitetura uma tabela de estados, que auxilia na tomada de decisão ao analisar os pacotes (SOEIRA; SILVA; TABORDA, 2013).

No início, o *firewall* baseado em estados analisava os protocolos dos pacotes, para assim verificar um acesso ilegítimo. A partir da evolução dos *firewalls*, diversas implementações foram melhorando a segurança e a forma como os pacotes são analisados, é o caso do SMLI (S*tatefull*

Multi-Layer Inspection). Este tipo de tecnologia faz uma averiguação em todas as camadas do modelo *OSI*, decodificando o pacote e aplicando técnicas para detectar um possível ataque (SOEIRA; SILVA; TABORDA, 2013).

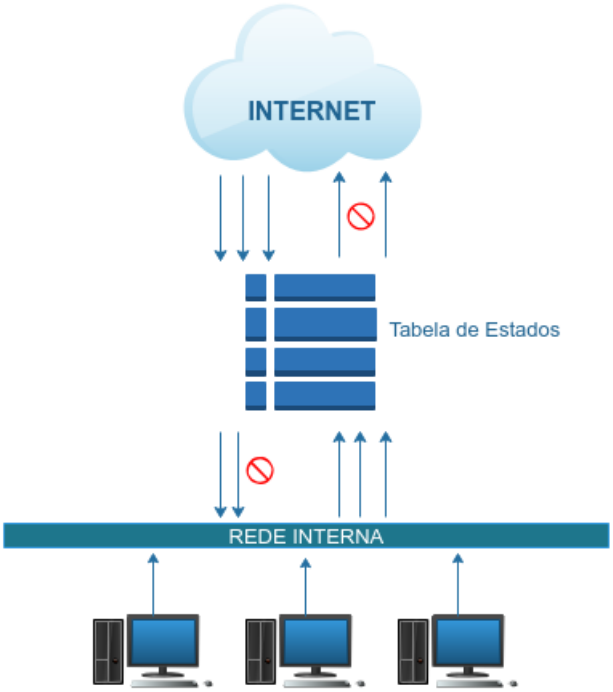


Figura 13 – *Firewall* baseado em estados (BAQUI, 2012).

A Figura 13 demonstra a tabela de estados analisando os pacotes e permitindo apenas aqueles que estão de acordo com as regras determinadas.

2.4.4 Arquitetura *dual homed-host*

Na arquitetura *dual homed-host*, um computador atua como um roteador, tendo uma interface para a rede interna e outra para a rede externa, desta forma, os pacotes externos não são repassados diretamente para um *host* na rede interna, pois são analisados e filtrados, provendo maior controle e eliminando aqueles indesejados. O administrador de rede pode gerenciar o tráfego e monitorar atividades de usuários. Uma desvantagem dessa implementação é a vulnerabilidade da rede, já que concentra-se em apenas uma máquina (SOEIRA; SILVA; TABORDA, 2013).

A Figura 14 ilustra a arquitetura deste tipo de *firewall*, demonstrando suas interfaces para a rede *LAN* e outra de saída para internet.

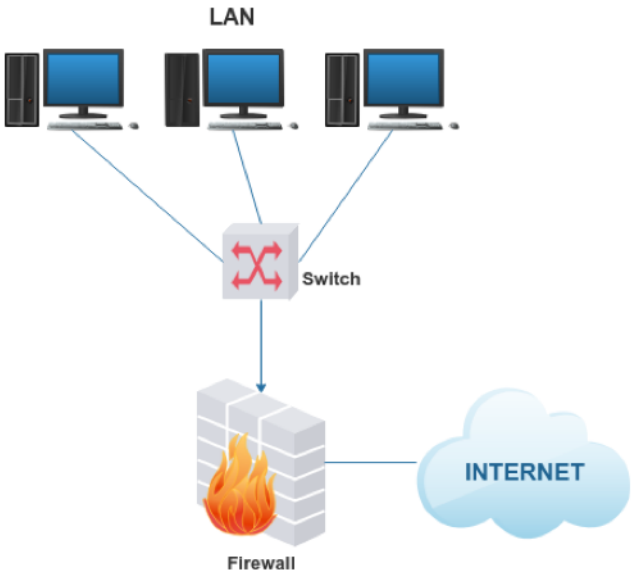


Figura 14 – *Firewall dual homed-host*. Adaptado de (SOEIRA; SILVA; TABORDA, 2013)

2.4.5 Arquitetura *screened host*

Apesar de ser uma arquitetura considerada segura, não tem uma implementação simples. Inicialmente, configura-se um servidor, que será uma *bridge* entre as redes internas e externas. Este intermediário é o *bastion host*. Para que se tenha acesso à rede interna, os usuários precisam autenticar-se primeiramente nessa máquina (SOEIRA; SILVA; TABORDA, 2013), (BAQUI, 2012).

Nessa implementação há dois níveis de proteção, um na camada de Rede e outra na camada de Aplicação, essa é uma das vantagens. Porém, se o *bastion host* for comprometido, toda a rede estará comprometida (SOEIRA; SILVA; TABORDA, 2013).

A Figura 15 demonstra a arquitetura deste tipo de *firewall*, onde o *bastion host* intermedia o acesso à rede interna.

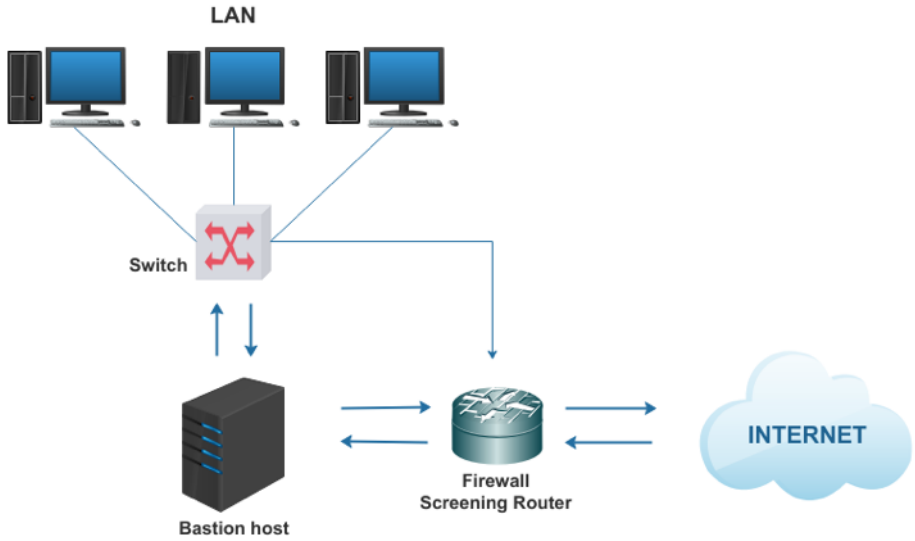


Figura 15 – *Firewall screened host*. Adaptado de (SOEIRA; SILVA; TABORDA, 2013)

2.4.6 Arquitetura *screened subnet*

Por último, a arquitetura *screened subnet*, a qual é utilizada como modelo na proposta deste trabalho, é semelhante à *screened host*, demonstrada na subseção 2.4.5. Também conhecida como *DMZ - Demilitarized Zone* - (Zona Desmilitarizada), que é um conceito de uma rede intermediária entre rede interna e externa. Nesta é acrescentada uma camada de segurança, adicionando um roteador entre a rede interna e o *bastion host*, fazendo combinação com o *screened router* e separando a área dos servidores (MELO, 2017).

A Figura 16 demonstra a inserção desta camada de proteção, tornando mais seguro o tráfego entre as redes interna e externa.

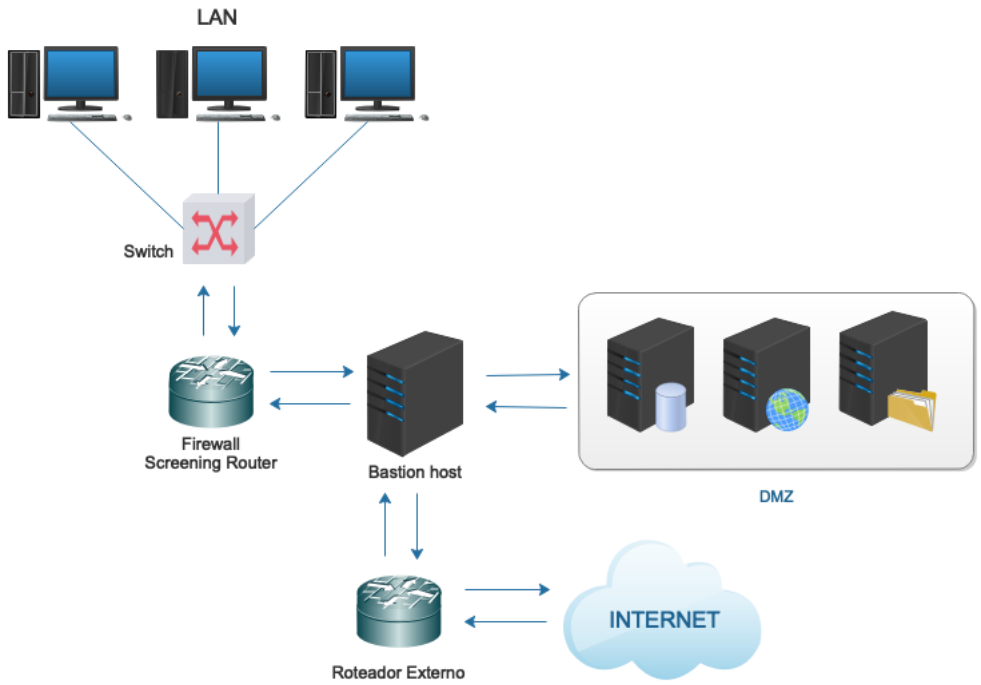


Figura 16 – *Firewall screened subnet*. Adaptado de (SOEIRA; SILVA; TABORDA, 2013)

3 Metodologia

Após apresentar os estudos que possibilitaram a realização deste trabalho, neste capítulo são descritos o cenário que compõem a arquitetura da solução proposta, as técnicas utilizadas para a análise e identificação do *flooding TCP/SYN*, os materiais de *hardware* e ferramentas de *software*, bem como a descrição dos conjuntos de testes realizados.

A solução proposta será denominada *FuzzSec* e terá suas entidades descritas na seção 3.3.

3.1 Materiais

Para o desenvolvimento do cenário proposto neste trabalho, utiliza-se os seguintes componentes de *hardware*:

Tabela 4 – Materiais de *hardware*

DESCRIÇÃO	**MARCA**	**CONFIGURAÇÃO**
Macbook Air	Apple	Processador Core i5 8gb RAM
MacMini	Apple	Processador Intel core i5 4gb RAM
Cabo par trançado CAT-5e	Cablefix	RJ45
Switch (Labin Unitins)	D-Link (DGS-1210-28)	-

E as seguintes ferramentas de *software*:

Tabela 5 – Materiais de *software*

DESCRIÇÃO	**VERSÃO**
Virtualizador VirtualBox	5.2
Sistema Operacional MacOS Mojave	10.14
Sistema Operacional Debian	9.5
Sistema Operacional Kali Linux Light	2018.4
Software hping3	3
Software T50 (Packet Injector)	5.8
Apache JMeter	5.0
RStudio	1.1.463
Plotly	1.0

Este trabalho utiliza somente ferramentas *open source* e gratuitas, possuindo assim custo zero na sua implementação. Os materiais e componentes descritos acima são a base para proporcionar a construção da solução *FuzzSec*, demonstrada nas seções a seguir, contudo há

ainda a possibilidade da exploração e implementação de módulos, utilizando as linguagens de programação *scripts*[9] em *shell* e *python*[10].

3.2 Definição do cenário

A Figura 17 demonstra o cenário, o qual foi definido para este trabalho, que no geral, caracteriza uma arquitetura *screened subnet*, detalhada na seção 2.4.6. O cenário é composto de duas ou mais redes locais (LAN), interligadas no *switch core*[11] através do *switch de acesso*[12], uma máquina com o SO Debian 9.6[13], com a implementação de *firewall* e a solução *FuzzSec*, que será explanada na seção 3.3.

Fazem parte ainda um servidor *web*, que se encontra numa *DMZ - Demilitarized Zone* - (Zona Desmilitarizada), que é um conceito de uma rede intermediária entre rede interna e externa. Nesta é acrescentada uma camada de segurança, adicionando um roteador entre a rede interna e o *bastion host*, fazendo combinação com o *screened router* e separando a área dos servidores (MELO, 2017). Este hospeda aplicação que será o alvo das requisições *flood TCP SYN*, pois esta área agrega todos os servidores provedores de serviços e sistemas, mas não há acesso direto, para que isso aconteça, é necessário fazer encaminhamento de pacotes através do *firewall*.

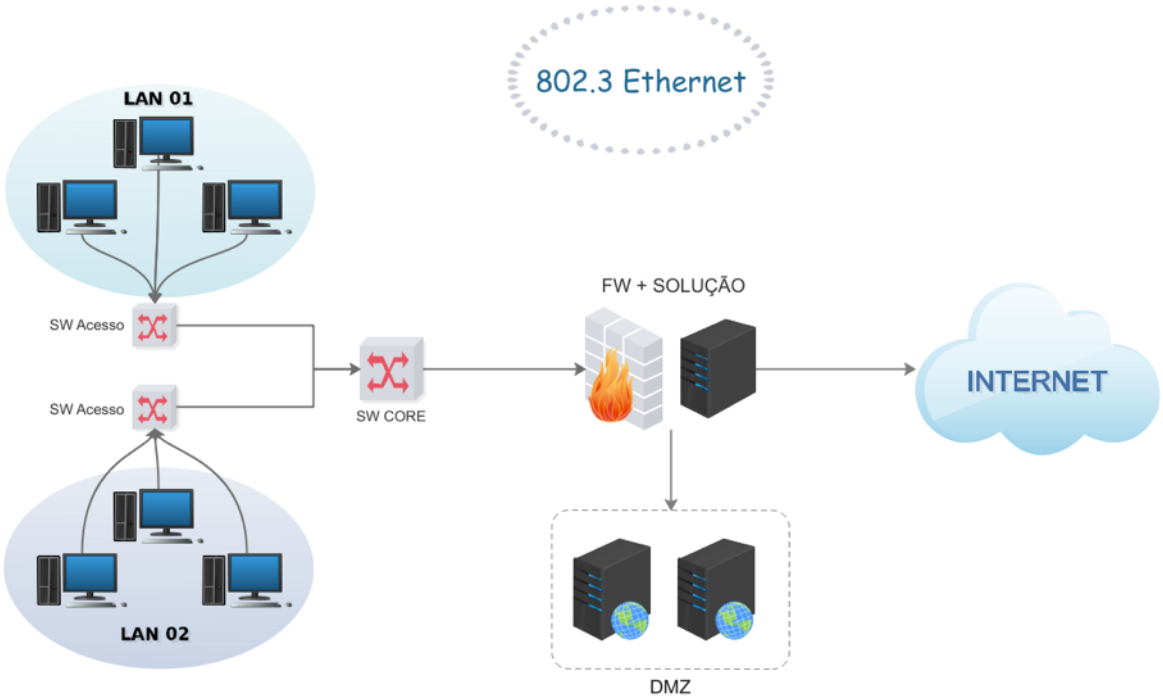


Figura 17 – Cenário da arquitetura da solução proposta.
Fonte: Autoria própria.

[9] https://www.shellscript.sh/
[10] https://www.python.org/about/
[11] Switch central, de alta velocidade, que interliga outras redes.
[12] Switch simples, que conectam os *hosts* na rede local.
[13] https://www.debian.org/intro/about

3.3 Entidades da arquitetura

Nesta seção é aberta a "caixa-preta"da *FuzzSec*, demonstrando os mecanismos utilizados para a coleta de dados, ativação das métricas, entradas para a técnica de tomada de decisão e *firewall*.

- Representando a *LAN* para realização dos testes, utilizou-se uma máquina, MacBook Air, com *software hping3*, T50[14] máquina virtual com Kali Linux Light 2018.4[15] para ataques e ferramenta *Apache JMeter 5.0*[16], para acessos legítimos;
- *Switch* para provimento do enlace, utilizada a própria estrutura da Universidade;
- *FuzzSec* - Máquina virtual com SO Debian versão 9.6, 512Mb de memória RAM e CPU com 1 core. Nesta máquina estão todos os *scripts* desenvolvidos para a realização deste trabalho. Nela está instalado o analisador de tráfego *tcpdump* para coletar o tráfego, *python 2.7* com as bibliotecas essenciais para a técnica de tomada de decisão: *numpy* 1.15.1[17] e *scikit-fuzzy* 0.2[18]. Por fim, interface de *firewall* linux do *framework netfilter*, *iptables* 1.4.20[19]
- DMZ - Máquina virtual com SO Debian 9.6, 512Mb de memória RAM e CPU com 1 core, denominada Alvo, com servidor web Apache 2.4 [20], aplicação em linguagem PHP 7[21] e banco de dados MariaDB 10.2[22].

