**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

# FUNCIONALIDADES E RECURSOS NATIVOS DA COMPUTAÇÃO EM NUVEM NA DETECÇÃO, IDENTIFICAÇÃO E MITIGAÇÃO DE ATAQUES A SERVIÇOS E A CLIENTES: UMA CONTRIBUIÇÃO PELO USO DE APRENDIZADO DE MÁQUINA

**JOÃO HENRIQUE GONÇALVES MEDEIROS CORRÊA**

**ORIENTADOR: PROF. DR. RODOLFO DA SILVA VILLAÇA**
**COORIENTADOR: PROF. DR. MOISÉS RENATO NUNES RIBEIRO**

Vitória – ES

Outubro/2021

# Funcionalidades e recursos nativos da computação em nuvem na detecção, identificação e mitigação de ataques a serviços e a clientes: uma contribuição pelo uso de aprendizado de máquina

João Henrique Gonçalves Medeiros Corrêa

Tese submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Aprovada em 29 de outubrode 2021:

Prof. Dr .Rodolfo da Silva Villaça
Orientador, participação remota

Prof. Dr. Moisés Renato Nunes Ribeiro
Coorientador, participação remota

Prof. Dr. Magnos Martinello
Membro Interno, participação remota

Prof. Dr. Jugurta Rosa Montalvão Filho
Membro Externo, participação remota

ANILTON SALLES GARCIA:39523799720 Assinado de forma digital por ANILTON SALLES GARCIA:39523799720
Dados: 2021.10.30 13:06:46 -03'00'

Prof. Dr. Anilton Salles Garcia
Membro Externo, participação remota

Prof. Dr. Rafael Silva Guimarães
Membro Externo, participação remota

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória/ES, 29 de outubro de 2021.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

PROTOCOLO DE ASSINATURA

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
MAGNOS MARTINELLO - SIAPE 1669875
Departamento de Informática - DI/CT
Em 01/11/2021 às 18:50

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/299115?tipoArquivo=O

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Corrêa, João Henrique Gonçalves Medeiros, 1987-
C824f Funcionalidades e recursos nativos da computação em nuvem na detecção, identificação e mitigação de ataques a serviços e a clientes: uma contribuição pelo uso de aprendizado de máquina / João Henrique Gonçalves Medeiros Corrêa. - 2021.
121 f. : il.

Orientador: Rodolfo da Silva Villaça.
Coorientador: Moisés Renato Nunes Ribeiro.
Tese (Doutorado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Computação em Nuvem. 2. Segurança de Redes. 3. Aprendizado de Máquinas. 4. Telemetria. I. Villaça, Rodolfo da Silva. II. Ribeiro, Moisés Renato Nunes. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

---

# AGRADECIMENTOS

Primeiramente a Deus, pela oportunidade de seus caminhos, por toda condução e proteção durante esse tempo.

À Maria Cristina, minha esposa e melhor amiga, por ter me acompanhado nessa caminhada e me apoiado em todas as minhas decisões. Ter aceitado em ir a uma cidade desconhecida, sem pessoas conhecidas, mas com a certeza de que seria o certo a se fazer. Te amo! Ao Samuel, ainda sendo gerado, mas que já oferece tanto amor, apoio e complementa o sentido de percorrer os caminhos.

À Ana e Pedro, meus pais, pelos seus conselhos, ensinamentos valiosos, e todo amor incondicional.

Aos meus irmãos, Filipe, José Pedro e Gustavo, que me acompanharam desde o início da minha vida.

A todos os meus amigos, seja de perto ou longe, os que fiz em Vitória e os que estão espalhados, que me acompanharam nessa jornada. Obrigado pela amizade sincera, por me indicarem, seja direta ou indiretamente, os melhores caminhos nas minhas dúvidas.

Aos meus orientadores, professores e amigos Rodolfo e Moisés, pela oportunidade de ser orientado por vocês e por todos os ensinamentos. Tenho certeza que cresci, no profissional e no pessoal, durante esse tempo e parte disso devido a vocês. Muito obrigado!

A todos do Núcleo de Estudos em Redes Definidas por Software (NERDS), da UFES, pelo aprendizado primordial para esse estudo. A todos os amigos que fiz durante esse tempo, pelas discussões, aprendizados, incentivos e por todos os cafés compartilhados. Em especial ao Alextian e Cristina (primeiros rostos do Nerds que encontrei na minha chegada a Vitória); ao Ricardo Carminatti, companheiro de mesa de trabalho, de discussões, indicações e aprimoramento; Diego Mafioletti, Victor e Rodolfo Valentim por todas as discussões aprofundadas, e que reverberaram nessa Tese. Ao Epaminondas, Diego Cardoso, Isabella, Pedro, Pablo. A todos os professores e alunos que fazem, ou fizeram, parte do NERDS. À própria UFES por

disponibilizar um local adequado para a realização dos meus estudos.

Agradeço a todos os professores com quem tive contato, que me ajudaram na elaboração e execução dessa Tese. Especialmente aos professores Magnos Martinello, Jugurta, Anilton e Rafael, que puderam auxiliar mais diretamente analisando, avaliando e corrigindo essa Tese.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Sem a bolsa recebida pela CAPES não teria sido possível realizar este trabalho. Agradeço também ao CNPq, Fapes e RNP por todo apoio financeiro aos projetos realizados no NERDS.

*"Se fosse fácil achar o caminho das pedras, tantas pedras no caminho não seria ruim."*

Humberto Gessinger

# RESUMO

Os ataques, sejam eles de negação de serviço ou de intrusão, são desafios permanentes em redes de computadores, com um agravamento maior devido à migração de serviços para ambientes de computação em nuvem. Esse novo paradigma de computação, com serviços que compartilham a mesma infraestrutura, potencializa os problemas gerados por esses ataques, levando a consequências desastrosas para usuários, empresas e corporações.

Na literatura, normalmente são requeridos *middleboxes* de rede, como *Deep Packet Inspectors* (DPI), para realizar a tarefa de detectar os ataques. Esses sistemas acabam sendo dependentes das assinaturas dos ataques, dos protocolos utilizados, além disso, têm uma grande dificuldade na localização da coleta do tráfego dentro do *data center*. Somado a isso, a inserção desses sistemas acarreta um aumento do tempo de serviço, afetando métricas relacionadas a Qualidade de Serviço (QoS) e de Experiência (QoE). Se o tráfego está sendo utilizado em conjunto com algoritmos de criptografia, o funcionamento do DPI fica prejudicado.

Diversas infraestruturas de nuvem têm sistemas de telemetria nativa poderosos, normalmente utilizados para monitoramento dos recursos e para realização de faturamento (*billing*). A Tese aqui defendida é que algoritmos de aprendizado de máquina auxiliam na análise aprofundada dos imensos volumes de informações oriundas do serviço nativo de coleta de dados da infraestrutura da nuvem, que disponibiliza o monitoramento de uma infinidade de métricas tanto de *hosts* físicos quanto virtuais.

Ou seja, os algoritmos de aprendizado de máquina são utilizados nos *datasets* coletados do serviço de telemetria nativa da infraestrutura da nuvem para realizar a detecção e identificação. Esses *datasets* contêm informações da máquina virtual, vítima hospedada no ambiente de nuvem. Após realizar a detecção e identificação, mecanismo do próprio ambiente de nuvem foi aplicado para mitigar os ataques, como exemplificamos com o *autoscaling*. Para desenvolver uma Prova de Conceito (PoC), foi utilizado um ambiente experimental com a plataforma de nuvem "OpenStack", com ataques de negação de serviço e de intrusão. Em relação aos dados da telemetria, estes foram utilizados como entrada de algoritmos de aprendizado de máquina, para classificar a presença de ataques, resultando em boa acurácia e boa relação entre os falsos positivos e os verdadeiros positivos. Por fim, verificou-se que o mecanismo de mitigação ofereceu maior disponibilidade para clientes durante os ataques de negação de serviço.

**Palavras-chave**: Computação em Nuvem. Segurança de Redes. Aprendizado de Máquina. Telemetria.

# ABSTRACT

Attacks, whether denial-of-service or intrusion, are a permanent challenge in computer networks, with a further escalation due to migration of services to cloud computing environments. This new computing paradigm, in which services share the same infrastructure, potentializes the problems generated by these attacks, leading to disastrous consequences for users, enterprises, and corporations.

In the literature, network middleboxes such as Deep Packet Inspectors are usually required to perform the task of detecting these attacks. These systems end up being dependent on attack signatures and specific protocols. Moreover, there is a great difficulty in locating the collection of traffic within the data center. Also, the insertion of these systems leads to an increase in service time, affecting metrics related to Quality-of-Service (QoS) and Experience (QoE). If traffic is being used in conjunction with encryption algorithms, the operation of these systems is impaired.

Several cloud infrastructures have powerful native telemetry systems, commonly used for resource monitoring and billing. Our thesis here is that machine learning algorithms help deepen the analysis of the massive volumes of data extracted from the native data collection service of the cloud infrastructure, which provides monitoring of a multitude of metrics from both physical and virtual hosts.

Thus, we use machine learning algorithms to process datasets collected from the service of native telemetry of the cloud infrastructure to perform the detection and identification. These datasets contain information from the victim virtual machine hosted in the cloud environment. After performing the detection and identification, mechanism of the cloud environment itself are used to mitigate attacks, as exemplified by autoscaling. To perform a proof-of-concept, we used an experimental environment, with the OpenStack cloud platform, with both DDoS and intrusion attacks. Telemetry data was used as input to machine learning algorithms to classify the presence of an attack. Results showed good accuracy and a good relationship between false positives and true positives to detect and identify attacks. Finally, the mitigation mechanism offered greater availability for clients during denial-of-service attacks.

**Keywords**: Cloud Computing. Network Security. Machine Learning. Telemetry.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# GLOSSÁRIO

**ADDoS** – *Application Layer DDoS attacks*

**API** – *Application Programming Interface*

**AUC** – *Area Under the Curve*

**DDoS** – *Distributed Denial-of-Service*

**DNS** – *Domain Name Server*

**DPI** – *Deep Packet Inspection*

**DoS** – *Denial-of-Service*

**FN** – *False Negative*

**FP** – *False Positive*

**HTTP** – *Hypertext Transfer Protocol*

**IDS** – *Intrusion Detection System*

**IP** – *Internet Protocol*

**IoT** – *Internet-of-Things*

**ML** – *Machine Learning*

**NFV** – *Network Function Virtualization*

**NTP** – *Network Time Protocol*

**OSI** – *Open System Interconnection*

**QoE** – *Quality-of-Experience*

**QoS** – *Quality-of-Service*

**RF** – *Random Forest*

**SDN** – *Software Defined Network*

**SFC** – *Service Function Chaining*

**SQL** – *Structured Query Language*

**SQLi** – *Structured Query Language injection*

**SSH** – *Secure Shell*

**SVM** – *Support Vector Machine*

**TCP** – *Transport Control Protocol*

**TN** – *True Negative*

**TP** – *True Positive*

**TTS** – *Time-to-Service*

**VM** – *Virtual Machine*

**VNF** – *Virtualized Network Function*

**kNN** – *k-Nearest Neighbor*

# SUMÁRIO

# Capítulo 1

## INTRODUÇÃO

De acordo com a empresa Verisign (VERISIGN, 2018), ataques de negação de serviço (DoS - *Denial-of-Service*) e de sua forma distribuída (DDoS - *Distributed Denial-of-Service*) são os que mais impactam nos negócios, principalmente serviços financeiros e das Tecnologias da Informação e Comunicação (TIC). A intenção desses ataques é fazer com que um serviço pare de responder a clientes que solicitam atendimento, ou seja, é uma tentativa de um usuário não legítimo de degradar ou negar recursos a um usuário legítimo (SHEA; LIU, 2012).

Segundo a empresa Akamai (AKAMAI, 2017), dispositivos de IoT (*Internet-of-Things*) também estão sendo utilizados para integrar *botnets*[1], com intuito de realizar ataques de negação de serviço do tipo DDoS, principalmente em serviços hospedados em ambiente de nuvem (BHARDWAJ; MIRANDA; GAVRILOVSKA, 2018; DOSHI; APTHORPE; FEAMSTER, 2018). Um exemplo dessas *botnets* é a Mirai[2] (ANTONAKAKIS et al., 2017). Em 2021, esses dispositivos de IoT integram uma nova botnet, a Simps[3], gerando danos ainda maiores às suas vítimas.

Além disso, com a migração de serviços e aplicações para infraestruturas de computação em nuvem, os ataques de DDoS se tornam problemas mais complexos. A arquitetura da infraestrutura da nuvem pode amplificar as vulnerabilidades e o impacto final em comparação com as infraestruturas descentralizadas (ARKKO, 2019). Essa observação é relevante, pois recursos físicos, de controle e de gerenciamento, são compartilhados intensamente na infraestrutura de computação em nuvem. Assim, o desempenho de vários serviços serão afetados indiretamente devido a um ataque de DDoS em um *tenant* específico. Dessa forma, há a necessidade de se

[1] *Botnet* é uma rede de computadores infectados e que respondem comandos de um usuário malicioso.

[2] Página web: <https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/>. Acessado em 15 de dezembro de 2017.

[3] Página web: <https://www.uptycs.com/blog/discovery-of-simps-botnet-leads-ties-to-keksec-group>. Acessado em 30 de setembro de 2021.

realizar a detecção e identificação de ataques DDoS para realizar algum tipo de mitigação e redução de danos.

Outra forma de ataque e que gera diversos problemas a serviços na Internet (e consequentemente para organizações, empresas e governos) são os ataques de intrusão (HUANG; SIEGEL; MADNICK, 2018). Esses ataques tem como objetivos obter informação privilegiada contida em servidores ou aplicações, gerar uma negação de serviço, ou utilizar o servidor invadido para realizar outros tipos de ataques contra outros servidores. Dentre os diversos tipos de ataques de intrusão, destacam-se três: Força Bruta no SSH, SQLi (Injeção SQL) e *Scan* de portas.

De acordo com o relatório da Akamai (AKAMAI, 2019) para o ano de 2019, 65,1% dos vetores de ataques contra *web* foram com SQLi. Segundo outro relatório, da F5 Labs and Communities (BODDY, 2018), a porta TCP do serviço SSH, a 22, é 2,7 vezes mais atacada que a porta 80 do HTTP, e 3 vezes mais atacada que a porta do serviço Telnet. Outro relatório, da Kaspersky[4], afirma que, no ano de 2020, mais de 60% dos vetores de ataques são de força bruta e de exploração de vulnerabilidades. Além dos problemas que esses ataques geram para o próprio servidor invadido, há também problema se esse servidor estiver instanciado em um ambiente de nuvem. A invasão pode abrir portas para que *hosts* que estão instanciados na mesma infraestrutura da nuvem, mas não disponíveis para a internet, possam ser alcançados pelos atacantes a partir do servidor invadido. Por isso, também há a necessidade da detecção e identificação desses ataques de intrusão para que ações de mitigação possam ser realizadas a fim de dirimir os efeitos ocasionados pelo ataque.

Nesse contexto, esta Tese investiga soluções que realizem a Detecção, Identificação e Mitigação desses ataques de DDoS e de intrusão. Neste âmbito, a Detecção será referida como a habilidade de indicar a presença anormal do sistema, a Identificação, a habilidade de identificar o tipo de ataque que está sendo realizado contra algum serviço na infraestrutura da nuvem, e a Mitigação refere-se à habilidade de, baseado na detecção e identificação, utilizar mecanismos para dirimir o ataque. Diante do exposto, esta Tese visa responder às respectivas perguntas: Está acontecendo algum ataque? Qual ataque está sendo realizado? E quais ações podem ser tomadas para diminuir ou cessar os efeitos deste ataque?

[4]Página web: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2021/09/13085018/Incident-Response-Analyst-Report-eng-2021.pdf>. Acessado em 30 de setembro de 2021

## 1.1 Problema de pesquisa

De acordo com Amaral *et al.* (AMARAL et al., 2016), existem três formas de realizar a detecção e identificação de ataques DDoS: (i) filtrando e verificando o cabeçalho dos pacotes da rede; (ii) filtrando e verificando os dados nos pacotes e suas requisições; (iii) aplicando algoritmos de Aprendizado de Máquina (*Machine Learning* - ML) nos dados e cabeçalhos dos pacotes. Para as técnicas (i) e (ii), existem diversos Sistemas de Detecção de Intrusão (*Intrusion Detection System* - IDS), como o Snort[5], que realiza a análise do tráfego em tempo real.

Zargar et al. (ZARGAR; JOSHI; TIPPER, 2013) afirma que ferramentas tradicionais, como os IDS, sofrem para distinguir ataques de DDoS na camada de aplicação, por exemplo o GET_-Flood, pois o ataque gera um tráfego com características similares às de clientes legítimos em alta escala. Além disso, para realizar a detecção de ataques na camada de aplicação, é necessário que o IDS realize uma inspeção mais detalhada de cada pacote (PROJECT, 2020). Essa inspeção detalhada gera um atraso nos serviços e degrada a Qualidade de Experiência (*Quality-of-Experience* - QoE) dos usuários (GOGUNSKA et al., 2018). Aliás, para a implementação de um IDS em um *data center*, são requeridos equipamentos de alto custo, especializados e com alta taxa de vazão, dificultando ainda mais a distinção dos ataques (PROJECT, 2020).

Dessa forma, é necessário um sistema de detecção, identificação e mitigação de ataques, para complementar (ou substituir) sistemas convencionais do tipo IDS. Esses utilizam informações dos pacotes, que necessitam ser coletadas em diferentes níveis hierárquicos (e em diferentes equipamentos) na rede do *data center*. Esse tráfego deve ser redirecionado, ou copiado para um servidor de monitoramento, a fim de que seja feita a análise dos pacotes. Ao realizar o redirecionamento do tráfego, além de inserir a latência do próprio redirecionamento, é inserido também a latência da inspeção de cada pacote pelo IDS, como dito anteriormente por Gogunska *et al.* (GOGUNSKA et al., 2018). Essa latência na inspeção pode ser agravada caso seja necessária a verificação de camadas superiores do pacote (PROJECT, 2020). No caso do tráfego ser copiado, não há inserção de latência, no entanto, ainda possui todo o tempo necessário da inspeção e processamento, para, por fim, ser realizada a detecção/identificação.

Além disso, a realização da inspeção dos pacotes de rede pode ser comprometida e consequentemente todo o processo de detecção/identificação/mitigação, caso os usuários da rede utilizem mecanismos de criptografia. Essas ferramentas criptográficas, principalmente as utilizadas na camada de rede do modelo OSI (*Open System Interconnection*), impedem que os IDS realizem a inspeção dos pacotes e da carga útil nas camadas criptografadas. Um exemplo disso

[5]Página web: <https://www.snort.org>. Acessado em 18 de agosto de 2019.

é o ataque GET_Flood realizado contra um servidor *web* que utiliza o HTTPS. Um IDS ou um *software* de inspeção profunda do pacote (*Deep Packet Inspection* - DPI) não conseguem fazer a análise necessária do cabeçalho e da carga útil do pacote HTTPS, justamente pelo o uso da criptografia, impedindo a detecção e a mitigação desse ataque.

## 1.2 Questões de Pesquisa

Diante do que foi exposto, esta Tese tem como Questões de Pesquisa:

- **Q1:** É possível realizar a detecção e identificação de ataques, sejam eles de negação de serviço ou de intrusão, em ambiente de nuvem utilizando ferramentas da própria infraestrutura da nuvem e que não degradem a qualidade de serviço e de experiência do usuário?
- **Q2:** Após realizar a detecção e identificação dos ataques de negação de serviço, é efetivo realizar a mitigação desses ataques utilizando artifícios habilitadores do ambiente de nuvem?

## 1.3 Hipóteses

Partindo das Questões de Pesquisa, foram então elaboradas as seguintes hipóteses:

- **Hipótese H1: É possível realizar a detecção e identificação de ataques de negação de serviço e de intrusão, utilizando informação de telemetria da infraestrutura da nuvem em conjunto com algoritmos de aprendizado de máquina.**
  - **Justificativa H1:** A computação em nuvem normalmente tem sistemas de telemetria, coletando dados sobre a utilização dos recursos computacionais (servidores físicos ou máquinas virtuais) e dos serviços da própria infraestrutura da nuvem. A telemetria nativa oferece ao administrador da infraestrutura da nuvem um rico conjunto de dados, com diversas informações do ambiente de nuvem e sem adicionar penalidade por monitorar esses recursos (SANTOS et al., 2018). Dessa forma, a hipótese levantada é que cada ataque realizado contra um serviço hospedado na infraestrutura da nuvem, ocasiona interferência em um subconjunto distinto de recursos computacionais (NICHOLS et al., 2016). A análise desses subconjuntos, observados pela telemetria, com auxílio de algoritmos de aprendizado de máquina, vão realizar a detecção e identificação de ataques de DDoS. Em relação aos ataques

de intrusão, apesar de gerarem uma quantidade menor de tráfego e, consequentemente, ser mais difícil detectar/identificar pela análise de rede (HUANG; SIEGEL; MADNICK, 2018), esses ataques também geram interferência em um subconjunto de recursos computacionais pertencentes à vítima e que podem ser monitorados e coletados pela telemetria nativa da infraestrutura da nuvem. Dessa forma, a utilização dos algoritmos de aprendizado de máquina, em conjunto com os dados coletados da telemetria, podem realizar a detecção e identificação.

- **Hipótese H2: É possível combinar a telemetria e os diversos mecanismos que a infraestrutura da nuvem oferece, por exemplo, o dimensionamento automático, em complementariedade com defesas existentes para realizar a mitigação dos ataques de negação de serviço.**
  - **Justificativa H2:** O ambiente de nuvem oferece mecanismos para que sejam tomadas diversas ações para mitigar ataques de DDoS, mas que em muitos casos não são utilizados para tal fim. A possibilidade de escalar a vítima automaticamente, ativar algum encadeamento de funções de rede, ou inserir dinamicamente alguma regra de bloqueio de tráfego são possibilidades que a infraestrutura da nuvem dispõe como tais ações (FAYAZ et al., 2015; SAHAY et al., 2017).

## 1.4 Objetivos

Diante das Questões de Pesquisas e das Hipóteses levantadas, esta Tese tem como objetivo realizar a detecção, identificação e mitigação de ataques de DDoS e de intrusão, utilizando tecnologias abertas e inerentes à computação em nuvem, combinado com algoritmos de aprendizado de máquina. Para atingir tal objetivo, este trabalho elenca os seguintes objetivos específicos:

- Investigar, propor e implementar mecanismos de pré-processamento e seleção de características dos dados disponíveis no sistema de telemetria existente na infraestrutura da nuvem;
- Investigar e implementar algoritmos de aprendizado de máquinas para realizar a detecção e identificação dos ataques de negação de serviço mediante a telemetria da infraestrutura da nuvem;
- Investigar e implementar técnicas de aprendizado de máquina para detectar ataques de intrusão com auxílio da telemetria do ambiente de nuvem;

- Investigar, propor e implementar sistema de mitigação dos ataques de DDoS e de intrusão através de funcionalidades da própria infraestrutura da nuvem;
- Avaliar a proposta por meio da criação de cenários e realização de experimentos como prova de conceito.

## 1.5 Proposta da Tese e Contribuições

Em resumo, a proposta desta Tese é que diferentes ataques de negação de serviço e de intrusão, realizados contra serviços hospedados em uma infraestrutura de computação em nuvem, geram diferentes níveis de utilização em grupos de recursos computacionais distintos. Dessa forma, se as métricas desses recursos computacionais, que está disponibilizado pelo serviço de telemetria da infraestrutura da nuvem, forem coletadas e analisadas, então é possível detectar e identificar esses ataques. Dessa forma, algoritmos de aprendizado de máquina podem auxiliar na tarefa de identificar esses grupos de recursos computacionais distintos, visto que há uma infinidade de recursos monitorados pela telemetria nativa da nuvem.

Somado a isso, outros recursos nativos da nuvem podem ser utilizados para mitigar os efeitos dos ataques. É o caso do dimensionamento automático, sendo um recurso que a nuvem dispõe para oferecer mais recursos quando há uma demanda variável ao serviço hospedado. No entanto, a proposta desta Tese é utilizar esse recurso para oferecer mais disponibilidade a clientes de serviços hospedados na infraestrutura da nuvem.

O esquema da proposta desta Tese pode ser representado pela Figura 1.1, sendo uma das contribuições deste trabalho retratada na parte superior do esquema: `Modelo ML (Detecção)`, `Modelo ML (Identificação)` e `Mitigação`. Nesse esquema, a `Nuvem (infraestrutura)` hospeda serviços, e a `Telemetria` monitora e coleta periodicamente informações sobre esses serviços. Essas informações, que são chamadas de `Entrada de Amostras`, são as entradas do sistema proposto nesta Tese. A `Amostra` passa pelo `Modelo ML (Detecção)`, que é responsável por verificar se está acontecendo ou não um ataque. Se o `Modelo ML (Detecção)` não detectar que está acontecendo um ataque, essa `Amostra` sai do sistema e não é mais verificada. Por outro lado, se for detectado, essa `Amostra` será encaminhada para a etapa seguinte, `Modelo ML (Identificação)`, que é responsável por identificar o tipo do ataque. Novamente, caso essa etapa identifique um tráfego normal, a `Amostra` sai do sistema, pois não é necessária a realização de uma ação de mitigação, caso contrário, após o `Modelo ML (Identificação)` identificar o tipo de ataque que está ocorrendo, logo será acionado a etapa de `Mitigação`, para que seja realizada uma ação contra o ataque. Assim, a `Mitigação` realiza uma ação diretamente na

`Nuvem (infraestrutura)`, que por sua vez afeta as métricas que são monitoradas. Portanto, no próximo ciclo (ou nos próximos ciclos), os efeitos da mitigação serão reavaliados após a coleta de novas amostras. Observa-se que a transição da etapa de `Modelo ML (Identificação)` para a `Mitigação`, representada pela linha tracejada, no contexto da presente Tese, não é realizada de forma orgânica, pois, como Prova de Conceito, a etapa da `Mitigação` é acionada sem a integração com a detecção/identificação realizada previamente. Ou seja, para o que está apresentado efetivamente nesta Tese, representado na Figura 1.1 pela linha pontilhada, a `Mitigação` é realizada a partir dos dados da `Telemetria`, por exemplo, a quantidade de uso do processador.

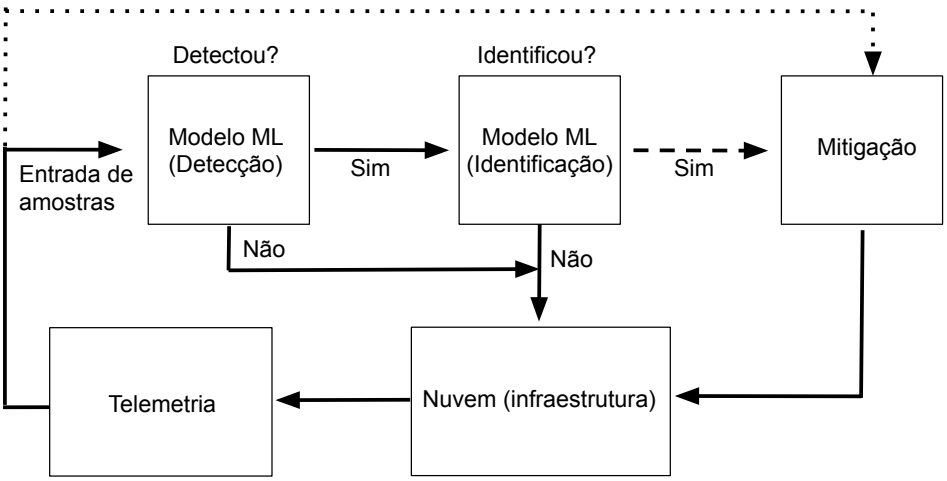



**Figura 1.1: Esquema de utilização da infraestrutura da nuvem para realizar a Detecção, Identificação e Mitigação de ataques**

Sendo assim, pode-se destacar como contribuição desta Tese a realização de detecção e identificação dos ataques, de negação de serviço e de intrusão, sem utilizar informações dos cabeçalhos e da carga útil dos pacotes de rede, e utilizando apenas o serviço nativo da telemetria da infraestrutura da nuvem combinado com algoritmos de aprendizado de máquina. Dessa forma, é dada uma nova finalidade para a telemetria nativa do ambiente de nuvem. Por não utilizar informações dos pacotes de rede, a proposta não gera latência e não é invasiva, por não inspecionar os pacotes de rede. Como não depende da assinatura dos ataques para realizar a detecção, outra contribuição é a possibilidade de detectar ataques desconhecidos, como os ataques *Zero-Day*. Destaca-se ainda que a utilização da telemetria da infraestrutura da nuvem é uma forma não invasiva de detecção e identificação, mantendo assim a privacidade dos dados trafegados. Em conjunto a isso, por não realizar a inspeção dos pacotes, contribui-se na detecção e identificação dos ataques que utilizam a criptografia para esconder a sua assinatura, como no caso específico do ataque GET_Flood que utiliza o HTTPS.

Além disso, este trabalho auxilia na utilização dos mecanismos da própria infraestrutura da

nuvem para realizar a mitigação dos ataques. Comumente, os serviços estão sendo migrados de infraestruturas físicas para a computação em nuvem, mas mantendo ainda as perspectivas da segurança dos serviços no paradigma anterior. Desse modo, diversos mecanismos e aspectos únicos que o ambiente de nuvem oferece não são utilizados para a mitigação de ataques.

Por fim, também contribui com a disponibilização de *datasets* que contêm as medições de telemetria da infraestrutura da nuvem durante os ataques. Esta Tese é relevante sobretudo porque não se tem notícia da existência de *datasets* públicos disponíveis com as informações da telemetria do ambiente de nuvem enquanto estão acontecendo ataques.

## 1.6 Metodologia

Para alcançar as hipóteses elencadas, este trabalho realizou uma avaliação experimental com a elaboração de um protótipo em nuvem privada baseado em tecnologias abertas, porém com funcionalidades equivalentes às nuvens públicas. Ademais, utilizou a telemetria em conjunto com algoritmos de aprendizado de máquina para realizar a detecção e identificação dos ataques de negação de serviço e de intrusão. Esses algoritmos de classificação, como os baseados em árvores de decisão ou em distância, podem detectar as variações provocadas pelos ataques nas métricas de telemetria da infraestrutura da nuvem.

Este trabalho emprega tecnologias de código aberto e *software* livre, especificamente, a plataforma de nuvem OpenStack, que oferece um módulo específico de telemetria, o Ceilometer. Além disso, são utilizadas ferramentas de código aberto para a geração do tráfego de ataque e de clientes legítimos, já consagradas na área de redes e bastante aplicadas em testes de segurança. Assim, são escolhidos ataques conhecidos e frequentemente realizados na Internet, visando testar a proposta com ataques atuais e que ainda causam problemas no ambiente de computação em nuvem. Nessa perspectiva, foram escolhidos os ataques SYN_Flood, GET_Flood e o Slowloris para os ataques de DDoS, e o ataque SQL_Injection como exemplo de ataque de intrusão. Os experimentos para verificar a proposta são necessários, pois não há *dataset* público disponível contendo informações da telemetria nativa da nuvem durante ataques.

Em relação aos algoritmos de aprendizado de máquina, são utilizados algoritmos de aprendizado supervisionado para classificação dos ataques. É verificado, com isso, a eficácia dos algoritmos por meio de indicadores de desempenho, como: Acurácia, Precisão, Revocação e *F1-Score*. Também são apresentadas matrizes de confusão e curva ROC (*Receiver Operating Characteristic*) para verificar o comportamento dos algoritmos na tarefa de classificação. Vale salientar que o modelo *Wrapper* (AGGARWAL, 2015) foi utilizado para realizar uma seleção

de características dos *datasets* gerados.

Por fim, serão utilizados ferramentas e dispositivos da própria infraestrutura da nuvem para realizar a mitigação dos ataques, como o dimensionamento automático.

## 1.7 Estrutura do Texto

Esta Tese está dividida em sete capítulos, o Capítulo 1 - a Introdução - com Questões de Pesquisa, Hipóteses, Objetivos Gerais e Específicos, Contribuições e Metodologia.

No Capítulo 2, são apresentados os conceitos sobre a computação em nuvem e sobre a especificidade da plataforma de nuvem OpenStack, que é uma ferramenta de código aberto e bastante utilizada pela comunidade. O Ceilometer é exposto no mesmo capítulo, já que é o módulo do OpenStack responsável pela telemetria da infraestrutura da nuvem. Em sequência, é discutido sobre a funcionalidade do dimensionamento automático (*autoscaling*). Além disso, são elucidados e debatidos os ataques de negação de serviço e de intrusão, e finalizando com a discussão sobre aprendizado de máquina.

No Capítulo seguinte, o 3, são expostos os trabalhos relacionados aos temas desta Tese. Posteriormente, no Capítulo 4, é apresentada a utilização do sistema de telemetria nativo da infraestrutura da nuvem para alimentar algoritmos de aprendizado de máquina e, assim, detectar e identificar os ataques.

Já no Capítulo 5 são demonstrados os cenários escolhidos, com a geração dos *datasets* e a combinação entre aprendizado de máquina e telemetria, para detectar e identificar ataques. No penúltimo, o Capítulo 6, é apresentado o método para mitigação dos ataques. Por fim, no Capítulo 7 são apresentados as conclusões e as propostas de trabalhos futuros.

# Capítulo 2

# REFERENCIAL TEÓRICO, MATERIAIS E MÉTODOS

Este capítulo apresenta um panorama sobre os conceitos que foram utilizados neste trabalho. Inicialmente, na Seção 2.1, decorrerá sobre o conceito de computação em nuvem, já na seção seguinte, será apresentado o *software* de ambiente de nuvem OpenStack, bem como o módulo Ceilometer, responsável pela telemetria da infraestrutura da nuvem OpenStack. Posteriormente, na Seção 2.3, haverá a discussão sobre as formas em que a computação em nuvem realiza o dimensionamento automático das máquinas virtuais hospedadas.

Na sequência, especificamente na Seção 2.4, é discutido sobre Ataques de Negação de Serviço, bem como os ataques utilizados no contexto desta Tese. Posteriormente, aborda os Ataques de Intrusão e os que foram utilizados neste trabalho. Após, na Seção 2.6, são apresentados os principais conceitos e algoritmos de Aprendizado de Máquina usados para a construção deste estudo. Por último, são feitas as considerações finais do capítulo.

## 2.1 Computação em Nuvem

Com o advento da tecnologia de virtualização de recursos computacionais, houve também o advento de um novo paradigma da computação: a computação em nuvem. De forma conceitual, mas direta, Tanenbaum e Bos (TANENBAUM; BOS, 2016) apresentam que a ideia fundamental da computação em nuvem é "terceirizar as suas necessidades de computação ou armazenamento para um centro de processamento de dados bem administrado e gerenciado por uma empresa especializada e gerida por experts na área". Nesse sentido, há uma troca em relação ao custo, na qual a empresa que oferece o serviço da nuvem arca com as despesas de manter os recursos disponíveis, preocupando-se com as máquinas físicas, a energia, o resfriamento e a

manutenção, mas, para oferecer esse serviço, cobra-se das empresas que pretendem terceirizar essa necessidade da computação.

De acordo com o NIST (*National Institute of Standards and Technology*), que é o instituto de padronização dos Estados Unidos, há uma lista de cinco características essenciais para a computação em nuvem: serviço automático de acordo com a demanda, ou seja, o usuário deve ser capaz de utilizar os recursos automaticamente, sem a interação humana; acesso amplo pela rede, em outras palavras, que o usuário possa acessar a infraestrutura virtual via a própria Internet; *pooling* de recursos, no qual o provedor do serviço disponibiliza recursos a múltiplos usuários, com a capacidade de alocar e realocar os recursos dinamicamente, e que esses recursos podem estar distribuídos geograficamente; elasticidade rápida, isto é, que a própria nuvem possa permitir ao usuário adquirir e liberar recursos elasticamente, inclusive automaticamente, de acordo com as demandas imediatas do usuário; e, serviço mensurado, no qual o provedor da nuvem monitora os recursos usados, adequando com o tipo de serviço contratado pelo usuário (MELL; GRANCE, 2011).