A Figura 18 demonstra as entidades envolvidas na solução proposta neste trabalho. A seguir, serão detalhados os módulos ilustrados.

[14] https://sourceforge.net/projects/t50/,
[15] https://www.kali.org/
[16] https://jmeter.apache.org/
[17] http://www.numpy.org/
[18] https://pythonhosted.org/scikit-fuzzy/
[19] http://ipset.netfilter.org/iptables.man.html
[20] https://www.apache.org/
[21] http://www.php.net/
[22] https://mariadb.org/

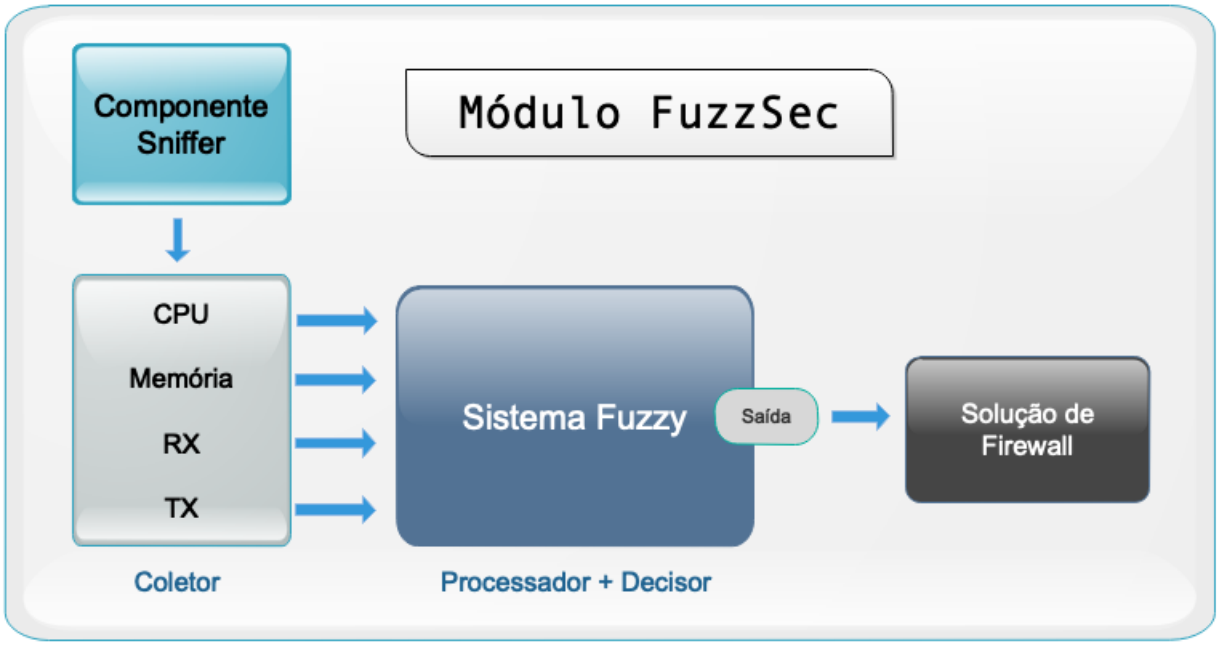


Figura 18 – Arquitetura da solução.
Fonte: Autoria própria.

3.4 Coletor

Para analisar o comportamento dos recursos da máquina onde está a *FuzzSec*, utilizou-se os parâmetros coletados através de um conjunto de *scripts* escrito nas linguagens C[23] e Python 2.7, demonstrados nos Algoritmos 1, 2 e 3.

Inicialmente é feita a coleta do tráfego de rede através da biblioteca *libpcap*, por meio da ferramenta *tcpdump*. Conforme demonstrado no Algoritmo 1, o analisador de tráfego está configurado para filtrar apenas a incidência da *flag TCP SYN*, a qual é objeto de análise neste trabalho.

Com isso, os dados são escritos em um arquivo, que é lido por outro *script*, fazendo uma verificação simplificada da quantidade de linhas gravadas. Se retornar um valor maior que 100.000 (cem mil), então é ativada a coleta dos parâmetros da máquina, pois o filtro já definiu que há um comportamento anormal, com incidência alta de abertura de conexões. Este limiar foi obtido através do conjunto de testes realizados no Cenário 02, descrito na seção 3.7.1, onde o número de pacotes foi parametrizado.

Seguindo, têm-se os *scripts* em Python 2.7 que serão executados apenas se houver uma identificação inicial de *flood TCP SYN*. Estes coletam a porcentagem de CPU utilizada, porcentagem de memória utilizada, além de dados da placa de rede, como RX (recepção de pacotes) e TX (transmissão de pacotes), medidos em kbps. Com o objetivo de coletar todos

[23] https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

estes dados em tempo real, cada parâmetro foi definido em uma função, para que todas sejam executadas paralelamente e a saída dos valores obtidos na coleta sejam gravadas em um arquivo, para posteriormente serem tabulados e servir de entrada para o sistema de apoio à tomada de decisão.

A coleta destes parâmetros durante os testes foi fundamental para determinar a saúde dos recursos da máquina e da aplicação. A partir destes dados tem-se os limiares necessários para a modelagem das regras, conjuntos e subconjuntos *fuzzy*, demonstrados na seção 3.5 os quais são determinantes no processo de decisão, e que são o foco desta pesquisa.

O Algoritmo 1 apresenta o processo inicial de coleta de tráfego com a ferramenta *tcpdump*. Atráves de um *daemon*, o *script* é disparado de 5 em 5 segundos e coleta durante 20 segundos, pois a biblioteca *libpcap*[24] utiliza recursos computacionais significativos para este contexto, logo ao final de cada coleta o processo é interrompido. O filtro utilizado para *TCP SYN* com a ferramenta *tcpdump* está descrito a seguir:

Algoritmo 1: ANALISADOR DE TRÁFEGO TCPDUMP

Entrada: Filtro de captura para *TCP SYN*
Saída: Arquivo txt com captura do tráfego

```
1  início
2      para i de 1 ate 20 faça
3          data_sniffer ← tcpdump(filter);   /* inicializa a coleta aplicando o
               filtro TCP SYN */
4          txt_file ← data_sniffer;          /* armazena os dados no arquivo */
5      fim
6      kill_sniffer()                        /* encerra o processo de coleta */
7  fim
8  espere(5);                                        /* Aguarda 5 segundos */
9  inicie a captura;                  /* Inicia novamente a captura do tráfego */
10 retorna Fim da execução
```

No Algoritmo 2 é demonstrado como foi feita a coleta das métricas de *CPU* e memória utilizadas, representados em porcentagem. Primeiramente inicializa-se a variável que irá armazenar a coleta em cada iteração, para que não haja nenhum valor indevido. O método *cpu_percent* obtém a *CPU* utilizada no momento da execução. Para isso utilizou-se o pacote *psutil*[25] da linguagem Python.

O processo de coleta da memória é semelhante, no entanto o método *get_mem* faz uma chamada de sistema com o comando *free -m -t*[26], com seus devidos filtros. Para esta execução,

[24] http://www.tcpdump.org/manpages/pcap.3pcap.html
[25] https://psutil.readthedocs.io/en/latest/
[26] http://man7.org/linux/man-pages/man1/free.1.html

utilizou-se o pacote *commands*[27] em Python 2.7.

Algoritmo 2: COLETA DE MÉTRICAS: CPU E MEMÓRIA

Entrada: Classe coletora GetData

Saída: CPU e memória utilizadas em porcentagem

```
1  início
2      cpu ← 0;                                          /* inicializa a variável */
3      memory ← 0;                                       /* inicializa a variável */
4      para i de 1 ate 200 faça
5          cpu_used ← GetData.cpu_percent();             /* coleta a CPU utilizada */
6          se cpu_used > 0 então
7              cpu ← cpu_used
8          fim
9          mem_used ← GetData.get_mem();                 /* coleta a memória utilizada */
10         se mem_used > 0 então
11             memory ← mem_used
12         fim
13         espere(1)                                     /* aguarde 1 segundo para a próxima coleta */
14     fim
15 fim
16 retorna cpu e memória
```

O Algoritmo 3 é semelhante ao anterior. São inicializadas as variáveis que irão receber os dados, após entram no *loop* para executar o processo de coleta dos valores de RX e TX da placa de rede. Como trata-se de uma chamada de sistema, fora parametrizado um *timer* de 1 segundo. Logo após, os dados coletados são inseridos em um arquivo *csv*. O módulo utilizado para a coleta foi o pacote de aplicativos de monitoramento do Linux desenvolvido em C *sysstat*[28],

[27] https://docs.python.org/2/library/commands.html
[28] http://man7.org/linux/man-pages/man5/sysstat.5.html

através do comando SAR.

Algoritmo 3: COLETA DE MÉTRICAS: RX E TX

Entrada: Classe coletora GetData

Saída: RX e TX da placa de rede em kbps

```
1  início
2      rx ← 0;                                   /* inicializa a variável */
3      tx ← 0;                                   /* inicializa a variável */
4      para i de 1 ate 200 faça
5          rx_now ← GetData.get_rx();            /* coleta RX/kbps utilizados */
6          se rx_now > 0 então
7              rx ← rx_now
8          fim
9          tx_now ← GetData.get_tx();            /* coleta TX/kbps utilizados */
10         se tx_now > 0 então
11             tx ← tx_now
12         fim
13         espere(1)              /* aguarde 1 segundo para a próxima coleta */
14     fim
15 fim
16 retorna rx e tx
```

3.5 Processador Fuzzy

O processador *fuzzy* é responsável por processar os dados coletados nas métricas para classificar se há ou não ataque *TCP SYN*. Para definir os limiares para os conjuntos *fuzzy* no processo de fuzificação (do inglês *fuzzification*), foram utilizados os parâmetros coletados no conjuntos de testes iniciais, de acordo com as estratégias explanadas na seção 3.7.1. A partir desses valores, foram definidas as variáveis linguísticas, apresentadas na Tabela 6.

Neste caso, foram utilizados valores mínimos e máximos da coleta de dados, definindo assim mais precisamente o sistema (RIOS, 2012). Para cada variável linguística estabelecida há termos linguísticos que determinam suas variações, apresentados na Tabela 7. Para a fuzificação destes termos, utilizou-se a função de pertinência triangular e o método Mamdani na inferência, descrito no capítulo 2.

3.5.1 Conjuntos *fuzzy* e variáveis linguísticas

A seguir serão apresentadas as variáveis, termos linguísticos e os conjuntos *fuzzy* adotados neste trabalho no processo de fuzificação. A Tabela 6 demonstra as variáveis de entrada para o sistema *fuzzy*, através da coleta de dados.

Tabela 6 – Variáveis Linguísticas

Variável	**Descrição**
CPU	Porcentagem do uso de CPU
Memória	Porcentagem do uso de memória
RX	Quantidade de kbps recebidos pela placa de rede
TX	Quantidade de kbps transmitidos pela placa de rede

A Tabela 7 apresenta os termos linguísticos utilizados para cada variável linguística. Estes termos traduzem os limiares descritos em números nos subconjuntos *fuzzy*.

Tabela 7 – Termos Linguísticos

Termos linguísticos	**Descrição**
mínimo	Tráfego e processamento mínimo
normal	Tráfego e processamento normal
intenso	Tráfego e processamento intenso
DDoS	Tráfego e processamento alto, ataque

A Tabela 8 apresenta os limiares de uso de *CPU* estabelecidos através do primeiro conjunto de testes, os quais foram ajustados posteriormente. A seção 4.1 detalha os testes realizados.

Tabela 8 – Subconjunto fuzzy para CPU (%)

Termos	**Subconjunto**
mínimo	porcentagem de uso de 2% a 7%
normal	porcentagem de uso de 12% a 36%
intenso	porcentagem de uso de 53% a 65%
DDoS	porcentagem de uso acima de 70%

A Tabela 9 apresenta os limiares de uso de memória estabelecidos através do primeiro conjunto de testes, os quais foram ajustados posteriormente. A seção 4.1 detalha os testes realizados.

Tabela 9 – Subconjunto fuzzy para Memória (%)

Termos	**Subconjunto**
mínimo	porcentagem de uso de 20% a 47
normal	porcentagem de uso de 34% a 66%
intenso	porcentagem de uso de 64% a 83%
DDoS	porcentagem de uso acima de 90%

A Tabela 10 apresenta os limiares classificados para entrada de dados na placa de rede, o que corresponde o parâmetro RX.

Tabela 10 – Subconjunto fuzzy para RX (kbps)

Termos	**Subconjunto**
mínimo	250 a 500
normal	500 a 750
intenso	750 a 1500
DDoS	taxa acima de 1600

A Tabela 11 apresenta os limiares classificados para saída de dados na placa de rede, o que corresponde o parâmetro TX.

Tabela 11 – Subconjunto fuzzy para TX (kbps)

Termos	**Subconjunto**
mínimo	250 a 500
normal	500 a 750
intenso	750 a 1500
DDoS	taxa acima de 1600

3.5.2 Regras Fuzzy

O conjunto de regras *fuzzy* representam uma ação que é tomada aplicando um consequente à um antecedente. Elas ocorrem no formato **SE** isso **ENTÃO** isso. Na regra geral, a quantidade de regras é mensurada seguindo o seguinte cálculo: x^y, onde x é a quantidade de variáveis linguísticas e y a quantidade de termos linguísticos. Para este cenário, totalizaram-se 256 regras, sendo 4^4.

Partindo da aplicação do cenário de política aberta, onde todo o tráfego é permitido, exceto aquilo que seja classificado como um ataque, tem-se uma quantidade menor de regras, pois muitas combinações dentro dos limiares mínimo e normal seriam desnecessárias e repetitivas. Portanto, há um conjunto de regras que devem garantir a classificação do ataque e estado de alerta com grau de pertinência tendendo ao ataque, e as demais regras suprem os outros comportamentos do sistema.

A Tabela 12 apresenta alguns exemplos das regras utilizadas na solução *FuzzSec*, observe que não foram utilizadas todas as variáveis linguísticas numa mesma regra. Isto se deu pelo fato de que os operadores lógicos *AND* e *OR* proveem facilidade na composição das regras, eliminado repetições desnecessárias. Além disso, o fator organizacional, para melhor leitura das mesmas foi decisor nesta forma de estruturação.

Tabela 12 – Regras fuzzy

Regras
SE cpu é ddos ou memória é ddos ENTÃO ataque sim
SE rx é ddos ou tx é ddos ENTÃO ataque sim
SE cpu é intenso e memória é ddos ENTÃO ataque sim
SE rx é intenso e tx é ddos ENTÃO ataque sim
SE rx é ddos e tx é intenso ENTÃO ataque sim
SE cpu é intenso ou memória é intenso ENTÃO alerta
SE rx é intenso ou tx é intenso ENTÃO alerta
SE cpu é intenso e memória é normal ENTÃO alerta
SE cpu é normal e memória é intenso ENTÃO alerta
SE cpu é mínimo ou memória é mínimo ENTÃO ataque não
SE rx é mínimo ou tx é mínimo ENTÃO ataque não
SE cpu é mínimo ou memória é normal ENTÃO ataque não
SE cpu é normal ou memória é mínimo ENTÃO ataque não
SE rx é normal ou tx é mínimo ENTÃO ataque não
SE rx é mínimo ou tx é normal ENTÃO ataque não

O Algoritmo 4 apresenta o processador *fuzzy*, no qual recebe os parâmetros coletados, aplica inferência com base nas regras e subconjuntos e retorna a saída desfuzificada, determinando o grau de pertinência da ocorrência de *flood TCP SYN*. O método *fuzzsec.fuzzy()* recebe as variáveis obtidas e faz uma abstração do processo.

Algoritmo 4: PROCESSADOR FUZZY

Entrada: porcentagem da CPU, porcentagem da $MEMORIA$, qtd de RX, qtd de TX
Saída: Classificação de $ATAQUE$

```
1 início
2     fuzzsec(CPU, MEMORIA, RX, TX)   /* realiza o input no processador */
3     ATAQUE ← fuzzsec.fuzzy();        /* processa e retorna a saída */
4 fim
5 retorna ATAQUE
```

3.5.3 Decisor

Após processadas as métricas coletadas, é gerada uma saída, na qual aplica-se no módulo decisor. Os termos linguísticos adotados para a saída segue o *MOS - Mean Opinion Score*, que originalmente foi desenvolvido para avaliar padrões de codificação, é uma medida da qualidade de voz, na qual aplica-se termos sequenciados de 0 a 5, sendo eles: Mau, pobre, razoável, bom e excelente (STREIJL; WINKLER; HANDS, 2016). Contextualizando para este trabalho, a Tabela 13 apresenta o subconjunto *fuzzy* para a saída do processador.