A computação em nuvem possui uma gama de diversidade, mas destacam-se os seguintes tipos: Infraestrutura como um Serviço; Plataforma como um Serviço; e *Software* como um Serviço. Quanto ao primeiro (IaaS - *Infrastructure-as-a-Service*), este é o modelo no qual o provedor da nuvem oferece ao cliente um acesso direto a uma máquina virtual, que pode ser usada da maneira que ele achar melhor. Nesse tipo, a infraestrutura da nuvem pode executar sistemas operacionais diferentes, possivelmente no mesmo *hardware*. O modelo Plataforma como um Serviço (PaaS - *Platform-as-a-Service*) proporciona ao usuário um ambiente pré-configurado, que já inclui um Sistema Operacional específico, banco de dados, servidor *web*, etc. E, por fim, o tipo de computação em nuvem, *Software* como um Serviço (SaaS - *Software-as-a-Service*) que oferece ao cliente acesso a um *software* específico, como o Google Docs, ou o Microsoft Office 365. Na Figura 2.1 são apresentados os três tipos de computação em nuvem, destacando, em cada modelo, o que deve ser gerenciado pelo provedor de serviço e o que é gerenciado pelo cliente.

Esses serviços de computação em nuvem podem ser de dois tipos: as nuvens públicas, que estão disponíveis para qualquer pessoa/empresa que queira usar o serviço e, consequentemente, pagar pelo uso; e as nuvens privadas, que são restritas a uma organização. Como exemplo de nuvens públicas, cita-se a Amazon Web Services (AWS), o Google Cloud e o Microsoft Azure. Já OpenStack e "CloudStack" são exemplos de nuvens privadas. Apesar de ter essa classificação, nada impede que as empresas com um ambiente OpenStack/CloudStack ofereçam serviços para outras empresas, classificando-os como nuvem pública. No contexto desta Tese,

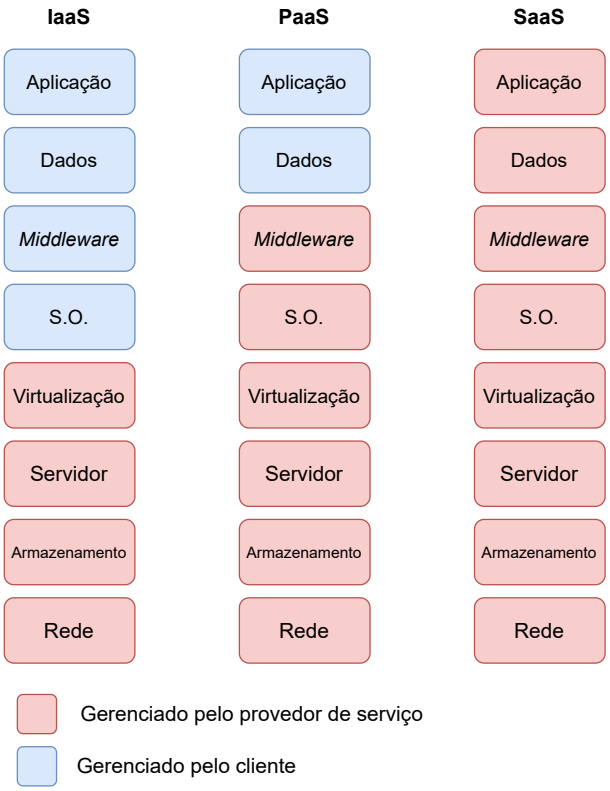



**Figura 2.1: Diferença entre modelos de computação em nuvem. Adaptado de (REDHAT, 2021)**

o ambiente de nuvem utilizado é o OpenStack, cujos motivos da escolha são apresentados na seção seguinte.

## 2.2 OpenStack

OpenStack[1] é uma plataforma de computação em nuvem pública e privada, de código aberto, que permite controlar grandes conjuntos de recursos de computação, armazenamento e redes pertencentes a um *data center*, tudo gerenciado através de uma interface dedicada que permite aos administradores controlar e provisionar recursos para múltiplos usuários. Entende-se por computação em nuvem como um modelo de computação onde as capacidades relacionadas a tecnologias da informação são escaláveis e elásticas, sendo que os serviços são providos para os usuários finais através da Internet (YI; LI; LI, 2015).

A plataforma OpenStack surgiu inicialmente de uma parceria entre a Rackspace e a NASA, mas atualmente mais de duzentas companhias de *hardware*, *software* e serviços apoiam o seu desenvolvimento, que é mantido por uma comunidade global de desenvolvedores. Essa comunidade é chamada de Fundação OpenStack (*OpenStack Foundation*) (OPENSTACK, 2019). O

[1]Página web: <https://www.openstack.org/>. Acessado em 15 de dezembro de 2017.

OpenStack opera sob o modelo de IaaS.

Um dos principais objetivos da plataforma é construir um serviço de computação em nuvem que pode ser instalado em *hardware* padrão ou "customizado". Sobre essa camada de *hardware* atuam serviços compartilhados do OpenStack e os componentes responsáveis por controlar os grupos de computadores, armazenamento e recursos de rede. No topo da estrutura do OpenStack estão as camadas de aplicações e as de administração de acesso, que são representadas por uma interface *web*.

Um dos motivos para que o OpenStack seja uma plataforma difundida e utilizada é que foi construída de forma modularizada, ou seja, cada função específica dentro do projeto é dividida em um módulo separado. Dessa forma, o desenvolvimento dos componentes pode variar, podendo ter graus de maturidade diferentes para cada módulo.

O OpenStack oferece inúmeros cenários para sua implantação, que vão desde um cenário "tudo em um" (*All-in-one*), no qual todos os componentes dessa plataforma estão dispostos em um único servidor, até cenários de múltiplos nós (*Multi-node*), nos quais seus componentes são separados em vários nós na rede.

Dentre os módulos do OpenStack, alguns são considerados componentes-chave para a implementação de um ambiente de nuvem, por serem responsáveis pela orquestração de recursos, processamento, armazenamento e rede. Os módulos-chave são: Nova (provisionamento de instâncias de computação), Keystone (gerência de acesso à infraestrutura da nuvem), Glance (gerência de imagens), Neutron (conectividade entre as redes virtuais), Horizon (painel de controle da infraestrutura da nuvem via página *web*) e Ceilometer (coleta de dados para monitoramento). A Figura 2.2 apresenta os serviços básicos: telemetria, comunicação e interação entre os diversos módulos do OpenStack (OPENSTACK, 2020).

### 2.2.1 Ceilometer

Como dito anteriormente, o Openstack é construído por diversos módulos, que realizam uma função específica dentro do ambiente de nuvem. O Ceilometer[2] é um módulo de serviço de coleta de dados, que normaliza e transforma os dados oriundos de vários serviços do OpenStack. O Ceilometer é um módulo componente do projeto de telemetria do OpenStack[3], e seus dados podem ser usados para fornecer faturamento ao cliente, rastreamento de recursos e capacidade

[2]Página web: <https://docs.openstack.org/ceilometer/latest/index.html>. Acessado em 15 de dezembro de 2017.

[3]A lista dos recursos monitorados pelo Ceilometer está disponível em: https://docs.openstack.org/ceilometer/latest/admin/telemetry-measurements.html

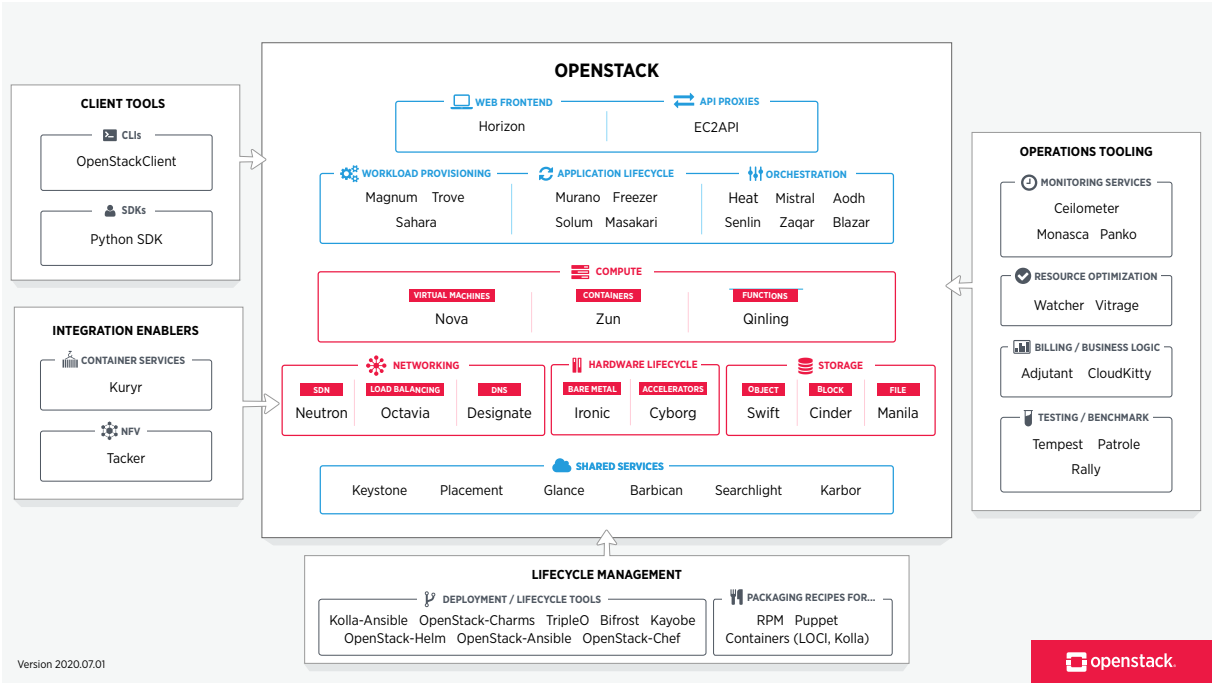



**Figura 2.2: Mapa de serviços do OpenStack (OPENSTACK, 2020)**

de gerar e monitorar alarme nos componentes centrais do OpenStack.

De acordo com a sua própria documentação (DOCUMENTATION, 2019), o sistema de coleta de dados de telemetria fornece as seguintes funções: (i) pesquisa eficiente dos dados de medição relacionados aos serviços do OpenStack; (ii) coleta dados de eventos e de medição através do monitoramento de notificações enviadas pelos serviços; (iii) realiza a publicação dos dados coletados em vários serviços, por exemplo em bancos de dados e em *message queues*.

O serviço de telemetria se divide em três componentes: *Compute agent*, *Central agent* e *Notification agent*. O primeiro é executado em cada nó de computação e realiza a captura das estatísticas de utilização dos recursos, tanto dos nós quanto das máquinas virtuais em execução. O segundo executa no nó central de gerenciamento, e realiza a pesquisa de estatísticas referentes a utilização dos recursos. Vários *Central agents* podem ser instanciados para escalar o serviço. Por fim, o terceiro funciona também no nó central de gerenciamento e consome mensagens das filas para construir eventos ou dados de medição. Os dados são publicados para metas definidas. Por padrão, os dados são publicados no Gnocchi[4], que é um banco de dados temporal, projetado para armazenar métricas em grande escala, além disso, possui propriedades de alta disponibilidade, garantindo que a telemetria esteja sempre ativa.

Em relação à Figura 2.3, vê-se um esquema de como o Ceilometer realiza a coleta dos dados oriundos de várias fontes. O Ceilometer criou duas formas de coletar os dados: via *Notifica-*

[4] A sua documentação pode ser encontrada em <https://gnocchi.osci.io/>

*tion agent* e via *Polling agent*. O primeiro resgata as mensagens geradas pelo *Notification bus* (que podem ser geradas pelos próprios elementos monitorados) e as transforma em amostras do Ceilometer, ou em eventos. A coleta via *Polling agent* pesquisará alguma API (*Application Programming Interface*) ou outra ferramenta para coletar informações dos serviços monitorados em um intervalo regular. Vale destacar que este método de coleta pode impor um grande consumo dos recursos computacionais, por isso tal método necessita ser otimizado.

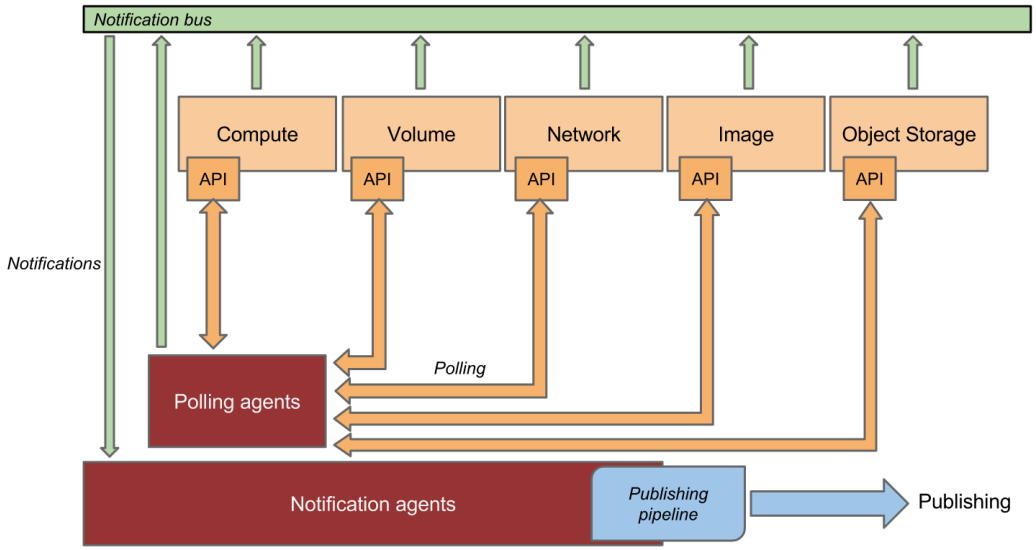



**Figura 2.3: Esquema da coleta dos dados pelo Ceilometer (DOCUMENTATION, 2019)**

## 2.3 Dimensionamento Automático

Uma das características marcantes na computação em nuvem é a escalabilidade, ou seja, a capacidade da infraestrutura da nuvem de se ajustar e ajustar os serviços hospedados nela, a mudanças na demanda. Esse dimensionamento consiste em aumentar ou reduzir a oferta de recursos computacionais dinamicamente para uma máquina virtual (*Virtual Machine* - VM) hospedada na infraestrutura da nuvem (QU; CALHEIROS; BUYYA, 2018). Essa característica oferece ao ambiente de nuvem uma utilização mais eficiente dos recursos computacionais, podendo disponibilizar maior quantidade de recursos para uma VM apenas quando esta realmente necessita. Nos momentos que essa VM não precisar, esse recurso ora disponibilizado, poderá ser destinado a outras máquinas virtuais.

Manter uma VM com pouco recurso pode ser inviável, pois pode ser insuficiente para atender demandas variáveis das aplicações em execução na máquina virtual. Por outro lado, oferecer o máximo de recursos para esta VM também não é adequado, pois em grande parte do tempo os recursos computacionais estarão ociosos. Sendo assim, a infraestrutura da nuvem deve ser capaz

de oferecer dinamicamente mais recursos computacionais quando a VM necessita. Da mesma forma, o ambiente de nuvem deve retirar os recursos computacionais ociosos dessa VM, gerando, assim, uma economia. Esse processo é chamado de Dimensionamento Automático, ou *autoscaling*.

O *autoscaling* pode ser classificado de duas formas: horizontal e vertical. Na horizontal, a infraestrutura da nuvem instancia ou remove uma VM, ou seja, quando é necessário mais poder computacional, uma outra VM é instanciada, e quando o recurso computacional está ocioso, a VM extra é removida (ARABNEJAD et al., 2016).

O segundo tipo de *autoscaling* é o vertical, em que a infraestrutura da nuvem disponibiliza mais recursos para a máquina virtual, ou seja, a vertical refere-se ao nível de recursos computacionais. Dessa maneira, quando a VM necessita mais poder computacional, a infraestrutura da nuvem aumenta a quantidade dos recursos (CPU e Memória, por exemplo) da própria VM, e quando a VM não mais precisar, a infraestrutura da nuvem desaloca esses recursos (ARABNEJAD et al., 2016). O *autoscaling* vertical é realizado no mesmo hospedeiro, dessa forma, limita-se à quantidade de VMs no mesmo hospedeiro (QU; CALHEIROS; BUYYA, 2018).

O OpenStack oferece a funcionalidade de dimensionamento automático, com o tipo horizontal. Além disso, o OpenStack oferece o dimensionamento vertical, mas na modalidade "a frio", em que a VM é desligada, aumenta e diminui os recursos computacionais, então a VM é religada. Apesar disso, existem soluções que oferecem o *autoscaling* vertical no OpenStack sem realizar o desligamento da máquina virtual (CARDOSO, 2019).

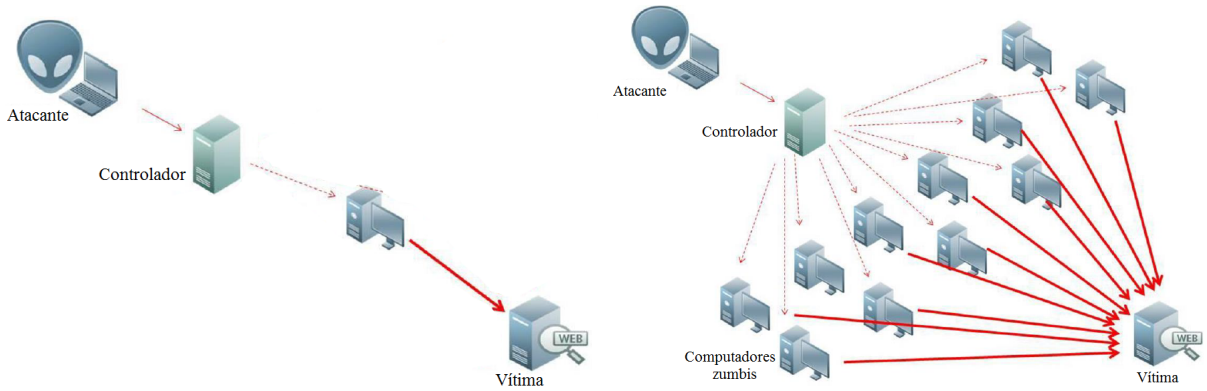
## 2.4 Ataques de Negação de Serviço

Os ataques de negação de serviço (DoS - *Denial-of-Service*) visam gerar a indisponibilidade de um serviço. Shea e Liu (SHEA; LIU, 2012) definem DoS como tentativas de um usuário malicioso de degradar ou negar recursos de um serviço ou servidor a um usuário legítimo. Essa negação, em fim último, ocorre quando uma aplicação não consegue servir a um usuário legítimo. Para isso, o usuário malicioso pode se utilizar de várias formas: gerar um consumo excessivo de recursos computacionais (por exemplo de CPU e de memória); consumir toda banda de rede disponível; preencher todos os espaços de atendimento em uma aplicação específica, etc.

Conceitualmente, o ataque de DoS pode acontecer de duas formas: quando apenas um atacante está provocando a indisponibilidade ou quando o ataque é realizado de forma distribuída, originada de vários lugares, sendo assim, um ataque de DDoS (*Distributed Denial-of-Service*).

A Figura 2.4 a seguir apresenta a realização do ataque DoS, na Figura 2.4(a), e o DDoS, representada na Figura 2.4(b). Nas duas representações há a figura do atacante, que é o usuário malicioso que tem o intuito de gerar a indisponibilidade. Possui também a presença do controlador, que é onde fica o serviço de comunicação e de onde serão geradas as instruções do ataque. Na Figura 2.4(a), este controlador se comunica apenas com uma máquina, sendo esta a responsável por gerar o ataque propriamente dito, enquanto que, na Figura 2.4(b), o controlador se comunica com os computadores zumbis, que são máquinas infectadas em outro momento e que obedecem aos comandos do atacante. Por fim, nas figuras apresentadas, têm-se a vítima, representada por um servidor *web*.

Atualmente, o DDoS é a forma mais usada, pois atinge o objetivo de gerar indisponibilidade mais rápido e sem a necessidade de despender grandes quantidades de recursos computacionais, porque, são utilizadas máquinas de terceiros para o ataque, máquinas essas, normalmente, infectadas por *malware*. Em comparação aos dois ataques, Almeida (ALMEIDA, 2013) afirma que o poder do ataque de DDoS é muito superior ao do DoS, por dois motivos: sistemas de detecção tem maior dificuldade de encontrar a origem do ataque e a quantidade de pacotes que o ataque de DDoS gera. Diante disso, no contexto desta Tese, os ataques de negação de serviço serão referenciados na sua forma distribuída, o DDoS.




(a) DoS (b) DDoS

**Figura 2.4: Esquema de ataques de negação de serviço de forma individualizada e distribuída. Adaptado (HELPSEC, 2016)**

### 2.4.1 Ataques de DDoS utilizados no contexto da Tese

Há diversos tipos de ataques de DDoS, variando a camada de ação ou o protocolo utilizado. De acordo com Bhosale *et al.* (BHOSALE; NENOVA; ILIEV, 2017), pode-se classificar os ataques de DDoS em três tipos:

- **Ataques de Inundação**: a característica desse tipo de ataque é a geração de uma grande

quantidade de tráfego para o alvo, gerando uma inundação de pacotes. O objetivo é esgotar os recursos computacionais do alvo, impossibilitando-os de responder a requisições legítimas. O SYN_Flood, o UDP_Flood e o PING_Flood são exemplos desse tipo de ataque.

- **Ataques de Amplificação**: são ataques em que o usuário malicioso cria requisições pequenas a servidores que se colocarão como amplificadores. Essas requisições têm o endereço IP de origem alterado para o IP da vítima. Assim, os servidores receberão as requisições e responderão para a vítima pacotes com grande quantidade de *bytes*, gerando um grande tráfego e ocasionando a indisponibilidade. O ataque tem esse nome proque há um fator de amplificação nos pacotes, em que requisições pequenas geram respostas com grande quantidade de *bytes*. Esse ataque pode ser utilizado em servidores de DNS e de NTP, por exemplo.

- **Ataques de Aplicação**: esses ataques visam explorar vulnerabilidades em protocolos da camada de aplicação do modelo OSI (*Open System Interconnection*) (TANENBAUM, 2003) e também nas aplicações. Tem a característica de necessitar de poucos recursos para gerar a indisponibilidade. Pode-se dividir os ataques de Aplicação em *low-* e *high-rate*. O GET_Flood, Slowloris e HTTP POST são exemplos de ataques de aplicação. Sendo o primeiro do tipo *high-rate* e os outros dois do tipo *low-rate*.

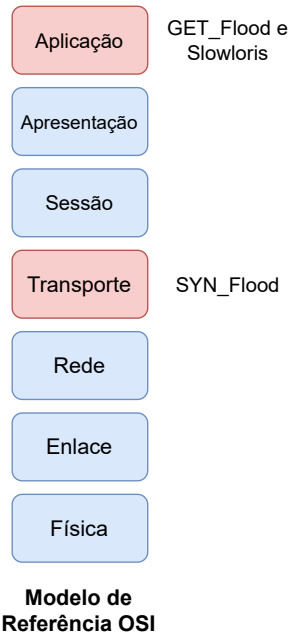



**Figura 2.5: Modelo de referência OSI, destacado as camadas afetadas pelos ataques**

Na conjuntura dessa Tese, foram escolhidos três tipos de ataques de DDoS, um ataque na camada de transporte e dois na camada de aplicação do modelo OSI. Na Figura 2.5 é apresen-

tado as camadas do modelo OSI, destacando as camadas atingidas pelos ataques escolhidos. O ataque na camada de transporte, sendo do tipo de inundação, é o SYN_Flood, e os outros dois do tipo de ataque de Aplicação são o GET_Flood[5] e o Slowloris. De acordo com a Kaspersky[6], o SYN_Flood é o ataque de DDoS mais realizado durante a primeira metade do ano de 2021. Ele representa a classe de ataques de inundação e tem como característica ser volumétrico, visto que é gerado um grande número de pacotes do protocolo TCP com a *flag* SYN ativada, com o objetivo de produzir uma exaustão dos recursos computacionais da vítima. Esse ataque está representado na Figura 2.6, em que um atacante envia diversos pacotes com a *flag* SYN ativada para o servidor *web*. Esse servidor recebe a solicitação de abertura de conexão, aloca recursos e responde com a *flag* SYN-ACK ativada, reconhecendo a abertura e prosseguindo nas etapas da conexão TCP. Como são várias solicitações de abertura de conexão, o servidor aloca recursos para todas, até que estes se esgotem, gerando, com isso, a indisponibilidade. Nos experimentos realizados nesta Tese, foi utilizada a ferramenta Hping3[7] para gerar o ataque SYN_Flood.

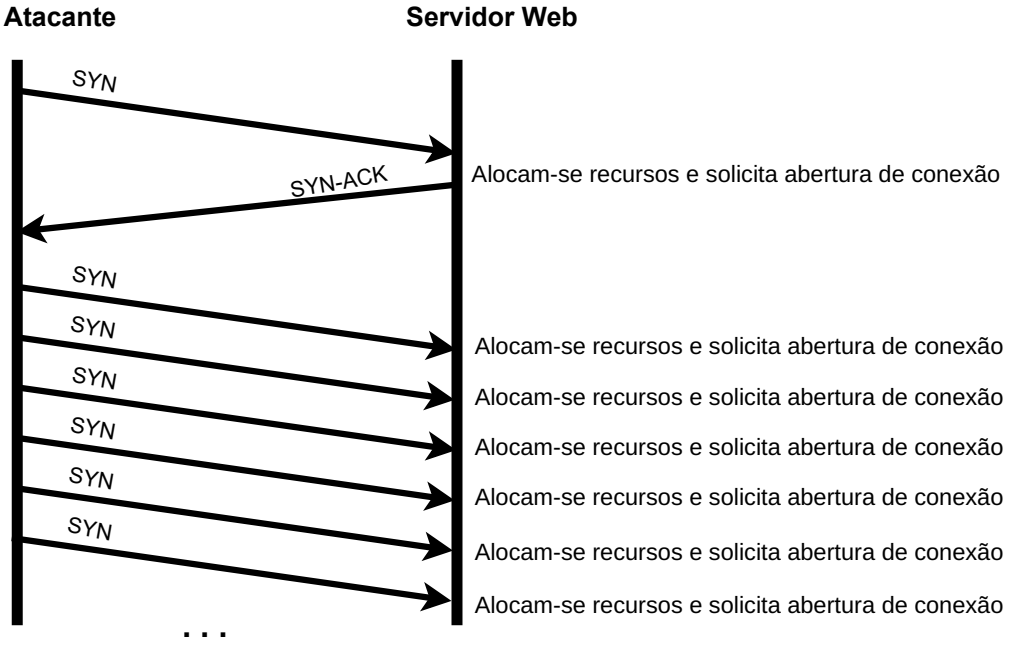



**Figura 2.6: Diagrama do ataque SYN_Flood**

O ataque GET_Flood, apesar de ser um ataque *flood*, gera menos tráfego na rede da vítima comparado aos ataques de camadas inferiores (AMJAD et al., 2019). Dentre os ataques de DDoS na camada de aplicação, o GET_Flood é considerado um ataque de alta taxa. Esse ataque realiza diversas requisições HTTP GET para o servidor *web*, principalmente para fazer com

[5]De acordo com a Kaspersky (KUPREEV; BADOVSKAYA; GUTNIKOV, 2019), no terceiro quarto de 2019, 70,7% e 1,7% de todos ataques realizados foram de SYN_Flood e GET_Flood, respectivamente.

[6]Disponível em: <https://securelist.com/ddos-attacks-in-q2-2021/103424/>. Acessado em 30 de setembro de 2021.

[7]Página web: <https://linux.die.net/man/8/hping3>. Acessado em 18 de agosto de 2019.

que requisições maliciosas se pareçam legítimas. O diagrama desse ataque está demonstrado na Figura 2.7, onde o atacante, inicialmente, solicita uma conexão TCP com o servidor, e que, após o estabelecimento total da conexão, o atacante envia uma solicitação HTTP GET para verificar se o servidor responderá à página *web*. Se o atacante receber a página, inicia-se o ataque com o envio de várias requisições HTTP GET sem esperar a resposta do servidor. Este tenta responder a todas as requisições, no entanto, por ser várias requisições, não consegue, gerando então a indisponibilidade. Nesta Tese, para gerar o ataque GET_Flood foi utilizada a ferramenta Goldeneye[8].

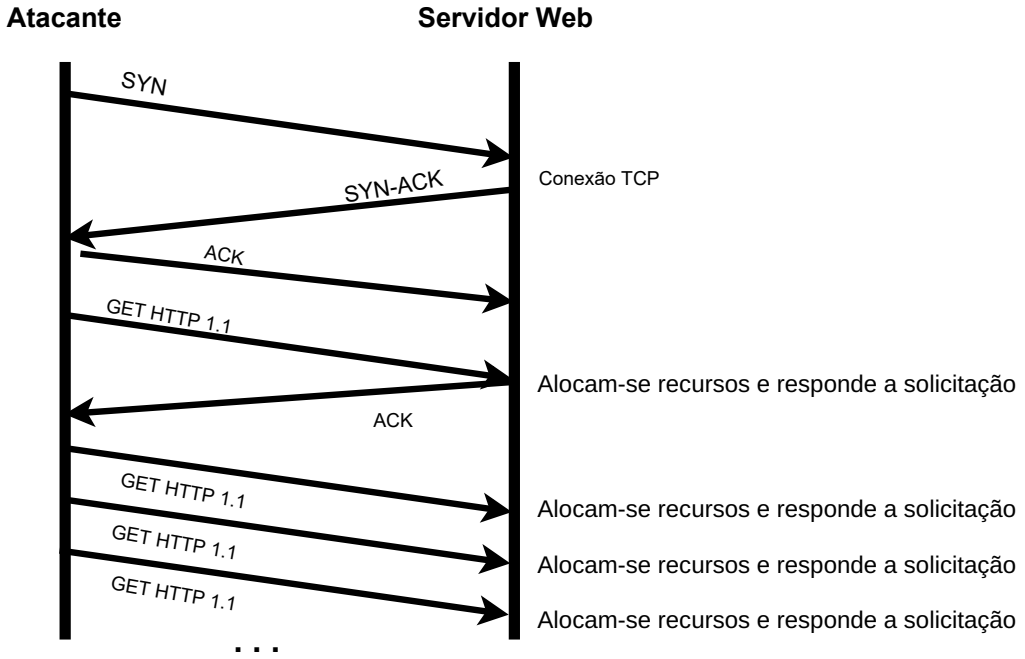



**Figura 2.7: Diagrama do ataque GET_Flood**

Ainda, apresentamos o ataque Slowloris, que é um ataque de DDoS na camada de aplicação, classificado como *low-rate*. De acordo com Dantas *et al.* (DANTAS; NIGAM; FONSECA, 2014), esse tipo de ataque procura explorar vulnerabilidades existentes em protocolos de aplicação, enviando menos tráfego na rede, mas ocupando espaço no atendimento do servidor. Com isso, clientes legítimos não encontram espaço para atendimento, tornando o serviço indisponível. O Slowloris age em cima do *timeout* da aplicação *web*, ou seja, age no tempo em que uma requisição pode permanecer na fila de atendimento da aplicação. A intenção desse ataque é permanecer no *buffer* de atendimento por tempo indeterminado. Isso pode ser observado na Figura 2.8, pois apresenta um diagrama do funcionamento do ataque Slowloris, que consiste no envio de requisições HTTP GET, com o HEADER incompleto, faltando os caracteres CR - *Carriage Return*, ASCII 13, */r*, e o LF - *Line Feed*, ASCII 10, */n* no final do pacote. Próximo

[8]Página web: <https://github.com/jseidl/GoldenEye>. Acessado em 18 de agosto de 2019.

ao *timeout* da aplicação, o ataque Slowloris envia novamente uma requisição HTTP GET incompleta, renovando o *timeout* e, assim, permanecendo no *buffer* de atendimento da aplicação. Como há várias requisições incompletas, o ataque ocupa todo o *pool* de atendimento, gerando a indisponibilidade para esse serviço.

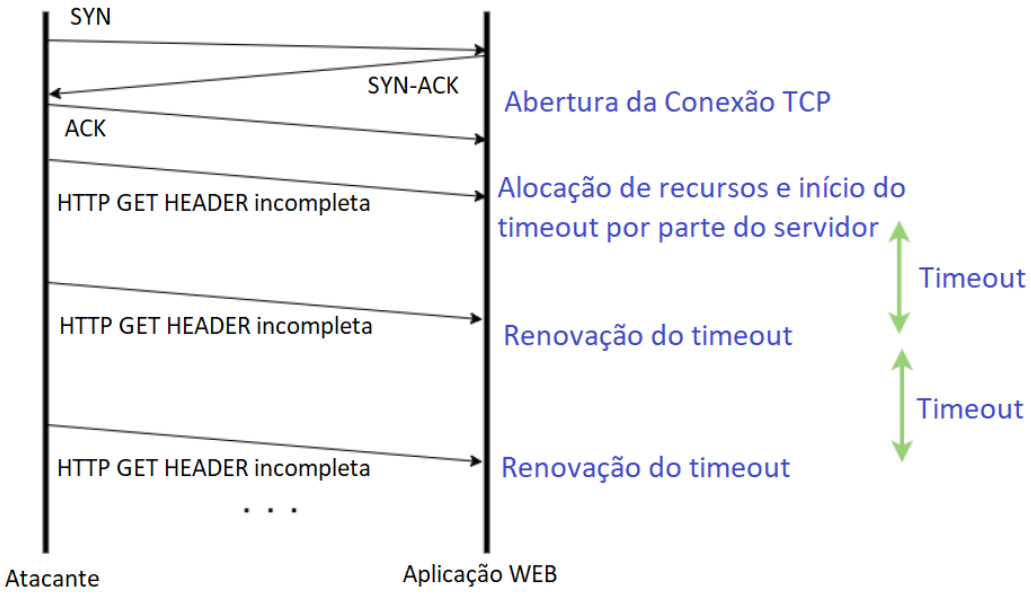



**Figura 2.8: Diagrama do ataque Slowloris**

## 2.5 Ataques de Intrusão

Além dos ataques de DDoS, os ataques de intrusão também são problemas em redes de computadores e, consequentemente, em infraestruturas de nuvem. A sua característica, como o seu nome já diz, é realizar uma invasão no próprio Sistema Operacional ou em aplicações, com o intuito de sequestrar a máquina ou as informações guardadas nela. Esses ataques, normalmente, utilizam-se de brechas de segurança ou má configuração dos serviços e das aplicações. Um exemplo é o ataque contra o serviço SSH (*Secure Shell*), que possibilita acesso remoto da máquina. Uma configuração errada, do protocolo ou de credenciais, permite que usuários maliciosos consigam acesso à máquina, possibilitando o roubo de informações confidenciais, os ataques de negação de serviço, ou até a utilização dessa máquina para cometer outros crimes. Esse tipo de ataque é extremamente comum, pois, de acordo com o relatório da F5 Labs and Communities (BODDY, 2018), a porta 22 TCP, específica do serviço SSH, é 2,7 vezes mais atacada que a porta 80 do HTTP.

De acordo com Huang *et al.* (HUANG; SIEGEL; MADNICK, 2018), os ataques de DDoS, quando são realizados, acabam gerando anomalias, principalmente relacionadas aos pacotes

de rede, que podem ser detectados em uma inspeção de pacotes. Por outro lado, os ataques de intrusão são menos detectáveis, visto que se assemelham ao tráfego de clientes legítimos. Pode-se destacar, entre os diversos tipos, os seguintes ataques de intrusão: contra o SSH, contra o Telnet, e a Injeção SQL (SQL_Injection).

### 2.5.1 Ataques de intrusão utilizados no contexto da Tese

Na Tese em questão, foi escolhido o ataque SQL_Injection (SQLi). Conforme o relatório da Akamai (AKAMAI, 2019) para o ano de 2019, 65,1% dos vetores de ataques contra *web* foram com SQLi, que é uma injeção de código malicioso no SQL. A linguagem SQL é utilizada para o gerenciamento de dados dentro de bancos de dados relacionais, estando presente na grande maioria das aplicações *web*. Os bancos de dados, como MySQL, Oracle e SQLite utilizam a linguagem SQL como padrão. Todo esse cenário permite a proliferação de ataques contra o SQL.

Os ataques de SQLi permitem que os usuários maliciosos tenham acesso aos dados contidos nos bancos de dados. Assim, estes usuários maliciosos podem obter seu conteúdo, modificar ou até mesmo destruir a informação. De acordo com Ma *et al.* (MA et al., 2019), existem dois tipos de ataques de injeção SQL: o usuário malicioso insere o código diretamente nas variáveis de entrada do usuário do banco de dados, as quais são concatenadas com o comando SQL. Quando comando for executado, o código malicioso também será. Já o segundo tipo de ataque ocorre de forma indireta, injetando o código malicioso que será armazenado nas tabelas ou nos documentos do banco de dados. O código armazenado é resgatado e conectado a um comando SQL, que faz a execução desse código malicioso.

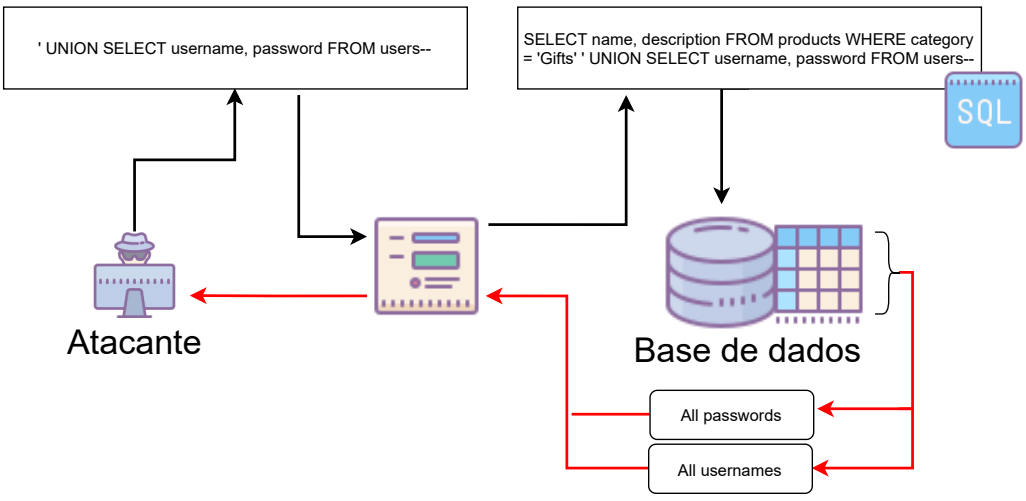



**Figura 2.9: Diagrama do ataque SQL_Injection**

A Figura 2.9 apresenta um diagrama do ataque SQLi. O atacante cria um código malici-

oso e insere no formulário de requisição SQL para o banco de dados. Quando esse código é executado, de acordo com o exemplo, todos os usuários e todas as senhas guardadas no banco de dados são resgatados pela requisição SQL e enviados para o usuário malicioso. Esse código malicioso é adaptado de acordo com o objetivo específico do atacante.

## 2.6 Aprendizado de Máquina

O aprendizado de máquina (*Machine learning* - em inglês) é a capacidade de fazer com que computadores tomem decisões sem ser explicitamente programados para tal. Isso é realizado por meio da programação de um conjunto de algoritmos que analisam dados e aprendem com eles (KLAINE et al., 2017), ou seja, a partir das técnicas de ML a máquina tenta encontrar padrões no conjunto de dados.

De modo geral, existem algumas fases para utilizar o aprendizado de máquina (SABAT, 2016):

- Seleção de dados: fase em que é realizada a coleta, identificação, limpeza, seleção e adequação dos dados que serão utilizados;
- Seleção de características: nessa fase, é realizada a escolha de dados que são menos sensíveis a ruídos e que sejam fáceis de serem manipulados, ocorrendo a separação dos dados que serão utilizados para o treinamento e os dados para os testes;
- Seleção de modelos: nessa fase, são escolhidos os algoritmos de aprendizado de máquina. Deve-se selecionar modelos mais simples e controlados para depois ir aperfeiçoando a partir dos resultados obtidos;
- Treinamento: fase em que serão identificados os parâmetros adequados para o algoritmo, fazendo-o trabalhar da melhor maneira possível;
- Validação: fase em que será colocado o treinamento em prática com a massa de dados, verificando o sucesso do treinamento;
- Aplicação: fase em que é aplicado o modelo com os dados sobre os quais não se sabe o resultado;
- Produção: a fase em que o modelo foi validado, testado e aplicado, sendo então colocado em produção.

O resultado obtido deriva de duas variáveis: a técnica de aprendizado utilizada e o treinamento realizado. Para que o aprendizado de máquina consiga realizar previsões e encontrar padrões, é necessário executar um treinamento, utilizando uma massa de dados confiável e mais próxima da realidade, para que, quando a máquina for executada, consiga verificar algum padrão que já foi encontrado durante a fase de treinamento. A outra variável que influencia no resultado do aprendizado de máquina são as técnicas utilizadas. Para cada problema, que pode ser resolvido por meio do aprendizado de máquina, deve ser utilizado um tipo de técnica que se encaixe nesse problema. Os tipos de técnicas e suas características serão discutidas a seguir.

### 2.6.1 Tipos de aprendizado de máquina

De modo geral, existem três tipos de técnicas para o aprendizado de máquina, são elas: aprendizado supervisionado (*Supervised Learning* - em inglês), aprendizado sem supervisão (*Unsupervised Learning* - em inglês) e aprendizado por reforço (*Reinforcement Learning* - em inglês) (FADLULLAH et al., 2017). Um esquema dos tipos está apresentado na Figura 2.10.

A técnica do aprendizado supervisionado, como o próprio nome indica, é uma técnica que requer uma supervisão para que os algoritmos aprendam seus parâmetros. Nesse tipo de aprendizado, os algoritmos recebem um conjunto de dados que contém a relação de uma determinada entrada com a sua respectiva saída, como resultado do processamento. Baseado nessa relação de entrada/saída, um modelo de dados é determinado pela máquina. Quando um conjunto de dados de entrada é coletado e inserido como entrada na máquina, o algoritmo realiza as suas previsões (NGUYEN; ARMITAGE, 2008; KLAINE et al., 2017).

A técnica de aprendizado sem supervisão consiste em oferecer ao algoritmo um conjunto de entradas, e o objetivo do algoritmo é inferir as saídas, sem a presença de um supervisor fornecendo as respostas corretas ou uma observação relacionada ao erro, isto é, sem *feedback* para o algoritmo (FADLULLAH et al., 2017; KLAINE et al., 2017). De maneira geral, o algoritmo sem supervisão encontra tendências e padrões que são incomuns aos olhos de uma supervisão humana. Como não há um rótulo ou padrão especificado no treinamento, os dados serão agrupados e ordenados de acordo com o padrão que a técnica sem supervisão encontrou.

Por último, existe a técnica de aprendizado por reforço, que junta os dois tipos anteriores. Essa técnica é bem semelhante do tipo sem supervisão, pois o algoritmo recebe apenas um conjunto de entradas, sem nenhum rótulo. A diferença está no processo de definição das tendências e padrões. Quando o algoritmo chega a um resultado, este é reforçado com um *feedback* positivo ou negativo. Ou seja, quando o padrão encontrado for bom, então o algoritmo recebe uma recompensa, caso seja um padrão negativo, o sistema recebe uma punição (KLAINE et al.,

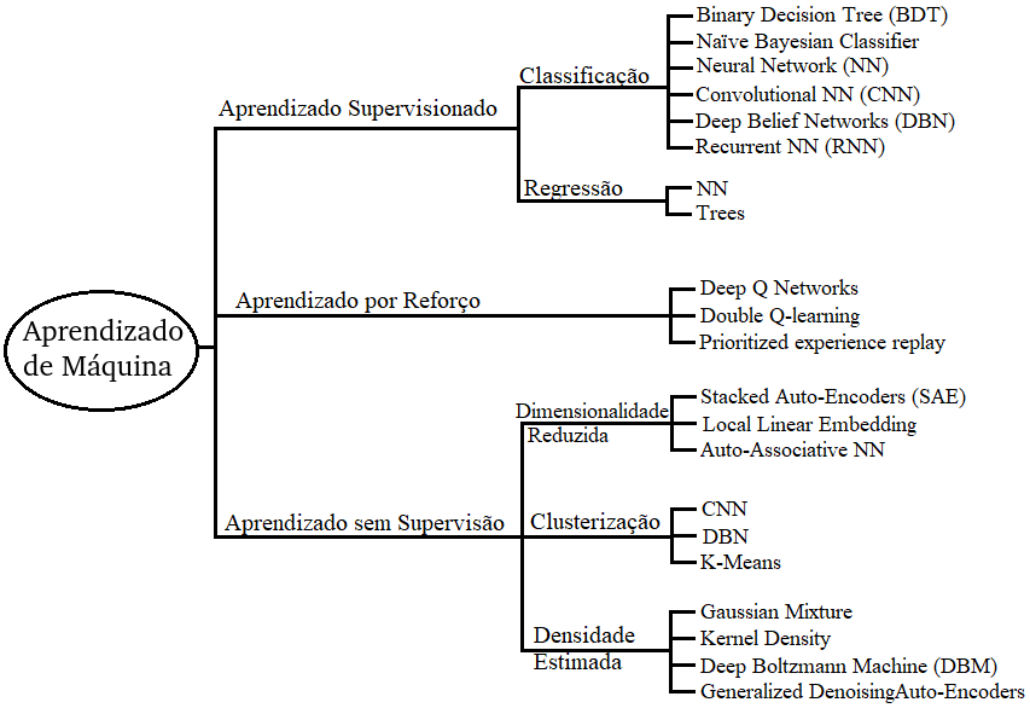



**Figura 2.10: Esquema com os tipos e técnicas de aprendizado de máquina. Adaptado de Fadlullah *et al.***

2017).

### 2.6.2 Algoritmos de Aprendizado de máquina no contexto da tese

No âmbito desta Tese, serão utilizadas técnicas de aprendizado de máquina do tipo supervisionado, particularmente os algoritmos de classificação. A proposta é que os algoritmos de aprendizado de máquina classifiquem se está ocorrendo um ataque ou se são apenas clientes. Além disso, se estiver acontecendo um ataque, que realize a classificação desse tipo de ataque. Para isso, nesta Tese, foram utilizados e avaliados vários algoritmos de aprendizado de máquina clássicos: k-Nearest Neighbor (kNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest (RF) e Multi-Layer Perceptron.

No entanto, como será discutido no Capítulo 5, dois algoritmos tiveram os melhores indicadores de desempenho no contexto desta Tese: o kNN e Random Forest. O kNN é um algoritmo de aprendizado de máquina supervisionado, utilizado para resolver problemas de classificação. No método, os mesmos tipos de objetos são classificados por estarem perto uns dos outros. Essa distância deve ser calculada de acordo com uma definição matemática, sendo a distância Euclidiana a mais usual. O algoritmo classifica a amostra com base na quantidade de $k$ objetos mais próximos. O kNN funciona da seguinte forma:

- Para cada amostra sem rótulo $v$;
- Encontrar as $k$ amostras mais próximas a $v$, usando uma métrica de distância $dist(v, x_i)$;
- Associar a $v$ a classe mais frequente entre as $k$ amostras (maioria de votos).

Para que este algoritmo funcione de forma eficiente, é preciso ter cuidado para escolher o valor de $k$. Pois, grandes valores de $k$ produzem uma separação mais suave das classes e podem reduzir o impacto de ruídos. No entanto, com o valor de $k$ menor, o modelo criado tem uma maior estabilidade na classificação (MISHRA et al., 2021).

O Random Forest, ou Florestas Aleatórias, é um algoritmo supervisionado, que pode ser utilizado para realizar a tarefa de classificação. O algoritmo é definido como um conjunto de árvores de decisão em que a aleatoriedade foi explicitamente inserida no processo de construção do modelo de cada árvore de decisão. Na classificação, cada nova amostra é passada sobre cada árvore de decisão. Cada árvore retorna uma classe (é considerado que a árvore votou para aquela classe). A floresta escolhe a classe que obteve mais votos (BREIMAN; CUTLER, 2001). Cada árvore é construída como segue:

- Selecione aleatoriamente $N$ amostras da base de treinamento. Estas amostras serão usadas para construir a árvore;
- Se há $M$ atributos, selecione $d$ atributos aleatoriamente ($d << M$) e independentemente para cada nó de divisão;
- Selecione o melhor atributo entre os $d$ atributos para fazer a separação no nó;
- Cada árvore deve ser construída até a máxima extensão possível. Não há procedimento de poda.

No artigo original (BREIMAN; CUTLER, 2001), foi mostrado que a taxa de erro da floresta depende de duas coisas: (i) a correlação entre quaisquer duas árvores na floresta, e que aumentar a correlação aumenta a taxa de erro da floresta; (ii) a força de cada árvore individual na floresta, pois, uma árvore com uma baixa taxa de erro é um classificador forte e, dessa forma, aumentar a força das árvores individuais diminui a taxa de erro da floresta. Em suma, com a criação de diversas árvores distintas, o algoritmo Random Forest oferece melhores indicadores de desempenho comparado a algoritmos de árvore de decisão (AGGARWAL, 2015).

Nesse contexto de uso de algoritmos de aprendizado de máquina, principalmente para o processo de classificação, faz-se necessário o uso de uma seleção de características. Isso porque

os dados reais podem conter características de relevância variável para a predição das classes. De acordo com Aggarwal (AGGARWAL, 2015), existem três métodos de realizar a seleção de características para o processo de classificação: os modelos *Filter, Wrapper* e *Embedded*. No caso desta Tese, foi escolhido o modelo *Wrapper*. A ideia dessa seleção é que um conjunto específico de característica ofereça um modelo de classificação mais preciso. Dessa forma, esse método de seleção de características utiliza um algoritmo de classificação específico, denotado por $A$. A estratégia básica do método *Wrapper* é refinar iterativamente um conjunto atual de características $F$, adicionando-lhe sucessivamente novas características. A estratégia pode ser resumida pelas duas etapas a seguir, que são executadas iterativamente:

- Crie um conjunto ampliado de características $F$, adicionando uma ou mais características ao atual conjunto de recursos;
- Use um algoritmo de classificação $A$ para avaliar a acurácia do conjunto de características $F$. Use a acurácia para aceitar ou rejeitar o acréscimo de F.

O modelo *Wrapper* foi utilizado no contexto desta Tese, como será discutido na Seção 5.1.2.

### 2.6.3 Indicadores de desempenho dos algoritmos de aprendizado de máquina

Como saber se a classificação realizada por um algoritmo de aprendizado de máquina foi bem sucedida? Existem alguns indicadores que permitem avaliar esse desempenho, na tarefa de classificar uma amostra (AGGARWAL, 2015). Os que serão utilizados nesta Tese são: Acurácia, Precisão, Revocação e *F1-Score*. As definições de cada métrica são apresentadas abaixo, sendo *true positives* (TP) o número de casos atuais positivos corretamente predito como positivo, *true negatives* (TN) é o número de casos atuais negativos corretamente predito como negativo, *false positives* (FP) é o número de casos atuais negativos incorretamente predito como positivo, e *false negative* (FN) é o número de casos atuais positivos incorretamente predito como negativo.

$$Acurácia = \frac{TP+TN}{TP+TN+FP+FN} \times 100\% \tag{2.1}$$

$$Precisão = \frac{TP}{TP+FP} \times 100\% \tag{2.2}$$

$$\text{Revocação} = \frac{TP}{TP+FN} \times 100\% \tag{2.3}$$

$$F1-Score = 2 * \frac{Precisão * \text{Revocação}}{Precisão + \text{Revocação}} \times 100\% \tag{2.4}$$

A métrica Acurácia mede a relação entre as amostras corretamente classificadas (TP + TN) e o número total de amostras no *dataset*. Precisão é a relação entre os casos positivos que foram corretamente classificados (TP) e o total de casos positivos estimados no *dataset* (TP + FP), ou seja, mede a capacidade dos algoritmos de ML de classificar apenas casos positivos reais como positivos. Revocação é a relação entre os casos positivos corretamente classificados (TP) e o número total existentes no *dataset* (TP + TN), ou seja, mede a capacidade do algoritmo de aprendizado de máquina reconhecer todos os casos corretamente, sejam eles positivos ou negativos. Por fim, *F1-Score* é a média harmônica entre a Precisão e a Revocação. Um modelo de ML ideal retorna os valores de todas essas métricas iguais a 100%.

## 2.7 Considerações do Capítulo

Foram evidenciadas neste capítulo temáticas acerca da proposta deste trabalho. Na primeira seção do capítulo, foram apresentados os conceitos relacionados à Computação em Nuvem, à sua classificação e tipificação. Em sequência, foi discutido sobre a plataforma de computação em nuvem OpenStack, bem como sobre o módulo Ceilometer, que é o módulo do OpenStack responsável pelas informações de telemetria da infraestrutura da nuvem. Posteriormente, abordou-se o mecanismo de dimensionamento automático, que é a forma da infraestrutura da nuvem oferecer mais recursos computacionais para a máquina virtual hospedada. Como visto, esse dimensionamento pode ser horizontal, aumentando a quantidade de máquinas virtuais, ou vertical, em que são incrementados os recursos computacionais da VM. Na sequência, os ataques de negação de serviço foram conceituados, tipificando aqueles que estão no contexto desta Tese. Em seguida, os ataques de intrusão também foram explicados e indicados os utilizados nesta Tese. Por fim, encontra-se uma exposição sobre aprendizado de máquina, as fases para utilizar o aprendizado de máquina, bem como os algoritmos utilizados no contexto desta Tese e suas métricas de avaliação.

Todos esses conceitos são importantes para o trabalho, visto que será proposta a utilização da telemetria nativa da computação em nuvem, combinada com algoritmos de aprendizado de máquina para detectar e identificar ataques. Uma vez detectado e identificado, será realizada

uma forma de mitigação, utilizando, por exemplo, o dimensionamento automático.

No próximo capítulo, os trabalhos disponíveis na literatura e que têm relação com a presente Tese, serão analisados e confrontados com as hipóteses apresentadas nesta pesquisa. Serão analisados trabalhos que envolvem a detecção, identificação e mitigação de ataques de DDoS, e de intrusão, e que utilizam algoritmos de aprendizado de máquina. Esses trabalhos podem, ou não, serem realizados no ambiente de nuvem.

# Capítulo 3

## TRABALHOS RELACIONADOS

Neste capítulo são descritos alguns trabalhos relacionados com a proposta desta Tese, levando em consideração as semelhanças e diferenças. Para melhor contextualizar o presente estudo, os trabalhos relacionados estão divididos em cinco categorias: (i) que utilizam algoritmos de aprendizado de máquina para a detecção e identificação de ataques de DDoS; (ii) que realizam detecção e identificação de ataques de DDoS em ambiente de computação em nuvem; (iii) que utilizam alguma forma de telemetria da computação em nuvem para realizar a detecção e identificação dos ataques; (iv) que realizam detecção e identificação de ataques de intrusão; (v) por fim, as formas de mitigação de ataques em ambientes de nuvem.

A Tabela 3.1 apresenta uma sumarização comparativa com importantes características dos trabalhos relacionados que usam ML com a proposta apresentada por esta Tese. Para realizar uma comparação relevante, foram escolhidos os seguintes parâmetros, separados por colunas: a primeira coluna verifica qual tipo de ataque é objeto de estudo e, em sequência, se realiza a detecção e identificação dos ataques. Depois é verificado se realiza alguma mitigação. Em sequência, é examinado se o trabalho relacionado utiliza informações dos pacotes de rede ou fluxos de rede. Considera-se nessa classificação, se na utilização de pacotes de rede, há alguma inspeção dos cabeçalhos e/ou da carga útil dos pacotes, enquanto que, para a classificação de fluxos de rede, verifica-se as informações geradas pelo agrupamento de pacotes com o mesmo endereço IP de origem e destino, porta de origem e destino e protocolo utilizado. Também é investigado se as técnicas propostas dependem de assinaturas dos ataques ou se são agnósticos para realizar a detecção e identificação. Posteriormente, é verificado se dados de telemetria da infraestrutura da nuvem são utilizados. E, por fim, é analisada a viabilidade de utilizar a técnica proposta em um ambiente de nuvem.

**Tabela 3.1: Comparação entre os trabalhos relacionados, utilizando ML, em oito aspectos diferentes**

| Trabalhos Relacionados | Tipo de Ataque | Detecta & Identifica | Mitigação | Packet Traces | Baseado em Fluxo | Agnóstico à Assinatura do Ataque | Telemetria | Computação em Nuvem |
|---|---|---|---|---|---|---|---|---|
| (MENDONÇA et al., 2019) | DDoS | Sim | Não | Não | Não | Sim | Não | Não |
| (SURESH; ANITHA, 2011) | DDoS | Sim | Não | Sim | Não | Não | Não | Não |
| (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021) | DDoS | Sim | Sim | Sim | Não | Não | Não | Não |
| (ZEKRI et al., 2017) | DDoS | Não | Não | Sim | Não | Não | Não | Sim |
| (HE; ZHANG; LEE, 2017) | DDoS | Não | Não | Sim | Não | Não | Não | Sim |
| (PHAN; PARK, 2019) | DDoS | Não | Sim | Não | Sim | Sim | Não | Sim |
| (BHARDWAJ; MIRANDA; GAVRILOVSKA, 2018) | DDoS | Não | Sim | Sim | Não | Não | Não | Não |
| (SINGH et al., 2020) | DDoS | Não | Não | Sim | Não | Não | Não | Sim |
| (ÖZÇELIK; CHALABIANLOO; GüR, 2017) | DDoS | Sim | Sim | Não | Sim | Sim | Não | Sim |
| (MIAO et al., 2015) | DDoS e Intrusão | Sim | Não | Não | Sim | Sim | Não | Sim |
| (JIA et al., 2020) | DDoS | Sim | Sim | Não | Sim | Sim | Não | Sim |
| (SOLANAS; HERNANDEZ-CASTRO; DUTTA, 2014) | DDoS[a] | Não | Não | Não | Não | Não | Sim | Sim |
| (NICHOLS et al., 2016) | *Malware* | Não | Não | Não | Não | Não | Sim | Sim |
| (JYOTHI et al., 2016) | DDoS | Sim | Sim | Sim | Não | Parcialmente[b] | Não | Não |
| (AZMANDIAN et al., 2015) | *Malware* | Não | Não | Não | Não | Sim | Não | Não |
| (URIAS; STOUT; LIN, 2016) | Intrusão | Sim | Não | Sim | Não | Sim | Não | Não |
| (BAO; AHN; PARK, 2018) | Intrusão | Sim | Sim | Sim | Não | Sim | Não | Sim |
| **Tese** | **DDoS e Intrusão** | **Sim** | **Sim** | **Não** | **Não** | **Sim** | **Sim** | **Sim** |

[a] Apenas ataque PING_Flood;
[b] Jyothi *et al.* também utiliza informações da rede, o que tem impacto sobre esta característica.

## 3.1 Detecção/Identificação de ataques de DDoS e aprendizado de máquina

Diversos *surveys*, como (BOUTABA et al., 2018; BUCZAK; GUVEN, 2016; SOMANI et al., 2017; SHAMELI-SENDI et al., 2015), apresentam a utilização de algoritmos de aprendizado de máquina para realizar a detecção e identificação de ataques de negação de serviço. Um ponto em comum em vários trabalhos relacionados é o uso de informações extraídas dos pacotes de rede, seja dos cabeçalhos, seja da carga útil, e que são utilizadas como entradas para o treino e teste em algoritmos de aprendizado de máquina, mas que podem ser diferenciados na sua fase de seleção de características. De acordo com Boutaba *et al.* (BOUTABA et al., 2018), vários trabalhos utilizam cabeçalhos dos protocolos, enquanto que outros utilizam apenas informações da carga útil. Outras abordagens, como em (BOERO; MARCHESE; ZAPPATORE, 2017), incluem informações oriundas do controlador das Redes Definidas por *Software* (*Software Defined Network* - SDN) e do protocolo Openflow para realizar o treino, e teste do algoritmo Support Vector Machine (SVM) para detectar anomalias, incluindo ataques de DDoS.

Em Mendonça *et al.* (MENDONÇA et al., 2019), são propostos algoritmos de ML para detectar e identificar DDoS no contexto de equipamentos de redes domésticos. Como entrada para o treino e teste dos algoritmos de aprendizado de máquina, diversos meta dados são coletados desses equipamentos de rede domésticos, como a quantidade de *Bytes* e de pacotes que estão entrando e saindo de uma interface. Ou seja, não há inspeção dos cabeçalhos e nem da carga útil dos pacotes e, além disso, a privacidade do usuário não é violada. Essa vantagem também é verificada na presente Tese, pois como há o uso da telemetria da computação em nuvem para detectar e identificar os ataques de DDoS, não há inspeção de pacotes de rede, garantindo assim a privacidade dos dados.

Suresh *et al.* (SURESH; ANITHA, 2011) usa o *dataset* CAIDA (TELESCOPE, 2021) para criar um modelo de ML e verificá-lo em um conjunto de dados do tráfego coletado em um ambiente seguro e inteligente. O estudo avalia diversos algoritmos de aprendizado de máquina utilizando ambos *datasets*. Enquanto que, Dimolianis *et al.* (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021) disseca os pacotes TCP, obtendo informações para classificar a presença do ataque SYN_Flood por meio de algoritmos de aprendizado de máquina. No entanto, Suresh *et al.* e Dimolianis *et al.* usam informações oriundas dos pacotes de rede para gerar e verificar o modelo de aprendizado de máquina, enquanto que a proposta desta Tese é agnóstica ao tráfego de rede, utilizando-se apenas das informações da telemetria da nuvem.

## 3.2 Computação em nuvem e detecção/identificação de ataques de DDoS

Diversos trabalhos relacionados realizam detecção e identificação de ataques de DDoS em ambiente de computação em nuvem, desses, destacam-se cinco trabalhos: Zekri *et al.* (ZEKRI et al., 2017), He *et al.* (HE; ZHANG; LEE, 2017), Phan *et al.* (PHAN; PARK, 2019), Bhardwaj *et al.*(BHARDWAJ; MIRANDA; GAVRILOVSKA, 2018) e Özçelik *et al.* (ÖZÇELIK; CHALABIANLOO; GüR, 2017). Zekri *et al.* (ZEKRI et al., 2017) utiliza fluxos de rede e o algoritmo de Árvore de Decisão (C.4.5) para detectar assinaturas de ataques de DDoS volumétricos em servidores hospedados no ambiente de nuvem OpenStack. Apesar de utilizar OpenStack, Zekri *et al.* usa dados oriundos dos pacotes para detectar ataques de DDoS, enquanto que esta Tese utiliza dados oriundos do serviço nativo de telemetria do OpenStack. Além disso, em Zekri *et al.* não há a realização da identificação do ataque, apenas a detecção de um tipo específico.

Em He *et al.* (HE; ZHANG; LEE, 2017), algoritmos de aprendizado de máquina são treinados utilizando dados obtidos em pacotes coletados durante diferentes tipos de ataques: *flooding*, *spoofing* e *brute force*. A principal diferença entre He *et al.* (HE; ZHANG; LEE, 2017) e a presente Tese, que se baseia na utilização da telemetria, é que em He *et al.* há a captura dos pacotes oriundos da rede, analisa os cabeçalhos e a carga útil e seleciona características relevantes para serem usadas como entradas nos algoritmos de ML. Phan (PHAN; PARK, 2019) apresenta um modelo híbrido para identificar ataques de DDoS usando os algoritmos Support Vector Machine (SVM) e Self Organized Map (SOM). Esse trabalho usa, além das características extraídas dos pacotes, métricas coletadas pelo controlador da rede SDN criada dentro da infraestrutura da nuvem Openstack. Apesar de ser possível coletar as métricas do controlador SDN no ambiente desta Tese, foi optado pela utilização apenas das métricas relacionadas aos servidores físicos e das máquinas virtuais hospedadas na nuvem.

Em Bhardwaj *et al.* (BHARDWAJ; MIRANDA; GAVRILOVSKA, 2018), o serviço ShadowNet é apresentado com objetivo de prevenir ataques de DDoS realizados por dispositivos IoT (*Internet-of-Things* - IoT), particularmente o ataque GET_Flood e UDP_Flood. ShadowNet trabalha como um proxy, sendo um *gateway* para cada dispositivo que envia dados para a infraestrutura da nuvem, criando assim um caminho rápido para o tráfego de equipamentos conhecidos e bloqueando os endereços IPs maliciosos. A proposta do ShadowNet é prevenir que aconteça um ataque de negação de serviço sem detectar ou identificar esse ataque. Em Singh *et al.* (SINGH et al., 2020), o *dataset* KDD (HETTICH; BAY, 1999) é utilizado para extrair as características e para detectar ataques presentes nesse *dataset*. Foi utilizado o algoritmo Naive Bayes para classificar entre tráfego normal ou ataque. Apesar de obter uma boa acurácia, Singh

*et al.* (SINGH et al., 2020) utiliza informações da rede, enquanto que a presente Tese utiliza informações da telemetria da infraestrutura da nuvem.

Em Özçelik (ÖZÇELIK; CHALABIANLOO; GüR, 2017), um sistema para detectar e mitigar ataques de DDoS gerados a partir de equipamentos IoT (o trabalho usa como caso de estudo a botnet Mirai), usando SDN e ambiente de computação em névoa, é proposto. É proposta a utilização de dois algoritmos, o *Threshold Random Walk with Credit Based Rate Limiting* (TRW-CB) e o *Rate Limiting*. No estudo (ÖZÇELIK; CHALABIANLOO; GüR, 2017), o primeiro algoritmo é para detectar a fase de escaneamento realizado por *malware*, e o segundo observa as vítimas infectadas que tentam se conectar ao maior número de diferentes *hosts*, em um curto espaço de tempo, para a propagação do vírus. A verificação é realizada para cada pacote TCP com a *flag* SYN em cada equipamento, utilizando-se os dois algoritmos apresentados anteriormente. O trabalho (ÖZÇELIK; CHALABIANLOO; GüR, 2017) difere na forma de detecção dos ataques, pois nesta Tese são utilizados os dados da telemetria disponíveis na computação em nuvem.

Miao *et al.* (MIAO et al., 2015) realiza uma caracterização de ataques baseado em rede, onde as informações são coletadas de múltiplos *data centers* geograficamente distribuídos. Os dados são coletados utilizando NetFlow e a identificação dos ataques é realizada pela análise de quatro características principais: quantidade de pacotes por segundo, número de clientes únicos ou número de conexões, informações das *flags* TCP e a comunicação com *hosts* maliciosos na Internet. Com essas informações, nove tipos de ataques são identificados, desde ataques de DDoS até SQL_Injection e SPAM. Miao *et al.* afirma que a utilização de dados provenientes do NetFlow dificulta a detecção de ataques de DDoS na camada de aplicação, principalmente aqueles em que não são apresentados nas amostras do NetFlow (por exemplo o HTTP Slowloris) (MIAO et al., 2015). A diferença entre o trabalho de Miao *et al.* e a presente Tese é a utilização dos dados para detectar/identificar ataques. Miao *et al.* utiliza informações de fluxos de rede, enquanto este trabalho utiliza dados da telemetria nativa da infraestrutura da nuvem.

Jia *et al.* (JIA et al., 2020) apresenta FlowGuard, um mecanismo que detecta, identifica e mitiga ataques de DDoS, principalmente oriundos de dispositivos IoT. Para detectar e identificar, Jia *et al.* realiza a coleta de informações dos fluxos que estão entrando na infraestrutura da nuvem, enquanto que, para realizar a mitigação desses ataques, Jia *et al.* utiliza regras de bloqueio e filtragem dos pacotes em seu sistema. Assim como os outros trabalhos, Jia *et al.* (JIA et al., 2020) não utiliza informações da telemetria nativa da infraestrutura da nuvem, sendo essa característica uma contribuição da presente Tese.

## 3.3 Sistema de telemetria na computação em nuvem, ML e detecção/identificação de ataques de DDoS

Existem alguns trabalhos relacionados usando a telemetria da computação em nuvem combinado com algoritmos de ML para detectar e identificar ataques. Em Solanas *et al.* (SOLANAS; HERNANDEZ-CASTRO; DUTTA, 2014), é utilizada uma infraestrutura da nuvem OpenStack e os algoritmos de aprendizado de máquina, para detectar atividades suspeitas, como o ataque de DDoS PING_Flood, a mineração de criptomoedas e as falhas de *hardware*. Em relação ao ataque PING_Flood, o objetivo é verificar se a origem desse tipo de ataque é algum *host* hospedado na própria infraestrutura da nuvem. Assim, as principais diferenças para este trabalho são: (i) Solanas *et al.* foca em apenas um ataque de DDoS, sendo esse um ataque volumétrico de Camada 3, o PING_Flood, enquanto que esta Tese verifica ataques de DDoS em camadas mais altas (Camadas 4 e 7); e (ii) a origem do ataque, pois em Solanas *et al.* a intenção é detectar se algum *host* hospedado no ambiente de nuvem está gerando o ataque, enquanto que o presente estudo verifica se o *host* é a vítima do ataque.

Nichols *et al.* (NICHOLS et al., 2016) utiliza dados de CPU, disco e interface de rede coletadas da telemetria oriundas de uma infraestrutura da nuvem OpenStack. Esses dados são utilizados para treino e teste dos algoritmos de aprendizado de máquina para descobrir *malware* hospedado no ambiente de nuvem. Dessa forma, Nichols *et al.* não se propõe a detectar ou identificar ataques de DDoS. Jyothi *et al.* (JYOTHI et al., 2016) utiliza dados de *hardware* para identificar ataques de DDoS, mas esses dados não são oriundos da telemetria da computação em nuvem, mas sim do próprio hipervisor. Desse modo, a ocorrência de ataques de DDoS é verificada utilizando algoritmos de clusterização e de classificação. O trabalho de Jyothi *et al.* (JYOTHI et al., 2016) necessita de uma ação intrusiva tanto nos hipervisores quanto nas VMs, podendo aumentar o *overhead* do sistema ou dos recursos físicos dos hipervisores. Um claro contraste é que, nesta Tese, são utilizados dados da telemetria nativa da infraestrutura da nuvem, que estão disponíveis em qualquer estrutura de nuvem, já que a maioria dos recursos computacionais em cada VM (como CPU, Memória, Disco e Interfaces de Rede) estão sempre sendo monitorados para fins de faturamento.

Em estudo de Azmandian *et al.* (AZMANDIAN et al., 2015), utilizam-se também dados oriundos de hipervisor para detectar *malware* em máquinas virtuais. Além disso, são utilizadas técnicas de seleção de características e de redução de dimensionalidade para que os dados possam ser utilizados para detectar ataques maliciosos. Há duas diferenças principais entre Azmandian *et al.* e este trabalho: (i) Azmandian *et al.* usa dados que não são nativos do hipervisor, enquanto que esta Tese utiliza dados oriundos do serviço de telemetria nativo existente

em diversas infraestruturas de nuvem, como o OpenStack; (ii) Azmandian *et al.* tenta detectar *malware* em máquinas virtuais (que não estão hospedadas no ambiente de nuvem), enquanto que esta Tese tenta detectar e identificar ataques hospedados em uma infraestrutura da nuvem.

## 3.4 Detecção e identificação de ataques de intrusão

Para os ataques de intrusão, o método mais utilizado para detectar e identificar esses ataques é a utilização de *honeypots* (FAN et al., 2018). Diversos trabalhos utilizam informações do tráfego de rede, seja para alguns protocolos particulares como na detecção de ataques contra contratos inteligentes na Ethereum (TORRES; STEICHEN; STATE, 2019), fraudes no Facebook (CRISTOFARO et al., 2014), ou em redes celulares (LIEBERGELD; LANGE; BORGAONKAR, 2014); ou avaliando a efetividade de diferentes tipos de *honeypots*: (HAN et al., 2016).

Em Urias *et al.* (URIAS; STOUT; LIN, 2016), é proposto um sistema que realiza a migração do tráfego suspeito para uma *deception network*. A verificação desse tráfego é realizada utilizando informações do controlador e do *switch* SDN, combinado com informações do hipervisor KVM. A presente Tese se diferencia do trabalho de Urias *et al.* pois ele utiliza informações da rede para detectar um tráfego suspeito.