Tabela 13 – Subconjunto *fuzzy* para Saída do processador

Termo	Subconjunto
Não ataque	valores entre 0 e 1
Alerta	valores entre 1 e 3
Ataque	valores entre 2 e 4

Esta saída é totalmente legível para qualquer outra solução que venha ser aplicada neste contexto de classificação de *flood*, podendo definir outras técnicas para realizar bloqueios. Ao aplicar a saída defuzificada no Algoritmo 5, é tomada uma decisão especificada para cada limiar, podendo continuar coletando as métricas e um modo de alerta ou realizar bloqueios com o *firewall iptables*, descrito na seção 3.6.

Algoritmo 5: DECISOR

Entrada: *SAIDA_DEFUZZY*

Saída: Tomada de decisão e execução de *script*

```
1  início
2      decisao ← decisor(SAIDA_DEFUZZY) /* verifica a incidência de ataque
        */
                /* se for apenas alerta, continua coletando os parâmetros */
3      se decisao == alerta então
4          coletor()
5      fim
                              /* se for ataque, ativa o mecanismo de defesa */
6      se decisao == ataque então
7          firewall_iptables()
8      fim
9  fim
10 retorna decisao
```

3.6 Mecanismo de defesa - IPTables

Na solução *FuzzSec* utilizou-se a ferramenta integrante do *framework Netfilter*, *IpSet*[29], para criar listas de *IP's* de forma mais rápida e simplificada, a fim de prover ao *firewall* as origens de possíveis ataques sem precisar definir milhares de regras.

Criou-se uma lista chamada *ips_blacklist* e adicionada uma regra ao *firewall iptables*, onde possibilita bloquear todo o tráfego originado dos *IP's* constantes nesta lista. Com isso, não é necessário atualizar a regra propriamente, basta adicionar o *IP* na lista que o *iptables* já busca automaticamente. O tempo definido para permanecer nesta lista foi de 604800 segundos, representando uma semana, podendo ser alterado conforme a política desejada.

[29] http://ipset.netfilter.org/

Quando é identificado o *flood TCP SYN*, após todo o processo de análise de tráfego, coleta de parâmetros de desempenho e processamento *fuzzy*, a solução proposta provê parâmetros adequados para que mecanismos de defesa possam bloquear a *flag* de forma geral. Este trabalho demonstra que há a possibilidade de integração do módulo *FuzzSec* em qualquer outra solução de *firewall*.

3.7 Implementação dos cenários

A metodologia deste trabalho consiste em apresentar uma proposta de solução, chamada *FuzzSec* para classificar e analisar o *flooding* do tipo *TCP SYN* em redes *ethernet IEEE*, propiciando respostas para que mecanismos de defesa, como o *firewall iptables*, possam aplicar regras de bloqueio contra este ataque.

Foram definidas três cenários (configurações) para a realização dos testes:

- Cenário 01: Este cenário trata-se da saturação total do serviço, quando houver, com requisições utilizando *hping3*. Este teste foi feito seguindo o Algoritmo 6. Nesta configuração não há como administrar a quantidade de requisições, pois o *hping3* dispara um *flood* de forma mais rápida possível, buscando a saturação do enlace.

- Cenário 02: Este cenário trata-se das requisições utilizando a ferramenta T50, em que é possível parametrizar quantidade de pacotes de forma mais detalhada, tendo controle sobre o número enviado.

- Cenário 03: Este cenário simula acessos de usuários legítimos, completando as requisições *TCP* corretamente. Para a realização deste teste foi utilizada a ferramente *Apache JMeter*, a qual disponibiliza inúmeros parâmetros para controlar as requisições, podendo assim identificar a carga máxima de acessos suportada pelo servidor. Esta estratégia visou o treinamento da solução, de forma a realizar a adequação dos parâmetros utilizados nos conjuntos *fuzzy*.

Para a realização dos testes, desenvolveu-se um simples cadastro de clientes na linguagem PHP 7, utilizando o banco de dados *MariaDB* e servidor de aplicação *Apache* 2.4. Conforme demonstrado no diagrama de caso de uso, representado pela Figura 19, esta aplicação simula um ambiente no qual o usuário/aplicação de *stress* realiza a abertura da página inicial, que já carrega a lista de clientes, gerando consultas no banco e logo após, é feito o cadastro de um cliente e novamente retora para a *index*, carregando a lista de clientes, consumindo recursos da máquina. Esta aplicação está hospedada em uma máquina virtual, com o SO Debian 9.6, denominada Alvo.

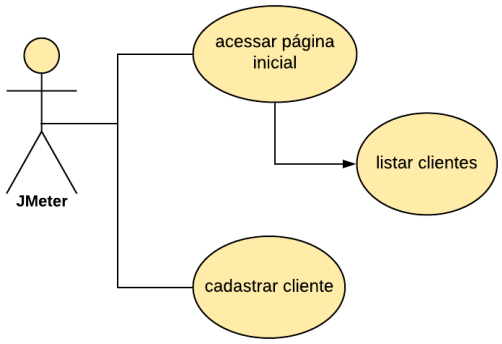


Figura 19 – Caso de Uso - CRUD em PHP - Aplicação alvo.

3.7.1 Parametrização dos testes e ajustes dos algoritmos

Com a finalidade de realizar ações automatizadas e parametrizadas para o cenário de testes, foram desenvolvidos *scripts* para verificar os dados de desempenho da máquina alvo, coletando as métricas em tempo real e também para realizar um ataque de negação de serviço do tipo *TCP SYN* com *hping3* e T50. Realizou-se dois conjuntos de testes. O primeiro foi para o treinamento do sistema, com a realização dos ajustes dos subconjuntos e regras *fuzzy* e o segundo teste, seguindo as mesmas parametrizações e quantidades, foi feito simulando o ambiente real.

O Algoritmo 6 representa um *flood* utilizando a ferramenta *hping3* e T50 de modo abstraído, onde requisições são direcionadas ao servidor alvo. A função *start_attack* representa o ataque, recebendo os parâmetros descritos na entrada do Algoritmo 6, utilizando *multithreads*, disparando diversas vezes simultaneamente.

Algoritmo 6: ATAQUE *TCP SYN*

Entrada: Quantidade de disparos simultâneos $THREADS$, endereço do alvo IP, porta de destino $PORT$, quantidade de pacotes PKT

Saída: Retorno de execução da requisição.

```
1 início
2     port ← 80;                                  /* define a porta de ataque */
3     server ← 172.16.90.150;                     /* informa o endereço do alvo */
4     para i de 1 ate THREADS faça
5         terminal ← start_attack(IP,PORT,PKT);   /* realiza o ataque */
6     fim
7 fim
8 retorna terminal
```

A simulação do ataque, descrita na seção 3.7, Cenário 01 utilizou-se a ferramenta *hping3*, a partir da implementação de uma aplicação *multithreading*, no qual foi executado na máquina cliente Macbook Air, conforme descrito na Seção 3.1. Para tal, fora parametrizado o valor de 10 e 20 *threads* simultâneas

Para o Cenário 02, com *flood* controlado, utilizou-se a ferramenta T50, no SO Kali Linux 2018 para realização dos testes. Este é parecido com o anterior, seguindo a mesma estrutura apresentada no Algoritmo 6, todavia com a possibilidade de definir limiares para o ataque *TCP SYN*. O valores parametrizados para os testes foram de 250.000 e 500.000 pacotes.

Para o Cenário 03, simulando acessos legítimos, o teste deu-se com a ferramenta *Apache JMeter*, que é uma ferramenta utilizada para realizar teste de *stress* em servidores e aplicações, disponibilizando vários parâmetros ajustáveis. Para este cenário, simulou-se uma interação do usuário com o sistema, na qual carrega a página inicial com a lista usuários, faz uma inserção e por fim retorna para a página inicial, gerando assim carga no banco de dados ao consultar e inserir dados. A Figura 20 apresenta o diagrama das atividades mencionadas.

Como trata-se de um ambiente de testes, a máquina Alvo dispõe de poucos recursos de *CPU* e memória, logo a quantidade de usuários simuladas foi condizente com o que provavelmente a máquina suportaria, podendo ter eventuais inundações.

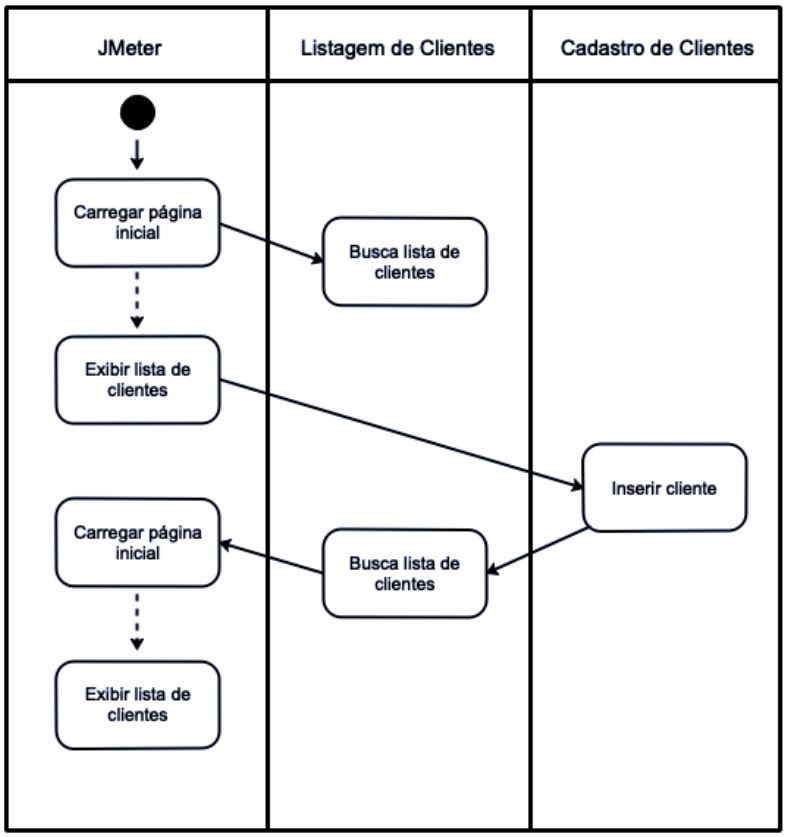


Figura 20 – Diagrama de atividades - Teste JMeter.

A Tabela 14 demonstra os parâmetros utilizados no teste.

Tabela 14 – Parâmetros JMeter no teste de carga

Nº DE USUÁRIOS (threads)	**TEMPO DE INICIALIZAÇÃO (s)**	**QTD ITERAÇÃO**
300	0.5	1
500	0.5	1

4 Resultados

Nesta seção serão apresentados os resultados obtidos durante a execução dos testes definidos no capítulo 3, bem como análise e tratamento dos dados. A princípio, na seção 4.1 são apresentados os testes realizados para ajustes do sistema *FuzzSec*, para melhoria dos parâmetros e regras. A seguir, na seção 4.2 são apresentados os dados referente à validação do modelo proposto, após ajustes dos parâmetros.

Adiante, são apresentadas as análises estatísticas para cada conjunto de testes. Para tabulação dos dados e geração dos gráficos, foi utilizada a linguagem R e o *software RStudio*[30], juntamente com a biblioteca *Plotly*[31].

4.1 Aprimoramento da Arquitetura

Nesta seção são apresentados os testes referentes ao ajuste do modelo para cada cenário determinado no capítulo 3.

4.1.1 Ajustes e resultados para o Cenário 01

A Tabela 15 demonstra as estatísticas obtidas com os testes de *flood* não controlado com a ferramenta *hping3*. O ataque realizado inundou a placa de rede, podendo ser visto valores altos nos parâmetros de RX e TX. Como é característico, não acontece o *three-way-handshake* da conexão *TCP*, então a aplicação Alvo não fez interação com o banco, permanecendo assim valores baixos na porcentagem de *CPU* e MEMORIA. Pode-se analisar, que na parte *B* da tabela, onde realizou-se o teste com 20 *threads* simultâneas, a taxa de uso de *CPU* se elevou, ocorrendo o processamento das requisições de conexão.

Após a identificação do *flood* pelo analisador de tráfego e coleta das métricas, ocorreu o processamento *fuzzy*, gerando a saída de 1.095. Este não foi um valor satisfatório, pois não caracterizou o ataque. Portanto, realizou-se ajustes nos parâmetros dos subconjuntos e regras, para que em uma nova tentativa, houvesse validação da solução.

[30] RStudio
[31] https://plot.ly/

Tabela 15 – Testes com Hping3 - Ajustes de parâmetros

HPING3 10 THREAD	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	14.3	3.44	771.73	771.85
Variância	8.73	0	12156.79	12150.87
Desvio Padrão	2.96	0.03	110.26	110.23
Mediana	14.3	3.42	794.12	794.43
Valor Máximo	22.2	3.47	859.33	859.52
Valor Mínimo	4.9	3.42	0	0
Intervalo de confiança	13.55 - 15.04	3.44 - 3.45	743.96 - 799.50	744.09 - 799.61
Saída Fuzzy	1.095			
HPING3 20 THREADS	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	16.89	3.83	763.49	763.69
Variância	69.18	0	17496.99	17495.71
Desvio Padrão	8.32	0.03	132.28	132.27
Mediana	16	3.81	788.79	789.09
Valor Máximo	62.1	3.86	932.94	926.11
Valor Mínimo	5.7	3.76	0	0
Intervalo de confiança	14.76 - 19.02	3.83 - 3.84	729.61 - 797.37	729.82 - 797.57
Saída Fuzzy	1.17			

A Figura 21 apresenta a inferência *fuzzy* para o teste inicial realizado com *hping3* parametrizado para 10 *threads*. O valor da saída, conforme demonstrado na Tabela 15 foi de 1.095, incindindo em uma pertinência alta para a saída de *não ataque*. Isso demonstrou uma ineficiência inicial da solução, determinando a necessidade de ajustes.

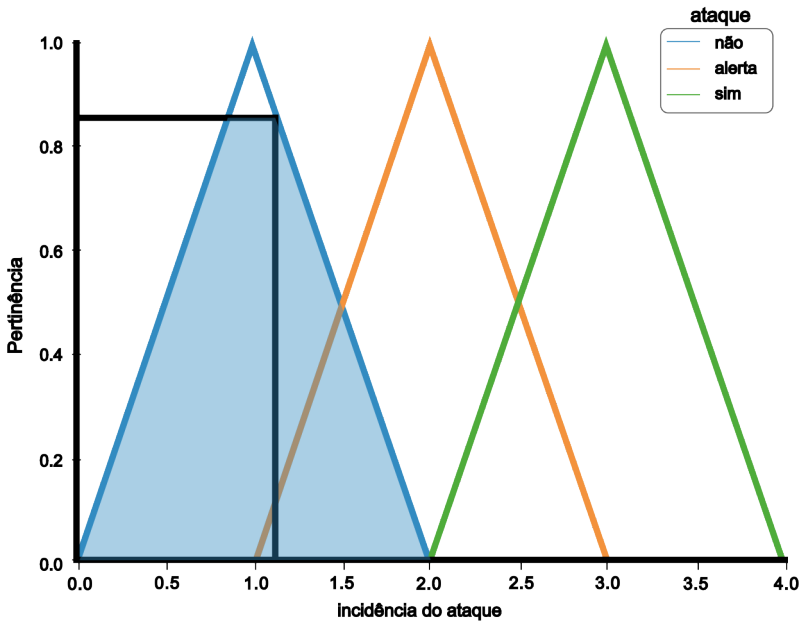
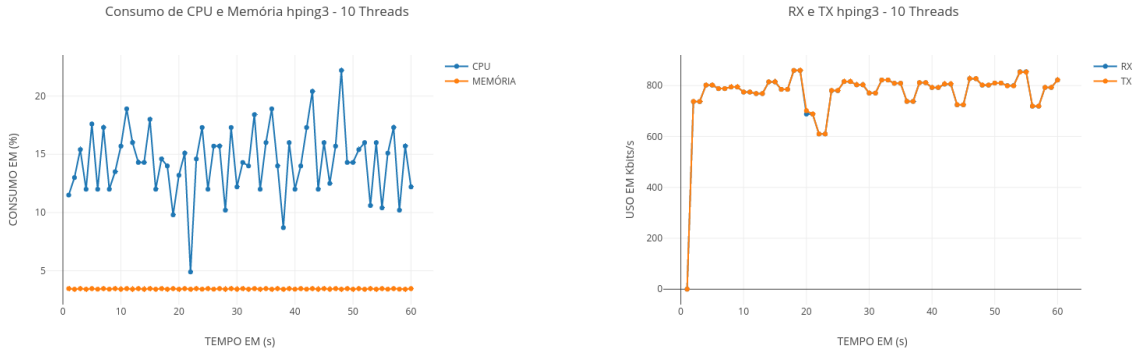


Figura 21 – Inferência *fuzzy* - Hping 10 *threads*

A Figura 22 apresenta o comportamento das variáveis linguísticas coletadas durante

o teste. Pode-se perceber no gráfico relativo à CPU e memória que devido ao alto número de requisições, o qual não é mensurado nesta ferramenta, houve um processamento maior do que o uso da memória. Para RX e TX, coletados em kbps/s, a máquina Alvo conseguiu processar as requisições recebidas neste cenário. A inferência *fuzzy* para este teste caracterizou pertinência para *não ataque*.