Bao *et al.* (BAO; AHN; PARK, 2018) utiliza recursos do ambiente de nuvem para instanciar sob demanda diferentes tipos de *honeypots* de acordo com recursos disponíveis da infraestrutura da nuvem e a necessidade de mais informações sobre os ataques. Bao *et al.* utiliza as tecnologias SDN e NFV para realizar a instanciação de novos *honeypots* e o redirecionamento do tráfego para as novas instâncias. Para iniciar o gatilho da criação de novas instâncias, Bao *et al.* utiliza informações do tráfego de rede, coletadas pelo controlador SDN. Essa é a diferença entre Bao *et al.* e a presente Tese, pois o presente trabalho utiliza a telemetria nativa oriunda da infraestrutura da nuvem para detectar e identificar ataques de intrusão.

## 3.5 Mitigação de ataques em ambientes de nuvem

Muitos dos trabalhos apresentados anteriormente, além de realizar a detecção e/ou identificação dos ataques, executam também alguma forma de mitigação. Phan *et al.* (PHAN; PARK, 2019), Bhardwaj *et al.* (BHARDWAJ; MIRANDA; GAVRILOVSKA, 2018), e Özçelik (ÖZÇELIK; CHALABIANLOO; GüR, 2017) utilizam a facilidade da rede SDN para implementar a mitigação dos ataques. São inseridas regras nos *switches* SDN, realizando a filtragem dos

pacotes maliciosos. Essas regras são inseridas o mais próximo da entrada da infraestrutura da nuvem, para que os efeitos dos ataques sejam diminuídos.

Jyothi *et al.* (JYOTHI et al., 2016) realiza a mitigação dos ataques inserindo regras em *firewall*, bloqueando e filtrando o tráfego malicioso. Além disso, são criadas listas de bloqueios com os endereços IPs conhecidos como maliciosos. A estratégia utilizada por Dimolianis *et al.* (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021) é semelhante, pois redireciona e filtra o tráfego TCP com a *flag* SYN, mitigando, assim, o ataque SYN_Flood. Bao *et al.* (BAO; AHN; PARK, 2018) utiliza a estratégia de instanciação de outros elementos de rede sob demanda, e faz a alteração do fluxo de tráfego para esses novos elementos. Na prática, Bao *et al.* efetua um encadeamento de funções de rede (*Service Function Chaining* - SFC) para mitigar esses ataques.

Em Dantas *et al.* (DANTAS; NIGAM; FONSECA, 2014), é proposto uma defesa contra o ataque Slowloris. Essa defesa, chamada SeVen, utiliza uma estratégia de seletividade, descartando, baseado em funções de probabilidade, requisições maliciosas. O objetivo dessa defesa não é detectar um ataque, mas apenas mitigar algum ataque Slowloris que possa estar acontecendo. De acordo com Dantas *et al.*, sem SeVen, a disponibilidade do serviço durante um ataque Slowloris é de 0% e o Tempo de Serviço (*Time-to-Service* - TTS), infinito, mas, com SeVen, 95% dos clientes legítimos obtiveram respostas do servidor *web*, e o TTS foi de 0,06 segundos em média.

Em Maurício e Lai (MAURICIO et al., 2017; LAI et al., 2016), são combinadas a utilização do SFC com a instanciação de IDS e Firewall para mitigar ataques. Tanto o SFC quanto a instanciação das funções de rede virtual são responsabilidade de um novo módulo na arquitetura, o Controlador de Segurança. Alharbi *et al.* (ALHARBI et al., 2017) apresenta algumas estratégias de mitigação dos ataques DDoS, como o uso de várias Funções de Rede Virtual (VNFs) para monitorar a rede, os recursos computacionais e o status de uma vítima. Apesar de ser uma boa proposta, em Alharbi *et al.* (ALHARBI et al., 2017), não há simulação, implementação ou resultados para avaliar sua proposta.

Em Tang *et al.* (TANG et al., 2015), é apresentada a utilização do dimensionamento automático (o *autoscaling*) para oferecer mais recursos computacionais para operadores de telecomunicações, suportando o número máximo de usuários no serviço. As políticas do dimensionamento seguem o Acordo de Nível de Serviço (*Service Level Agreement* - SLA), aumentando ou diminuindo os recursos computacionais de acordo com a necessidade do serviço. Apesar de Tang *et al.* (TANG et al., 2015) não propor a utilização do dimensionamento automático para mitigar ataques, essa habilidade da infraestrutura da nuvem pode ser utilizada para esse fim (CORRÊA

et al., 2019b).

## 3.6 Considerações do Capítulo

Este capítulo apresentou uma revisão dos trabalhos relacionados ao tema desta Tese. Percebe-se, a partir da Tabela 3.1 e de vários trabalhos disponíveis na literatura, que a detecção e identificação dos ataques de negação de serviço e de intrusão é um tema bastante debatido. Vários trabalhos utilizam informações dos cabeçalhos ou da carga útil dos pacotes, sendo necessário percorrer toda a pilha de protocolos. Outros trabalhos utilizam informações de fluxo de rede, ou seja, a quantidade de *bytes* ou de pacotes por segundo de um determinado fluxo agrupado por endereço IP de origem e destino, a porta TCP/UDP de origem e destino. Independente da forma que são coletados os dados, o que se percebe é que esses trabalhos ficam dependentes da assinatura do ataque realizado, e se há alguma alteração na dinâmica do ataque, estes trabalhos não terão sucesso na detecção e identificação. Além disso, quando há a coleta e inspeção do pacote, há uma invasão na privacidade do que está sendo trafegado na rede.

Ao propor, por esta Tese, a utilização de informação coletada na telemetria nativa da nuvem, a detecção e identificação dos ataques de negação de serviço e de intrusão pode ser realizado independente da assinatura do ataque, tornando-se agnóstico ao ataque. Dessa forma, se um determinado ataque alterar seu padrão de geração dos pacotes, a telemetria da nuvem continuará a registrar os efeitos dos ataques, possibilitando assim a detecção e identificação. Além do que, como não há uma análise microscópica, ou seja, não é realizado a inspeção dos pacotes, a privacidade dos dados trafegados está garantida quando se utiliza dos dados oriundos da telemetria.

Somado a isso, verifica-se que diversos trabalhos apresentados ao longo deste capítulo podem se tornar inviáveis pelo uso da criptografia, escondendo os cabeçalhos e carga útil necessários para a detecção e identificação dos ataques. Obviamente, quando os dados da telemetria é utilizado para detectar e identificar os ataques, não há problemas se o tráfego está ou não criptografado.

De acordo com a Tabela 3.1, apenas dois trabalhos utilizam informações da telemetria da infraestrutura da nuvem: Solanas *et al.* (SOLANAS; HERNANDEZ-CASTRO; DUTTA, 2014) e Nichols *et al.* (NICHOLS et al., 2016). Apesar de também utilizar as informações da telemetria da infraestrutura da nuvem, Nichols *et al.* (NICHOLS et al., 2016) verifica a existência de *malware* nas máquinas virtuais instanciadas no ambiente de nuvem, e Solanas *et al.* (SOLANAS; HERNANDEZ-CASTRO; DUTTA, 2014) verifica se as máquinas virtuais da sua própria

infraestrutura da nuvem são geradoras de um ataque de DDoS. Dessa forma, a presente Tese se diferencia desses trabalhos por duas razões: (i) detecta e identifica os ataques de DDoS e intrusão utilizando a telemetria da infraestrutura da nuvem; (ii) oferece esse serviço com a ótica da vítima, ou seja, verifica se a máquina virtual hospedada no ambiente de nuvem é a vítima dos ataques de DDoS e intrusão.

Em relação à mitigação, todas as formas de mitigação apresentadas no capítulo vão ao encontro do que o trabalho de Sabrine *et al.* (Sabrine; Abderrahmane; Fouzi, 2019) apresenta em seu estudo comparativo recente. Sabrine *et al.* afirma que existem dois grandes métodos para mitigar ataques em ambiente de nuvem: (i) mitigação baseada em métodos relacionados às Redes Definidas por *Software* (bloqueio, filtragem, redirecionamento de tráfego, etc.) e as vantagens da visão centralizada da rede SDN; (ii) mitigação baseada em métodos de dimensionamento de recursos computacionais (CPU, Memória, disco, etc.). O que corrobora ao objetivo desta Tese em relação à mitigação, pois a proposta ora apresentada é de utilizar recursos disponíveis pela própria infraestrutura da nuvem para realizar a mitigação. Recursos estes criados para outros fins, mas que podem ser utilizados para mitigar ataques. Por exemplo, o dimensionamento automático, o encadeamento de funções de rede, a instanciação de novas máquinas, contendo um *Deep Packet Inspector* ou até uma *Honeypot*. Todas essas funcionalidades, que são nativas, tornam a ter um novo uso, relacionado a segurança do ambiente de nuvem.

No próximo capítulo será apresentado como é possível utilizar a telemetria, uma ferramenta nativa do ambiente de nuvem, como nova fronteira para a segurança de nuvens computacionais.

# Capítulo 4

## TELEMETRIA COMO NOVA FRONTEIRA PARA A SEGURANÇA DE NUVENS COMPUTACIONAIS

Como discutido no Capítulo 3, e de acordo com Amaral *et al.* (AMARAL et al., 2016) e com Boutaba *et al.* (BOUTABA et al., 2018), diversos trabalhos utilizam algoritmos de aprendizado de máquina para detectar/identificar ataques, mas em sua maioria utilizam como entrada para esses algoritmos campos do cabeçalho ou da carga útil dos pacotes de rede. Esta Tese também propõe a utilização de algoritmos de ML para detectar/identificar ataques de DDoS e de intrusão, mas tendo como entrada informações da telemetria nativa da infraestrutura da nuvem.

A telemetria da infraestrutura da nuvem contém informações coletadas das máquinas virtuais e dos servidores físicos que hospedam a nuvem. Essas informações podem ser relacionadas aos recursos computacionais, por exemplo, a utilização de CPU e de memória, dos servidores físicos onde a nuvem está hospedada, ou das máquinas virtuais disponibilizadas na infraestrutura. Por exemplo, é representado na Figura 4.1 um pequeno ambiente de nuvem, composto por três servidores físicos, duas redes virtuais e 2 máquinas virtuais em cada uma das redes. Verifica-se, ainda na Figura 4.1, a presença do sistema de telemetria. Esse sistema é responsável por coletar diversas métricas, tanto dos servidores, das redes e máquinas virtuais, e disponibilizar essa informação para outros serviços, com o de monitoramento e de faturamento. Salienta-se que todo ambiente de nuvem pode prover informação da telemetria.

Além disso, a telemetria do ambiente de nuvem é nativa, ou seja, não é necessária a inserção de nenhum *software* para realizar o monitoramento, diferente da estratégia da captura dos pacotes em que é necessário inserir um *software* coletor para ter a informação necessária. A telemetria já é utilizada normalmente para realizar um monitoramento dos recursos da infraestrutura da nuvem e do faturamento (*billing*) dos clientes, por exemplo. Dessa forma, a proposta deste trabalho é utilizar informações que já estão disponíveis para uma outra tarefa, a

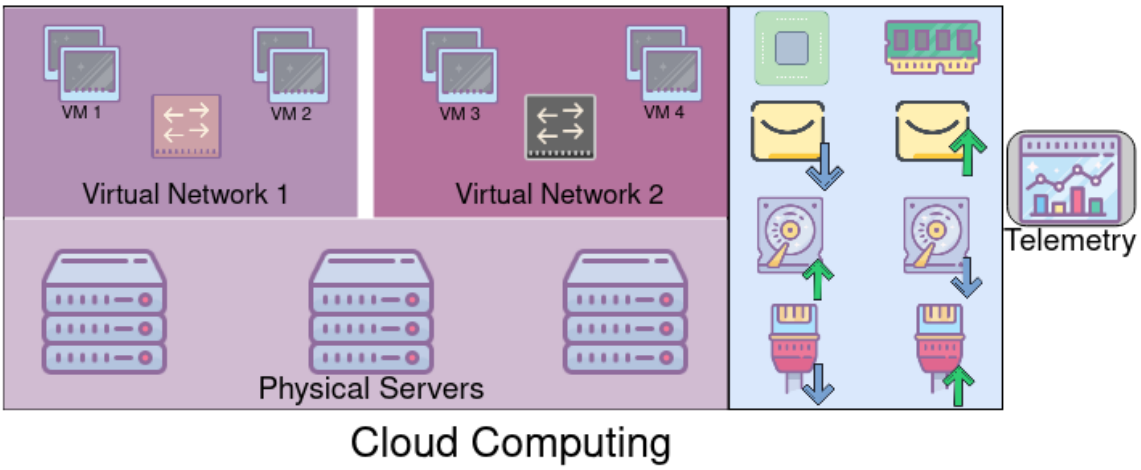



**Figura 4.1: Exemplo de um ambiente de nuvem e o sistema de telemetria**

de detecção e identificação de ataques de DDoS. A ideia é dar um novo uso para a telemetria do ambiente de nuvem. Tendo em consideração as hipóteses levantadas na Seção 1.3, o presente capítulo tem o objetivo de iniciar a discussão em relação à **Hipótese 1**.

Além disso, é disposta neste capítulo a Seção 4.1, onde são apresentados os fundamentos para que seja possível realizar a detecção/identificação de ataques, sejam eles de DDoS ou de intrusão. Em sequência, na Seção 4.2, são apresentados o cenário e os experimentos realizados, com o intuito de exibir um comparativo entre a atividade dos recursos computacionais no momento em que apenas o cliente legítimo está realizando requisições e em outro período em que apenas o atacante está presente. Na Seção 4.3, os resultados dessa comparação são expostos e discutidos e, em sucessão, é discorrido, na Seção 4.4, sobre as características importantes da utilização dos dados da telemetria nativa da infraestrutura da nuvem. No final do capítulo são apontadas as considerações finais do autor.

## 4.1 Formalização

A hipótese central da presente Tese é que diferentes tipos de tráfego destinado a máquinas virtuais, hospedadas em uma infraestrutura de computação em nuvem, geram também diferentes níveis de utilização em grupos de recursos computacionais distintos. Desse modo, é exposta a primeira definição dessa Tese:

**Definição 1** *Para todo ataque, existe um recurso computacional y, tal que se x é um ataque, então gera um padrão anômalo em y.*

Partindo da **Definição 1**, pode-se realizar uma especificação dessa definição, utilizando-

se da Lógica Proposicional e de Predicados. Considerando o domínio do discurso Ataques e Recursos Computacionais, pode-se definir dois predicados:

$A(x)$: "x é um ataque";

$G(x,y)$: "x gera utilização em y".

Por consequência, apresentamos a proposição $P$, que fundamenta a Tese:

$$P : \forall x : \exists y : (A(x) \rightarrow (G(x,y))$$

Portanto, se $x$ é um ataque, então há um recurso computacional $y$ sendo acionado por esse ataque. Isso vale para todos os ataques no domínio do discurso (representado pelo quantificador universal $\forall$)[1]. Da mesma maneira, existe um ou um grupo (representado pelo quantificador existencial $\exists$)[2] de recursos computacionais que sinalizam um uso por parte desse ataque $x$. Por exemplo, o ataque GET_Flood, que é um elemento que pertence ao domínio do discurso dos ataques, visa atingir o serviço *web*. A hipótese desta Tese se baseia na possibilidade de que esse ataque irá atingir pelo menos um recurso computacional monitorado pelo serviço de telemetria do ambiente de nuvem ($G(x,y)$) e que pertence ao domínio do discurso dos recursos computacionais.

Sendo assim, a hipótese do presente estudo é que, a partir da observação dos recursos computacionais, é possível detectar e identificar o ataque realizado. Por outro lado, essa análise do conjunto de recursos computacionais que estão sendo utilizados pelo ataque pode não ser trivial, pois a alteração no recurso pode ser sutil, ou pode ser necessária a inserção (ou exclusão) de um recurso em um subconjunto, ou até se a análise for feita pelo administrador, esta pode ser falha. Logo, é recomendado que essa análise dos conjuntos de recursos computacionais seja uma análise fundamentada, através de algoritmos de aprendizado de máquina. Nesse contexto, apresenta-se a segunda definição desta Tese:

**Definição 2** *Para todo recurso computacional, monitorado pela telemetria nativa da nuvem e a proposição P seja verdadeira, existe um algoritmo de aprendizado de máquina que pode detectar ou também identificar o ataque.*

[1] O quantificador universal $\forall$ tem valor verdadeiro quando todos os elementos no domínio do discurso também forem verdadeiros. Ou seja, não há um Contra-Exemplo no domínio do discurso que torne esse quantificador falso (FILHO, 2002).

[2] O quantificador existencial $\exists$ tem valor verdadeiro se pelo menos um elemento no domínio do discurso for verdadeiro. Esse elemento (ou grupo de elementos) é chamado de Testemunha (FILHO, 2002).

Diante da **Definição 2**, e ampliando o domínio do discurso para Algoritmos de Aprendizado de Máquina, pode-se definir mais três predicados:

$R(y)$: "o recurso $y$ é monitorado pela telemetria nativa da nuvem";

$D(z,x)$: "o algoritmo de aprendizado de máquina $z$ detecta o ataque $x$";

$I(z,x)$: "o algoritmo de aprendizado de máquina $z$ identifica o ataque $x$".

Dessa forma, em função da proposição $P$, tem-se a proposição $Q$, que complementa esta Tese:

$$Q : \forall y : ((R(y) \wedge P) \rightarrow \exists z : (D(z,x) \vee I(z,x)))$$

Portanto, se todos os recursos computacionais são monitorados pela telemetria e a proposição $P$ é verdadeira, então existe um ou um grupo de algoritmos de aprendizado de máquina capazes de realizar a classificação de que está acontecendo um ataque. Além disso, esse algoritmo de aprendizado de máquina pode ser eficaz também para identificar qual o ataque está sendo realizado.

A **Definição 1**, e consequentemente a **Definição 2**, podem considerar clientes legítimos ao invés de ataques. Neste contexto, consideram-se clientes legítimos as requisições enviadas com o intuito de serem atendidas, mas que não geram indisponibilidade do serviço por si só. Os clientes dos sistemas computacionais têm o seu próprio padrão de uso, gerando um nível de utilização em recursos computacionais distintos.

Em resumo, esta Tese se fundamenta no fato de que ataques e clientes geram graus de utilização distintos nos recursos computacionais. Os recursos e os graus distintos de utilização que podem ser caracterizados por algoritmos de aprendizado de máquina, sendo possível realizar a detecção e a identificação desses ataques.

### 4.1.1 Exemplificação

A Figura 4.2 apresenta, de forma gráfica, a hipótese de se utilizar a telemetria da computação em nuvem para detectar/identificar ataques, visto que, à princípio, cada ataque afeta diferentes subconjuntos de recursos computacionais. Considere, na Figura 4.2, que os recursos computacionais monitorados fazem parte do conjunto $R$, que é o círculo em branco. Os

recursos computacionais mais utilizados durante o tráfego de clientes legítimos estão sombreados pelo polígono em azul e fazem parte do subconjunto $R_a$. Da mesma forma, os recursos computacionais mais utilizados durante o Ataque 1 estão sombreados pelo polígono em rosa, representado pelo subconjunto $R_b$. Por fim, o conjunto de recursos computacionais utilizados durante o Ataque 2 está sombreado pelo polígono em laranja e é representado pelo subconjunto $R_c$. Esse reconhecimento e a separação em subconjuntos de recursos computacionais são realizados pelos algoritmos de aprendizado de máquina.

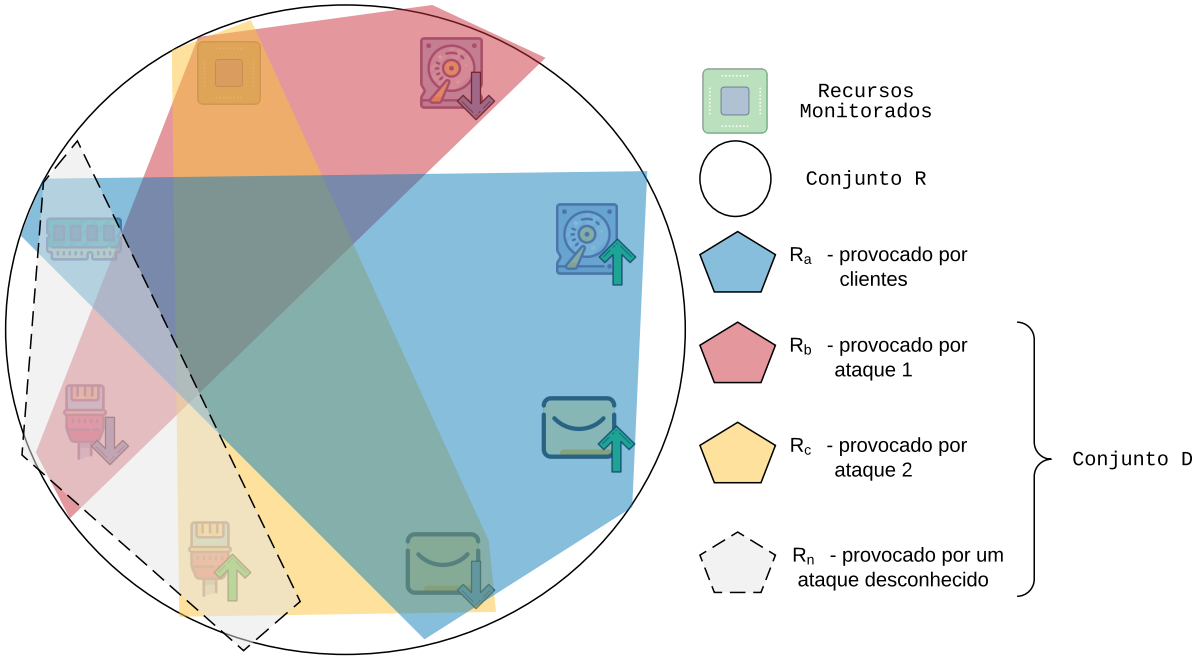



**Figura 4.2: Exemplo de utilização de diversos subconjuntos de recursos computacionais durante ataques distintos**

Nesse sentido, se forem identificados os subconjuntos de recursos computacionais, é possível detectar e identificar o ataque que está sendo realizado. Além disso, observando ainda a Figura 4.2, essa abordagem possibilita a detecção de ataques em que o sistema não conhece previamente. Por exemplo, na Figura 4.2, ocorre um aumento de utilização da memória RAM, na quantidade de Bytes enviados e recebidos pela interface de rede. Consequentemente, analisando esse subconjunto de recursos computacionais $R_n$ (dispostos na figura com sombreamento do polígono em branco), o sistema pode inferir que há um comportamento anormal, e que pode caracterizar um ataque não conhecido previamente. Assim, o sistema conseguirá detectar o ataque, mas, por não conhecer o seu tipo, não será possível realizar a sua identificação.

Como há diversos recursos computacionais que são monitorados pelos sistema de telemetria da infraestrutura da nuvem, e da mesma maneira pode haver diversos subconjuntos de recursos computacionais, é proposta a utilização de algoritmos de aprendizado de máquina para realizar

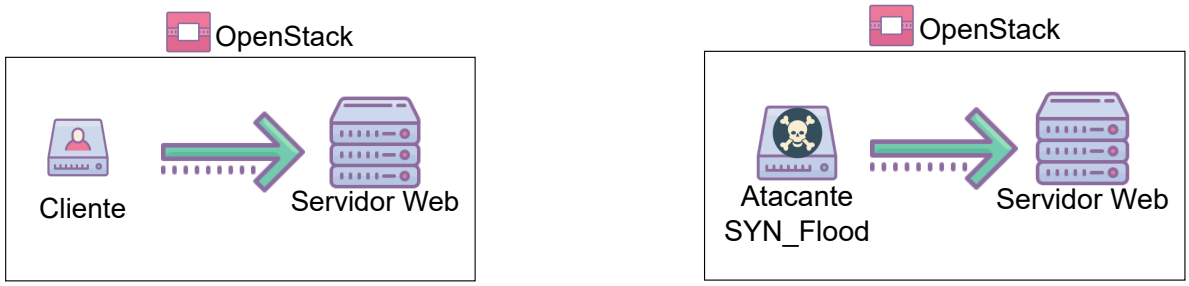
a detecção e identificação dos ataques. Para complementar a proposta desta Tese com a detecção/identificação desse ataque, uma medida de mitigação pode ser acionada. Essa mitigação utiliza ferramentas da própria infraestrutura da nuvem, como o dimensionamento automático, para diminuir ou eliminar os efeitos dos ataques.

## 4.2 Cenário de experimento e geração do dataset de comparação entre cliente e atacante

Para verificar a hipótese de que tráfego diversos, de clientes e de um ataque, por exemplo, provocam níveis distintos de utilizações em diferentes subconjuntos de recursos computacionais, foi criado um cenário com três máquinas virtuais no OpenStack. Uma VM com o servidor *web* Apache2, uma outra VM, para realizar requisições de clientes legítimos, e uma terceira VM, para realizar o ataque de DDoS. Nesse cenário, foram realizados dois experimentos: o primeiro apenas cliente legítimo realiza requisições, e o segundo experimento apenas o atacante realiza as requisições, sendo ambos os experimentos com duração de 30 minutos. No primeiro experimento, os clientes legítimos realizaram requisições HTTP GET, sem a presença de um ataque, enviando em uma taxa suficiente para consumir 100% de CPU do servidor *web*, mas sem gerar indisponibilidade do serviço. Neste cenário foi utilizada a ferramenta Siege[3], que gera requisições HTTP GET legítimas, simulando clientes legítimos.

A execução das requisições de clientes legítimos teve duração de 30 minutos, e os dados da telemetria, coletados pelo Ceilometer, foram guardados em um *dataset*. Finalizada a etapa dos clientes legítimos, foi realizado o segundo experimento contendo apenas o ataque SYN_-Flood isolado, sem a presença de clientes legítimos. Este segundo experimento foi feito com a ferramenta Hping3[4], que serve para gerar ataques de DDoS, entre os quais o SYN_Flood. Esse experimento teve duração total de 30 minutos, mas divididos em três etapas, de 10 minutos cada: nos primeiros 10 minutos, o ataque foi realizado em taxa baixa de geração das requisições, ou seja, a cada 300 milissegundos o ataque envia uma nova requisição. Na segunda etapa, o ataque foi gerado com um intervalo de 250 milissegundos. E na última etapa, o ataque foi gerado com a máxima taxa que o *host* atacante poderia gerar causando indisponibilidade no serviço *web*. Na geração do tráfego dos clientes e dos atacantes, os valores foram escolhidos empiricamente, com o intuito de comparar a utilização dos recursos computacionais entre cliente e atacante. Na Figura 4.3 é apresentado os cenários criados, sendo a Figura 4.3(a) o experimento em que apenas cliente realiza requisições, e a Figura 4.3(b) o experimento do ataque SYN_Flood.

[3]Página web: <https://www.joedog.org/siege-home/>. Acessado em 18 de agosto de 2019.
[4]Página web: <https://linux.die.net/man/8/hping3>. Acessado em 18 de agosto de 2019.




(a) Apenas cliente gerando requisições

(b) Apenas atacante gerando requisições

**Figura 4.3: Cenários de geração do *dataset* de comparação entre cliente e atacante**

Nesse experimento, os dados também foram coletados da telemetria e guardados em um segundo *dataset*. Em ambos os experimentos, o Ceilometer armazenou as métricas a cada 5 segundos, totalizando assim 360 amostras durante os 30 minutos de cada experimento. Esse intervalo de amostragem foi escolhido por ser um valor em que o consumo de recursos físicos por parte do Ceilometer não é tão elevado e mantém uma precisão da métrica coletada. O objetivo dos dois experimentos é verificar a influência dos clientes legítimos em alta taxa e do ataque de DDoS SYN_Flood nas métricas disponíveis no serviço de telemetria, e, dessa forma, verificar se há níveis de utilização diferente em subconjuntos de recursos computacionais, criando assim diferentes subconjuntos $R$, como discutido na Seção 4.1.

É importante destacar que não se tem notícia de *datasets* públicos disponíveis com a telemetria da infraestrutura da nuvem durante os ataques. Por isso, para a verificação da presente Tese é necessária a criação desses *datasets* com as informações da telemetria. Sendo assim, pode-se contabilizar uma contribuição desta Tese a disponibilização destes *datasets*[5].

## 4.3 Resultados comparativos

Após a realização e a geração dos *datasets* com experimentos de apenas cliente e de apenas atacante, os resultados foram inseridos em gráficos para ter uma comparação visual entre os recursos monitorados. A título de exemplificação, das 58 métricas que o Ceilometer pode resgatar, 8 foram comparadas e apresentadas nos gráficos a seguir. Foram escolhidas as seguintes métricas: utilização de CPU e Memória, quantidade de requisições de leitura e de escrita no disco, quantidade de Bytes e pacotes enviados e recebidos da interface de rede. As comparações para cada métrica estão apresentadas nas Figuras: 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10 e 4.11. Nestas, as linhas em azul, representam os experimentos realizados apenas com clientes legítimos, e as linhas em vermelho representam o ataque de DDoS SYN_Flood, já a linha tracejada cinza

[5]Página web: <https://jhenriquecorrea.github.io/datasets/>. Acessado em 20 de junho de 2020.

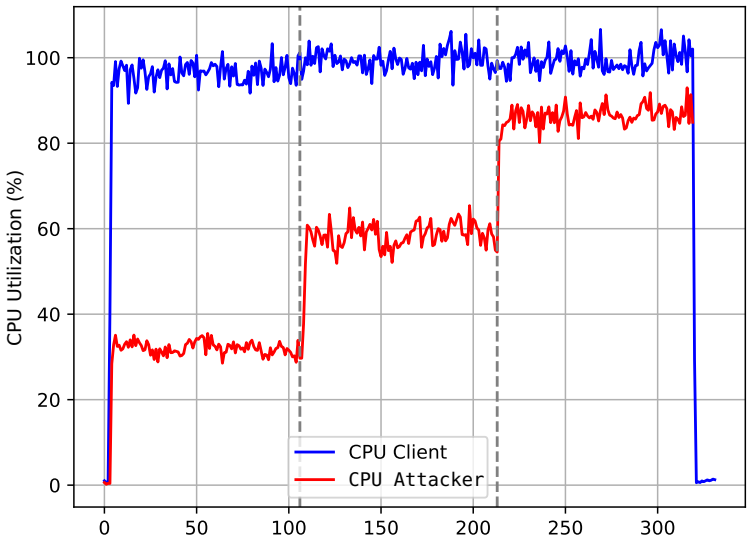



**Figura 4.4: Comparação entre a utilização de CPU nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

na vertical é a linha que separa cada etapa do ataque.

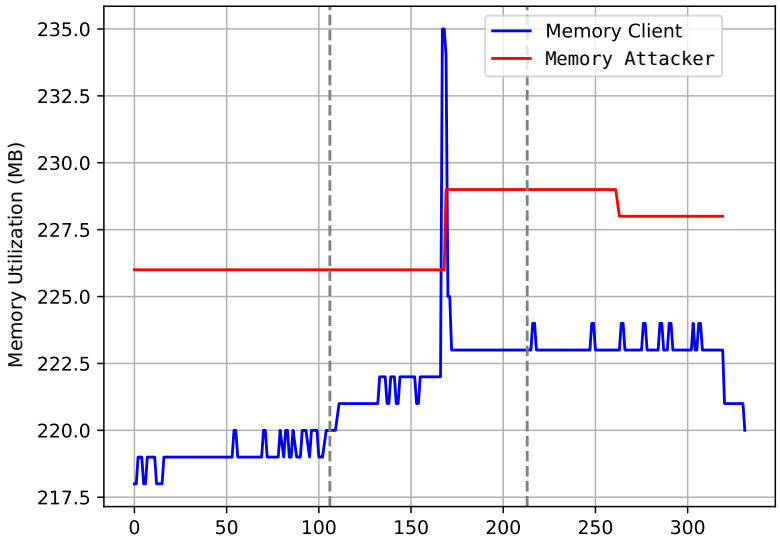



**Figura 4.5: Comparação entre a utilização de memória nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

A Figura 4.4 apresenta a utilização de CPU do servidor *web* nos dois experimentos. Como as requisições dos clientes estão sendo geradas na capacidade máxima do servidor *web*, verifica-se uma utilização de CPU próxima a 100%. Em relação ao cenário de ataque, na primeira etapa a utilização de CPU, está próximo a 35%, na segunda etapa, está em torno de 60% e, por fim, na última etapa, está em torno de 90% de utilização. O comportamento do recurso de CPU nesses experimentos foram os esperados, visto que para o tráfego dos clientes, a intenção é

gerar requisições que ocupem a CPU em 100%. Por outro lado, percebe-se os níveis diferentes dos ataques também nesse consumo de CPU, e se houver inserção de mais atacantes o servidor *web* estará totalmente indisponível.

A Figura 4.5 mostra a quantidade de memória utilizada pelo servidor *web*. Pode-se notar que não há variação significativa na utilização da memória pelo servidor *web* em ambos os experimentos. Esse resultado pode ser relacionado à forma como o servidor *web* realiza o gerenciamento de memória. O Apache implementa um sistema para evitar o consumo de toda a memória disponível no servidor, e esse sistema utiliza apenas a quantidade de memória solicitada pelo Apache no momento da inicialização do serviço. Então, o Apache realiza esse esquema precisamente para evitar que a memória do servidor seja degradada quando há um ataque do SYN_Flood. Dessa forma, vale salientar que se for utilizado outro servidor *web* esse comportamento da memória pode se alterar.

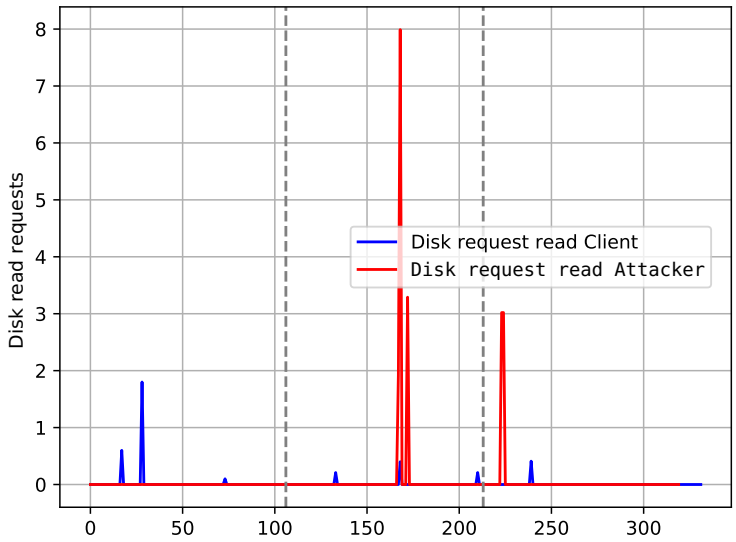



**Figura 4.6: Comparação entre a quantidade de requisições de leitura no disco nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

A Figura 4.6 apresenta o número de requisições de leitura no disco em ambos os experimentos. Nesse recurso, não há variação significativa em ambos os casos e, aparentemente, essa não é uma boa métrica para distinguir clientes legítimos e atacantes.

Ao contrário da quantidade de requisições de leitura no disco, as requisições de escrita no disco, apresentada na Figura 4.7, mostra grande diferença entre os dois experimentos. Naquele com apenas clientes legítimos, é verificada grande variabilidade na utilização do recurso, enquanto que no experimento com apenas ataque de DDoS o recurso computacional apresenta baixa variabilidade. Essa diferença pode ser explicada pelo fato que o servidor web escreve

em um arquivo de *log* todas as páginas requisitadas com sucesso. No ataque SYN_Flood, não há requisições da página, apenas o início da conexão TCP. Devido a essa diferença entre os cenários, esse recurso computacional pode ser um notável indicador para diferenciar clientes legítimos neste tipo de ataque.