(a) CPU e Memória

(b) RX e TX

Figura 22 – Gráficos das Variáveis Linguísticas - Hping 10 *threads*

A Figura 23 apresenta a inferência *fuzzy* para o teste inicial realizado com *hping3* parametrizado para 20 *threads*. O valor da saída *fuzzy*, conforme demonstrado na Tabela 15 foi de 1.17, um pouco maior que o teste anterior, incindindo em uma pertinência alta para *não ataque* e baixa pertinência para *alerta*.

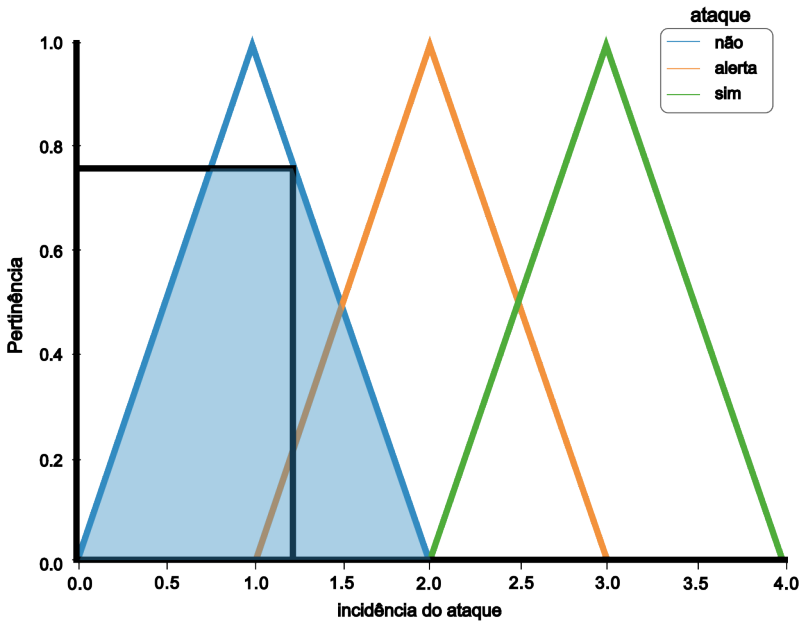
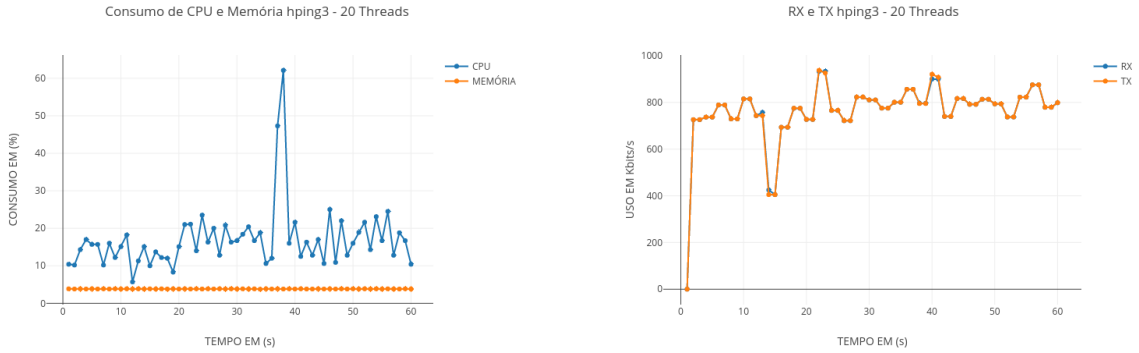


Figura 23 – Inferência *fuzzy* - Hping 20 *threads*

A Figura 24 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. No gráfico relativo à *CPU*, houve um ponto de acentuação no processamento. A memória

permanece em baixo uso, pois não houve demanda das requisições. Para RX e TX, a máquina Alvo conseguiu processar as requisições recebidas, sem ocorrer a queda, mas os altos valores indicaram indícios de *alerta*. A inferência *fuzzy* para este teste caracterizou pertinência para *não ataque*, novamente sendo objeto de ajustes dos parâmetros.



(a) CPU e Memória

(b) RX e TX

Figura 24 – Gráficos das Variáveis Linguísticas - Hping 20 *threads*

4.1.2 Ajustes e resultados para o Cenário 02

A Tabela 16 apresenta os dados obtidos no conjunto de testes do Cenário 02, com fluxo controlado. Em pouco se difere do anterior, pois tem o mesmo objetivo, que é inundar o sistema. Este tem uma peculiaridade, apenas os valores de RX estão altos, ou seja, foram enviados pacotes de forma tão rápida que o alvo não conseguiu devolver nenhuma solicitação, ficando assim com um *flood* apenas nos pacotes recebidos.

Tabela 16 – Testes com T50 - 250 e 500 mil pacotes

T50 250.00 pacotes	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	4.55	4.51	623.61	0.27
Variância	11.26	0	412670.6	0.01
Desvio Padrão	3.36	0.03	642.39	0.08
Mediana	4.8	4.49	559.14	0.25
Valor Máximo	9.5	4.54	1414.29	0.43
Valor Mínimo	1	4.49	0.12	0.2
Intervalo de confiança	2.69 - 6.40	4.50 - 4.53	267.87 - 979.36	0.23 - 0.32
Saída Fuzzy	1.02			
T50 500.000 pacotes	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	4.8	4.81	808.75	0.27
Variância	9.9	0	425575.6	0.01
Desvio Padrão	3.15	0.02	652.36	0.08
Mediana	4.85	4.81	1254.8	0.3
Valor Máximo	10.8	4.83	1459.63	0.34
Valor Mínimo	1	4.79	0	0
Intervalo de confiança	3.62 - 5.98	4.80 - 4.82	565.15 - 1052.34	0.24 - 0.29
Saída Fuzzy	1			

A Figura 25 apresenta a inferência *fuzzy* para o teste realizado com *T50* parametrizado para 250.000 pacotes. O valor da saída, conforme demonstrado na Tabela 16 foi de 1.02, com saída diferente do esperado, caracterizando *não ataque*.

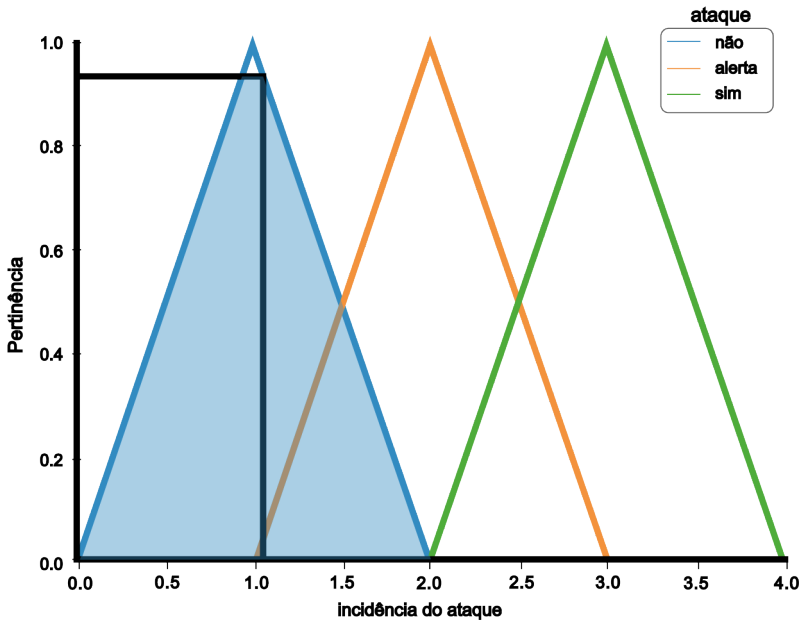
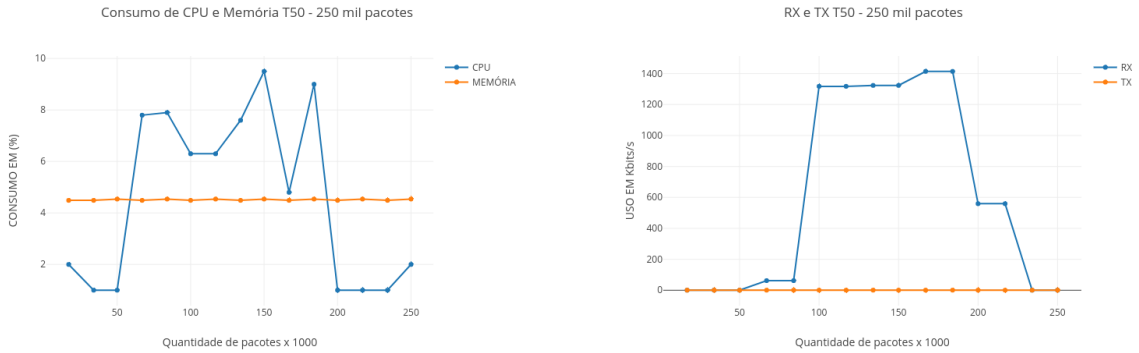


Figura 25 – Inferência *fuzzy* - T50 250000 pacotes

A Figura 26 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Neste cenário, houve uma variação constante do uso de *CPU*, mas com baixos valores. A memória manteve-se constante, mas também valores baixos, sem muito processamento. No

gráfico referente às variáveis RX e TX, observa-se que houve uma inundação da máquina, onde RX se manteve alto e TX se manteve baixo, denotando que máquina Alvo não conseguiu processar e devolver as requisições, ocorrendo o *flood*.



(a) CPU e Memória

(b) RX e TX

Figura 26 – Gráficos das Variáveis Linguísticas - T50 250.000*pacotes*

A Figura 27 apresenta a inferência *fuzzy* para o teste realizado com *T50* parametrizado para 500.000 pacotes. O valor da saída *fuzzy*, conforme demonstrado na Tabela 16 foi de 1.00, caracterizando *não ataque*, necessitando ajustar os parâmetros.

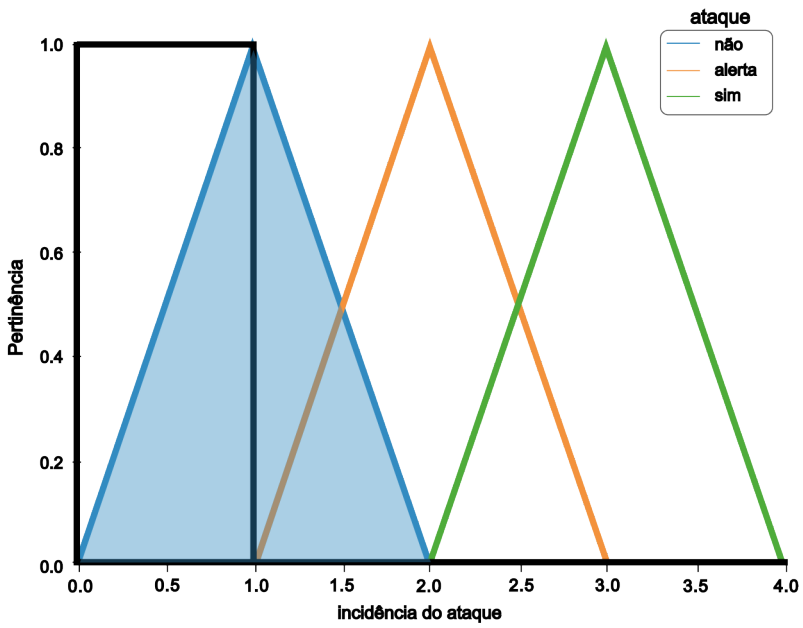
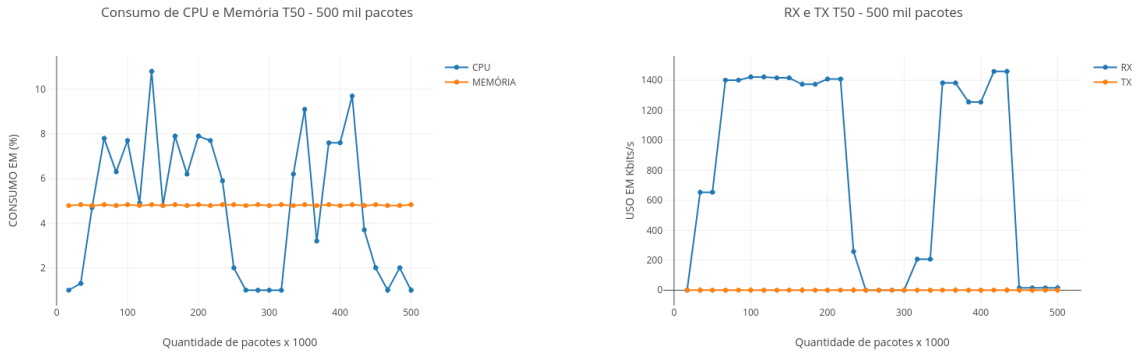


Figura 27 – Inferência *fuzzy* - T50 500.000 pacotes

A Figura 28 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Neste cenário, houve uma variação constante do uso de *CPU*, com valores um pouco mais altos que o anterior. A memória manteve-se constante, mas também valores baixos, sem muito processamento. No gráfico referente às variáveis RX e TX, observa-se que houve uma inundação

da máquina após certa quantidade de pacotes, onde RX manteve-se alto e TX manteve-se baixo, denotando que máquina Alvo não conseguiu processar e devolver as requisições, ocorrendo o *flood*.



(a) CPU e Memória

(b) RX e TX

Figura 28 – Gráficos das Variáveis Linguísticas - T50 500.000*pacotes*

4.1.3 Ajustes e resultados para o Cenário 03

Este teste, assim como os demais, realizou-se seguindo a parametrização detalhada na seção 3.7.1, onde foi simulado uma carga de acessos legítimos, verificando assim o limiar de usuários suportados pela aplicação e o comportamento do sistema *FuzzSec* quando houve um grande número de requisições e interações com o Alvo. Este conjunto de testes teve suas métricas coletadas diretamente no Alvo, para viabilizar apresentação de valores reais da *CPU* e memória devido a interação com o banco.

Tabela 17 – Testes com JMeter - 300 e 500 usuários

JMeter 300 usuários	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	13.04	28.65	26.33	156.95
Variância	529.7	0.21	3438	86.913
Desvio Padrão	23.02	0.46	58.63	294.81
Mediana	2.1	28.52	0.13	0.22
Valor Máximo	100	30.47	251.32	1196.3
Valor Mínimo	1	28.32	0	0
Intervalo de confiança	5.37 - 20.72	28.50 - 28.81	6.78 - 45.88	58.65 - 255.24
Saída Fuzzy	1.219			
JMeter 500 usuários	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	26.61	40.49	35.05	387.44
Variância	1219.43	140.62	2268.14	291.097
Desvio Padrão	34.92	11.86	47.62	539.53
Mediana	7.55	30.76	1.72	39.01
Valor Máximo	100	57.81	149.11	1777.37
Valor Mínimo	1	30.08	0	0
Intervalo de confiança	16.68 - 36.53	37.12 - 43.86	21.51 - 48.58	234.11 - 540.78
Saída Fuzzy	1.226			

A Tabela 17 demonstra a análise estatística do conjunto de testes resultante do Cenário 03, onde foi simulado o acesso legítimo de usuários ao sistema, realizando interações com o banco de dados.

A Figura 29 apresenta o resultado da inferência *fuzzy* neste conjunto de teste. Como demonstrado na Tabela 17, o valor de saída *fuzzy* foi de 1.219, denotando *não ataque*, com leve incidência para um *alerta*. Mesmo sendo um conjunto de testes iniciais para ajustes dos parâmetros do sistema, este é o resultado esperado para estes acessos legítimos, onde ocorra dentro do fluxo normal.