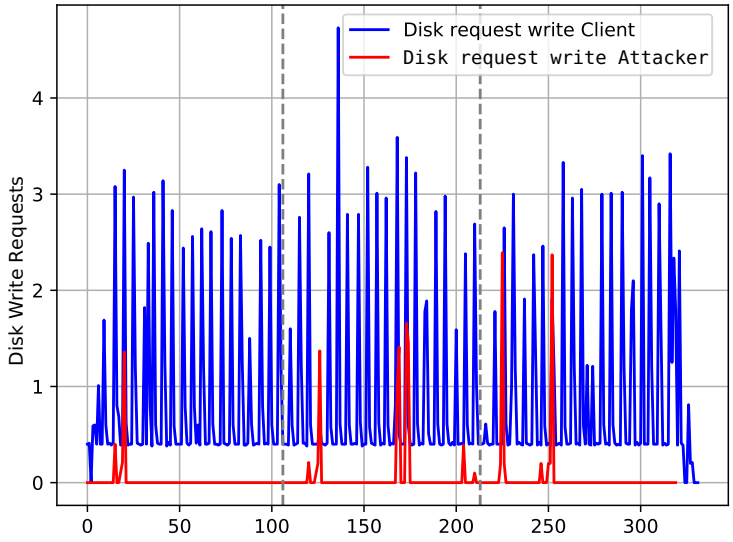



**Figura 4.7: Comparação entre a quantidade de requisições de escrita no disco nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

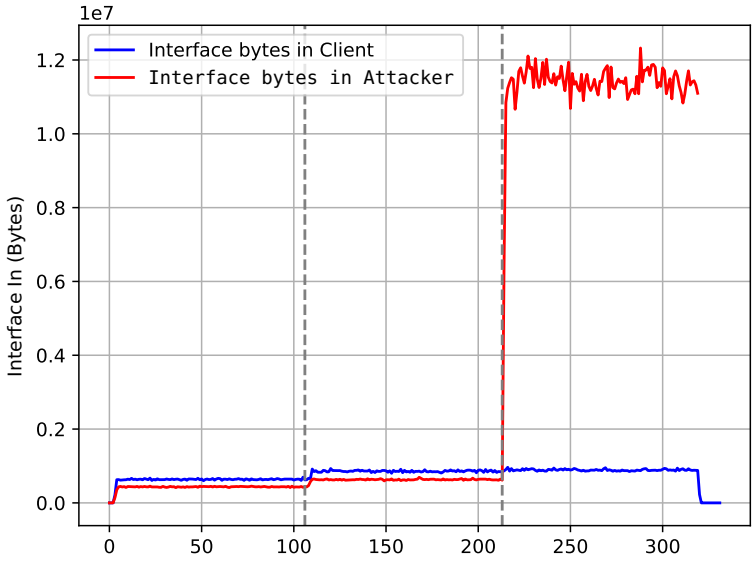



**Figura 4.8: Comparação entre a quantidade de Bytes recebidos na interface de rede nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

O quinto recurso computacional, apresentado na Figura 4.8, é a quantidade de Bytes recebidos na interface de rede do servidor web. Como esperado, um grande número de Bytes pode

ser verificado durante o experimento com apenas o ataque de DDoS, especialmente na terceira etapa em que o ataque é gerado com a máxima capacidade. A característica do ataque, de gerar várias requisições TCP com a *flag* SYN ativada é demonstrada pela quantidade de Bytes que o servidor *web* recebe. Diante dessa marcante características vários trabalhos apresentados no Capítulo 3 utilizam, de alguma forma, essa métrica.

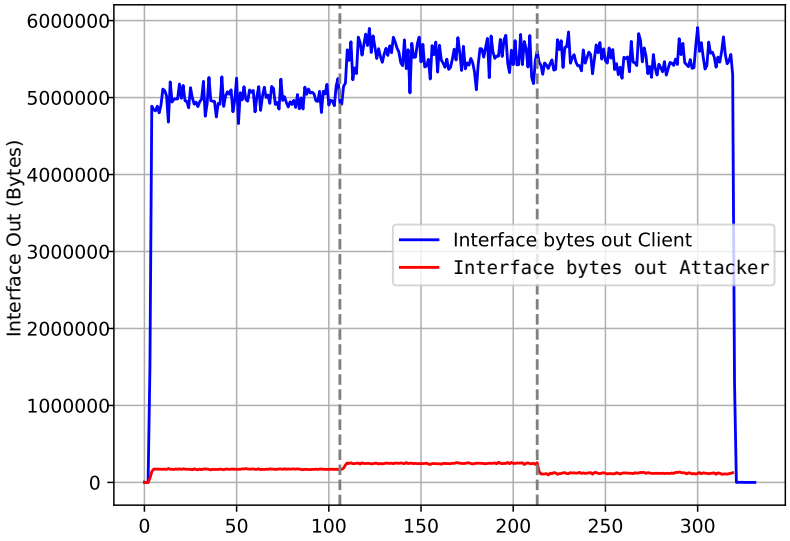



**Figura 4.9: Comparação entre a quantidade de Bytes enviados pela interface de rede nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

A Figura 4.9 apresenta a quantidade de Bytes enviados na interface de rede do servidor web. No experimento com apenas cliente legítimo, em que há também a resposta do servidor web, o número de Bytes enviados pela interface é expressiva. Por outro lado, durante o ataque SYN_Flood, o servidor web responde para cada conexão apenas um pacote com as *flags* SYN e ACK ativadas. Dessa forma, a quantidade de Bytes enviados pela interface de rede é muito menor. Essa métrica é um indicativo de que atacante e cliente geram níveis de utilização em recursos computacionais diferentes.

A quantidade de pacotes recebidos na interface de rede do servidor web é apresentada na Figura 4.10. O comportamento deste recurso é bastante similar ao número de Bytes recebidos na interface (Figura 4.8). Em contrapartida, a quantidade de pacotes enviados pela interface (Figura 4.11) é bastante diferente da quantidade de Bytes enviados pela interface (Figura 4.9), especialmente durante o experimento de ataque. Esse recurso computacional mostra um número crescente de pacotes enviados pela interface durante a primeira e a segunda etapa do experimento de ataque do SYN_Flood. Quando o ataque está em sua terceira etapa, gerada na capacidade total do atacante, o servidor web não consegue responder a todas as solicitações, e

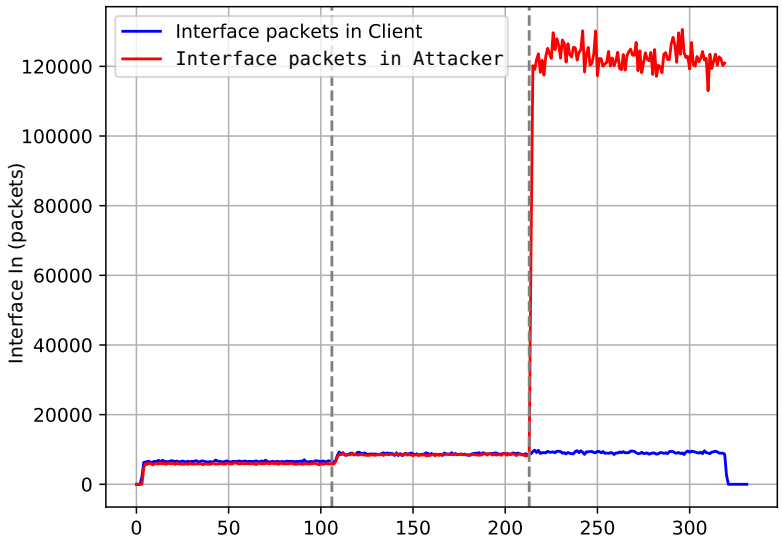



**Figura 4.10: Comparação entre a quantidade de pacotes recebidos na interface de rede nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

o número de pacotes enviados diminui.

Sendo assim, como apresentado, diversos recursos computacionais desenvolvem comportamentos diferentes sob ataque e sob utilização intensa de clientes legítimos, tornando possível realizar a distinção entre cliente legítimo e atacante. Essa distinção é viável de ser realizada utilizando algoritmos de aprendizado de máquina, como discutido na Seção 4.1.

## 4.4 Destaques e aspectos práticos

Como observação final, esta seção destaca as características importantes da utilização dos dados da telemetria nativa da infraestrutura da nuvem. Conforme será discutido no Capítulo 3, há vários trabalhos relacionados que visam detectar e identificar ataques usando algoritmos de aprendizado de máquina. As principais vantagens desta Tese, em relação à abordagem da literatura tradicional, podem ser resumidas em três pontos: o método de coleta de dados, o volume de dados a serem processados e a privacidade desses dados.

Em relação ao método de coleta de dados, as abordagens tradicionais que utilizam a inspeção de pacotes de rede tem uma escolha crítica a ser feita: "Onde instalar o coletor de dados da rede?". Então, o administrador da rede deve conhecer esse local e todos os pacotes devem ser coletados. Dessa forma, quando os pacotes são enviados para análise, uma latência maior pode ser experimentada pelos usuários. Esta Tese capta dados da telemetria já existente em cada serviço de computação em nuvem. O único parâmetro que deve ser selecionado é a taxa

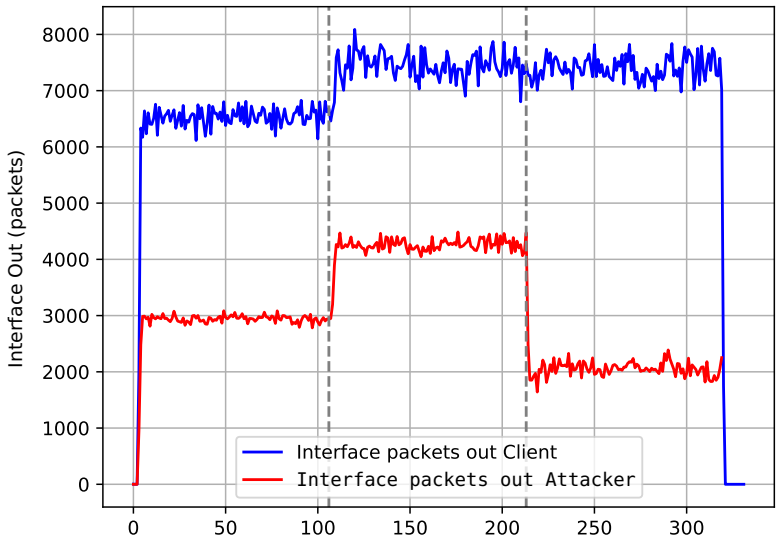



**Figura 4.11: Comparação entre a quantidade de pacotes enviados pela interface de rede nos dois experimentos: apenas clientes legítimos (linha azul) e apenas ataque de DDoS (linha vermelha) (CORRÊA et al., 2019a)**

de amostragem dos recursos monitorados, mas não há relação entre essa seleção e a latência que pode ser experimentada pelos usuários dos serviços da infraestrutura da nuvem.

Em relação ao volume de dados a serem coletados e analisados, nas abordagens tradicionais, um volume muito grande de dados deve ser capturado e analisado para as tarefas de detecção e identificação de ataques. A análise de cada pacote de rede pode interferir no desempenho da rede, diminuindo a Qualidade de Experiência (QoE) dos usuários finais. Se a intenção do administrador é realizar esta análise somente para serviços específicos que são hospedados, então um filtro deve ser instalado no local de captura, e isto levará a uma grande gama de configurações, pois precisará de uma regra para cada serviço a ser monitorado. Na abordagem proposta por esta Tese, o volume de dados a serem coletados e analisados é determinístico. Os dados de telemetria serão coletados para cada servidor instanciado na infraestrutura da nuvem e terá seu volume de dados com tamanho relacionado ao tempo de coleta e não pelo volume de tráfego do servidor. É importante destacar aqui que esses dados já são coletados pelo serviço de telemetria para fins de gerenciamento, e esta Tese está fornecendo um novo uso a eles.

Em relação à privacidade dos dados, há duas preocupações principais: o uso de criptografia por alguns serviços ou usuários e a inspeção do conteúdo dos pacotes no *data center*. Se a criptografia for realizada em camadas superiores, como SSL/TLS, o conteúdo dos pacotes nas camadas mais altas não estará disponível. Esse problema pode ser agravado se a criptografia for realizada em camadas inferiores, como as transações que usam IPSec. Assim sendo, qualquer protocolo e *payload* acima da camada de rede não estarão disponíveis para serem usados para

a detecção e identificação de ataques. Evidentemente, esse problema pode ser contornado pela realização da captura logo após a remoção da criptografia, mas isso interferirá diretamente na posição da captura dos pacotes. É notável o fato que a proposta desta Tese é agnóstica aos protocolos e serviços utilizados na infraestrutura da nuvem.

Na detecção de ataques, as ferramentas IDS utilizam informações do cabeçalho ou da carga útil de cada pacote da rede para elaborar regras correspondentes baseadas na assinatura (ou comportamento) desses ataques. Essas regras são geralmente estáticas e desenvolvidas para detectar usando a característica específica de cada ataque. Como resultado, um grande número de correspondências precisa ser feito para cada pacote, aumentando a latência da inspeção do pacote, pois o IDS estará realizando a verificação no meio do caminho entre o cliente e o servidor. Por outro lado, o uso de algoritmos de aprendizado de máquina, como proposto nesta Tese, é independente do próprio tráfego, já que as métricas utilizadas para realizar a detecção/identificação estão disponíveis no serviço de telemetria. Além disso, os algoritmos de ML utilizam a análise multivariada para realizar uma tarefa, que é potencialmente mais poderosa do que uma análise do pacote de rede.

Como exemplo, as Figuras 4.12 e 4.13 apresentam uma topologia de *data center* em Fat-tree (MYSORE et al., 2009), em que a primeira utiliza o monitoramento de recursos da forma tradicional, e a segunda utiliza o serviço de telemetria da infraestrutura da nuvem no *data center*.

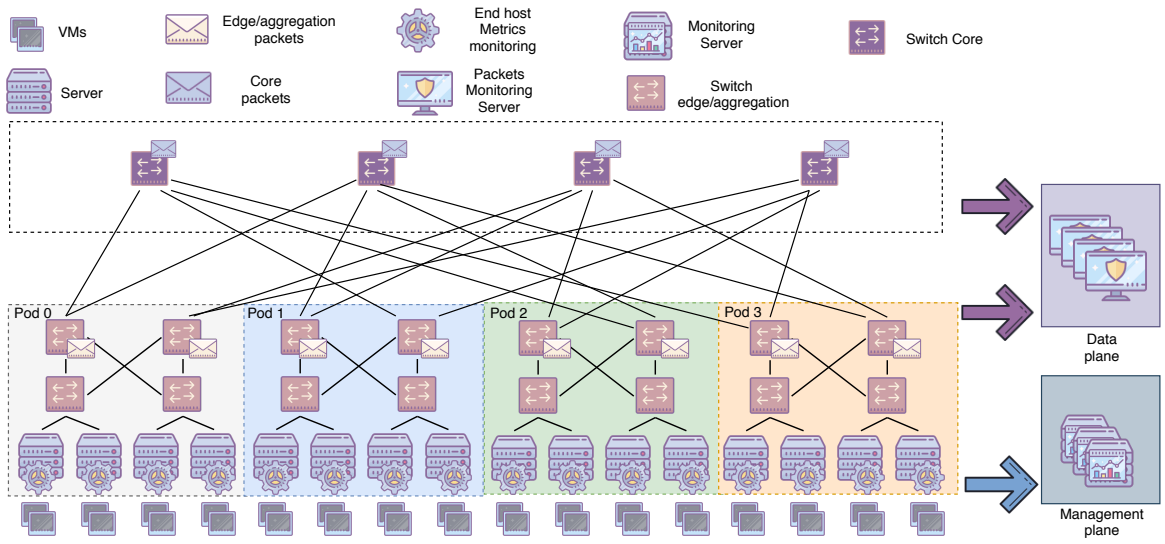



**Figura 4.12: Monitoramento Tradicional em um exemplo Fat-tree (CORRÊA et al., 2021)**

Aparece na Figura 4.12 o maior problema relacionado ao uso do IDS e coleta dos dados de rede: onde colocar os coletores de pacotes nos dispositivos de rede no plano de dados. O administrador da rede tem de fazer múltiplas escolhas para instalar esses mecanismos de coleta

para realizá-la. A primeira solução estaria nos próprios servidores, ou seja, em 16 locais diferentes, gerando uma sobrecarga para realizar essa administração. Se o administrador optar por coletar nos *switches* de agregação, em cada *pod* da topologia o administrador deverá colocar a monitoração em ambos os *switches* (correspondências em bege na Figura 4.12). Finalmente, se o administrador de rede preferir coletar os pacotes nos *switches* de núcleo, para que possa ter uma visão global do *data center*, deverá monitorar os quatro *switches* de núcleo (correspondências em roxo na Figura 4.12). Além disso, o sistema de coleta deve ter mecanismos para não interferir e gerar gargalo para o *data center*, copiando os pacotes a uma velocidade que não gere latência extra para os usuários da nuvem.

Na Figura 4.13, está representada a abordagem proposta nesta Tese, na qual a própria infraestrutura da nuvem realiza o monitoramento dos recursos computacionais. Ou seja, todos os *hosts* físicos e máquinas virtuais são monitorados por serviços de telemetria, como o Ceilometer. Assim, para obter os dados usados como entrada nos algoritmos de aprendizado de máquina, basta solicitar essas informações ao serviço de telemetria, não se importando com o quê e como os recursos existem no próprio ambiente de nuvem. Além disso, não gera nenhuma preocupação com a privacidade dos dados dos usuários, pois o serviço de telemetria recupera informações diretamente da infraestrutura da nuvem, sem nenhuma inspeção em arquivos ou pacotes de rede. Por fim, o ambiente de nuvem em si fornece uma visão global do *data center*, tornando a abordagem deste trabalho (usando telemetria) mais fácil de gerenciar do que a abordagem tradicional (usando IDS).

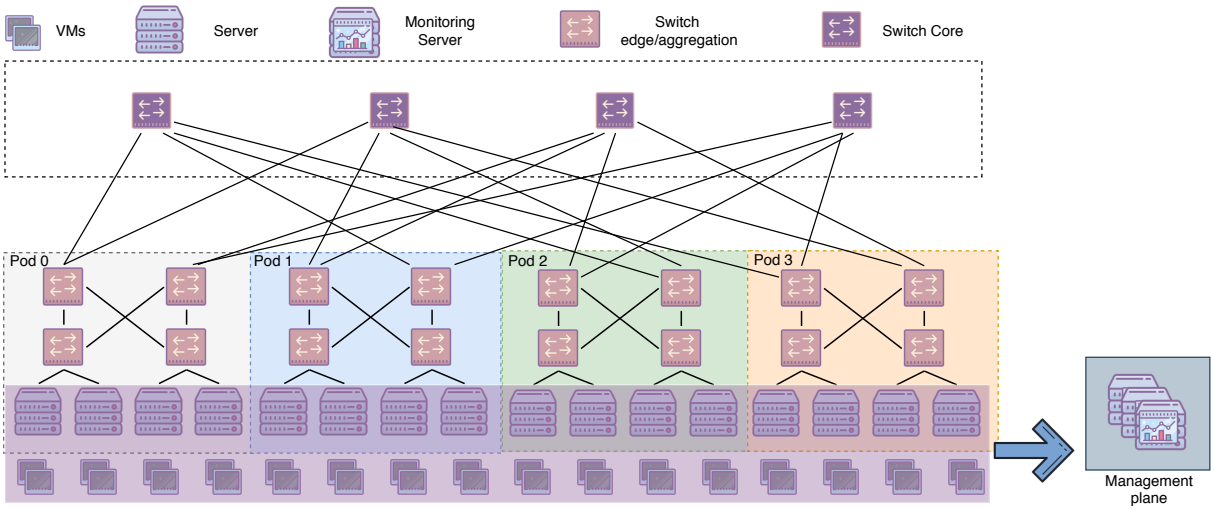



**Figura 4.13: Telemetria da infraestrutura de Nuvem em um exemplo Fat-tree (CORRÊA et al., 2021)**

## 4.5 Considerações do Capítulo

Este capítulo apresentou uma formalização para a utilização de dados da telemetria nativa da infraestrutura da nuvem para realizar a detecção e identificação de ataques, visto que cada ataque gera utilização em um subconjunto específico de recursos computacionais. Se esse subconjunto for caracterizado e identificado por meio do monitoramento da nuvem, é possível detectar/identificar esse ataque.

Posteriormente, foram realizados dois experimentos, um com tráfego apenas de cliente legítimo e outro de apenas atacante. Em ambos os experimentos, os dados dos recursos computacionais da vítima foram coletados. Então, foi realizada uma comparação entre os dois experimentos, mostrando que há uma clara utilização em recursos distintos quando há apenas cliente legítimo ou apenas atacante, viabilizando assim a proposta apresentada anteriormente. Por último, foi realizada a discussão das importantes vantagens da abordagem desta Tese, bem como os aspectos práticos.

No próximo capítulo, são criados cenários, com tráfego de clientes e atacantes, para que algoritmos de aprendizado de máquina, utilizando informação da telemetria nativa da nuvem, realizem a detecção e identificação dos ataques, de DDoS e de intrusão.

# Capítulo 5

## DETECÇÃO E IDENTIFICAÇÃO DE ATAQUES USANDO ML SOBRE TELEMETRIA

Diante do que foi apresentado no Capítulo 4, este capítulo visa discutir a utilização de dados oriundos da telemetria nativa da computação em nuvem, combinado com algoritmos de aprendizado de máquina, para detectar e identificar ataques de negação de serviço e de intrusão, comprovando a **Hipótese 1** apresentada na Seção 1.3.

O objetivo é verificar se a utilização de dados da telemetria da infraestrutura da nuvem conseguem ter sucesso para realizar a detecção e identificação dos ataques em VMs hospedadas na infraestrutura da nuvem. No contexto deste capítulo, quando se trata de detecção, refere-se à habilidade de indicar a presença de algum comportamento anormal do sistema causado por um ataque de DDoS ou de intrusão. Por sua vez, o termo "identificação"se refere à capacidade de identificar qual tipo de ataque está sendo realizado. A fase de identificação complementa a fase de detecção e será fundamental para aplicar uma estratégia de mitigação focada em um tipo particular de ataques de DDoS, em vez de usar estratégias gerais de mitigação que podem impactar também os usuários legítimos (DANTAS; NIGAM; FONSECA, 2014; CORRÊA et al., 2019b).

A Figura 5.1 apresenta, de forma geral, a primeira etapa dos experimentos para a geração dos modelos de ML para a fase de detecção e identificação dos ataques. As métricas disponíveis na telemetria do OpenStack são coletadas durante a realização dos experimentos e guardadas em *datasets*. Então, são utilizados algoritmos de aprendizado de máquina para criar os modelos em cada fase da proposta: um modelo para a fase de detecção e outro para a fase de identificação.

De modo geral, a Figura 5.2 apresenta a segunda etapa, na qual serão utilizados os modelos de ML gerados anteriormente. Ou seja, a partir da telemetria do OpenStack, são coletadas as

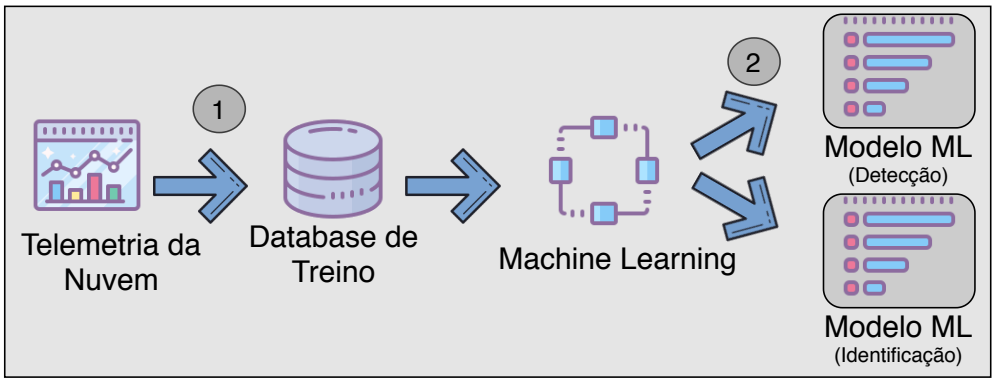



**Figura 5.1: Diagrama da geração dos modelos de ML**

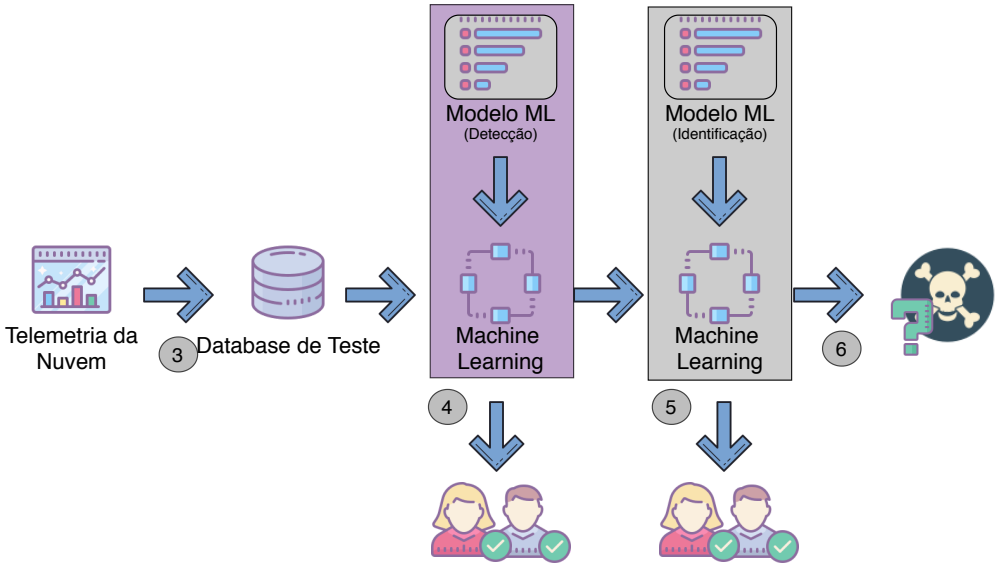



**Figura 5.2: Diagrama da utilização da telemetria para detectar e identificar ataques de DDoS e de intrusão**

métricas do serviço hospedado na infraestrutura da nuvem e que está recebendo requisições de clientes e atacantes. Criado esse *dataset*, são então realizadas as duas fases da proposta: a detecção do ataque e a identificação do tipo de ataque, que estão representadas na figura pelos passos 4 e 5, respectivamente. Para a fase de detecção, o modelo de ML tem duas possíveis saídas: a amostra é de um momento de funcionamento normal ou está acontecendo um ataque. Para o primeiro caso, não é realizada mais nenhuma ação. Mas, se a amostra for detectada como de ataque, então é ativada a segunda fase, a de identificação. Nesta, o modelo de ML verifica qual o tipo de ataque está acontecendo. Nesse ponto, o modelo de ML pode ainda tentar identificar um funcionamento normal que foi classificado erroneamente pela fase anterior.

O restante deste capítulo está disposto da seguinte forma: na Seção 5.1, são apresentados a metodologia e os resultados para os ataques de DDoS. Especificamente, na Seção 5.1.1, é apresentado o *setup* utilizado nos experimentos, como foram gerados os *datasets*, a seleção de

**Tabela 5.1: Datasets e Traces do tráfego do ataque de DDoS**

| Datasets | Treino[a] | Teste[a] | Zero-Day[a] |
|---|---|---|---|
| Duração | 5h47m | 16h | 5h |
| Utilização da CPU do Servidor Web (por clientes legítimos) | 50% / 100% | 50% / 100% | 50% / 100% |
| # VMs Cliente | 2/4 | 2/4 | 2/4 |
| # VMs Atacante | 2/4/6 | 2/4/6 | 2/4/6 |
| Intervalo do SYN_Flood por VM | 300 $\mu$s / 250 $\mu$s / flood | 300 $\mu$s / 250 $\mu$s / flood | - |
| Requisições GET_Flood por VM | 36000 | 36000 | - |
| Intervalo do PING_Flood por VM | - | - | 300 $\mu$s / 250 $\mu$s / flood |
| Regime de teste entre Cliente-Atacante | Misto | Misto | Misto |
| Tamanho do Dataset | 383.7 KB | 1.1 MB | 302.1 KB |
| Tamanho do Trace de rede | - | 218 GB | - |

[a] Disponíveis em https://jhenriquecorrea.github.io/datasets/

características e os algoritmos de aprendizado de máquina, bem como seus indicadores de desempenho. Na Seção 5.1.4, são apresentados e discutidos os resultados das fases de detecção e identificação. Além disso, é demonstrada uma comparação com o IDS Snort (PROJECT, 2020) e, por fim, a verificação da proposta com ataques que os modelos de ML não conhecem, simulando um *Zero-Day Attack*. Na Seção 5.2, é exposto o cenário, a metodologia e os resultados da proposta de utilizar dados da telemetria para detectar ataques de intrusão, especificamente o SQL_Injection. Por último, são apresentadas as considerações finais do capítulo.

## 5.1 Ataques de DDoS

### 5.1.1 Cenário de testes e geração dos datasets

A Tabela 5.1 sumariza os *datasets* utilizados nesta seção. Neles, há três instâncias diferentes geradas para diferentes propósitos: Treino, Teste e *Zero-Day*. Observa-se que foi coletado também, durante os experimentos do *dataset* de Teste, o *trace* dos pacotes do plano de dados, com a diferença, com ordem de magnitude 5, entre o *trace* e a telemetria, sendo esse um dos fatores que motivou a investigação de utilizar algoritmos de ML com a telemetria em vez de utilizar os *traces* de rede. É importante salientar que não se tem notícia de *datasets* públicos disponíveis com a telemetria da infraestrutura da nuvem durante os ataques, o que justifica a

necessidade da realização desses experimentos[1].

Os *datasets* de telemetria descritos na Tabela 5.1 foram todos coletados a cada 5 segundos. Esse intervalo de amostragem foi um compromisso empírico entre a precisão da métrica coletada e o uso de recursos computacionais onde os serviços estão hospedados para a elaboração dos modelos de ML. Intervalos de amostragem mais curtos podem permitir reações mais rápidas ao ataque de DDoS, mas, provavelmente, tornarão o sistema mais suscetível a picos instantâneos na utilização do servidor *web*. A quantidade de dados coletados evidentemente depende deste intervalo de amostragem da telemetria, bem como do tempo para o seu processamento.

Quatro máquinas virtuais clientes foram utilizadas para realizar requisições HTTP legítimas para o servidor *web*. Em cada máquina virtual, 300 clientes legítimos foram simulados, totalizando 1200 clientes neste *setup*. A ferramenta Siege[2] foi utilizada e realiza requisições GET para o servidor *web*. Nesse *setup*, cada conexão-cliente gera uma nova requisição GET em um tempo aleatório entre 0 e 3 segundos, durante todo o experimento. Dessa forma, diversos clientes legítimos estão requisitando serviço simultaneamente para o servidor *web*. Na Figura 5.3 são apresentadas as variações dos clientes gerados. Na Figura 5.3(a) apenas duas máquinas virtuais são utilizadas, simulando 600 clientes que visam gerar uma utilização média de 50% na CPU do servidor *web*. Enquanto que, na Figura 5.3(b), as quatro máquinas virtuais são utilizadas, totalizando 1200 clientes e esperando gerar 100% de utilização do servidor *web*.


(a) Cliente com 2 VMs (b) Cliente com 4 VMs

**Figura 5.3: Diferentes regimes de geração do tráfego cliente**

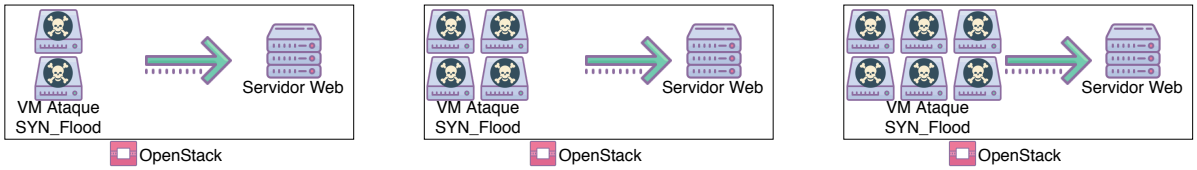
Para gerar o ataque de DDoS, seis máquinas virtuais são utilizadas, alternando entre os ataques SYN_Flood e GET_Flood. Na Figura 5.4 são apresentados as variações dos ataques gerados. Para gerar o ataque SYN_Flood, as máquinas virtuais utilizaram a ferramenta Hping3[3] em três níveis com diferentes intervalos entre as requisições maliciosas: 300 $\mu$s, 250 $\mu$s e o intervalo mínimo possível, o que gera uma taxa máxima de ataque. Cada nível de ataque é variado

[1] É importante destacar que o *replay* de *datasets* públicos de rede não é trivial, visto que diversas ferramentas geram os pacotes fora da ordem ou não obedecem o *timestamp*, tornando inviável realizar a repetição desses *datasets* (FIORINO et al., 2021).

[2] Página web: <https://www.joedog.org/siege-home/>. Acessado em 18 de agosto de 2019.

[3] Página web: <https://linux.die.net/man/8/hping3>. Acessado em 18 de agosto de 2019.

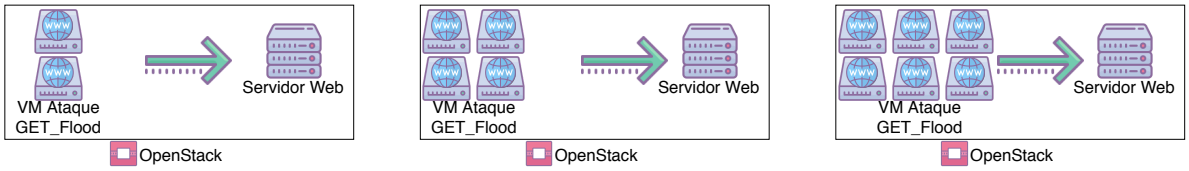
também com a quantidade de máquinas virtuais, representado na Figura 5.4(a) a utilização de duas máquinas virtuais, gerando as requisições maliciosas a cada 300 $\mu$s. Na Figura 5.4(b) as quatro máquinas virtuais geram o ataque SYN_Flood a cada 250 $\mu$s e, por fim, a Figura 5.4(c) apresenta a utilização de seis máquinas virtuais atacando o servidor *web*, com requisições sendo geradas no intervalo mínimo possível.




(a) Ataque SYN_Flood com 2 VMs

(b) Ataque SYN_Flood com 4 VMs

(c) Ataque SYN_Flood com 6 VMs




(d) Ataque GET_Flood com 2 VMs

(e) Ataque GET_Flood com 4 VMs

(f) Ataque GET_Flood com 6 VMs

**Figura 5.4: Diferentes regimes de geração do tráfego do ataque SYN_Flood e GET_Flood**

Para gerar o ataque GET_Flood, as 6 VMs utilizaram a ferramenta Goldeneye[4]. Cada uma dessas máquinas virtuais emula 200 atacantes diferentes, e cada atacante cria 180 conexões distintas que realizam requisições simultaneamente durante todo o experimento. Assim, para o ataque GET_Flood, cada VM gera 36.000 (200x180) conexões diferentes com o servidor *web* durante o experimento. Na Figura 5.4(d) o ataque GET_Flood é realizado por duas máquinas virtuais, enquanto, na Figura 5.4(e) são utilizadas quatro máquinas virtuais para gerar o ataque. Por fim, na Figura 5.4(f) seis máquinas virtuais são utilizadas para gerar o ataque GET_Flood. Essa variação de quantidade de máquinas virtuais, de clientes e de atacantes, é para gerar regimes de tráfego diferentes. Vale salientar que toda a geração do tráfego de cliente e atacante, na organização do experimento, é uma escolha empírica da carga. Todas as VMs, incluindo o servidor *web*, estão na mesma rede, conectados apenas por um *switch* virtual na infraestrutura da nuvem OpenStack. Esse cenário foi implementado em um servidor Intel Xeon com 32 GB de memória RAM, utilizando um ambiente de nuvem OpenStack (versão Rocky).

No contexto desta Tese, consideram-se clientes legítimos as requisições HTTP GET realizadas para o servidor *web* e que não tem o intuito de gerar indisponibilidade no servidor *web*. As requisições maliciosas ou de ataque significam pacotes com a *flag* TCP SYN para o ataque SYN_Flood, e mensagens HTTP GET para o ataque GET_Flood e geram indisponibilidade.