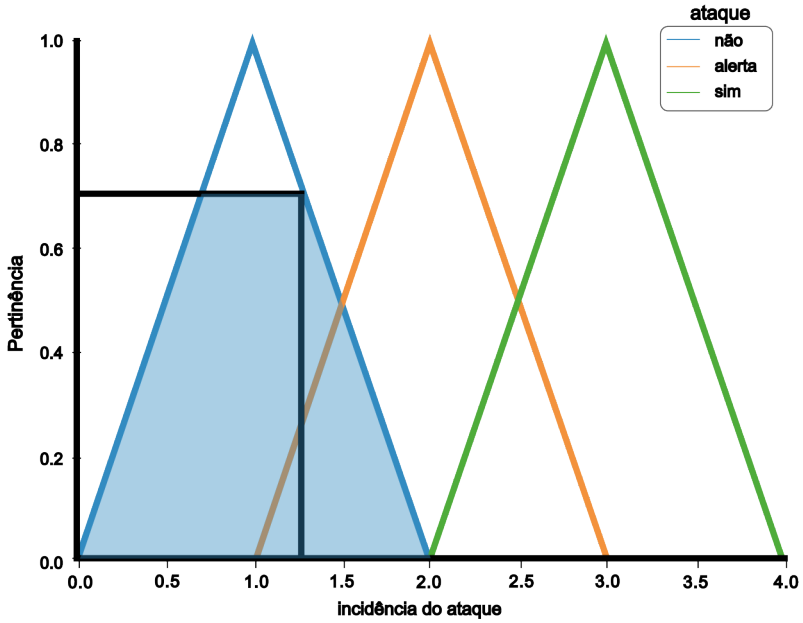
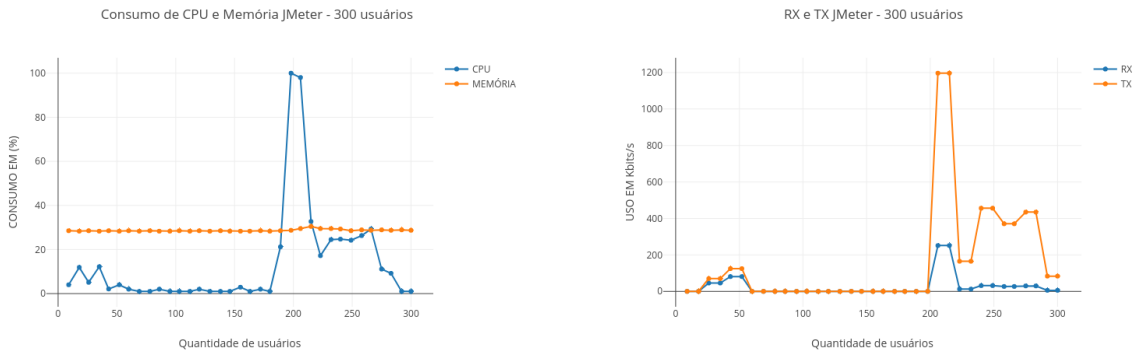


Figura 29 – Inferência *fuzzy* - JMeter 300 usuários

A Figura 30 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. O uso de *CPU* foi acentuado e poucos pontos, permanecendo baixo e contínuo no restante do teste. A memória manteve-se constante, não apresentando grande processamento. No gráfico referente às variáveis RX e TX, observa-se que em certo ponto houve um pico no recebimento de pacotes e poucas respostas às requisições, implicando que a máquina não conseguiu processar todo o tráfego.



(a) CPU e Memória

(b) RX e TX

Figura 30 – Gráficos das Variáveis Linguísticas - JMeter 300 usuários

A Figura 31 apresenta o resultado da inferência *fuzzy* neste conjunto de teste. Como demonstrado na Tabela 17, o valor de saída *fuzzy* foi de 1.226, denotando *não ataque*, com leve incidência para um *alerta*. Mesmo sendo um conjunto de testes iniciais para ajustes dos parâmetros do sistema, este é o resultado esperado para este acessos legítimos, onde ocorra dentro do fluxo normal.

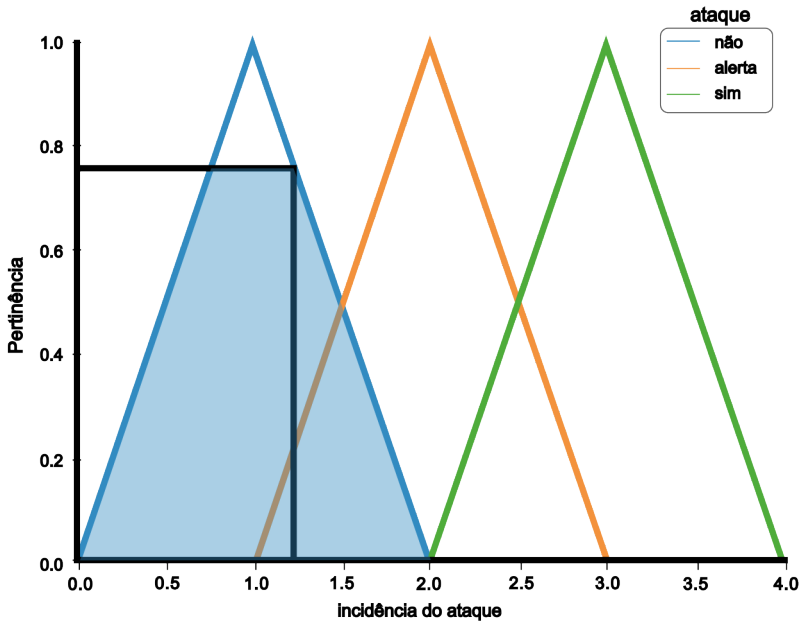
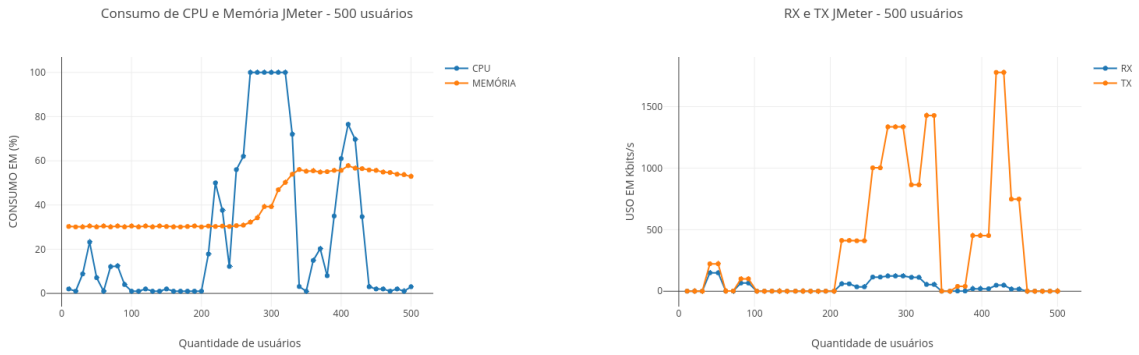


Figura 31 – Inferência *fuzzy* - JMeter 500 usuários

A Figura 32 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. O uso de *CPU* foi maior em mais pontos que no teste anterior e a memória acentuou-se um pouco. Nas variáveis RX e TX, observa-se o aumento significativo no recebimento de pacotes e pouca taxa de transmissão, denotando que neste teste, a máquina não suportou o processamento desta quantidade de usuários.



(a) CPU e Memória

(b) RX e TX

Figura 32 – Gráficos das Variáveis Linguísticas - JMeter 500 usuários

Após análise realizada sobre os dados obtidos no primeiro conjunto de testes para os três cenários definidos, verificou-se que o resultado obtido foi próximo do esperado para este primeiro conjunto, os quais imprescindivelmente necessitariam de ajustes. Portanto, foram feitos os ajustes nos conjuntos *fuzzy* e base de regras, para que o módulo processador gere saídas mais satisfatórias e próximas do esperado para validação da solução.

Na seção 4.2 serão demonstrados os novos conjuntos estatísticos obtidos após os ajustes nos parâmetros do sistema *FuzzSec*.

4.2 Validação da Arquitetura

Após efetuados os conjuntos de testes para obtenção de limiares e variáveis linguísticas, verificou-se a necessidade de realizar ajustes nos parâmetros do sistema *fuzzy*. Os valores de RX e TX foram os principais, nos quais apresentaram valores muito altos durante o *flood TCP SYN*.

Para validação do modelo proposto, realizaram-se novos testes, seguindo os mesmos parâmetros já definidos, conforme descrito na seção 3.7.1. As subseções a seguir apresentarão de forma mais detalhada cada conjunto de resultados obtidos nos novos teste para validação da arquitetura.

4.2.1 Testes de validação do Cenário 01

Com os ajustes realizados nos parâmetros do sistema *FuzzSec*, foi possível obter uma gama de resultados mais satisfatórios para cada cenário. A Tabela 18 apresenta as estatísticas das variáveis linguísticas coletadas durante os testes efetuados com a ferramenta *hping3*, com 10, 20 e 40 *threads*, com a duração de 60 segundos cada execução. Para cada execução com as diferentes quantidades de *threads* há uma estatística e uma saída do sistema *fuzzy*, que indica a pertinência para a variável linguística de saída *ataque*.

Tabela 18 – Testes com Hping3 - Validação

hping3 10 THREADS	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	30.54	13.18	1173.5	1173.08
Variância	64.45	0.01	57396.5	57.240
Desvio Padrão	8.03	0.01	239.58	239.25
Mediana	31.6	13.08	1233.23	1233.14
Valor Máximo	51.3	13.28	1463.75	1464.08
Valor Mínimo	12.2	13.08	94.77	94.81
Intervalo de confiança	28.47 - 32.62	13.15 - 13.20	1111.61 - 1235.38	1111.27 - 1234.88
Saída Fuzzy	2.363			
hping3 20 THREADS	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	72.3	9.29	3120.91	3128.44
Variância	518.81	0.02	2509708	253.053
Desvio Padrão	22.78	0.14	1584.21	1590.76
Mediana	69	9.36	3395.83	3399.28
Valor Máximo	100	9.56	6556.69	6582.44
Valor Mínimo	35.3	9.16	269.74	264.66
Intervalo de confiança	67.36 - 77.25	9.26 - 9.33	2777.12 - 3464.71	2783.22 - 3473.66
Saída Fuzzy	2.680			
hping3 40 THREADS	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	59.33	14.37	3280.3	3285.7
Variância	220.26	0.02	1195778	1204349
Desvio Padrão	14.84	0.15	1093.52	1097.43
Mediana	61.5	14.44	3145.78	3144.58
Valor Máximo	87.5	14.64	5602.13	5615.33
Valor Mínimo	27	14.24	0.13	0.22
Intervalo de confiança	55.20 - 63.46	14.33 - 14.41	2975.86 - 3584.74	2980.17 - 3591.22
Saída Fuzzy	2.470			

A Figura 33 apresenta a inferência *fuzzy* para o teste realizado com *hping3* parametrizado para 10 *threads*. O valor da saída, conforme demonstrado na Tabela 18 foi de 2.363, incindindo em uma pertinência menor para *ataque*, caraterizando então um *alerta*.

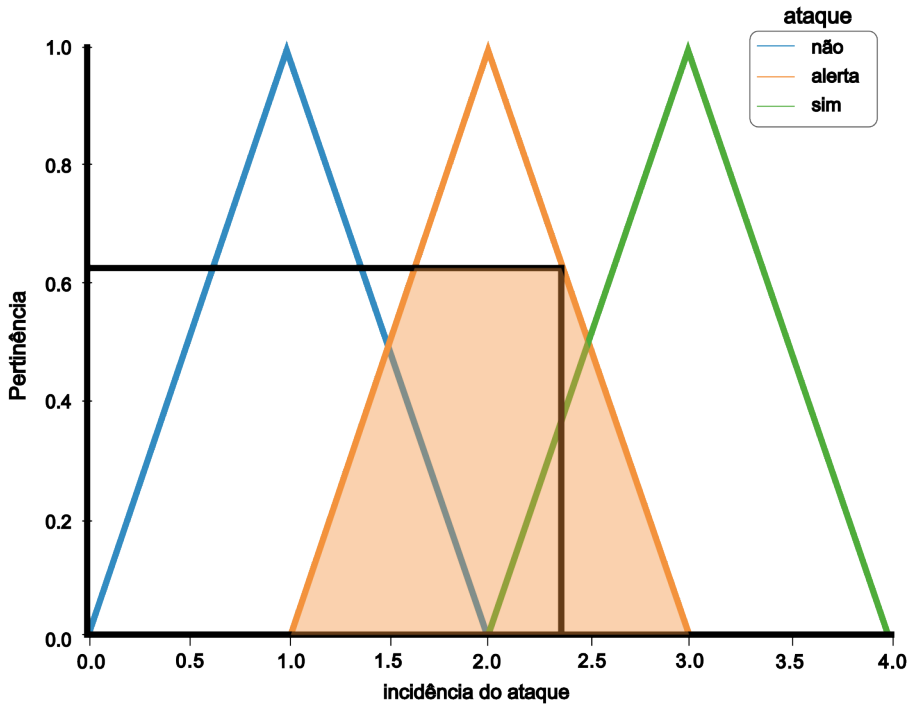
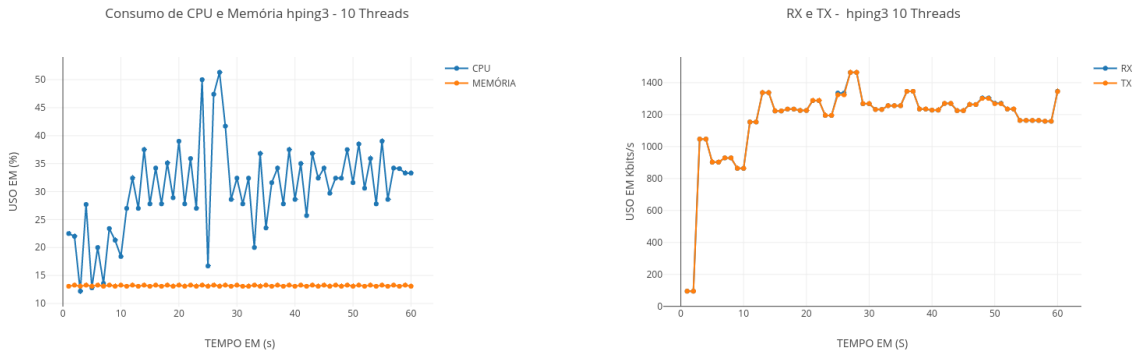


Figura 33 – Inferência *fuzzy* - Hping 10 *threads*

A Figura 34 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Pode-se perceber no gráfico relativo à *CPU* e memória que devido ao alto número de requisições, o qual não é mensurado nesta ferramenta, houve um processamento maior do que o uso da memória. Para RX e TX, a máquina Alvo conseguiu processar as requisições recebidas neste cenário, mas que por fim, caraterizou um *alerta*.



(a) CPU e Memória

(b) RX e TX

Figura 34 – Gráficos das Variáveis Linguísticas - Hping 10 *threads*

A Figura 35 apresenta a inferência *fuzzy* para o teste realizado com *hping3* parametrizado para 20 *threads*. O valor da saída, conforme demonstrado na Tabela 18 foi de 2.680, incindindo em uma pertinência maior para *ataque*, caracterizando positivamente que houve um ataque.

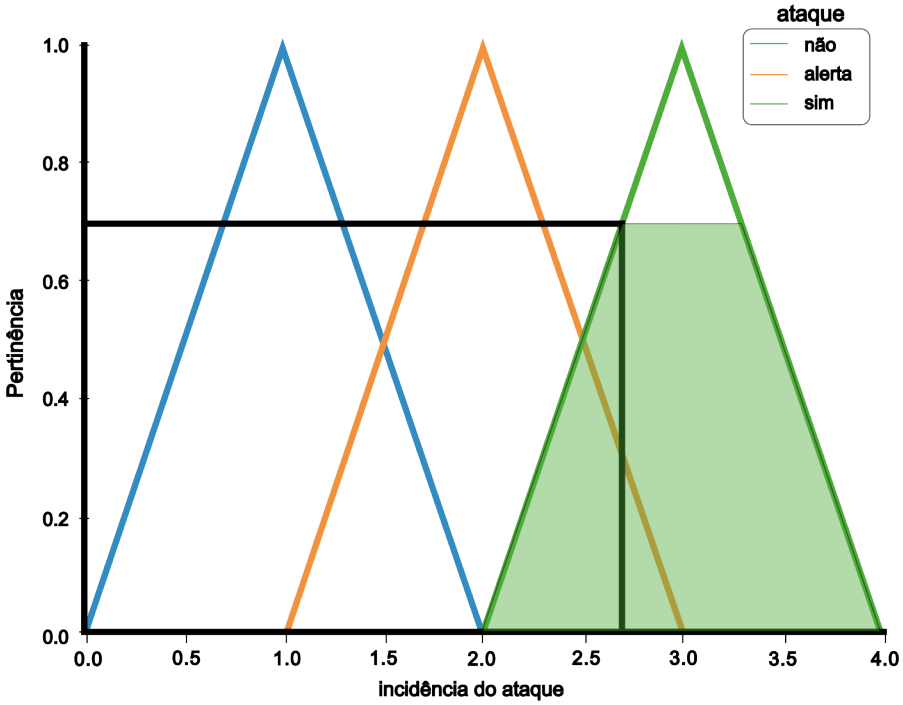
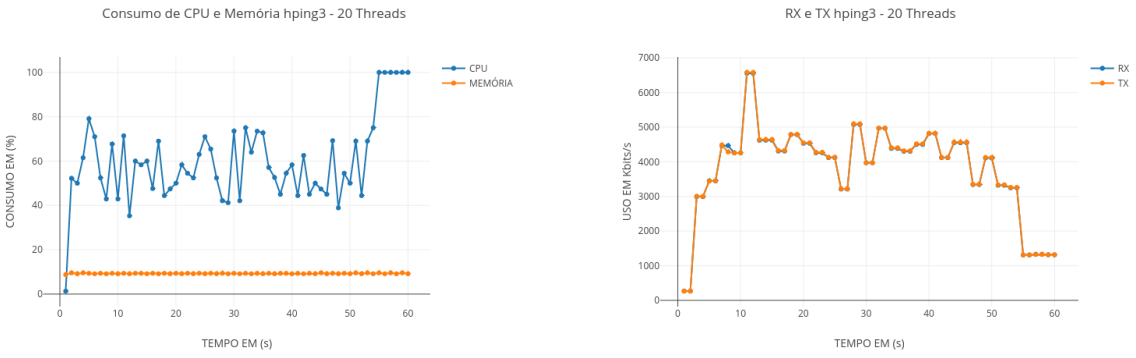


Figura 35 – Inferência *fuzzy* - Hping 20 *threads*

A Figura 36 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Pode-se perceber no gráfico relativo à *CPU* que o processamento foi significativo, ocorrendo um número maior do que o uso da memória. Para RX e TX, a máquina Alvo conseguiu processar as requisições recebidas neste cenário, mas caraterizou um *ataque* com alto grau de pertinência.



(a) CPU e Memória

(b) RX e TX

Figura 36 – Gráficos das Variáveis Linguísticas - Hping 20 *threads*

A Figura 37 apresenta a inferência *fuzzy* para o teste realizado com *hping3* parametrizado para 40 *threads*. Este teste foi feito de forma extra para a validação desta arquitetura, uma vez que o valor de saída *fuzzy* foi diferente do esperado, já que o sistema classificou como *ataque* o teste com 20 *threads*. O valor da saída, conforme demonstrado na Tabela 18 foi de 2.470, incindindo em uma pertinência pouco maior para *alerta*, mas caracterizando positivamente a ocorrência de um *ataque*.

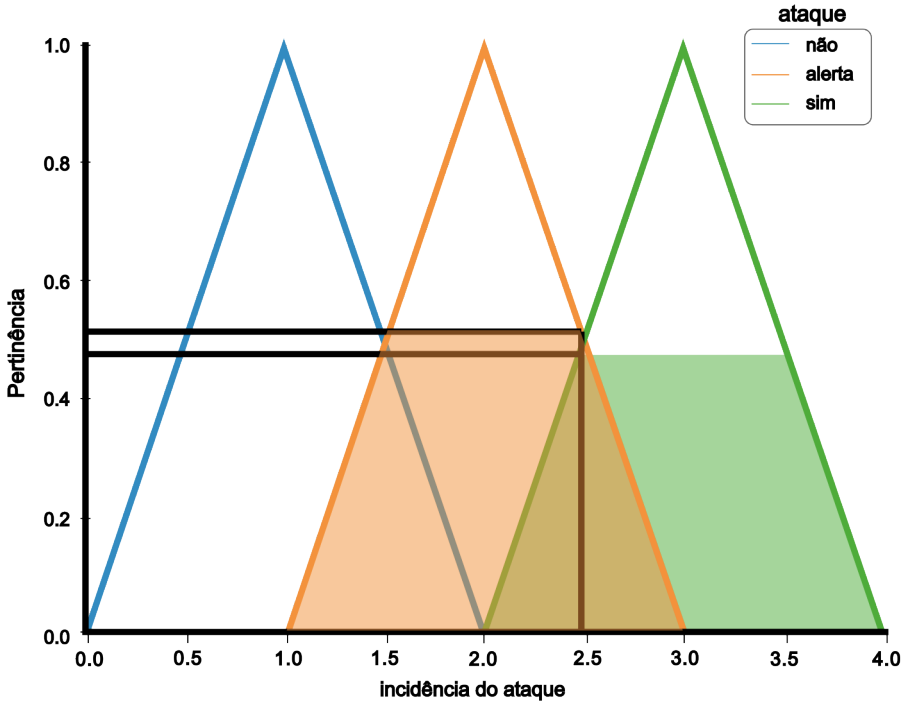
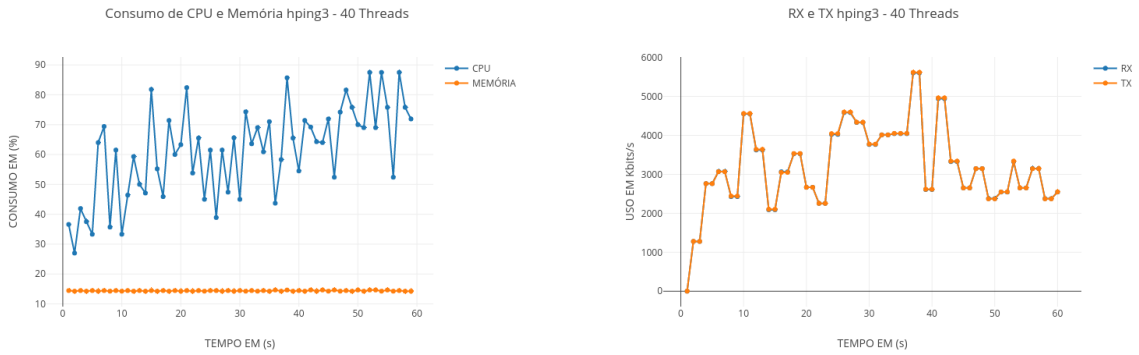


Figura 37 – Inferência *fuzzy* - Hping 40 *threads*

A Figura 38 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. No gráfico relativo à *CPU*, observa-se que uso foi bastante intenso no processamento das requisições. Para RX e TX, novamente se assemelha aos anteriores, porém, difere do resultado esperado, o qual deveria apresentar valores maiores.



(a) CPU e Memória

(b) RX e TX

Figura 38 – Gráficos das Variáveis Linguísticas - Hping 40 *threads*

4.2.2 Teste de validação para o Cenário 02

A Tabela 19 apresenta as estatísticas das variáveis linguísticas coletadas durante os testes efetuados com a ferramenta *T50*, com 250.000 e 500.000 pacotes parametrizados, sendo que cada execução independe do tempo, fazendo o disparo até atingir a quantidade solicitada. Para

cada envio de pacotes há uma estatística e uma saída do sistema *fuzzy*, que indica a pertinência para a variável linguística de saída *ataque*.

Tabela 19 – Testes com T50 - Validação

T50 - 250.000 pacotes	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	9.46	18.95	1171.71	0.36
Variância	18.34	0.04	222605.3	0.01
Desvio Padrão	4.28	0.2	471.81	0.1
Mediana	10	18.96	1321.91	0.38
Valor Máximo	15.9	19.16	1483.33	0.47
Valor Mínimo	1	18.76	0	0
Intervalo de confiança	7.61 - 11.31	18.87 - 19.04	967.68 - 1375.73	0.32 - 0.40
Saída Fuzzy	2.307			
T50 - 500.000 pacotes	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	11.76	20.11	1417.74	0.4
Variância	5.98	0.04	3703.8	0
Desvio Padrão	2.44	0.2	60.86	0.03
Mediana	11.7	19.92	1439.15	0.39
Valor Máximo	19	20.32	1486.14	0.49
Valor Mínimo	5.1	19.92	1235.66	0.37
Intervalo de confiança	11.16 - 12.36	20.06 - 20.16	1402.78 - 1432.70	0.39 - 0.41
Saída Fuzzy	2.193			

A Figura 39 apresenta a inferência *fuzzy* para o teste realizado com *T50* parametrizado para 250.000 pacotes. O valor da saída, conforme demonstrado na Tabela 19 foi de 2.307, incindindo em uma pertinência menor para *ataque*, caraterizando então um *alerta*.

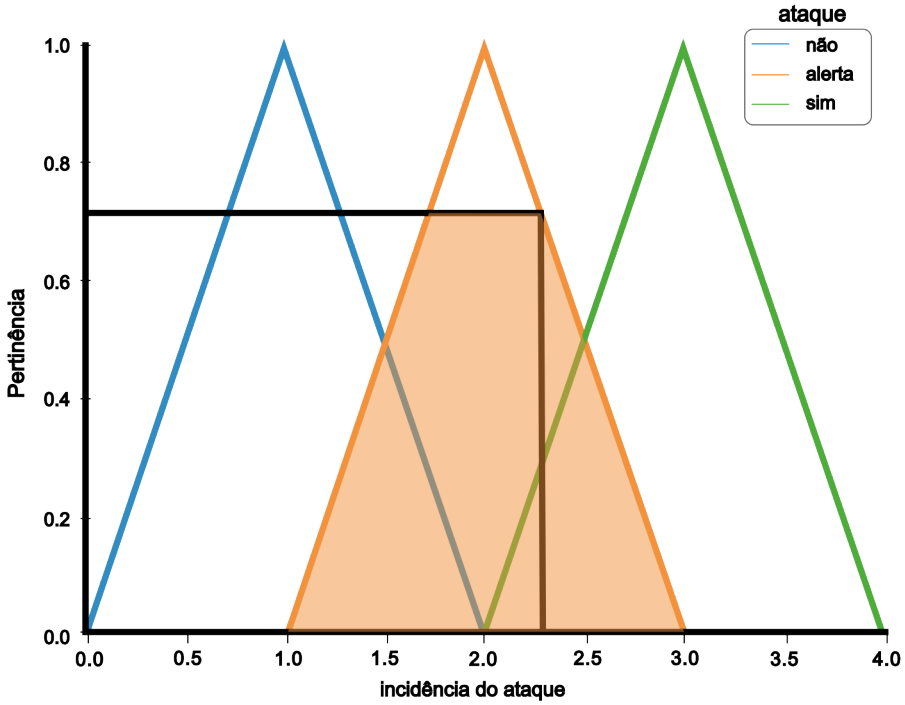
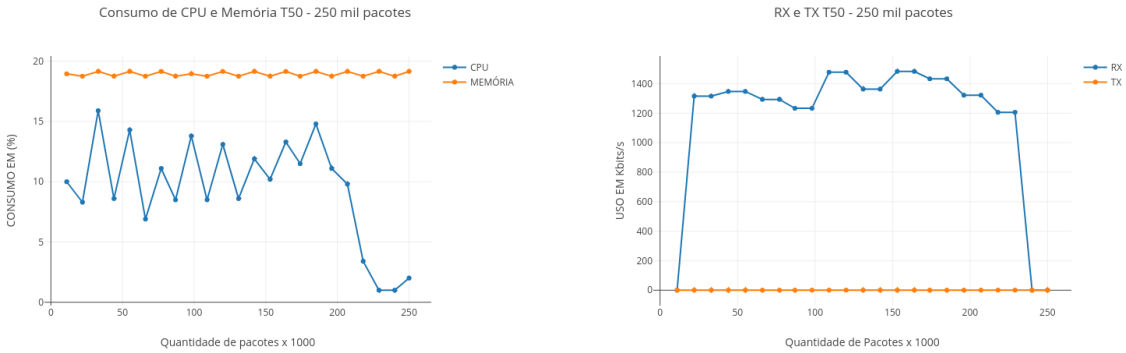


Figura 39 – Inferência *fuzzy* - T50 250000 pacotes

A Figura 40 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Neste cenário, houve uma variação constante do uso de *CPU*, mas com baixos valores. A memória manteve-se constante, mas também apresentou valores baixos, sem muito processamento. No gráfico referente às variáveis RX e TX, observa-se que houve uma inundação da máquina, onde RX manteve-se alto e TX manteve-se baixo, denotando que máquina Alvo não conseguiu processar e devolver as requisições, ocorrendo o *flood*.



(a) CPU e Memória

(b) RX e TX

Figura 40 – Gráficos das Variáveis Linguísticas - T50 com 250000 pacotes

A Figura 41 apresenta a inferência *fuzzy* para o teste realizado com *T50* parametrizado para 500.000 pacotes. O valor da saída, conforme demonstrado na Tabela 19 foi de 2.193. Neste caso, ocorreu um falso-negativo, pois a saída *fuzzy* deveria caracterizar um *ataque*, porém a

pertinência maior ocorreu para *alerta*. As linhas horizontais ilustram esta pertinência para cada saída.

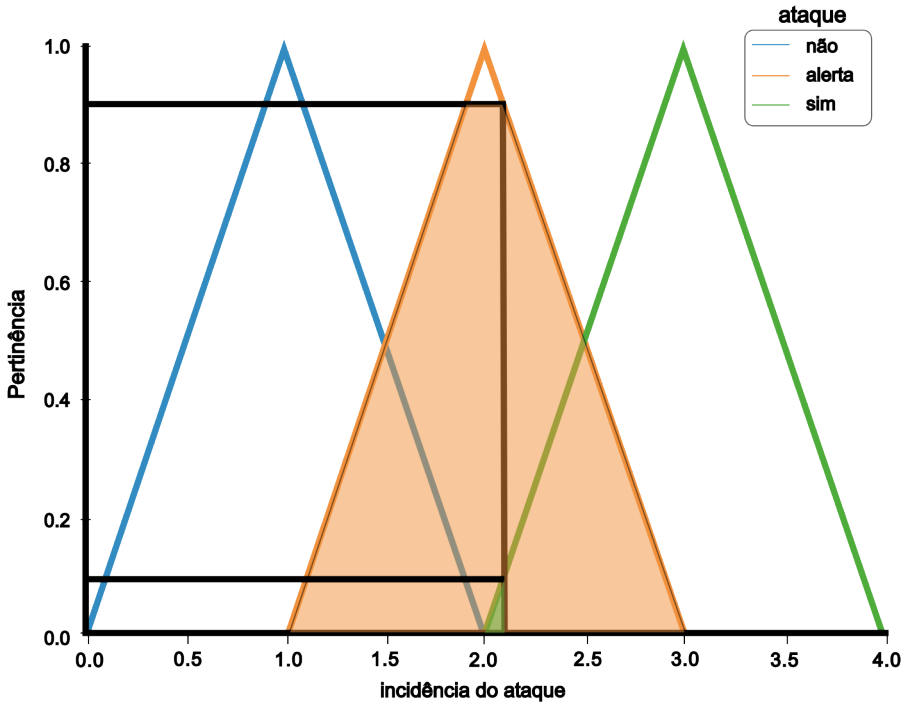
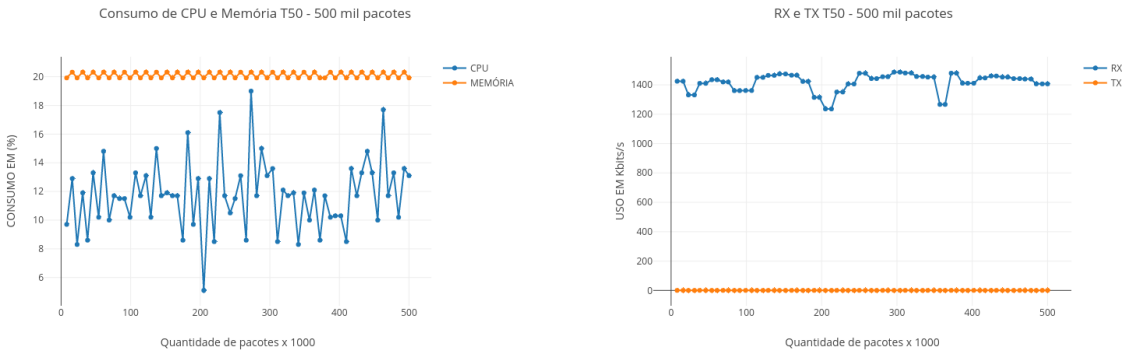


Figura 41 – Inferência *fuzzy* - T50 500000 pacotes

A Figura 42 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Semelhante ao teste anterior, o gráfico relativo à *CPU* demonstra uma taxa de uso variável, mas com baixos valores e memória constante. Para RX e TX, novamente se assemelha ao anterior, porém com taxas mais constantes no envio de requisições à máquina Alvo, ocorrendo o *flood*, uma vez que a variável TX apresentou valores quase zerados.



(a) CPU e Memória

(b) RX e TX

Figura 42 – Gráficos das Variáveis Linguísticas - T50 com 500000 pacotes

4.2.3 Teste de validação para o Cenário 03

A Tabela 20 apresenta as estatísticas das variáveis linguísticas coletadas durante os testes efetuados com a ferramenta *JMeter*, com 300 e 500 usuários, simulando acesso legítimo à máquina Alvo, com o sistema web. Este conjunto de testes é independente de tempo, no entanto a ferramenta faz a simulação de acessos de forma mais rápida que um usuário real. Para cada conjunto de usuários há uma estatística e uma saída do sistema *fuzzy*, que indica a pertinência para a variável linguística de saída *ataque*. Neste cenário, deve-se ocorrer o menor número possível de classificação de *ataques*, para que não haja falso-positivo.