[4]Página web: <https://github.com/jseidl/GoldenEye>. Acessado em 18 de agosto de 2019.

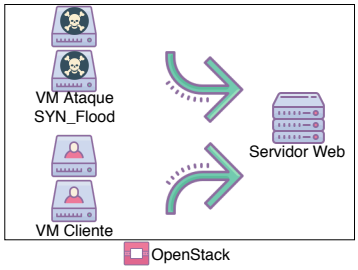
Dois experimentos foram realizados: o primeiro experimento para gerar o *dataset* de treino dos algoritmos de aprendizado de máquina, e o segundo para verificar se os modelos de ML criados poderiam detectar e identificar os tipos de ataques de DDoS. No *dataset* de treino, que tem como duração 5h47min, clientes legítimos realizam requisições para o servidor web de duas formas diferentes: (i) a realização das diversas requisições para o servidor *web* foram limitadas a capacidade de utilização da CPU do servidor *web* em 50% e (ii) a realização das diversas requisições ao servidor web não teve limitação, ocupando assim 100% da CPU do servidor *web*. No primeiro caso, foi escolhido essa porcentagem para representar uma utilização normal do servidor *web*. No segundo caso pode haver negação de serviço, mas é devido à sobrecarga de clientes legítimos em vez de ataques de DDoS. O objetivo desse experimento é confundir os modelos entre clientes legítimos e ataques maliciosos de DDoS, já que ambos causaram indisponibilidade ao servidor *web*. Por fim, destaca-se que a ocupação de 50% e 100% é uma média, sendo esta uma estimativa, visto que o tráfego cliente é gerado baseado em aleatoriedade.

Nesse *dataset* o ataque foi gerado com dois objetivos: (i) simular ataques de DDoS em alta taxa, mas não capaz de gerar indisponibilidade do serviço aos clientes legítimos, gerando apenas uma degradação do desempenho do servidor *web*; e (ii) simular ataques de DDoS em altíssima taxa, capaz de deixar o serviço indisponível para clientes legítimos.

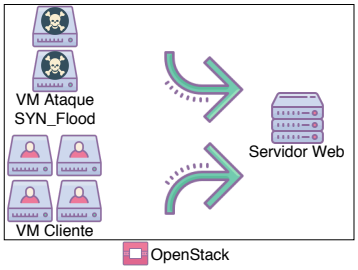
Para alcançar a variação do tráfego de cliente e de atacante, a quantidade de máquinas virtuais foram variadas. Por exemplo, para a geração das requisições de clientes legítimos, visando consumir 100% da CPU do servidor *web*, todas as VMs destinadas ao tráfego cliente foram utilizadas, como apresentado na Figura 5.3(b), enquanto que, para consumir apenas 50% da CPU do servidor *web*, apenas metade das VMs geradoras do tráfego cliente foram usadas, de acordo o que foi apresentado na Figura 5.3(a). Esse procedimento também foi utilizado na geração dos ataques de DDoS SYN_Flood e GET_Flood, como apresentado na Figura 5.4. Esse processo de variar a carga da geração do conjunto de dados do *dataset* visa evitar o viés no modelo de ML e confundir os algoritmos de aprendizado de máquina na detecção e identificação dos ataques de DDoS. Toda a variação dos cenários, com diferentes regimes de tráfego, visa abarcar as possibilidades de um cenário real.

Partindo dos cenários apresentados nas Figuras 5.3 e 5.4, diversas combinações desses cenários foram construídos, variando tanto o tráfego de cliente quanto de atacante. Na Figura 5.5, são apresentados os diferentes *setups* para a geração do ataque SYN_Flood, variando a quantidade de máquinas atacantes e também de clientes. Nas Figuras 5.5(a) e 5.5(b), foram utilizadas duas máquinas virtuais para gerar o ataque SYN_Flood, e houve uma variação da quantidade de máquinas virtuais de clientes legítimos. Nas Figuras 5.5(c) e 5.5(d), o nível do ataque aumenta,

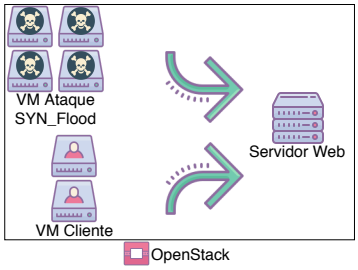
em relação ao cenário descrito anteriormente, com a utilização de quatro máquinas virtuais, e, novamente, com a variação de duas e quatro máquinas virtuais de cliente. Por fim, nas Figuras 5.5(e) e 5.5(f), são apresentados os cenários do ataque SYN_Flood, gerado por seis máquinas virtuais, mas variando a quantidade de máquinas virtuais de clientes legítimos.



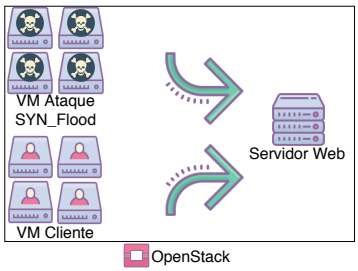

(a) Ataque SYN_Flood e Cliente com 2 VMs



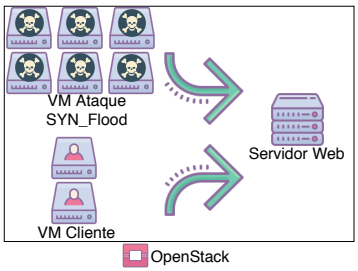

(b) Ataque SYN_Flood com 2 e Cliente com 4 VMs



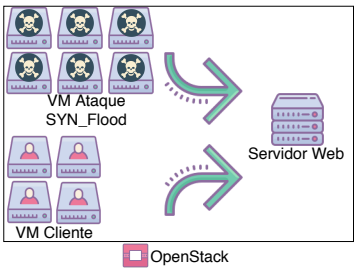

(c) Ataque SYN_Flood com 4 e Cliente com 2 VMs




(d) Ataque SYN_Flood e Cliente com 4 VMs



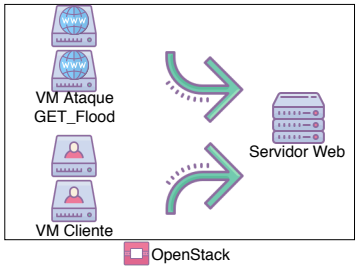

(e) Ataque SYN_Flood com 6 e Cliente com 2 VMs



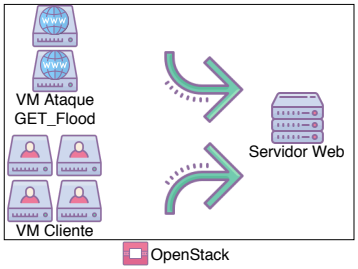

(f) Ataque SYN_Flood com 6 e Cliente com 4 VMs

**Figura 5.5: Diferentes regimes de geração do tráfego cliente e do ataque SYN_Flood**

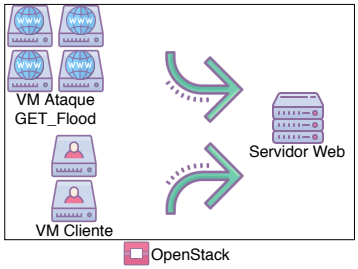
Por fim, na Figura 5.6, também foi variada a quantidade de máquinas virtuais de cliente e de geração do ataque GET_Flood. Nas Figuras 5.6(a) e 5.6(b), duas e quatro máquinas virtuais, respectivamente, variam a geração de clientes legítimos, enquanto duas máquinas virtuais geram o ataque GET_Flood. Nas Figuras 5.6(c) e 5.6(d), quatro máquinas virtuais são utilizadas para gerar o ataque, e há uma variação da quantidade de clientes legítimos gerados, variando entre duas e quatro máquinas virtuais. Por fim, nas Figuras 5.6(e) e 5.6(f), são apresentados os cenários com seis máquinas virtuais gerando o ataque GET_Flood, com duas e quatro máquinas virtuais gerando o tráfego de cliente legítimo. Dessa forma, com essa diversidade de cenários durante a elaboração dos *datasets* de treino e de teste, o objetivo é confundir os algoritmos de aprendizado de máquina, e eliminar eventuais vieses que possam existir nos *datasets*. Considerando o exemplo do cenário da Figura 5.6(b), o algoritmo deverá detectar e identificar que está acontecendo um ataque GET_Flood, mesmo que exista mais requisições clientes do que atacante, em virtude das quantidades de máquinas virtuais clientes ser o dobro das VMs atacantes. Todos esses cenários foram realizados dez vezes cada um, em diferentes sequências e duração, durante a realização do experimento para gerar o *dataset* de treino.



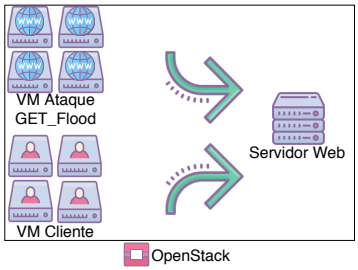

(a) Ataque GET_Flood e Cliente com 2 VMs



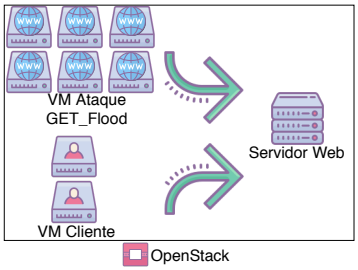

(b) Ataque GET_Flood com 2 e Cliente com 4 VMs



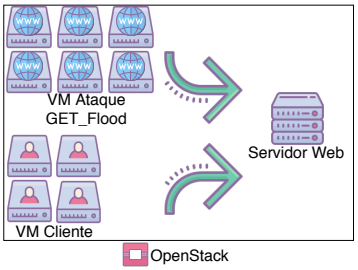

(c) Ataque GET_Flood com 4 e Cliente com 2 VMs




(d) Ataque GET_Flood e Cliente com 4 VMs




(e) Ataque GET_Flood com 6 e Cliente com 2 VMs




(f) Ataque GET_Flood com 6 e Cliente com 4 VMs

**Figura 5.6: Diferentes regimes de geração do tráfego cliente e do ataque GET_Flood**

Para gerar o *dataset* de teste, outro experimento foi realizado seguindo as mesmas variações explicadas anteriormente, com a diferença da duração dos cenários. No total, esse *dataset* de teste teve duração de 16 horas, obtendo assim um total de 11.555 amostras coletadas pelo Ceilometer. No *dataset*, para realizar a fase de detecção, as amostras foram rotuladas em duas classes: ataque e não ataque. Além disso, outro rótulo foi utilizado para identificar o tipo de ataque de DDoS: SYN_Flood ou GET_Flood. Esse segundo rótulo é utilizado para a fase de identificação, em que o objetivo é verificar especificamente qual o tipo de ataque.

Para a fase de identificação, foi utilizada a estratégia de considerar a probabilidade de clientes legítimos serem detectados erroneamente como maliciosos na primeira fase. Assim, a fase de identificação que analisa o tipo de ataque de DDoS também deve reanalisar a classificação de clientes legítimos. Como resultado, a fase de identificação pode ter as seguintes saídas: SYN_Flood, GET_Flood ou cliente legítimo.

#### 5.1.1.1 Zero-Day dataset

Levando em consideração, o que foi apresentado no Capítulo 4, um ataque de DDoS vai gerar algum tipo de anormalidade em um subconjunto $R_x$ de recursos computacionais monitorados pelo Ceilometer. Consequentemente, seria possível detectar um ataque de DDoS mesmo que o modelo de ML criado não conhecesse previamente esse ataque. O objetivo desta seção

é verificar se essa hipótese é verdadeira. Diante disso, um novo *dataset* foi gerado utilizando um novo tipo de ataque, o PING_Flood, que não foi utilizado durante a fase de treinamento dos modelos. Esse *dataset* está apresentado na última coluna da Tabela 5.1. Dessa forma, está sendo emulado um ataque *Zero-Day*. Esse *dataset* foi criado no mesmo ambiente dos experimentos anteriores e em três cenários diferentes: apenas clientes legítimos utilizando o servidor *web*, apenas o ataque de DDoS PING_Flood e os dois realizados juntos.

### 5.1.2 Seleção de características

Como discutido na Seção 2.2, a telemetria do OpenStack pode oferecer diversas métricas. Para selecionar as melhores métricas e melhorar o desempenho na tarefa dos algoritmos de aprendizado de máquina, foi realizada uma seleção de características, como apresentado no Capítulo 2. Essa seleção utilizou a biblioteca Python Scikit-Learn[5], especificamente o método selectKBest[6], que seleciona um subconjunto com *k* métricas diferentes. O selectKBest implementa o modelo *Wrapper* de seleção de características (AGGARWAL, 2015). Ou seja, todo o conjunto de métricas disponíveis foram combinadas em subconjuntos distintos, com o intuito de avaliar a acurácia desse subconjunto na tarefa de classificar as amostras. A seleção de características foi realizada nas seguintes etapas:

- É selecionado *k* métricas do conjunto de métricas disponibilizada pela telemetria;
- Com este subconjunto de *k* métricas, o *dataset* é dividido, aleatoriamente: 70% do *dataset* para treino e 30% para teste;
- É realizado o treinamento do algoritmo Nearest Neighbor (NN);
- Com o *dataset* de teste é verificado a acurácia do algoritmo na classificação das amostras com o subconjunto de *k* métricas.

As etapas descritas anteriormente foram repetidas até que toda a combinação de subconjunto de *k* métricas distintas fossem testadas. As acurácias para cada subconjunto de métricas foram comparadas, e, dessa forma, o critério para a seleção de qual *k* métricas seriam utilizadas na detecção e identificação foi a que tivesse a maior acurácia. O código-fonte e o *dataset* estão disponíveis para que os resultados apresentados possam ser reproduzidos[7].

[5]Página web: <https://scikit-learn.org/stable/>. Acessado em 18 de agosto de 2019.

[6]Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html>. Acessado em 18 de agosto de 2019.

[7]Página web: <https://jhenriquecorrea.github.io/datasets/>. Acessado em 20 de junho de 2020.

Como o objetivo desta seção é realizar a detecção e identificação de um ataque de DDoS, foram realizadas duas seleções de características diferentes. Para a fase de detecção, os atributos selecionados (métricas) foram: a taxa de Bytes enviados e recebidos pela interface de rede e a taxa de pacotes recebidos pela interface de rede. Para a fase de identificação, a seleção de características indicou que a melhor escolha são as seguintes métricas: utilização de CPU e de memória, número de requisições de leitura no disco, número de requisições de escrita no disco, a taxa de Bytes recebidos e enviados da interface de rede, e a taxa de pacotes recebidos e enviados da interface de rede.

### 5.1.3 Algoritmos de ML e indicadores de desempenho dos algoritmos

Uma parte importante da proposta envolve a avaliação dos algoritmos de aprendizado de máquina nas etapas de detecção e identificação dos ataques de negação de serviço. Foram utilizados e avaliados vários algoritmos de ML clássicos: k-Nearest Neighbor (kNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest (RF), e Multi-Layer Perceptron. Esses algoritmos foram escolhidos como representantes das classes de algoritmos de aprendizado de máquina, baseado em distância, função de probabilidade, esquema de decisão e uma rede neural.

Os seguintes indicadores de desempenho foram escolhidos para avaliar os algoritmos de aprendizado de máquina na tarefa de detectar e identificar ataques de DDoS: Acurácia, Precisão, Revocação e *F1-Score*. Esses indicadores de desempenho foram definidos e explicados no Capítulo 2.

O *true positives* (TP) o número de casos atuais positivos corretamente predito como positivo, *true negatives* (TN) é o número de casos atuais negativos corretamente predito como negativo, *false positives* (FP) é o número de casos atuais negativos incorretamente predito como positivo, e *false negative* (FN) é o número de casos atuais positivos incorretamente predito como negativo. Neste contexto, na fase de detecção, os casos positivos são os ataques de DDoS e os casos negativos são de clientes legítimos. Já na fase de identificação, cada métrica é computada para cada classe (SYN_Flood, GET_Flood e cliente legítimo), considerando as amostras da classe avaliada como casos positivos e as amostras restantes como casos negativos. Em seguida, a média ponderada de cada métrica é calculada considerando a quantidade de amostras em cada classe.

### 5.1.4 Avaliação e Resultados da Detecção e Identificação

Nesta seção, o teste de detecção e identificação dos ataques de negação de serviço é realizado, apresentado e discutido. Além disso, IDS convencional é testado de forma *offline*, utilizando o *trace* do tráfego correspondente descrito na Tabela 5.1. Finalmente, o modelo de ML é verificado contendo um ataque que não foi treinado, simulando um ataque *Zero-Day*. Dos algoritmos de aprendizado de máquina elencados anteriormente, dois apresentaram os melhores indicadores de desempenho: k-Nearest Neighbor (kNN) e Random Forest (RF). Dessa forma, os resultados apresentados utilizarão esses dois classificadores.

Para realizar a detecção e identificação dos ataques de DDoS, foram escolhidos os parâmetros dos algoritmos de aprendizado de máquina: neste processo de escolha, para o algoritmo kNN, o valor de $k$ foi variado entre 1 e 15, sempre utilizando valores ímpares e com o mesmo número de exemplos em cada classe, buscando a melhor acurácia na detecção e identificação. Da mesma forma, foi variado o número de árvores no algoritmo Random Forest entre 80 e 120, novamente buscando a melhor acurácia. Assim, os seguintes parâmetros obtiveram os melhores indicadores de desempenho: (i) para o algoritmo kNN, foi utilizado $k$ igual a 3 e distância Euclidiana; (ii) para o algoritmo Random Forest, 100 árvores diferentes foram utilizadas. Vale destacar que a implementação desses algoritmos está disponível na biblioteca Python's Scikit-Learn[8]. De acordo com a metodologia apresentada na Figura 5.2, a utilização dos algoritmos de ML tem dois objetivos: a detecção da ocorrência de um ataque de DDoS e a identificação de qual tipo de ataque está acontecendo. Esse é um ponto relevante, pois apenas as amostras classificadas como ataque de DDoS na fase de detecção são repassadas para a fase identificação, e as amostras que foram classificadas como cliente legítimo não são processadas novamente na fase de identificação.

#### 5.1.4.1 Fase de Detecção

Na Tabela 5.2, as métricas Acurácia, Revocação, Precisão e *F1-Score* para os algoritmos kNN e Random Forest na fase de detecção são apresentadas. Nessa fase, o objetivo é verificar se está ou não acontecendo algum ataque de DDoS no ambiente de nuvem. Todos os algoritmos tiveram uma Acurácia com valores altos, sendo de 87%. O valor da Revocação para os algoritmos kNN e RF ficou acima de 87%, ou seja, uma Revocação alta, detectando corretamente a maioria das amostras de ataque. Em relação à avaliação da métrica Precisão, que está relacionada à detecção de clientes legítimos com sucesso, os dois algoritmos obtiveram bons resultados, acima de 86%.

[8]Página web: <https://scikit-learn.org/stable/>. Acessado em 18 de agosto de 2019.

**Tabela 5.2: Resultado dos algoritmos de ML na fase de Detecção**

| Algoritmo | Acurácia | Revocação | Precisão | F1-Score |
|---|---|---|---|---|
| KNN | 87,37% | 87,37% | 86,18% | 85,91% |
| RF | 87,28% | 87,28% | 86,16% | 85,51% |

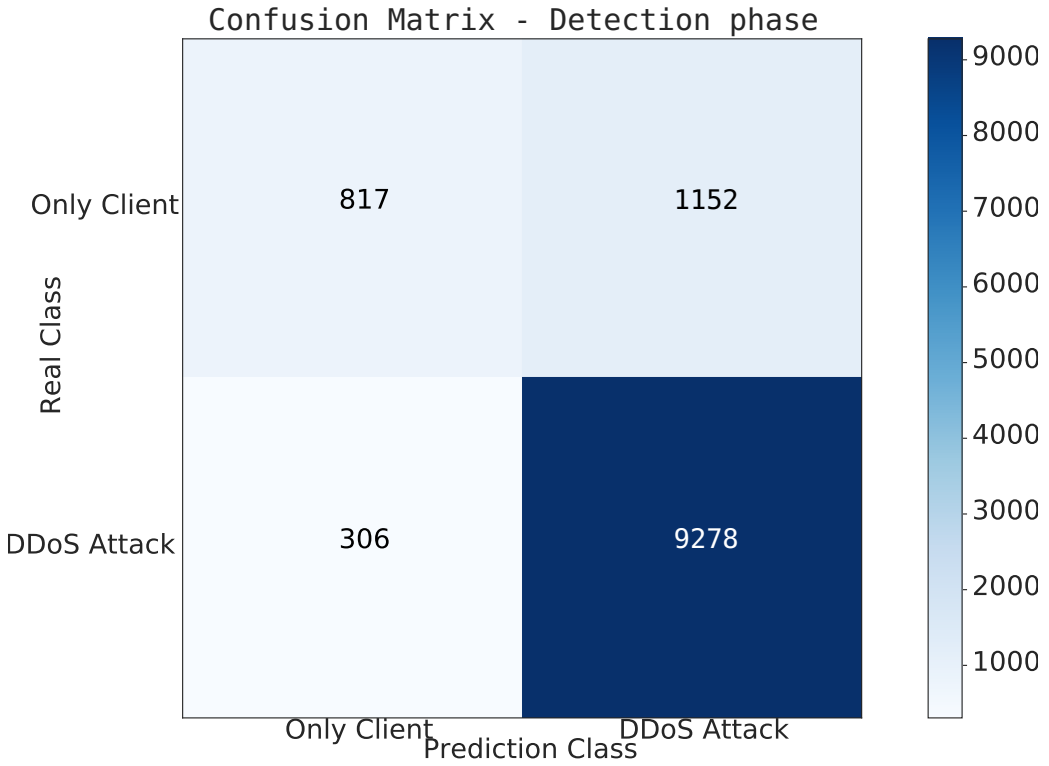



**Figura 5.7: Matriz de Confusão para o algoritmo kNN na fase de Detecção (CORRÊA et al., 2021)**

De acordo com a Figura 5.7, que apresenta a matriz de confusão gerada pela execução do algoritmo kNN para a fase de detecção, de um total de 1.969 amostras clientes, 817 amostras foram classificadas corretamente como cliente legítimo, mas 1.152 foram detectadas erroneamente como ataque de DDoS. Essa classificação errônea é devido ao ataque GET_Flood que imita o comportamento de clientes legítimos. Por outro lado, 9.278 amostras são corretamente detectadas como ataque, e apenas 306 amostras são classificadas erroneamente como de clientes. Esses resultados mostram que, apesar de ser um classificador simples utilizando a distância Euclidiana, o kNN atinge um bom desempenho com uma taxa de erro relativamente baixa.

A Figura 5.8 apresenta a curva ROC (*Receiver Operating Characteristic*) para o algoritmo kNN na fase de detecção. Essa curva apresenta uma avaliação na relação entre *True Positive* (eixo Y) e *False Positive* (eixo X). O canto superior esquerdo do gráfico é o ponto ”ideal”, com uma taxa de falso positivo em zero e a taxa de verdadeiro positivo em um. São refletidos na Figura 5.8 os resultados que o algoritmo kNN obteve, com uma boa relação entre os falsos posi-

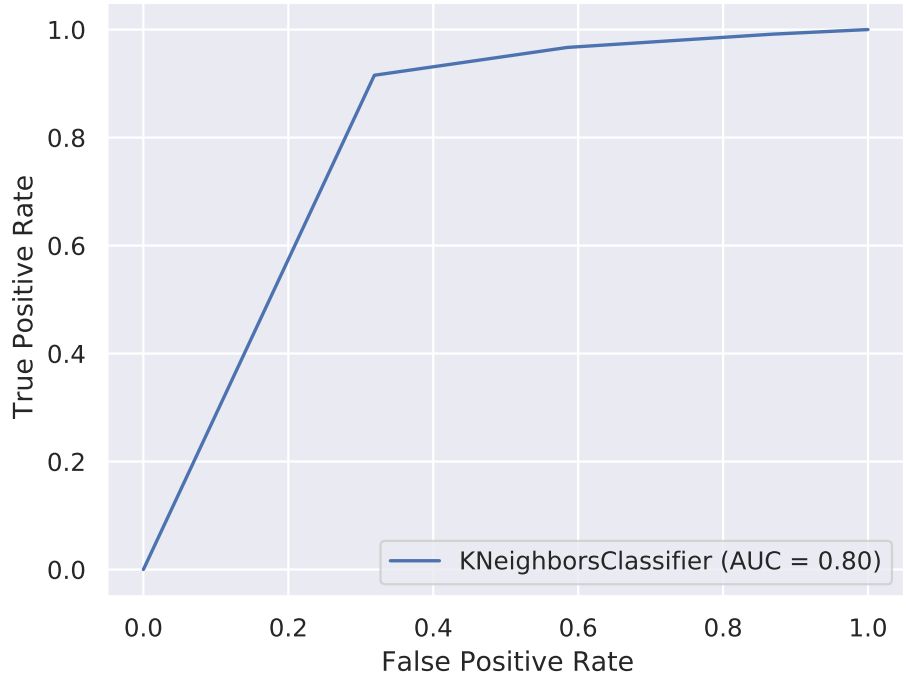



**Figura 5.8: Curva ROC para o algoritmo kNN na fase de Detecção (CORRÊA et al., 2021)**

**Tabela 5.3: Resultados gerais para os algoritmos de ML na fase de Identificação**

| Algoritmos | Acurácia | Revocação | Precisão | F1-Score |
|---|---|---|---|---|
| KNN | 89,46% | 89,46% | 88,68% | 88,15% |
| RF | 87,84% | 87,84% | 80,10% | 83,34% |

tivos e os verdadeiros positivos, a curva ROC sobe rapidamente para o canto superior esquerdo da figura, e obtendo um AUC (*Area Under the Curve*) 0.8 e gerando um bom resultado para o classificador.

### 5.1.4.2 Fase de Identificação

Para a fase de identificação do tipo de ataques de DDoS, três resultados são possíveis: SYN_Flood, GET_Flood, ou cliente legítimo. Os resultados gerais são apresentados na Tabela 5.3. Nessa etapa, a métrica Acurácia para os algoritmos kNN e RF foi de 89,46% e 87,84%, respectivamente. De modo geral, o algoritmo kNN apresenta resultados melhores comparado com o *Random Forest*.

A Figura 5.9 apresenta a matriz de confusão para o algoritmo kNN na fase de identificação. Pode-se verificar que 1.152 amostras de clientes legítimos foram classificadas erroneamente como ataque durante a fase de detecção. Dessas amostras, 337 foram reclassificadas correta-

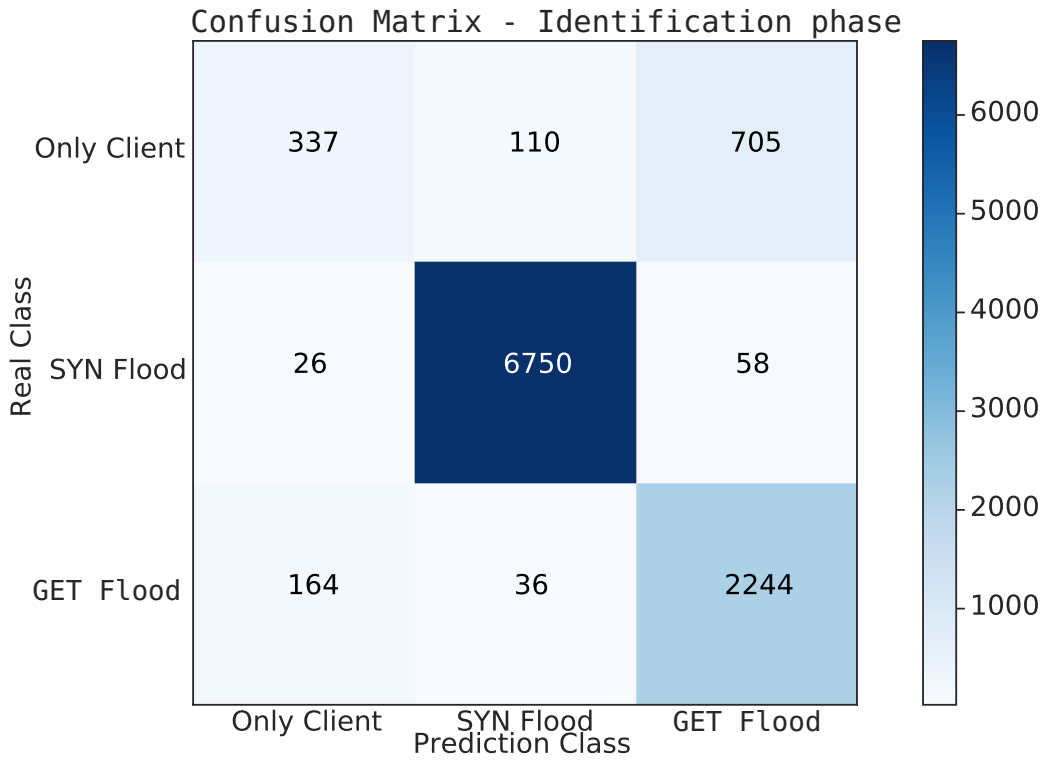



**Figura 5.9: Matriz de Confusão para o algoritmo kNN na fase de Identificação (CORRÊA et al., 2021)**

mente como clientes legítimos na fase de identificação, enquanto que 815 foram identificadas erroneamente como ataque de DDoS (110 como SYN_Flood e 705 como GET_Flood). Das 6.834 amostras de ataque de DDoS SYN_Flood, o algoritmo kNN identificou corretamente 6.750. De modo semelhante, das 2.444 amostras do ataque GET_Flood, 2.244 foram identificadas corretamente pelo algoritmo kNN. Como pode ser observado na Figura 5.9, a maioria das amostras de clientes foram identificadas como ataque de DDoS GET_Flood, pois, como afirmado por Zargar *et al.* (ZARGAR; JOSHI; TIPPER, 2013), este tipo de ataque tem um comportamento semelhante ao dos clientes legítimos, dessa forma, refletindo também nas métricas obtidas pelo serviço de telemetria. Por outro lado, pode ser verificado que o ataque de DDoS SYN_Flood é mais fácil de identificar, pois, há acerto maior na classificação deste ataque.

#### 5.1.4.3 Comparativo com Snort

Com a finalidade de ilustrar como um IDS tradicional pode ser comparado com a proposta apresentada, foi escolhida a utilização do Snort IDS[9] (version 2.9.13), que é uma solução de código aberto, multiplataforma e um dos IDS mais popular na comunidade de redes de computadores. O Snort é fundamentado em regras que realizam um filtro de pacotes de rede, baseado

[9]Página web: <https://www.snort.org/>. Acessado em 18 de agosto de 2019.

nas assinaturas conhecidas dos ataques. Essas assinaturas são escritas como regras e a identificação é explícita: uma ou mais regra para cada tipo de ataque. Observa-se que, para permitir esta comparação, durante a geração do *dataset* de teste, os pacotes de rede também foram coletados em um arquivo .pcap, para serem analisados pelo Snort. Essa informação está apresentada na última linha da Tabela 5.1.

Duas regras diferentes para detectar os ataques SYN_Flood e GET_Flood foram encontrados na literatura e em sites na Internet, especializados em segurança de rede. É importante salientar que não foram realizadas modificações das regras, sendo utilizadas nesta Tese de acordo como foram apresentadas na literatura e nos sites. A regra para detectar o ataque SYN_Flood foi obtida em Buchanan (BUCHANAN, 2014) e pode ser verificada na Regra 5.1, escrita na linguagem padrão do IDS Snort. Nessa regra, são filtrados todos os pacotes com o destino a porta 80 (que é a porta padrão do HTTP). Um alerta de ataque de DDoS SYN_Flood é gerado quando mais do que 60 pacotes são ativados por essa regra, durante um intervalo de 60 segundos.

**Regra 5.1: Regra Snort para a geração de alertas para o ataque de DDoS SYN_Flood.**

```
alert tcp any any -> any 80 (msg:"SYN_Flood DDoS attack attempt"; flow:to_server;
    detection_filter:track by_dst, count 60, seconds 60; sid:5000002; rev:1)
```

Para detectar o ataque de DDoS GET_Flood, a Regra 5.2 foi utilizada, obtida de Ruhl e Johnjg12 (RUHL; JOHNJG12, 2015). Novamente, a Regra 5.2 está escrita de acordo com a linguagem requerida pelo IDS Snort. Essa regra realiza a detecção fazendo uma verificação dos pacotes IP de acordo com: (i) o endereço IP do servidor web (a vítima) como destino; (ii) a porta de destino do TCP igual a 80; (iii) pacotes oriundos de uma mesma fonte; e (iv) pacotes com conexões já estabelecidas. Nessa regra, o alerta é gerado se mais de 30 HTTP GET acontecessem durante o intervalo de 30 segundos.

**Regra 5.2: Regra Snort para a geração de alertas para o ataque de DDoS GET_Flood.**

```
alert tcp any any -> [web server IP address] 80 (msg:"GET_Flood DDoS aatack
    attempt"; flow:to_server,established; content:"GET"; nocase; http_method;
    detection_filter:track by_src, count 30, seconds 30; metadata: service http;
    sid: 5000004)
```

Relacionado à Regra 5.2, é importante notar que é requerido do IDS uma inspeção de cada pacote até a camada de aplicação. De acordo com o manual do usuário do Snort (PROJECT, 2020), essa tarefa de inspeção em camadas altas pode ocasionar um aumento na latência na rede ou do número de pacotes descartados devido à alta carga da ferramenta IDS, visto que o IDS

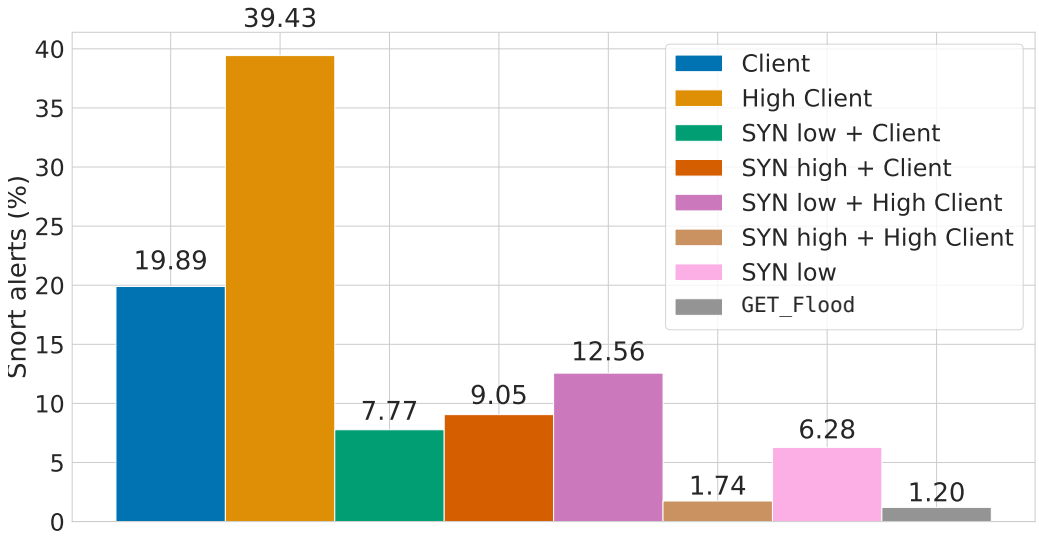



**Figura 5.10: Porcentagem dos alertas gerados pela Regra 5.2 (ataque de DDoS GET_Flood) (CORRÊA et al., 2021)**

pode estar localizado no meio do caminho entre a origem e o destino.

O *trace* dos pacotes mencionados na Tabela 5.1 foi processado *offline* pelo Snort com duas regras diferentes. Como esperado, quando se concentrava no ataque SYN_Flood, utilizando a Regra 5.1, Snort gerou alertas do ataque SYN_Flood, alcançando assim resultados muito satisfatórios, ou seja, alertas foram gerados mesmo quando as taxas de geração do ataque SYN_-Flood eram baixas. Isso se deve principalmente porque este tipo de ataque tem uma assinatura simples e fácil de ser descrita na Regra 5.1. Por outro lado, a detecção do ataque GET_Flood utilizando a Regra 5.2, Snort está longe de apresentar um desempenho satisfatório, pois não conseguiu gerar alertas para o ataque GET_Flood. Além disso, há de se destacar que essa regra em particular se aprofunda nos campos dos pacotes e impacta na latência da rede (PROJECT, 2020).