Tabela 20 – Testes com JMeter - Validação

JMeter 300 usuários	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	5.25	24.7	180.24	254.46
Variância	54.12	0.01	111008.9	237330.1
Desvio Padrão	7.36	0.32	333.18	487.17
Mediana	2	24.6	5.58	2.76
Valor Máximo	40	25.2	1191.26	1720.33
Valor Mínimo	1	24.2	0.13	0.22
Intervalo de confiança	3.04 - 7.46	24.61 24.80	80.14 - 280.34	108.10 - 400.82
Saída Fuzzy	2.126			
JMeter 300 usuários	**CPU (%)**	**MEMORIA (%)**	**RX (kbps)**	**TX (kbps)**
Média	8.03	27.66	384.71	482.19
Variância	59.02	0.04	225690.3	370613.4
Desvio Padrão	7.68	0.21	475.07	608.78
Mediana	5.1	27.72	122.4	121.57
Valor Máximo	26.7	28.12	1312.27	1656.83
Valor Mínimo	1	27.16	0.13	0.22
Intervalo de confiança	5.91 - 10.15	27.60 - 27.71	253.77 - 515.66	314.39 - 649.99
Saída Fuzzy	2.223			

A Figura 43 apresenta a inferência *fuzzy* para o teste realizado com *JMeter* parametrizado para 300 usuários. O valor da saída, conforme demonstrado na Tabela 20 foi de 2.126. A saída *fuzzy* indica uma pertinência alta para *alerta*, uma vez que este número de acessos simultâneos no sistema ocasionou fluxo elevado de pacotes.

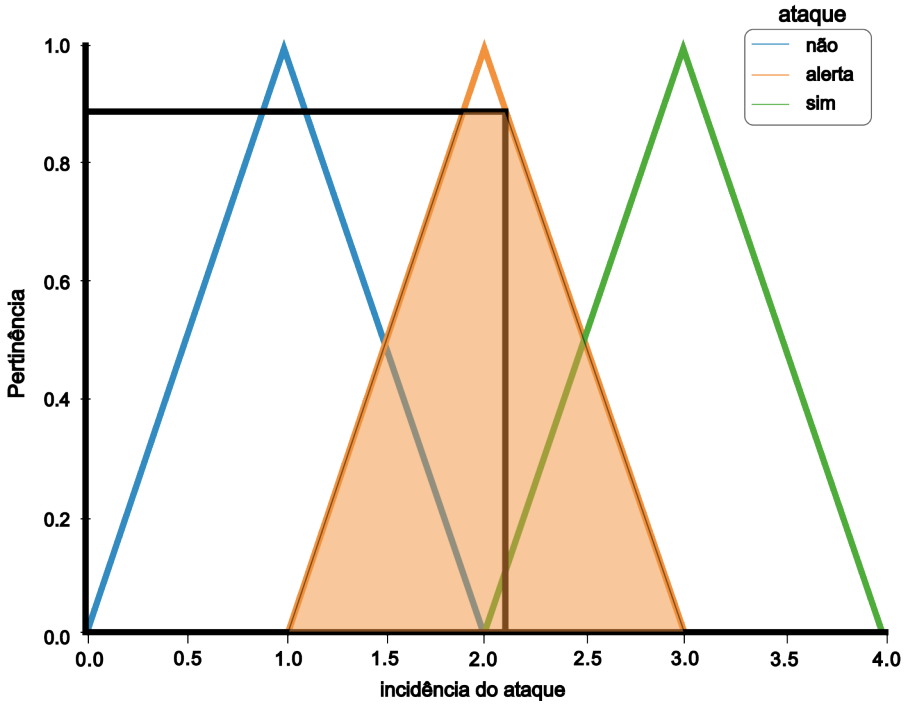
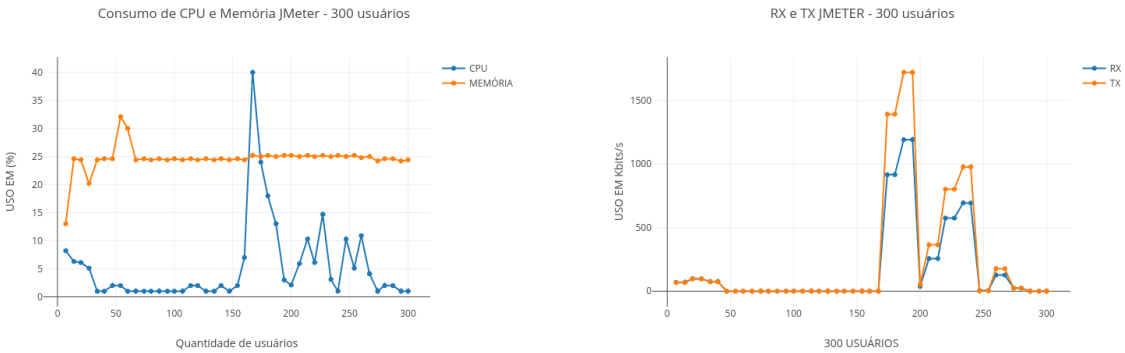


Figura 43 – Inferência *fuzzy* - JMeter 300 usuários

A Figura 44 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. Como neste cenário houve interação com o banco de dados, o uso de memória foi mais acentuado em relação aos outros testes. O uso de *CPU*, no geral, apresentou valores baixos, com apenas uma acentuação. No gráfico referente às variáveis RX e TX, observa-se que inicialmente, os valores são mínimos, pois ocorre primeiro requisições à página inicial do sistema. Logo após, ocorrem valores mais altos, com semelhanças entre pacotes de entrada e saída, onde foram enviados os dados referente ao cadastro no sistema, parametrizado no teste.



(a) CPU e Memória

(b) RX e TX

Figura 44 – Gráficos das Variáveis Linguísticas - JMeter 300 usuários

A Figura 45 apresenta a inferência *fuzzy* para o teste realizado com *JMeter* parametrizado para 500 usuários. O valor da saída, conforme demonstrado na Tabela 20 foi de 2.223. A saída

fuzzy indica uma pertinência alta para *alerta*, porém menor que a anterior, tendendo um pouco para pertinência relativa à *ataque*. Contudo, a saída *fuzzy* classificou com êxito como um *alerta*.

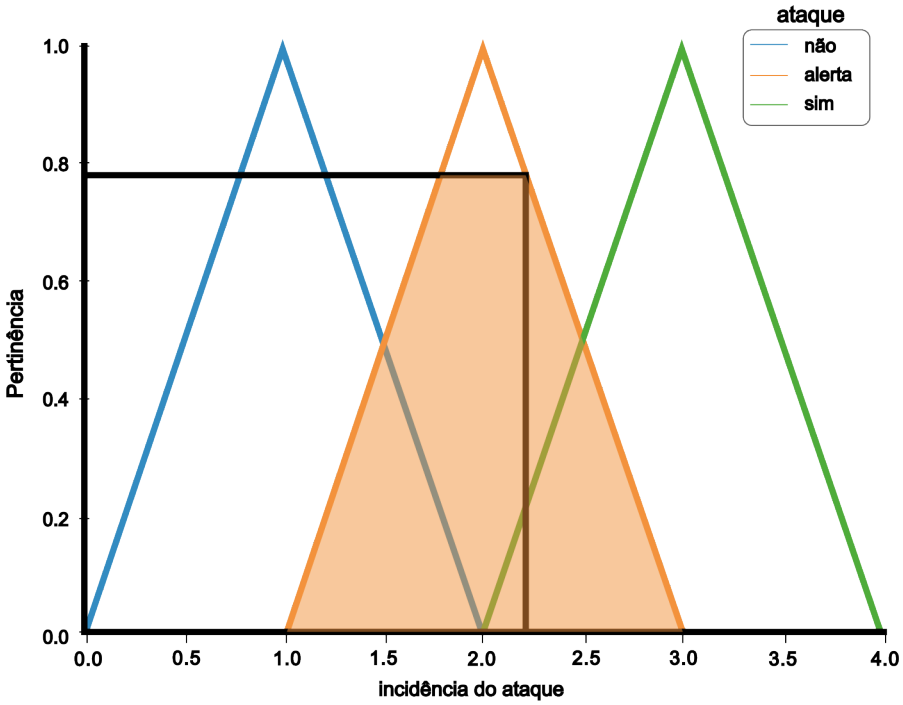
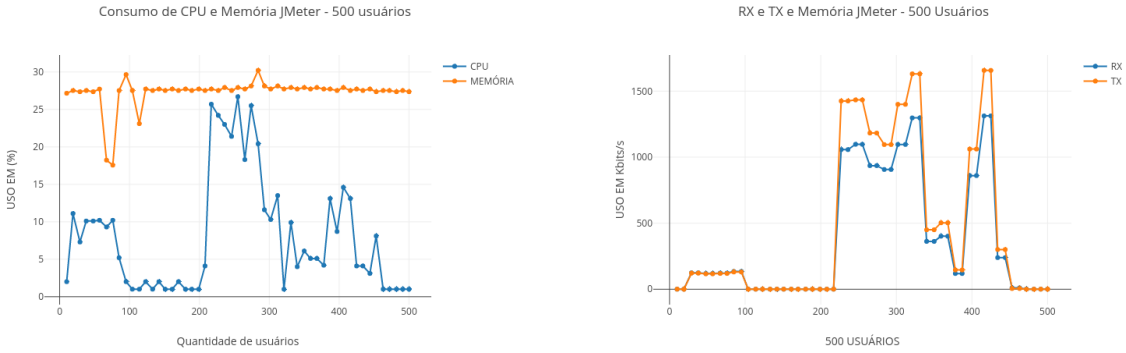


Figura 45 – Inferência *fuzzy* - JMeter 500 usuários

A Figura 46 apresenta o comportamento das variáveis linguísticas coletadas durante o teste. O uso de memória também foi mais acentuado em relação aos outros testes, devido às interações com a base de dados. Percebe-se neste comportamento, em relação ao anterior, que o uso de *CPU* foi mais significativo no que tange o processamento de requisições legítimas. No gráfico referente às variáveis RX e TX, observa-se que inicialmente, os valores são mínimos, pois ocorre primeiro requisições à página inicial do sistema. Logo após, ocorrem valores mais altos, com semelhanças entre pacotes de entrada e saída, onde foram enviados os dados referente ao cadastro no sistema, todavia maiores que o teste com 300 usuários, devido à quantidade de requisições e dados trocados.



(a) CPU e Memória

(b) RX e TX

Figura 46 – Gráficos das Variáveis Linguísticas - JMeter 500 usuários

A Tabela 21 apresenta de forma geral as classificações obtidas com a solução *FuzzSec*, proposta neste trabalho. Os cenários definidos neste trabalho não contemplaram situação de tráfego *mínimo*, devido a não necessidade. São também apresentados os percentuais de acurácia e taxa de erros da solução, onde classificou corretamente 5, dos 7 conjuntos de testes realizados na validação da arquitetura.

Tabela 21 – Resultado das Classificações obtidas com *FuzzSec*

CONJUNTO DE TESTES	SAÍDA FUZZY	CLASS. ESPERADA	CLASS. FUZZSEC
hping3 10 threads	2.363	Ataque	Alerta
hping3 20 threads	2.680	Ataque	Ataque
hping3 40 threads	2.470	Ataque	Ataque
T50 250.000 pacotes	2.307	Alerta	Alerta
T50 500.00 pacotes	2.193	Ataque	Alerta
JMeter 300 usuários	2.126	Alerta	Alerta
JMeter 500 usuários	2.223	Alerta	Alerta
		Acurácia	71,42%
		Taxa de erros	28,58%

A Figura 47 apresenta a porcentagem para cada tipo de classificação obtida com a solução proposta.

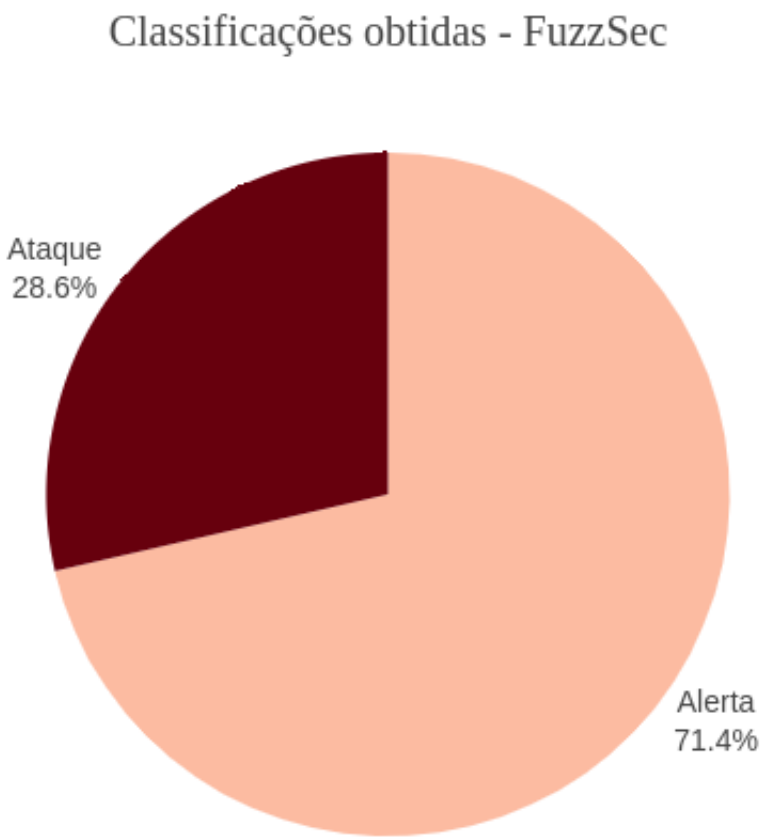


Figura 47 – Porcentagem de classificação

5 Considerações Finais

Quando se pensa em ataque distribuído de negação de serviço, *DDoS*, logo imagina-se um cenário global, no qual *hackers* sequestram máquinas e executam códigos maliciosos para atingir um alvo e o tornar indisponível. Normalmente é assim que acontece. Mas, há um cenário que é pouquíssimo abordado, é que um ataque vindo de dentro da própria instituição/organização. Pode acontecer de um funcionário descontente, que está para sair da empresa ou um aluno mal intencionado nas dependências da universidade realizar um ataque interno, com o intuito de tornar indisponível os serviços providos pela instituição. Foi este o cenário abordado no presente trabalho, de forma a entender, desmitificar e apresentar uma proposta de solução para combater o *flood TCP SYN*.

As técnicas e ferramentas utilizadas para obtenção dos resultados são conhecidas e difundidas em diversos tipos de aplicações, logo tornou-se fácil a obtenção de material para difundir os estudos e aplicá-las no desenvolvimento. Para tornar possível a identificação e análise do *flooding TCP SYN*, foi necessário primeiramente analisar o tráfego da rede e filtrar somente aquilo que era desejado, no caso da *flag TCP SYN* para abertura de conexões. A verificação da incidência de ataques iniciou-se de forma simples, com a contagem das linhas do arquivo do *sniffer*, para então dar o *start* na coleta das métricas, as quais serviram de entrada para o sistema *fuzzy*. Estes dados passaram pelo processo de fuzificação, processador *fuzzy* e deffuzyficação, para então entrar no decisor, ativando ou não o mecanismo de defesa.

Após todos esse processo, foram analisadas as métricas e saídas, gerando tabelas estatísticas e gráficos de inferência *fuzzy* para melhor compreensão. Os resultados *fuzzy* inicialmente não apresentaram valores satisfatórios devido à falta de acurácia do sistema, necessitando realizar um ajuste fino nos parâmetros e posteriormente validá-los. Após estes ajustes, os resultados adquiridos com os mesmos conjuntos de testes apresentaram maior taxa de êxito na análise e classificação do *flood TCP SYN*, provendo saídas adequadas para qualquer módulo de *firewall* que venha a ser integrado com esta solução. Com isso, o módulo *FuzzSec* pode ser utilizado em ambiente de produção, uma vez os resultados de validação se mostraram bastante satisfatórios. Com isso, foram cumpridos os objetivos propostos neste trabalho, de forma geral, determinando se houve tentativa de ataque através da análise do tráfego e em seguida classificando-o para tomada de decisão.

5.1 Trabalhos futuros

Pretende-se como trabalhos futuros:

- Realizar novos testes para treinar o conjunto da solução e melhorar ainda mais a eficácia da classificação do *flood TCP SYN*, através dos ajustes de parâmetros e conjuntos *fuzzy*;
- Utilizar como SO da solução o FreeBSD com *firewall* IPFW, o qual fazia parte da proposta inicial;
- Realizar estudos para possibilidade de embarcar a solução em um *Raspberry Pi*.
- Comparar a solução *FuzzSec* com uma solução IDS/IPS de mercado, tal como o Suricata e OSSEC.

Referências

BAQUI, R. B. T. *Segurança de Redes Linux com Firewall*. Monografia (Especialista em Configuração e Gerenciamento de Servidores e Equipamentos de Redes) — Centro Federal de Educação Tecnológica do Paraná, 2012.

BARDAL, R. M. *Estudo sobre ataques de negação de serviço e uma abordagem prática.* Monografia (Curso de Especialização Semipresencial em Configuração e Gerenciamento de Servidores e Equipamentos de Rede) — Universidade Tecnológica Federal do Paraná, 2014.

BHATTACHARYYA, D. K.; KALITA, J. K. *DDOS Attacks. Evolution, Detection, Prevention, Reaction and Tolerance*. [S.l.]: CRC Press, 2016.

BULLOCK JESSEY. PARKER, J. T. *Wireshark para Profissionais de Segurança. Usando Wireshark e o Metasploit Framework*. [S.l.]: Editora Novatec, 2017. ISBN 9788575225998.