A Figura 5.10 apresenta o comportamento do Snort IDS utilizando a Regra 5.2 durante uma combinação de cenário: apenas clientes legítimos, SYN_Flood, GET_Flood, e a combinação de clientes e atacantes. O eixo vertical representa a porcentagem do número total de alertas gerados pelo Snort IDS durante o experimento. Nota-se que a maioria dos alertas gerados pelo Snort (59,32%) foram gerados quando apenas clientes legítimos estavam acessando o servidor web (`Client` + `High Client`). `High Client` são clientes legítimos acessando o servidor web em altas taxas. A barra em cinza, rotulada como `GET_Flood`, sumariza o número de alertas gerados quando o ataque GET_Flood era simulado. Snort não foi capaz de gerar alertas durante o ataque de DDoS GET_Flood, gerando apenas 1,2% dos alertas, o que pode ser visto como um alto número de falsos negativos, pois, nesse momento o tráfego era de ataque. Isso confirma Zargar *et al.* (ZARGAR; JOSHI; TIPPER, 2013), que afirmam ser difícil para um IDS convencional

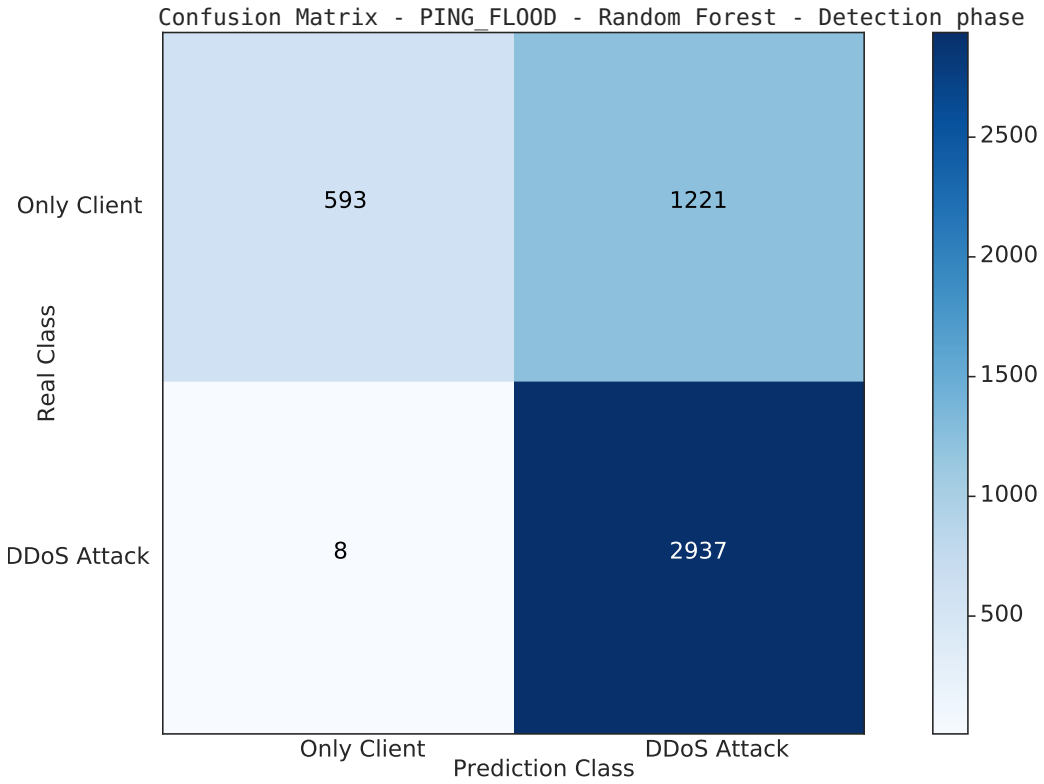



**Figura 5.11: Matriz de Confusão para o algoritmo Random Forest para o ataque PING_Flood (CORRÊA et al., 2021)**

detectar ataques na camada de aplicação, como o GET_Flood, uma vez que tem características muito semelhantes às de clientes legítimos que sobrecarregam o servidor *web*.

Além disso, é importante destacar que o Snort é incapaz de diferenciar entre cliente legítimo e atacantes, identificando apenas grandes volumes de tráfego que correspondem à assinatura do ataque descrito em cada regra. Essa configuração é feita pelo gerente da rede e pode ser facilmente sujeita a erros.

Em resumo, a capacidade do Snort de detectar um ataque de DDoS depende totalmente da configuração de cada regra e da assinatura dos ataques de DDoS. Se um pacote corresponder à assinatura, ele conta para essa regra e pode gerar um alerta. Percebe-se que a Regra 5.2 é invasiva em relação à inspeção da carga útil, e isso pode causar não apenas violações de privacidade, mas também latência maior ou pacotes descartados devido a uma sobrecarga da ferramenta IDS (PROJECT, 2020). Ressalta-se que os resultados obtidos pelo Snort não podem ser comparados diretamente à classificação realizada pelos algoritmos de aprendizado de máquina.

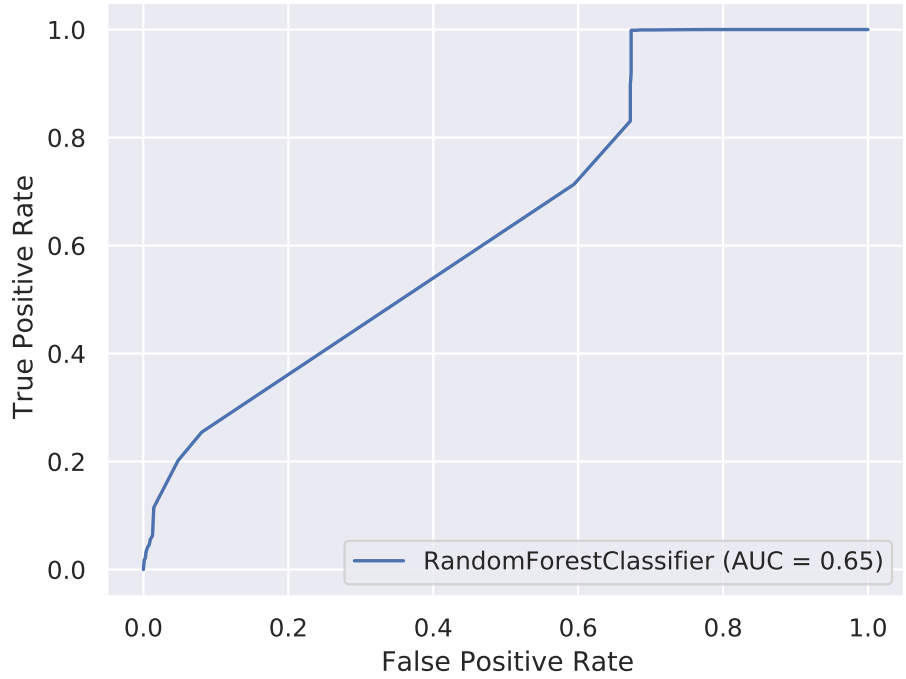



**Figura 5.12: Curva ROC para o algoritmo RF no ataque PING_Flood (CORRÊA et al., 2021)**

#### 5.1.4.4 Algoritmo de ML com ataque *Zero-Day*

Nos experimentos, o melhor resultado para a detecção do ataque PING_Flood foi apresentado pelo algoritmo Random Forest, com uma Acurácia de 74% e uma Precisão de 80%. A Figura 5.11 apresenta a matriz de confusão gerada pela execução do algoritmo RF. Sendo 2.937 amostras classificadas corretamente como ataque de DDoS, e apenas 8 classificados como clientes legítimos. A Figura 5.12 expõe a curva ROC para o classificador Random Forest. Nessa avaliação, a curva ROC demonstra uma evolução um pouco pior comparado com ataques em que o modelo de ML conhece na fase de detecção, como mostrado na Figura 5.8. O AUC obtido pelo classificador RF, na fase de detecção do ataque PING_Flood, foi de 0.65. Apesar de esse resultado não ser considerado bom para detectar ataques já conhecidos, o modelo de ML demonstra potencial para detectar corretamente ataques ”desconhecidos”.

Assim, é possível afirmar que o modelo de ML criado com o uso de ataques de DDoS SYN_Flood e GET_Flood foi capaz de detectar a ocorrência do ataque PING_Flood com boa Acurácia. Entretanto, o modelo de ML não consegue identificar o subconjunto dos recursos computacionais atingidos por este ataque, dado que não foi treinado para essa tarefa.

Como avaliação inicial, pode-se observar que a abordagem de detectar ataques de DDoS proposto nesta seção tem um potencial para generalizar a detecção de diferentes tipos de ataques

de DDoS mesmo que o modelo de ML não tenha sido treinado para eles. Essa afirmação é possível de ser explicada, já que a maioria dos ataques de DDoS aumentam a utilização de alguns recursos computacionais na infraestrutura da nuvem, e isso pode ser detectado como uma condição anormal por um modelo de ML previamente treinado com clientes legítimos. Assim, mesmo sem a necessidade de gerar um novo modelo, a proposta pode detectar a ocorrência de um ataque de DDoS que não era previamente conhecido. Essa detecção é importante e permite a possibilidade de realizar alguma mitigação, seja filtrando/descartando o tráfego, enviando para serviços de *traffic scrubbing*, ou mesmo encaminhando pacotes para um serviço de DPI para verificação adicional. Essas formas tradicionais de identificação ou mitigação, especialmente as que utilizam inspeção de pacotes, podem ser inseridas sob demanda no caminho (utilizando o *Service Function Chaining*), logo após a detecção do ataque na infraestrutura da nuvem. Assim, leva a uma diminuição na necessidade de recursos dedicados e adiciona latência apenas durante o ataque, visto que será instanciado sob demanda e não interferindo no QoE do usuário.

## 5.2 Ataques de Intrusão

### 5.2.1 Metodologia

Assim como descrito para os ataques de DDoS, foi utilizada também a telemetria da nuvem para detectar ataques de intrusão, em conjunto com algoritmos de aprendizado de máquina. Para isso, foram realizados experimentos para a geração de *datasets* de teste e de treino para verificar a proposta de detectar ataques de intrusão. A Tabela 5.4 sumariza as informações para a geração dos *datasets*. Nesses experimentos, uma máquina virtual foi inserida com uma aplicação *web*, com acesso a um banco de dados por meio de requisições SQL. Essa aplicação, chamada *Damn Vulnerable Web Application*[10] (DVWA), é escrita em PHP/MySQL e é uma aplicação voltada para testes e prática de segurança.

Para a geração do tráfego cliente, a ferramenta Siege também foi utilizada, com a diferença que os clientes simulados não enviam apenas requisições HTTP GET, mas também consultas SQL. O cliente gerar consultas SQL é importante para trazer uma justiça em relação do tráfego do cliente para com o do atacante. Sabe-se que uma consulta SQL gera busca em um banco de dados, ocasionando uso de recursos computacionais. Dessa forma, se clientes também realizam consultas SQL, então os experimentos são gerados de forma justa entre atacante e cliente. Assim como nos experimentos de DDoS, apresentados na Seção 5.1, nos experimentos com ataque de intrusão foram utilizadas duas e quatro máquinas virtuais clientes. A quantidade de VMs de

[10]Página web: <https://dvwa.co.uk/>. Acessado em 30 de setembro de 2021.

**Tabela 5.4: Datasets do ataque de intrusão**

| Datasets | Treino | Teste |
|---|---|---|
| Duração | 10h | 24h |
| Utilização da CPU do Servidor SQL (por clientes legítimos) | 50% / 100% | 50% / 100% |
| # VMs Cliente | 2/4 | 2/4 |
| # VMs Atacante | 2/4/6 | 2/4/6 |

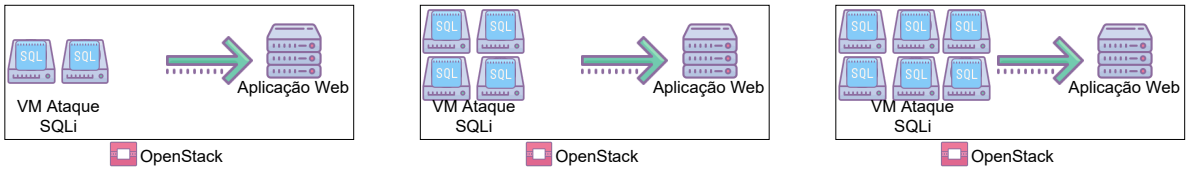
clientes é variada para criar, no *dataset*, diferentes níveis de uso, simulando vários regimes de tráfego. Duas máquinas virtuais clientes são utilizadas, gerando requisições legítimas consumindo assim 50% de CPU do servidor, como representado na Figura 5.13(a). E as quatro VMs são utilizadas para gerar tráfego legítimo capaz de consumir 100% de CPU do servidor, sem gerar qualquer tipo de indisponibilidade, como apresentado na Figura 5.13(b). Essas informações estão sumarizadas na terceira e quarta linha da Tabela 5.4.


(a) Cliente com 2 VMs (b) Cliente com 4 VMs

**Figura 5.13: Diferentes regimes de geração do tráfego cliente no ataque de intrusão**

Para a geração do ataque SQL_Injection, foi criado um programa em Python que realiza requisições SQL para a aplicação *web* com código malicioso inserido. O objetivo da requisição é resgatar todas as informações contidas no banco de dados, mesmo sem ter autorização. O ataque envia novamente uma nova requisição SQL quando a anterior é totalmente respondida. Dessa forma, a aplicação *web* sempre estará recebendo, processando e respondendo requisições com SQL_Injection. Foram utilizadas ao todo 6 máquinas virtuais gerando o tráfego de ataque, e essas máquinas foram variadas para gerar diferentes níveis de uso: duas, quatro e seis VMs atacando simultaneamente, significando assim um ataque de baixa, média e alta intensidade, respectivamente. Esses diferentes regimes de tráfego estão apresentados nas Figuras 5.14(a), 5.14(b) e 5.14(c), representando, respectivamente, o uso de duas, quatro e seis máquinas virtuais para gerar o ataque SQL_Injection. Essa informação está presente na última linha da Tabela 5.4.

Foram realizados dois experimentos com esse cenário: um para a geração do *dataset* de treino e outro para gerar o de teste. O experimento do *dataset* de treino teve uma duração



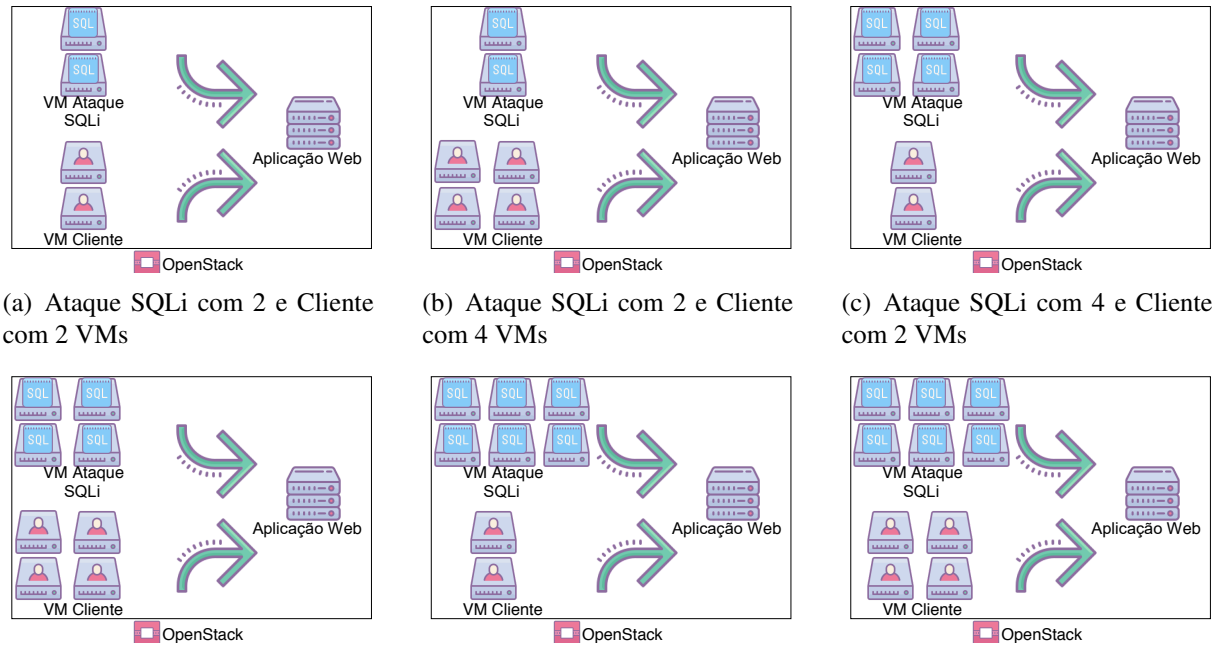

(a) Ataque SQLi com 2 VMs (b) Ataque SQLi com 4 VMs (c) Ataque SQLi com 6 VMs

**Figura 5.14: Diferentes regimes de geração do tráfego do ataque SQL_Injection**

aproximada de 10 horas, enquanto que os experimentos para a geração do *dataset* de teste teve uma duração aproximada de 24 horas. Durante os experimentos, foram combinados momentos em que apenas clientes realizavam requisições, momento em que apenas o ataque acontecia, e, por fim, a combinação de clientes e de atacantes. Da mesma maneira, como explicado anteriormente, os níveis de clientes e de atacantes foram variados, simulando diferentes regimes de tráfego durante os experimentos. A representação do regime de tráfego durante os experimentos está na Figura 5.15. Nas Figuras 5.15(a) e 5.15(b) foram utilizadas duas máquinas virtuais para gerar o ataque SQL_Injection. e houve uma variação da quantidade de máquinas virtuais de clientes legítimos. Já nas Figuras 5.15(c) e 5.15(d), a quantidade de máquinas virtuais para gerar o ataque aumenta para quatro, sendo variada a quantidade de VMs clientes em duas e quatro, respectivamente. Por fim, nas Figuras 5.15(e) e 5.15(f), são apresentados os cenários em que seis máquinas virtuais são utilizadas para gerar o ataque SQL_Injection, com a variação do tráfego de cliente, alternando entre duas e quatro máquinas virtuais. Todos esses cenários foram realizados dez vezes cada um, em diferentes sequências e duração, durante toda a execução do experimento. Ao longo dos experimentos para a geração dos dois *datasets*, os dados da telemetria da nuvem foram coletados a cada 5 segundos. Esse valor é um compromisso empírico entre a precisão da métrica coletada e o uso dos recursos computacionais para a coleta das métricas.

A mesma seleção de características apresentada na Seção 5.1.2 também foi realizada no contexto dos ataques de intrusão, apresentando o seguinte conjunto de métricas: utilização de CPU e de memória, número de requisições de leitura no disco, número de requisições de escrita no disco, a taxa de Bytes recebidos e enviados da interface de rede, e a taxa de pacotes recebidos e enviados da interface de rede.

Foram utilizados e avaliados algoritmos de aprendizado de máquina supervisionado, com a tarefa de classificação: k-Nearest Neighbor (kNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest (RF), e Multi-Layer Perceptron. No entanto, o kNN obteve o melhor resultado, utilizando os seguintes indicadores de desempenho: Acurácia, Precisão, Revocação e *F1-Score*. Por esse motivo, apenas os resultados obtidos pelo kNN são apresentados.




(a) Ataque SQLi com 2 e Cliente com 2 VMs

(b) Ataque SQLi com 2 e Cliente com 4 VMs

(c) Ataque SQLi com 4 e Cliente com 2 VMs

(d) Ataque SQLi com 4 e Cliente com 4 VMs

(e) Ataque SQLi com 6 e Cliente com 2 VMs

(f) Ataque SQLi com 6 e Cliente com 4 VMs

**Figura 5.15: Diferentes regimes de geração do tráfego cliente e do ataque SQL SQL_Injection**

### 5.2.2 Avaliação e Resultados

Para o algoritmo de aprendizado de máquina kNN, o $k$ foi escolhido variando entre 1 e 15, sempre em valores ímpares, buscando a melhor acurácia. O valor do $k$ que obteve os melhores indicadores de desempenho foi igual a 3, utilizando o cálculo da distância Euclidiana. A verificação da classificação dos ataques de intrusão foi utilizando a biblioteca Python's Scikit-Learn.

Para o ataque SQL_Injection, o algoritmo kNN obteve uma Acurácia de 94,71%, tendo uma Precisão e uma Revocação de 94,72%, e um *F1-Score* de 94,71%. A Figura 5.16 apresenta a matriz de confusão gerada pela execução do algoritmo kNN durante o ataque SQLi. Nesta matriz de confusão, das 8.711 amostras de clientes, 8.205 foram classificadas corretamente, e 506 erroneamente classificadas como ataque SQLi. De modo semelhante, das 8.792 amostras de ataque, 8.373 foram classificadas como sendo do ataque SQL_Injection e apenas 419 amostras são classificadas erradas como clientes. Percebe-se que o algoritmo kNN obtém melhores resultados na classificação do ataque SQLi, que nos ataques de DDoS SYN_Flood e GET_Flood, de acordo com os resultados apresentados nesta sessão e os apresentados anteriormente na Sessão 5.1.4. Esse resultado pode ser explicado pela característica do ataque SQL_Injection, que força a aplicação *web* a realizar uma pesquisa em todo o banco de dados, e, dessa forma, gera maior utilização dos recursos computacionais do que os ataques de DDoS mencionados anteriormente.

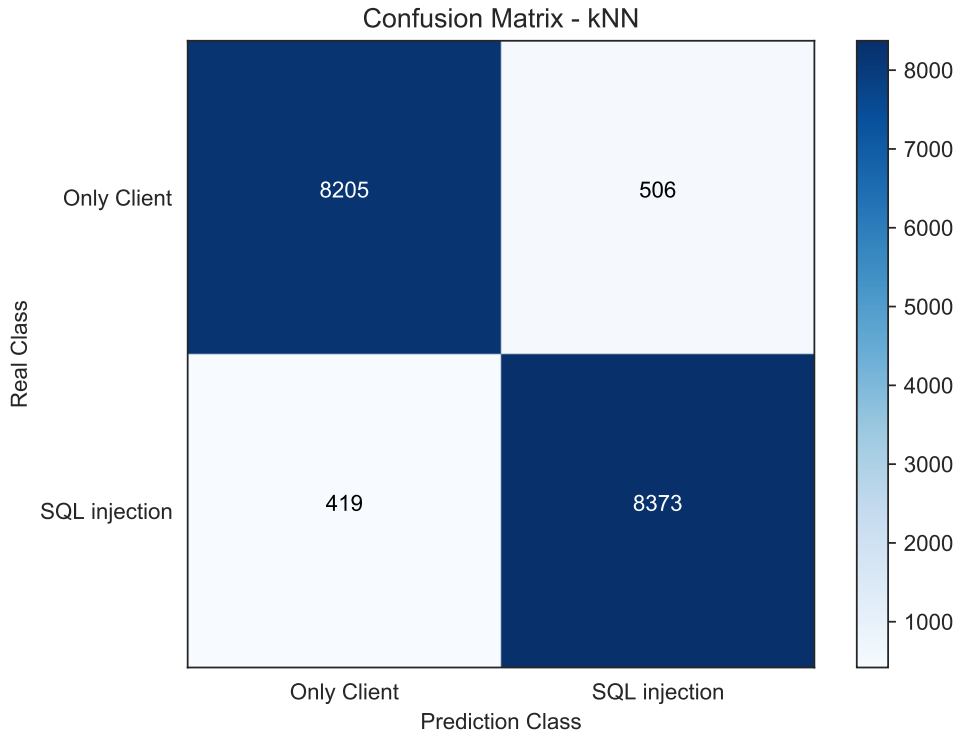



**Figura 5.16: Matriz de Confusão para o algoritmo kNN na fase de Detecção no ataque SQLi**

A Figura 5.17 apresenta a curva ROC (*Recceiver Operation CHaracteristic*) para o algoritmo kNN durante o ataque SQL_Injection. Esse resultado acompanha os indicadores de desempenho, pois a curva se aproxima rapidamente do canto superior esquerdo, indicando uma boa execução da tarefa de classificar as amostras de clientes e de ataque, e na relação entre falsos positivos e verdadeiros positivos. O AUC (*Area Under the Curve*) obtido foi de 0.99, gerando um ótimo resultado para o classificador kNN.

## 5.3 Considerações do Capítulo

Neste capítulo, foi discutida a utilização de algoritmos de aprendizado de máquina, combinados com métricas coletadas pela telemetria, para realizar a detecção e identificação de ataques de negação de serviço e de intrusão. Primeiro foi apresentado o cenário de testes para os ataques de DDoS e como foram gerados os *datasets*. Foram elaborados um total de três *datasets*: o primeiro, para realizar o treino e a geração do modelo dos algoritmos de ML; o segundo, para a fase de teste, verificando os modelos criados; e, por fim, um terceiro *dataset* contendo um ataque que não foi treinado previamente. Além disso, para efeito de comparação, durante a geração do *dataset* de teste, foram coletados os pacotes da rede para a utilização em um IDS tradicional.

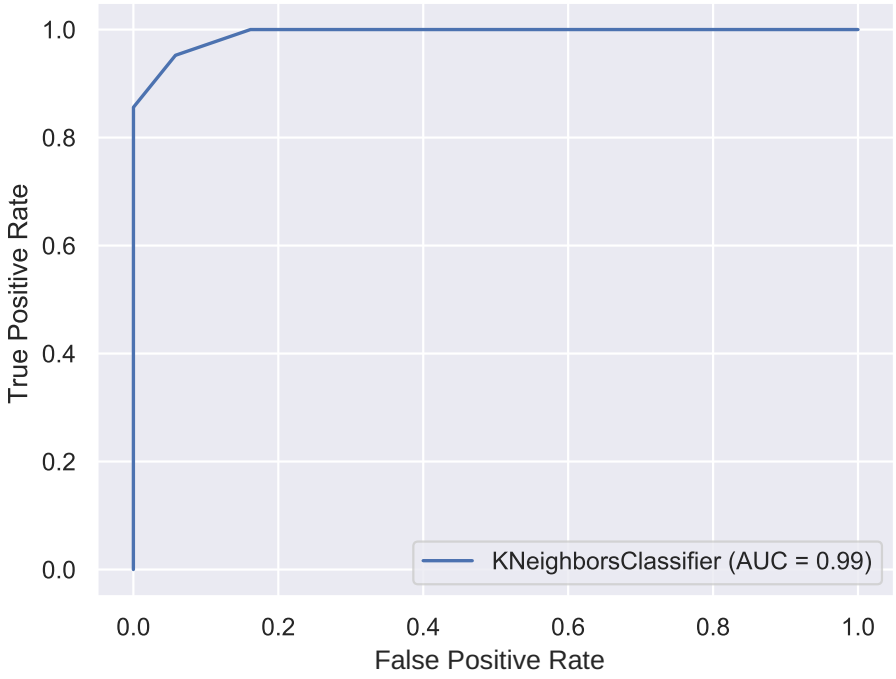



**Figura 5.17: Curva ROC para o algoritmo kNN na fase de Detecção no ataque SQLi**

Posteriormente, foi apresentada a seleção de características, reduzindo a quantidade de recursos computacionais utilizados nos modelos dos algoritmos de aprendizado de máquina, gerando um conjunto de características para serem utilizados na fase de detecção e produzindo outro conjunto de características para ser utilizado na fase de identificação. Além disso, foram apresentados os algoritmos de ML utilizados, bem como, os indicadores de desempenho desses algoritmos.

Em sequência, foram apresentados a avaliação e os resultados da execução dos algoritmos de aprendizado de máquina nos *datasets*, tanto na fase de detecção quanto na fase de identificação. Além disso, foi realizado um comparativo com o IDS Snort, utilizando a captura dos pacotes de rede. Ademais, com intuito de propor uma generalização da proposta, foram verificados os modelos de ML criados anteriormente em ataques desconhecidos pelo modelo. E essa verificação mostrou-se promissora.

Após os resultados para os ataques de DDoS, foram expostos no capítulo os cenários dos experimentos para os ataques de intrusão e discutida a metodologia de como foram gerados os *datasets*. Depois, os resultados dos indicadores de desempenho do algoritmo de aprendizado de máquina kNN foram apresentados e discutidos, demonstrando que, para o ataque SQL_-Injection a proposta de utilizar informação da telemetria é efetiva, obtendo resultados melhores que os ataques de DDoS.

Diante da efetividade de detectar e identificar ataques de DDoS e de intrusão, ações devem ser tomadas para diminuir, ou até cessar, os efeitos desses ataques. Dessa forma, o próximo capítulo se destina a discutir as possibilidades de mitigação dos ataques e como a infraestrutura da nuvem pode proporcionar ferramentas para que ações de mitigação possam ser tomadas, de acordo com a detecção e identificação desses ataques realizada previamente.

# Capítulo 6

## MITIGAÇÃO DE ATAQUES USANDO FERRAMENTAS DA NUVEM

Após a detecção e identificação dos ataques, como apresentado no Capítulo 5, é necessário algum tipo de ação para diminuir os seus efeitos, ou até cessá-los. Esta Tese propõe, de acordo com a **Hipótese H2**, apresentada na Seção 1.3, que os próprios mecanismos da infraestrutura da nuvem, podem tomar ações de mitigação contra esses ataques. Dessa maneira, o presente capítulo tem como objetivo apresentar e discutir a utilização do mecanismo de dimensionamento automático, disponível no próprio ambiente de nuvem, como forma de mitigação. Somado a isso, os cenários, os experimentos e resultados são expostos neste capítulo, mostrando ser possível utilizar essa estratégia para mitigar os ataques. Como discutido na Sessão 1.5, a transição da etapa de detecção e identificação, apresentada no capítulo anterior, para a mitigação não é realizada de forma orgânica, ou seja, a mitigação, neste capítulo, não é acionada a partir da detecção/identificação. A proposta do capítulo é apresentar a possibilidade, através de experimentos, de utilizar recursos nativos da infraestrutura da nuvem para realizar a mitigação de ataques.

## 6.1 Dimensionamento automático como mitigação

Dimensionamento automático, o *autoscaling*, como apresentado na Seção 2.3, pode ser utilizado como forma de mitigação dos ataques de DDoS, oferecendo mais recursos computacionais para o serviço hospedado na infraestrutura da nuvem. Além disso, defesas comumente utilizadas para um tipo de ataque podem ter sua ação aprimorada utilizando o dimensionamento automático. Essa combinação entre a defesa e a infraestrutura com o *autoscaling* pode ser proveitosa principalmente contra ataques de negação de serviço na camada de aplicação, que

tem como característica a geração de requisições similares as de clientes legítimos (ZARGAR; JOSHI; TIPPER, 2013). Os ataques DDoS na camada de aplicação podem ser classificados de duas formas: *low-* e *high-rate*.

Os ataques de DDoS na camada de aplicação já foram discutidos na Seção 2.4.1. Para o tipo de ataque *high-rate*, o principal expoente é o GET_Flood, enquanto que para o tipo *low-rate*, o principal ataque é o Slowloris (SHOREY et al., 2018).

Diante da dificuldade de identificar os clientes legítimos e maliciosos nos ataques de DDoS na camada de aplicação, uma possível defesa consiste em não realizar o descarte (bloqueio) de qualquer tipo de tráfego, pois existe uma alta probabilidade de clientes legítimos serem afetados. Como discutido no Capítulo 3, para o ataque do tipo *low-rate*, como o Slowloris, existe uma defesa, chamada SeVen (DANTAS; NIGAM; FONSECA, 2014), que se especializa em mitigar ataques *low-rate* utilizando estratégia de seletividade, descartando requisições baseado em funções de probabilidade. No entanto, a utilização da defesa SeVen contra ataques *high-rate* não é efetiva. Dessa forma, a proposta da presente Seção é combinar o dimensionamento automático oferecido pela infraestrutura da nuvem e a defesa seletiva SeVen, para criar uma forma de mitigação eficiente contra ataques de DDoS na camada de aplicação do tipo *low-* e *high-rate*.

Sendo assim, foram criados alguns cenários de testes para verificar as formas de mitigação, individualmente, SeVen e *autoscaling* contra os ataques Slowloris e GET_Flood, e verificar as duas defesas em conjunto. Os cenários, os experimentos, os resultados e a discussão serão apresentados a seguir.

### 6.1.1 Cenário de testes

Uma versão do OpenStack (Rocky) foi instalada em um servidor Dell PowerEdge T430, com 32 GB de memória RAM, 2 TB de HD e processador Intel Xeon E5-2609. Foi utilizado o sistema operacional Ubuntu Server, na versão 16.04.3. Para a avaliação da utilização do dimensionamento automático da infraestrutura da nuvem e de uma estratégia de seletividade, foram elaborados alguns cenários:

- **Cenário 1**: A defesa SeVen é introduzida, mas sem a característica de *autoscaling* estar ativada. Esse cenário tem como objetivo avaliar e verificar a efetividade da defesa SeVen em relação a ataques *high-rate*;
- **Cenário 2**: SeVen é removido e apenas o *autoscaling* é utilizado. Nesse cenário, o objetivo é avaliar se os ataques de DDoS do tipo *low-rate* podem ser mitigados utilizando o

processo de *autoscaling*;

- **Cenário 3**: SeVen e *autoscaling* são usados, combinando ambas as defesas para prover uma proteção completa contra ataques de DDoS *low-* e *high-rate*.

Em cada um desses três cenários, os experimentos foram realizados utilizando dois tipos de ataques de DDoS: *low-* e *high-rate*, os dois isolados e depois combinados. Para o ataque *low-rate*, foi escolhido o Slowloris, com seu próprio *script* em Perl (ANONYMOUS, 2013), enquanto que para o ataque *high-rate*, o ataque GET_Flood foi escolhido. A ferramenta GoldenEye[1] foi utilizada para gerar o ataque GET_Flood.

Em todos os cenários, duas métricas diferentes foram utilizadas para avaliar a eficiência das defesas: (i) a disponibilidade do serviço para clientes legítimos, ou seja, a porcentagem dos clientes legítimos que requisitaram o serviço e obtiveram sucesso na resposta; (ii) tempo de Serviço (*Time-to-Service* - TTS), que está relacionado ao tempo de resposta do serviço, em segundos. Em um servidor *web*, o TTS é o tempo que o cliente requisita uma página *web*, o servidor recebe essa requisição, processa e a resposta retorna ao cliente. As métricas Disponibilidade e o TTS são coletadas pela ferramenta Siege[2], que simula os clientes legítimos e produz um relatório da sua utilização.

Um total de 200 clientes simultâneos[3] foram simulados durante 30 minutos de experimento utilizando a ferramenta Siege. Cada cliente requisita repetidamente a página *web* do servidor de forma aleatória. Ou seja, no momento em que o cliente solicita e recebe a resposta da página, ele espera um tempo aleatório (variando entre 0 a 4 segundos) para enviar outra requisição.

Em todos os cenários, duas máquinas virtuais foram criadas, cada uma com 1 vCPU e 1 GB de memória RAM. A primeira VM foi utilizada para realizar os ataques de DDoS, já a segunda, para simular os clientes legítimos realizando as requisições para o servidor *web*. E uma terceira máquina virtual foi criada, também com 1 vCPU e 1 GB de RAM, que agirá como o servidor *web* sofrendo ataque. O servidor Apache na versão 2.4 foi utilizado nessa VM.

Para o cenário 1, apresentado na Figura 6.1, a defesa SeVen foi colocada próxima ao servidor Apache, protegendo o servidor *web* hospedado no ambiente de nuvem OpenStack.

Enquanto que no cenário 2, apresentado na Figura 6.2, a defesa SeVen foi removida e o servidor *web* foi instanciado com a funcionalidade de *autoscaling*. Dessa forma, o módulo de

[1]Página web: <https://github.com/jseidl/GoldenEye>. Acessado em 18 de agosto de 2019.

[2]Página web: <https://www.joedog.org/siege-home/>. Acessado em 18 de agosto de 2019.

[3]A simulação de 200 clientes não gera mais que 10% de utilização da CPU do servidor *web*. A escolha desse número se dá para que não seja os clientes que acionem o processo de *autoscaling*, mas que seja um número considerável de clientes legítimos.