COUTO, A. V. d. *Uma abordagem de Gerenciamento de Redes baseado no Monitoramento de Fluxos de Tráfego Netflow com o suporte de Técnicas de Business Intelligence*. Dissertação (Mestrado em Engenharia Elétrica - Faculdade de Tecnologia - Departamento de Engenharia Elétrica) — Universidade de Brasília, 2012.

IBRAHIM, I. *Conjunto de Protocolo TCP/IP e suas falhas*. Monografia (Curso de Especialização em Teleinformática e redes de Computadores) — Universidade Tecnológica Federal do Paraná, 2011.

KUROSE JIM. ROSS, K. W. *Redes de Computadores e a Internet – Uma abordagem Top Down*. 6. ed. [S.l.]: Editora Pearson, 2013.

LISKA, A.; STOWE, G. *Segurança de DNS*. [S.l.]: Editora Novatec, 2016. 304 p.

MARRO ALESSANDRO ASSI, A. M. d. C. S. E. R. d. S. C. G. S. B. R. d. O. N. Lógica fuzzy: Conceitos e aplicações. *Departamento de Informática e Matemática Aplicada (DIMAp)*, 2010. Universidade Federal do Rio Grande do Norte (UFRN).

MCNAB, C. *Avaliação de segurança de redes*. [S.l.]: Editora Novatec, 2017.

MELO, S. *Exploração de vulnerabilidades em Redes TCP/IP*. 3. ed. [S.l.]: Alta Books, 2017.

NETO, L. B. et al. Minicurso de sistema especialista nebuloso. In: *XXXVIII SIMPÓSIO BRASILEIRO PESQUISA OPERACIONAL*. [S.l.: s.n.], 2006.

PEDRYCZ W. GOMIDE, F. *Fuzzy Systems Engineering: Toward Human-Centric Computing*. 1. ed. [S.l.]: Wiley/IEEE Press, 2007. ISBN 978-0-471-78857-7.

RIGNEL D. G. DE S., C. G. P. L. C. Revista eletrônica de sistemas de informação e gestão tecnológica. *Uma introdução a lógica fuzzy.*, v. 1, 2011.

RIOS, V. d. M. *Seleção de redes sem fio baseada em técnicas de apoio à decisão*. Dissertação (Mestrado em Engenharia Elétrica - Faculdade de Tecnologia - Departamento de Engenharia Elétrica) — Universidade de Brasília, 2012.

SADOK D. FEITOSA, E. S. E. Tráfego internet não desejado: Conceitos, caracterização e soluções. *Centro de Informática – Grupo de Pesquisa em Redes e Telecomunicações (GPRT)*, 2008.

SANTOS, A. F. P. D. *Identificação e Análise de Comportamentos Anômalos*. Tese (Doutorado em Modelagem Computacional) — Laboratório Nacional de Computação Científica, 2009.

SILVA, E. d. C. d. *Detecção de ataques de negação de serviço utilizando aprendizado de máquina*. Dissertação (Mestrado em Informática - Centro de Ciências Exatas e Tecnológicas) — Universidade Federal do Estado do Rio de Janeiro, 2016.

SOEIRA, B. V. et al. *Implementação do Firewall Cisco ASA 5500*. Monografia (Curso Superior de Tecnologia em Sistemas de Telecomunicações) — Universidade Tecnológica Federal do Paraná, 2013.

SOLHA, L. E. V. A. et al. Tudo que você precisa saber sobre os ataques ddos. *RNP - Rede Nacional de Ensino e Pesquisa*, v. 4, n. 2, 2000. ISSN 1518-5974.

SOUSA, J. N. d. P. E. *Aplicação de Lógica Fuzzy em Sistemas de Controle de Tráfego Metropolitano em Rodovias Dotadas de Faixas Exclusivas para Ônibus*. Dissertação (Mestrado em Ciências em Engenharia de Transportes) — Universidade Federal do Rio de Janeiro, 2005.

STREIJL, R. et al. Mean opinion score (mos) revisited: methods and applications, limitations and alternatives. *Multimedia Systems Journal*, v. 22, n. 2, 2016.

TRILLAS, E.; ECIOLAZA, L. *Fuzzy Logic. An Introductory Course for Engineering Students.* [S.l.]: Springer, 2015.

VAZ, A. M. *Estudos das Funções de pertinência para conjuntos fuzzy utilizados em Controladores Semafóricos Fuzzy*. Dissertação (Mestrado em Transportes Urbanos) — Universidade de Brasília, 2006.

Apêndices

APÊNDICE A – INSTALAÇÃO E CONFIGURAÇÃO DO AMBIENTE

O processo de instalação das ferramentas e inclusão dos *scripts* utilizados, estão detalhados nas seções subsequentes.

A.1 Aplicação Alvo: Instalando Apache2.4 + PHP7 + MariaDB10

Primeiramente atualize os repositório do sistema operacional Debian9 com os comandos abaixo:

```
# apt-get update
# apt-get upgrade
```

Em seguida, instale o servidor web Apache2 e outros pacotes necessários, instale o PHP7 e alguns módulos importantes. E o servidor e cliente MariaDB. Abaixo os comandos necessários para instalação de todos os pacotes:

```
# apt-get install apache2 libapache2-mod-php7.0 -y

# apt-get install php7.0 php7.0-cli php7.0-curl php7.0-gd php7.0-intl
    php7.0-json php7.0-mbstring php7.0-mcrypt php7.0-mysql php7.0-opcache
    php7.0-readline php7.0-xml php7.0-xsl php7.0-zip php7.0-bz2
    libapache2-mod-php7.0 -y

# apt-get install mariadb-server mariadb-client -y
```

Reinicie o servidor Apache2 executando o comando:

```
# systemctl restart apache2
```

Agora, tanto o servidor da Web Apache2, incluindo o PHP 7, quanto o servidor MariaDB estão prontos para uso. Por padrão, o diretório da web é "/var/www/html".

A.1.1 *Script* para coleta dos acessos legítimos ao banco (metricas.py)

Abaixo segue o *script* usado para coleta das métricas:

```
import csv
```

```
import time
from threading import Thread
import commands
import psutil

class Info:
    def __init__(self):
        self.data = ['0', '0', '0', '0']

    #cria o arquivo csv
    def create_csv(self):
           try:
               with open("metric_data.csv", "wb") as fp:
                   c = csv.writer(fp, delimiter=',')
           except:
               print('erro ao criar arquivo')

           finally:
               fp.close

    #salva os dados, abrindo o arquivo criado
    def save_data(self):
        print(self.data)
        try:
            with open("metric_data.csv", "ab") as fp:
                c = csv.writer(fp, delimiter=',')
                c.writerow(self.data)
        except:
            print('erro ao inserir no arquivo')
        finally:
            fp.close

    #pega cpu em porcentagem. Caso tenha mais de 1 core e queira pega-los
        individualmente, coloque TRUE no parametro 'percpu'
    def cpu(self):
           self.data[0] = '0'

           for i in range(0, 200, 1):
               cpu_percent = psutil.cpu_percent(percpu=False)

               if(cpu_percent > 0):
                   self.data[0] = cpu_percent
```

```
                self.save_data()

            time.sleep(1)

    #pega a memoria usada e converte para porcentagem
    def memory(self):
        cmd = ' free -m | grep Mem'
        init.data[1] = '0'

        for i in range (1, 200, 1):
            out = commands.getoutput(cmd)
            mem_percent = round(float(out.split()[2])*100/512, 2)

            if(mem_percent > 0):
                self.data[1] = mem_percent
            time.sleep(1)

    #pega o RX e TX da placa de rede em Kbits/s. Se quiser pegar a qtd de
        pacotes, altere o pametro print{$5, $6} para $3 e $4
    def network(self):
        init.data[2] = '0'
        init.data[3] = '0'

        cmd = 'sar -n DEV 1 1 | grep ens192 | tail -n 1 | gawk \'{print $5, $6}\''

        for i in range (1, 200, 1):
            out = commands.getoutput(cmd)
            rx = float(out.split()[0])
            tx = float(out.split()[1])

            if(rx > 0):
                self.data[2] = rx

            if(tx > 0):
                self.data[3] = tx

            time.sleep(1)


    def main(self):

        #define as threads
```

```
        self.create_csv()
        tCpu = Thread(target=self.cpu, args=())
        tMem = Thread(target=self.memory, args=())
        tNetwork = Thread(target=self.network, args=())

        tCpu.start()
        tMem.start()
        tNetwork.start()

        tMem.join()
        tCpu.join()
        tNetwork.join()

init = Info()
init.main()
```

A.2 *Firewall*: Instalando IPTables + IpSet + Python2.7

Primeiro, precisamos instalar alguns pacotes de software necessários. Como visto no comando abaixo:

```
# apt-get install -y iptables-persistent
# invoke-rc.d netfilter-persistent save
# service netfilter-persistent start
```

Quando os pacotes acima estiverem instalados, aparecerá um novo diretório em /etc/iptables/. Esse diretório mantém as regras de filtro do *IPTables* que serão recarregadas na inicialização do sistema.

Para que as novas regras tenham efeito, basta reiniciar o serviço *netfilter-persistent* conforme mostrado abaixo.

```
# service netfilter-persistent restart
# iptables -L
```

Para instalação do IpSet use o comando abaixo:

```
# apt-get install ipset
```

Para rodar o script que possui a lógica *fuzzy* é necessário a instalação da biblioteca *Scikit-fuzzy* e pra seu correto funcionamento todas as dependências a seguir devem ser instaladas:

```
# pip install numpy
# pip install setuptools
# pip install matplotlib
# pip install scipy
```

E por fim:

```
# pip install scikit-fuzzy
```

A.2.1 *Script* do processador *fuzzy* (fuzzsec.py)

Abaixo segue o *script* usado para o processamento da lógica fuzzy:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

#Variaveis linguisticas e saida.
cpu = ctrl.Antecedent(np.arange(0, 101, 1), 'cpu')
memoria = ctrl.Antecedent(np.arange(0, 101, 1), 'memoria')
rx = ctrl.Antecedent(np.arange(0, 101, 1), 'rx')
tx = ctrl.Antecedent(np.arange(0, 401, 1), 'tx')
ataque = ctrl.Consequent(np.arange(0, 5, 1), 'ataque')

#subconjunto da saida
ataque['nao'] = fuzz.trimf(ataque.universe, [1, 1, 1])
ataque['alerta'] = fuzz.trimf(ataque.universe, [1, 2, 3])
ataque['sim'] = fuzz.trimf(ataque.universe, [2, 3, 4])

#subconjunto fuzzy para CPU
cpu['minimo'] = fuzz.trimf(cpu.universe, [0, 2, 7])
cpu['normal'] = fuzz.trimf(cpu.universe, [5, 12, 36])
cpu['intenso'] = fuzz.trimf(cpu.universe, [12, 53, 65])
cpu['ddos'] = fuzz.trimf(cpu.universe, [53, 70, 100])

#subconjunto fuzzy para MEMORIA
memoria['minimo'] = fuzz.trimf(memoria.universe, [0, 20, 47])
memoria['normal'] = fuzz.trimf(memoria.universe, [34, 47, 66])
memoria['intenso'] = fuzz.trimf(memoria.universe, [63, 66, 83])
memoria['ddos'] = fuzz.trimf(memoria.universe, [66, 83, 100])

#subconjunto fuzzy para RX
rx['minimo'] = fuzz.trimf(rx.universe, [0, 20, 47])
rx['normal'] = fuzz.trimf(rx.universe, [34, 47, 66])
rx['intenso'] = fuzz.trimf(rx.universe, [63, 66, 83])
rx['ddos'] = fuzz.trimf(rx.universe, [66, 83, 100])

#subconjunto fuzzy para TX
tx['minimo'] = fuzz.trimf(tx.universe, [0, 0.58, 16.09])
tx['normal'] = fuzz.trimf(tx.universe, [13.69, 16.09, 34.82])
tx['intenso'] = fuzz.trimf(tx.universe, [30, 78.17, 274.43])
tx['ddos'] = fuzz.trimf(tx.universe, [200, 274.43, 400])
```

```
#Secao de Regras

#Regras que garantem o bloqueio
rule1 = ctrl.Rule(rx['ddos'] | tx['ddos'], ataque['sim'])
rule2 = ctrl.Rule(cpu['ddos'] | memoria['ddos'], ataque['sim'])
rule3 = ctrl.Rule(cpu['ddos'] & memoria['intenso'], ataque['sim'])
rule4 = ctrl.Rule(cpu['intenso'] & memoria['ddos'], ataque['sim'])
rule5 = ctrl.Rule(rx['intenso'] & tx['ddos'], ataque['sim'])
rule6 = ctrl.Rule(rx['ddos'] & tx['intenso'], ataque['sim'])

#regras de alerta
rule7 = ctrl.Rule(cpu['intenso'] | memoria['intenso'], ataque['alerta'])
rule8 = ctrl.Rule(rx['intenso'] | tx['intenso'], ataque['alerta'])
rule9 = ctrl.Rule(cpu['intenso'] & memoria['normal'], ataque['alerta'])
rule10 = ctrl.Rule(cpu['normal'] & memoria['intenso'], ataque['alerta'])
rule11 = ctrl.Rule(cpu['normal'] & memoria['intenso'], ataque['alerta'])

#regras sem bloqueio
rule12 = ctrl.Rule(cpu['minimo'] | memoria['minimo'], ataque['nao'])
rule13 = ctrl.Rule(rx['minimo'] | tx['minimo'], ataque['nao'])
rule14 = ctrl.Rule(cpu['minimo'] | memoria['normal'], ataque['nao'])
rule15 = ctrl.Rule(cpu['normal'] | memoria['minimo'], ataque['nao'])
rule16 = ctrl.Rule(rx['normal'] | tx['minimo'], ataque['nao'])
rule17 = ctrl.Rule(rx['minimo'] | tx['normal'], ataque['nao'])

rules = []

for i in range (1, 17):
    rules.append(eval("rule" + str(i)))

#Insercao das regras no sistema fuzzy
ataque_ctrl = ctrl.ControlSystem(rules)
sistema_fuzzy = ctrl.ControlSystemSimulation(ataque_ctrl)

#Entradas ficticias (teste)
#sistema_fuzzy.input['cpu'] = 90
#sistema_fuzzy.input['memoria'] = 80
#sistema_fuzzy.input['rx'] = 10
#sistema_fuzzy.input['tx'] = 20
```

```
#Entradas do arquivo
with open('dados_metrica.csv', 'rb') as metricas:
   reader = csv.reader(metricas)
   for linha in reader:
      print (linha)

sistema_fuzzy.compute()

saida = (sistema_fuzzy.output['ataque'])

if (saida >= 2 AND saida < 3):
   os.system("python metricas.py")

if (saida > 3):
   os.system("nucleo.py")
```

A.2.2 *Script* principal (nucleo.py)

Abaixo segue o *script* principal, responsável por disparar o *sniffer* e verificar sua saída:

```
import os
import time
import commands

THRESHOLD = 100000

def call_sniffer():
    os.system("./sniffer")
    time.sleep(20)

def verify_sniffer():
    cmd = "wc -l dados_sniffer.txt"
    out = commands.getoutput(cmd)
    out = round(int(out.split()[0]))

    if(out > THRESHOLD):
        get_ips()
        insert_ips()
    else:
        return 0

def get_ips():
```

```
    os.system('cat dados_sniffer.txt | awk
       \'{match($0,/[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+/); ip =
       substr($0,RSTART,RLENGTH); print ip}\' > lista_ips.txt')

def insert_ips():
   file = open('lista_ips.txt', 'r')
   text = file.readlines()
   for ip in text:
      os.system("ipset -! add ips_blacklist " + ip)

verify_sniffer()
#get_ips()
#time.sleep(0.2)
#insert_ip()

def main():
    call_sniffer()
    ret = verify_sniffer()

    if (ret == 0):
       time.sleep(5)
       main()
```

A.2.3 *Script* para inicializar *sniffer* tcpdump, com filtro *TCP SYN* (sniffer.c)

Abaixo segue o *script* responsável por inicializar *sniffer tcpdump* com o filtro *TCP SYN*:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>

void *funcao_thread(void *arg){
    printf("Iniciou o sniffer...\n");
     system("tcpdump -i enp0s3 \"tcp[tcpflags] & (tcp-syn) != 0\" >>
         dados_sniffer.txt");
     pthread_exit(0);
 }

void *funcao2_thread(void *arg){
     system("tail -f dados.txt.\n");
```

```
    pthread_exit(0);
 }

int main(void){
    int id;
    pthread_t thread1, thread2;

    printf("Thread progenitora.\n");
    pthread_create(&thread1, NULL, &funcao_thread, NULL);
    pthread_create(&thread2, NULL, &funcao2_thread, NULL);
    sleep(20); //tempo de vida da thread progenitora (tempo de coleta do
        sniffer);
    system("killall -9 tcpdump");
    exit(0);
```