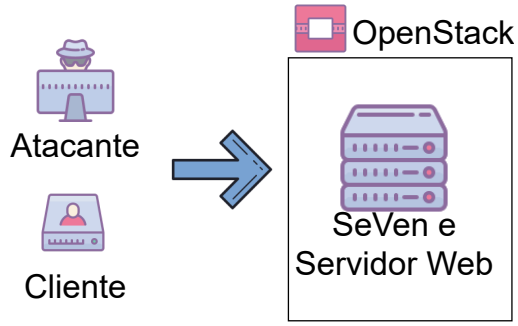



**Figura 6.1: Cenário de experimento apenas com a defesa SeVen**

monitoramento Ceilometer do OpenStack verifica a utilização da CPU, com um limiar de 70%; se esse limite for excedido, uma nova máquina virtual com o servidor *web* será instanciada. O valor do limiar da CPU poderia ser alterado para qualquer outro valor, sendo escolhido para esse experimento o valor de 70%[4]. Como será instanciado uma nova máquina virtual, é necessário realizar um balanceamento de carga entre as várias instâncias dos servidores *web*. Para isso, uma outra máquina virtual foi colocada na frente dos servidores *web*, utilizando o serviço do HAProxy, e com o algoritmo de balanceamento de carga Round Robin.

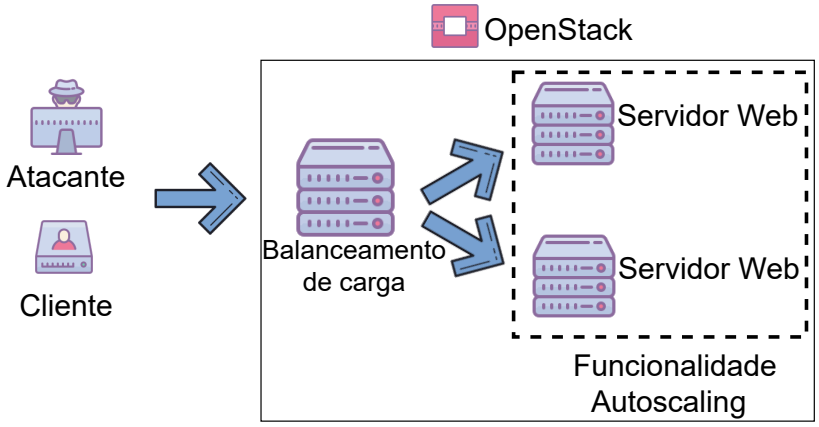



**Figura 6.2: Cenário de experimento apenas com a funcionalidade *autoscaling***

No cenário 3, apresentado na Figura 6.3, a máquina virtual em que está hospedado o servidor *web* foi iniciada contendo a defesa SeVen e a funcionalidade de *autoscaling*. O objetivo da utilização do SeVen e do dimensionamento automático é a possibilidade de criar uma defesa contra ataques *low-* e *high-rate* na camada de aplicação, em que o SeVen oferece disponibilidade para o ataque Slowloris, e a instanciação de outro servidor *web* automaticamente é a mitigação dos efeitos do ataque GET_Flood.

[4]O valor do limiar deve ser objeto de uma melhor verificação, pois, caso seja baixo, o mecanismo de dimensionamento automático pode ser iniciado sem necessidade, criando novas instâncias sem realmente precisar. Por outro lado, se o limitar for alto, a inserção de novas instâncias pode demorar, fazendo com que o dimensionamento automático não seja muito efetivo.

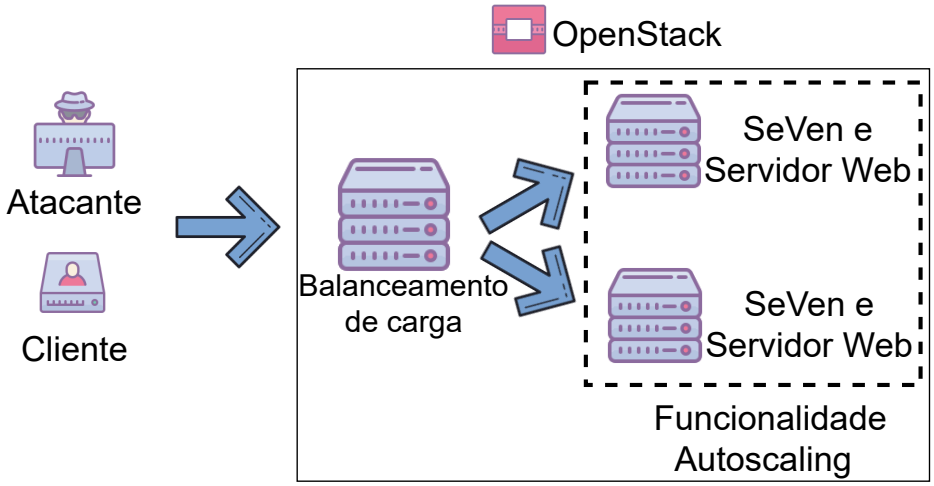



**Figura 6.3: Cenário de experimento com a funcionalidade *autoscaling* e a defesa SeVen (CORRÊA et al., 2019b)**

**Tabela 6.1: Resultados dos experimentos utilizando apenas SeVen, com ataques *low-* e *high-rate***

| Cenário | Disponibilidade (%) | *Time-to-Service (s)* |
|---|---|---|
| *low-rate* | 95 | 0,06 |
| *high-rate* | 14,2 | 8,8 |
| Ambos | 13,6 | 9,1 |

## 6.1.2 Resultados

### 6.1.2.1 Resultados do cenário 1

Os resultados dos experimentos do cenário 1 são apresentados na Tabela 6.1, sendo os resultados contra *low-rate* os obtidos por Dantas *et al.* (DANTAS; NIGAM; FONSECA, 2014). Nos experimentos, foi verificada a deficiência da utilização da defesa SeVen em ataques de DDoS do tipo *high-rate*, tendo a média da disponibilidade em 14,2% durante o ataque GET_Flood. Isso se deve ao fato de que a defesa não consegue atender à maioria das requisições para o servidor *web*, visto que é a característica do ataque *high-rate* de enviar várias requisições simultaneamente. Dos 14,2% dos clientes legítimos atendidos, a média do TTS foi de 8,8 segundos. Em outras palavras, o cliente realiza a requisição, o servidor *web* recebe, processa e responde novamente ao cliente em, média, 8,8 segundos. Esse tempo de resposta é extremamente alto para um cliente *web*.

Nos experimentos utilizando SeVen defendendo contra o ataque combinado de Slowloris e GET_Flood, a defesa ofereceu uma disponibilidade apenas para 13,6% dos clientes legítimos, apresentando uma piora comparado ao cenário com apenas o ataque GET_Flood. Igualmente ocorreu para o TTS, levando 9,1 segundos para o cliente receber a resposta da página do ser-

vidor. Portanto, como apresentado, não é recomendada a utilização da defesa SeVen contra ataques *high-rate*. Sendo assim, a utilização do *autoscaling* pode auxiliar a defesa SeVen a ter um bom desempenho contra ataques de maior taxa.

#### 6.1.2.2 Resultados do cenário 2

A característica de *autoscaling* é ativada quando algum recurso computacional, como a utilização da CPU ou de memória, ultrapassa algum limite estabelecido previamente. Assim, os experimentos realizados nesse cenário visam verificar os ataques de DDoS que geram um aumento na utilização desses recursos em um servidor *web* e se este aumento foi capaz de ativar este *trigger*.

Experimentos foram realizados com diferentes tipos de ataques: *low-rate*, *high-rate* e ambos combinados. Os resultados incluem disponibilidade, TTS e a taxa de utilização da CPU, contendo a média e o valor máximo obtido. Esses resultados são apresentados na Tabela 6.2.

Para o ataque Slowloris, foi verificada uma disponibilidade de 1,77% dos clientes legítimos. Nesse experimento, não foi encontrado aumento na utilização da CPU, apresentando uma média de 1% e um pico de 7% de uso. Isso se deve à característica principal do ataque Slowloris, que não envia grande número de requisições à vítima, resultando em baixa utilização da CPU. Devido à baixa utilização do processador, o serviço de *autoscaling* não foi acionado, pois o sistema não considerou necessário criar outra instância de servidor *web* na infraestrutura da nuvem. Para o TTS, foi verificado que um cliente leva 13,7 segundos para requisitar a página *web* e receber a resposta dessa requisição, sendo considerado um alto tempo de resposta no cenário de serviço *web* na infraestrutura da nuvem.

No experimento utilizando o ataque GET_Flood, o cenário de *autoscaling* oferece uma disponibilidade para clientes legítimos de 100%. Por consequência, constatou-se que a média de utilização da CPU era de 68%, e os valores mais altos atingiram 100%. Durante os 30 minutos do experimento, um total de 5 (cinco) instâncias criadas pela funcionalidade de dimensionamento foi observado. Quanto ao TTS, um cliente legítimo leva apenas 0,87 segundos para solicitar e receber a página *web*.

No cenário com os dois ataques, Slowloris e GET_Flood, o resultado foi similar ao cenário com apenas o ataque GET_Flood. Como nos cenários anteriores, o ataque GET_Flood gera uma quantidade maior de pacotes na rede do que o Slowloris, deixando-o sem eficácia. Portanto, foi observada uma disponibilidade de 100% para clientes legítimos, com 0,85 segundos para o TTS. A infraestrutura da nuvem, durante os 30 minutos do experimento, instanciou um total de

**Tabela 6.2: Resultados utilizando apenas *autoscaling*, com ataques *low-* e *high-rate***

| Cenários | Disponibilidade (%) | *Time-to-Service (s)* | Média de CPU (%) | CPU máximo (%) |
|---|---|---|---|---|
| *low-rate* | 1,77 | 13,7 | 1 | 7 |
| *high-rate* | 100 | 0,87 | 68 | 100 |
| Ambos | 100 | 0,85 | 67 | 100 |

5 (cinco) servidores *web*, com uma média de 67% de utilização da CPU, com um valor máximo de 100%.

A partir desses resultados, é possível observar que a funcionalidade de *autoscaling* consegue ser efetiva contra ataques de *high-rate*, como o GET_Flood. Dessa forma, a funcionalidade de dimensionamento automático pode ser utilizada para mitigar ataques de DDoS.

#### 6.1.2.3 Resultados do cenário 3

Como apresentado anteriormente, SeVen consegue mitigar ataques *low-rate*, mas não os ataques *high-rate*. Por outro lado, a funcionalidade de *autoscaling* provê uma disponibilidade adequada para clientes legítimos diante de ataques do tipo *high-rate*, mas é ineficaz contra ataques *low-rate*. Assim, a complementaridade entre a funcionalidade de *autoscaling* da infraestrutura da nuvem e a seletividade da defesa SeVen consegue gerar uma defesa efetiva contra os dois tipos de ataques de DDoS na camada de aplicação.

A Tabela 6.3 apresenta os resultados obtidos durante os experimentos em que o SeVen e *autoscaling* foram utilizados para mitigar os ataques, e a combinação desses ataques. Os resultados de disponibilidade e TTS são apresentados, além da média e do valor máximo da utilização da CPU.

Quando há apenas o ataque Slowloris, no cenário de uso combinado, foi observada uma disponibilidade média de 95%, com TTS apenas 0,09 segundos. Nesses experimentos, o recurso *autoscaling* não foi acionado devido ao fato de que a utilização da CPU foi de apenas 13%, em média, com um máximo de 20% durante o ataque Slowloris.

No experimento com apenas o ataque GET_Flood, foi verificado que 100% dos clientes que requisitam o serviço obtiveram resposta, com 0,91 segundos de TTS. A funcionalidade *autoscaling* foi ativada, instanciando um total de 5 (cinco) servidores *web* durante o experimento, obtendo uma média de utilização da CPU de 68%, e 100% como utilização máxima.

Finalmente, os experimentos com os dois tipos de ataques de DDoS, *low-* e *high-rate*,

**Tabela 6.3: Resultados dos experimentos utilizando SeVen e o *autoscaling*, com os ataques *low-* e *high-rate***

| Cenários | Disponibilidade (%) | *Time-to-Service (s)* | Média de CPU (%) | CPU máximo (%) |
|---|---|---|---|---|
| *low-rate* | 95 | 0,07 | 13 | 20 |
| *high-rate* | 100 | 0,91 | 68 | 100 |
| Ambos | 100 | 0,87 | 66 | 100 |

simultaneamente, foram realizados. É importante ressaltar que, agora, ambas as defesas foram utilizadas. Nesse caso, foi obtido 100% de disponibilidade para clientes legítimos, com um TTS de 0,87 segundos. O uso da funcionalidade de *autoscaling* foi verificado, sendo instanciado um total de 5 (cinco) servidores *web* durante o experimento.

#### 6.1.2.4 Discussão dos resultados

A média da disponibilidade dos clientes legítimos no serviço *web* na infraestrutura da nuvem é apresentado na Figura 6.4. Os três cenários elaborados para os experimentos são exibidos, realizando dois tipos diferentes de ataques de DDoS: Slowloris e GET_Flood, isolados e executados juntos. O Slowloris é representado pelas barras em azul, enquanto que o ataque GET_Flood é representado pelas barras em laranja, e a combinação dos dois ataques é apresentada pelas barras em verde.

O cenário 1, em que apenas SeVen foi utilizado, está representado pelas três primeiras barras, da esquerda para a direita. Já o cenário 2, utilizando apenas o *autoscaling*, está representado pelas três barras ao centro. Finalmente, as últimas três barras, representam os resultados do cenário 3, em que foram incluídos SeVen e *autoscaling.*

De acordo com a Figura 6.4, o ataque Slowloris é efetivo no cenário em que a defesa SeVen não está presente (cenário 2). Nos experimentos com o ataque GET_Flood, se apenas SeVen for utilizado, a disponibilidade é baixa. No entanto, no cenário em que está presente o *autoscaling*, o ataque GET_Flood não é efetivo, pois o *autoscaling* mitiga esse tipo de ataque oferecendo mais recursos para o serviço. Nesse cenário, a disponibilidade para clientes legítimos foi de 100%. Por fim, no que os dois tipos de ataques são realizados em conjunto, no cenário em que *autoscaling* e a defesa SeVen são utilizadas, a mitigação é efetiva apresentando uma disponibilidade de 100% para os clientes legítimos. Estes resultados mostram que, para o cenário proposto, a estratégia de aumentar a quantidade de instâncias do serviço, de acordo com a demanda e automaticamente, tornou-se eficaz para o ataque GET_Flood.

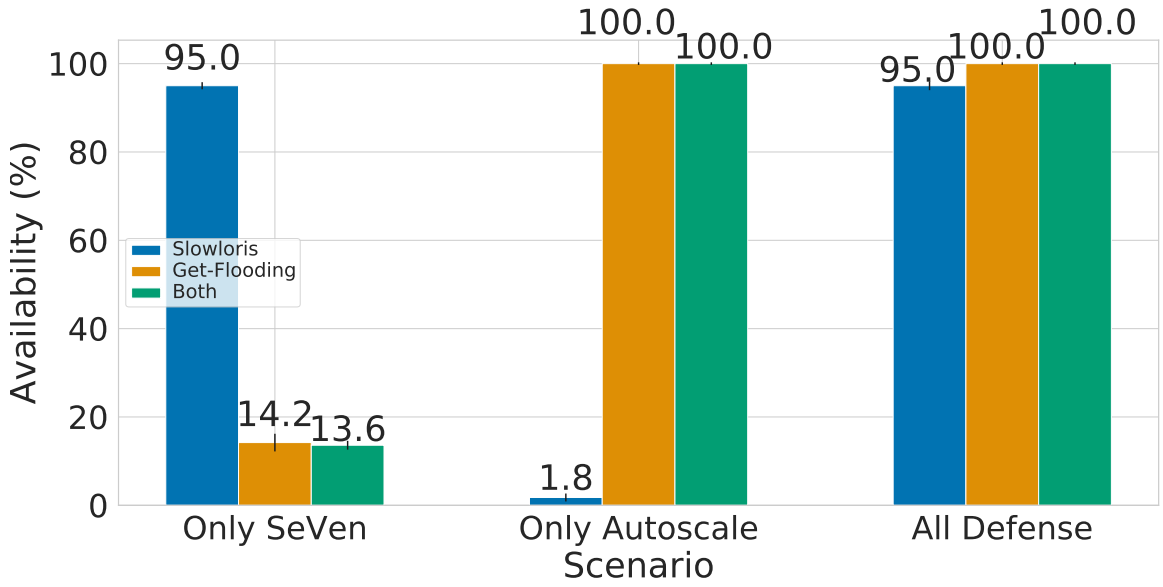



**Figura 6.4: Disponibilidade do serviço *web* na infraestrutura da nuvem para clientes legítimos nos três cenários (CORRÊA et al., 2019b)**

Em relação à métrica *Time-to-Service*, os resultados (em segundos) são sumarizados na Figura 6.5. O ataque Slowloris está representado nas barras em azul, enquanto que o ataque GET_Flood está representado pelas barras em laranja. Por último, os experimentos utilizando ambos os ataques estão representados pelas barras em verde.

A Figura 6.5 apresenta todos os cenários em que os experimentos foram realizados: as três primeiras barras são do cenário 1 (apenas SeVen); as três barras no meio são utilizando apenas o *autoscaling* (cenário 2); e, finalmente, as últimas três barras referem-se aos experimentos com ambas defesas (cenário 3). Quando há apenas o ataque Slowloris, um TTS extremamente elevado é verificado em cenários em que o SeVen não está presente, mas, se SeVen estiver, o TTS diminui. No momento em que há somente o ataque GET_Flood, no cenário 1, o TTS é considerado alto para clientes legítimos esperarem uma resposta do servidor. No cenário 3, usando o recurso de *autoscaling* da infraestrutura da nuvem como uma defesa contra ataques de *high-rate*, combinado com a defesa SeVen, foi alcançado um TTS de 0,87 segundos. Esse resultado mostra que o uso de múltiplas instâncias do serviço da infraestrutura da nuvem, sendo automaticamente alocados de acordo com as necessidades da aplicação, não tem influência no tempo de resposta aos clientes legítimos.

Dessa maneira, como observação final, a estratégia de utilizar o *autoscaling* da infraestrutura fornecida pelo ambiente de nuvem para adicionar dinamicamente mais VMs é uma forma eficaz de mitigar ataques de DDoS. Uma deficiência nessa estratégia pode ser identificada: existe a necessidade de limitar o recurso de *autoscaling* imposto pela própria infraestrutura. Assim, se ocorrer um ataque com grande volume de tráfego e a infraestrutura da nuvem não

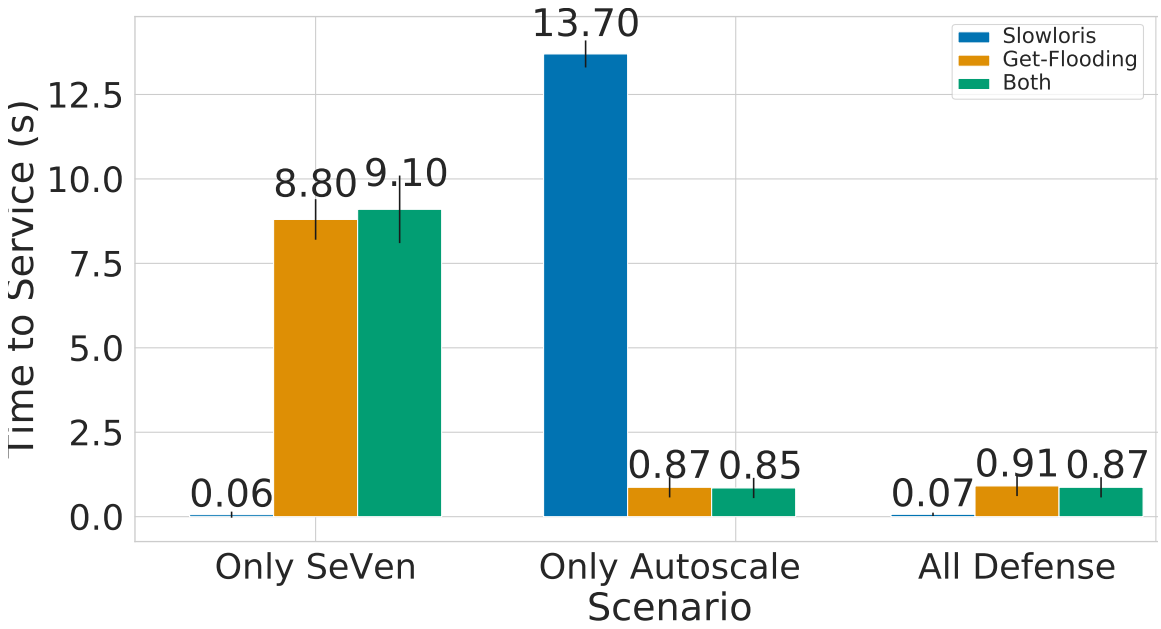



**Figura 6.5: *Time-to-Service*, em segundos, em todos cenários (CORRÊA et al., 2019b)**

for suficientemente grande, a indisponibilidade continuará a ser observada pelos clientes legítimos. Sendo assim, é necessário que a nuvem tenha um esquema de auto gerenciamento da sua infraestrutura, sabendo organizar e dedicar os recursos necessários para o dimensionamento automático e também para os outros clientes hospedado nesta nuvem. Nesse caso, outras alternativas para mitigar esses ataques podem ser empregadas, como as já propostas na literatura e que a infraestrutura da nuvem pode dar suporte na implementação.

## 6.2 Considerações do Capítulo

Este capítulo apresentou mecanismos da computação em nuvem para realizar a mitigação dos ataques após ser realizada a detecção e identificação desses ataques. Foi apresentado o mecanismo de *autoscaling*, que pode oferecer mais recursos com objetivo de mitigar ataques. Dessa forma, clientes legítimos obtêm uma maior disponibilidade quando o mecanismo de dimensionamento automático é utilizado. Já o próximo capítulo versará sobre as conclusões obtidas com esta Tese. Além disso, serão apresentadas as indicações de trabalhos futuros.

# Capítulo 7

## CONCLUSÃO

Diante do que foi exposto e dos resultados obtidos, verifica-se que a hipótese levantada por essa Tese é verdadeira. Ou seja, que funcionalidades e recursos nativos da computação em nuvem podem ser utilizados para detectar, identificar e mitigar ataques a serviços hospedados no ambiente de nuvem e aos clientes desses serviços. Inicialmente, no Capítulo 4, foi verificada a relação entre os ataques e as consequências geradas nos recursos computacionais da vítima, e que é possível fazer distinção entre o ataque e o tráfego normal, analisando a utilização dos recursos afetados pelo ataque. Ou seja, ataques diferentes vão gerar níveis de utilização em subconjuntos distintos de recursos computacionais, dessa forma, se um ataque é realizado contra um serviço hospedado em uma infraestrutura da nuvem, e que as informações da telemetria desse serviço estão sendo coletados, então algoritmos de aprendizado de máquina podem detectar/identificar esses subconjuntos de recursos computacionais, e, consequentemente, o ataque.

Posteriormente, no Capítulo 5, foram realizados experimentos para verificar a proposta de utilizar informação da telemetria para detectar e identificar ataques. Para isso, durante esses experimentos, dados da telemetria foram coletados e guardados em um *dataset*. Diante desse *dataset*, foi realizado um pré-processamento e uma seleção de características, descrito na Sessão 5.1.2, alcançando assim o primeiro objetivo específico elencado na Sessão 1.4. Após a seleção de características, algoritmos de aprendizado de máquina foram selecionados e treinados para realizar a detecção e identificação dos ataques de negação de serviço. Na fase de detecção, o algoritmo kNN e RF obtiveram uma acurácia maior que 85%, e, na segunda fase, obtiveram uma acurácia acima de 88% identificando os ataques de DDoS SYN_Flood e GET_Flood. Esses resultados, apresentados na Sessão 5.1.4, atinge o segundo objetivo específico apresentado na Sessão 1.4. Ainda no Capítulo 5, foi apresentado que os algoritmos de aprendizado de máquina, com os dados da telemetria, obtiveram uma acurácia de 94% na tarefa de classificar o ataque de intrusão SQL_Injection. Esse resultado, da Sessão 5.2, demonstra que alguns ataques

de intrusão podem ser corretamente classificados pela telemetria nativa da infraestrutura da nuvem, alcançando o terceiro objetivo específico desta Tese. Em resumo, a **Hipótese 1** levantada na Seção 1.3 foi provada, tendo os três primeiros objetivos específicos atingidos.

É importante destacar que o uso da telemetria tem um limite, imposto justamente no grau de utilização dos recursos computacionais pelos ataques. Se um ataque, seja de negação de serviço ou de intrusão, não altere a utilização de recursos computacionais, então não será possível detectar, e, muito menos, identificar esse ataque. Por exemplo, o ataque de intrusão de força bruta contra o protocolo SSH é um exemplo de ataque que apenas a telemetria da infraestrutura da nuvem não conseguirá detectar esse ataque. Nesse tipo de cenário, será necessário combinar a telemetria com outras informações como as de fluxo ou dos pacotes de rede.

No Capítulo 6, foi apresentado o mecanismo da infraestrutura da nuvem para realizar a mitigação dos ataques, verificando que o dimensionamento automático, o *autoscaling*, consegue prover disponibilidade a clientes legítimos aumentando a quantidade de servidores *web* dinamicamente mesmo quando está sofrendo um ataque de GET_Flood. Dessa forma, o quarto objetivo específico foi atendido. E, de modo geral, o quinto objetivo específico também foi atendido, pela construção e realização dos experimentos. Sendo assim, como demonstrado na Sessão 6.1.2, foi iniciada a verificação da **Hipótese 2**, mas que precisa ainda ser aprofundada com outros recursos disponíveis na infraestrutura da nuvem.

Destaca-se que há duas limitações na utilização do *autoscaling* para realizara a mitigação: (i) o ataque não aumentar a utilização do recurso computacional que ultrapasse o limiar pré-estabelecido para ocorrer o dimensionamento automático, como foi o caso do ataque Slowloris; (ii) limite de recursos destinados para o *autoscaling* imposto pela própria infraestrutura física. Se for gerado um volume grande de ataque, pode ser que a infraestrutura da nuvem não tenha a capacidade de mitigar esse ataque, e os clientes experimentarem indisponibilidade.

Portanto, respondendo às duas primeiras Questões de Pesquisa elencadas na Seção 1.2: é sim possível realizar a detecção e identificação de ataques de negação de serviço e de intrusão, utilizando a telemetria nativa da infraestrutura de nuvem. E também é possível, utilizando os artifícios habilitadores do ambiente de nuvem, a mitigação, tomando uma ação baseada na detecção e identificação desses ataques.

Por fim, destaca-se como uma vantagem da abordagem da presente Tese a possibilidade de detectar (e, consequentemente, realizar a mitigação) ataques em que o modelo de aprendizado de máquina não foi treinado para detectar. Na Sessão 5.1.4.4, foi apresentado o experimento com o ataque PING_Flood e que os algoritmos de aprendizado de máquina não foram treinados para detectarem esse ataque. Os algoritmos obtiveram uma acurácia de 74% na detecção desse

ataque, sendo uma evidência da possibilidade de generalização da proposta de detectar ataques desconhecidos.

## 7.1 Trabalhos Futuros

A partir desta Tese, pode-se observar quatro caminhos de trabalhos futuros: (i) criação de um processo de escolha dos mecanismos de mitigação; (ii) integração de um sistema único de controle, que permita realizar a detecção, identificação e mitigação em tempo real; (iii) detectar a existência de *hosts* infectados que integram *botnets*; (iv) replicar esta proposta em contêineres, utilizando a telemetria oferecida pelo próprio gerenciador de contêiners. Aprofundando-se na primeira proposta, existem alguns questionamentos: quais seriam as ações de mitigação recomendadas para cada tipo de ataque? Apesar de existir na literatura diversas propostas de mitigação, inclusive apresentadas no Capítulo 6, qual seria a melhor utilizada para cada tipo de ataque? Como seria esse processo de escolha? Seriam tomadas mais de uma ação? Se a ação não surtir efeito, qual outra possibilidade de mitigação poderia ser utilizada? Existe uma lógica/inteligência por trás desse processo de escolha? Nessa direção, esta Tese trouxe inteligência para as fases de detecção e identificação dos ataques, mas o processo de mitigação também pode se aproveitar da inteligência na escolha da ação a ser tomada. Somado a isto, essa indicação de trabalho futuro deve considerar a avaliação de desempenho e de capacidade, com possibilidade de auto gerenciamento da nuvem, com processos elásticos de alocação de capacidade na infraestrutura da nuvem. Este trabalho futuro visa complementar, com mais pesquisa, investigação e experimentos a **Hipótese 2** levantada nesta Tese, de que, com recursos nativos da infraestrutura da nuvem é possível realizar a mitigação de ataques.

Na segunda possibilidade de trabalhos futuros, indica-se a criação de um sistema único de controle, que faça a detecção, identificação e mitigação dos ataques, de forma orgânica e integrada com a própria infraestrutura da nuvem. Após o primeiro trabalho futuro efetivamente executado, essa segunda indicação de trabalho posterior tende a ser de desenvolvimento, integrando com as APIs (*Application Programming Interface*) que a própria infraestrutura da nuvem oferece. Com essa integração, a detecção, identificação e mitigação podem ser realizadas em tempo real. Além disso, com o sistema único criado, testes com mais ataques de DDoS e de intrusão podem ser realizados, aumentando a gama de ataques testados pelo sistema. Por fim, neste cenário, indica-se a utilização de outras métricas disponíveis na telemetria, por exemplo, as informações das redes virtuais internas no ambiente da nuvem, como as métricas de estatísticas de fluxos de rede.

Dessa forma, com a inteligência no processo de escolha dos mecanismos de mitigação e com um sistema único de controle, integrado totalmente à infraestrutura da nuvem, essa arquitetura criada pode ser utilizada como um ambiente de testes de segurança. Ou seja, a utilização dessa infraestrutura para instanciar *honeypots*[1], que serão atacados por usuários maliciosos reais, mas que existe toda uma infraestrutura que permite controlar os efeitos da ação desse atacante. Dessa forma, o usuário malicioso irá realizar ataques, em que o sistema único de controle ficará monitorando e, assim, controlará os efeitos do ataque, permitindo que prossiga até que seja verificado um nível mais danoso do ataque. Nesse momento, o próprio sistema de controle irá cessar a conexão com o atacante. Com isso, todas as informações que foram inseridas, e todos os passos executados pelo atacante poderão ser resgatadas pelo ambiente de testes e servindo para atualização dos modelos de reconhecimento dos ataques.

Nesta Tese, o objetivo é verificar se algum serviço hospedado na infraestrutura da nuvem é vítima de um ataque. Na terceira proposta de trabalhos futuros, a ideia é inverter a posição da detecção e verificar se existe na infraestrutura da nuvem algum *host* que foi infectado e está integrando uma *botnet* com objetivo de gerar ataques. A proposta é que, se é possível detectar esse cenário também utilizando a telemetria nativa da nuvem. Pois, se é possível detectar que está acontecendo um ataque, como provado pela presente Tese, em que há uma mudança de comportamento nos recursos computacionais monitorado pela telemetria, então, a geração do ataque também pode gerar mudança de comportamento na telemetria. E, a partir da observação desses dados, com auxílio dos algoritmos de aprendizado de máquina, é possível detectar que algum elemento hospedado na própria nuvem é um gerador de ataques.

Em conclusão, a última proposta de trabalhos futuros é a evolução natural dos serviços: a migração para contêineres. Diversos serviços na Internet deixaram de utilizar máquinas virtuais, e estão utilizando o conceito de micro-serviço em contêineres, utilizando apenas os recursos necessários para a aplicação e com a garantia do isolamento entre os processos no servidor físico. Essa migração alcançou diversas plataformas de nuvem, que estão oferecendo não apenas máquinas virtuais, mas também contêineres, como o caso da AWS, IBM Cloud, Microsoft Azure. Dessa forma, a proposta é utilizar as informações de telemetria oferecido pelos gerenciados de contêineres, por exemplo, o Kubernets[2], para detectar e identificar ataques. Essa mudança acarreta também a utilização de novas métricas, inclusive relacionadas especificamente a aplicação em execução no contêiner, possibilitando a detecção e identificação de outros tipos de ataques.

[1] *Honeypots* são servidores, com falhas conhecidas, que ficam disponíveis na Internet com o intuito de receber ataques e, com isso, conhecer e estudar sobre esses ataques.

[2] Página web: <https://kubernetes.io>

## 7.2 Artigos publicados

A seguir, os artigos publicados que estão ligados à Tese:

1. **CORREA, J. H. G. M.** ; CIARELLI, P. M.; RIBEIRO, M. R. N.; VILLACA, R. S.. ML-Based DDoS Detection and Identification using Native Cloud Telemetry Macroscopic Monitoring. In Journal of Network and Systems Management, v. 29, p. 13, 2021. **Qualis B1.**

2. **CORREA, J. H. G. M.**; SOUSA JUNIOR, E. A. ; Fonseca, Iguatemi Eduardo da ; NIGAM, V. ; RIBEIRO, M. R. N. ; VILLACA, R. S. . Selectivity and Autoscaling as Complementary Defenses for DDoS Protection to Cloud Services. In: IEEE 8th International Conference on Cloud Networking (CloudNet), 2019, Coimbra. EEE 8th International Conference on Cloud Networking (CloudNet 2019). New York: IEEE, 2019. **Qualis B1**.

3. **CORREA, J. H. G. M.**; SOUSA JUNIOR, E. A. ; SANTOS, P. B. ; NIGAM, V. ; FONSECA, I. E. ; GUIMARAES, R. L. ; MARTINELLO, M. ; RIBEIRO, M. R. N. ; VILLACA, R. S. . Dispositivos conectados a serviço web em nuvem: complementariedade entre infraestrutura e seletividade para proteção contra DDoS. In: XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2018), 2018, Campos do Jordão/SP. Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC 2018). Porto Alegre/RS: SBC, 2018.

Além disso, cita-se o prêmio recebido pelo autor durante o LANCOMM (*Latin American Student Workshop on Data Communication Networks*), sendo um dos seis melhores trabalhos apresentados no *Workshop*, cujo o título é "On Machine Learning DoS Attack Identification from Cloud Computing Telemetry", e que contém os resultados preliminares da Tese (CORRÊA et al., 2019a). O certificado desse prêmio está reproduzido na Figura 7.1.

A seguir, tem-se outra lista de artigos publicados, minicursos e capítulos de livro em que o autor da presente Tese participou, os quais oferecem assuntos relevantes para a Tese, mas não estão diretamente ligados:

1. SANTOS, P. B. ; **CORREA, J. H. G. M.** ; CARDOSO, D. G. ; QUEIROZ, F. M. ; PICORETTI, R. ; CARMO, A. P. ; RIBEIRO, M. R. N. ; VILLACA, R. S. . Monitoramento de Recursos para Aplicações de Robótica em Espaços Inteligentes baseados em uma Nuvem OpenStack. In: XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas


**Figura 7.1: Certificado do LANCOMM pelo prêmio recebido**

Distribuídos (SBRC 2018), 2018, Campos do Jordão/SP. XVI Workshop em Clouds e Aplicações (WCGA 2018). Porto Alegre/RS: SBC, 2018.

2. COMARELA, G. ; FRANCO, G. ; TROIS, C. ; LIBERATO, A. ; MARTINELLO, M. ; **CORREA, J. H. G. M.**; VILLACA, R. S. . Introdução à Ciência de Dados: Uma Visão Pragmática utilizando Python, Aplicações e Oportunidades em Redes de Computadores. In: Alberto Egon Schaeffer Filho; Weverton Luis da Costa Cordeiro; Miguel Elias Mitre Campista. (Org.). Minicursos do SBRC 2019. 37ed.Porto Alegre: Sociedade Brasileira de Computação, 2019, v. 1, p. 246-295. **Minicurso do SBRC e Capítulo de livro nos anais.**

3. SOUZA, RENAN ; TROIS, CELIO ; TURCHETTI, ROGERIO ; MARTINELLO, MAGNOS ; **CORREA, JOAO HENRIQUE G.** ; MAFIOLETTI, DIEGO ROSSI ; BONA, LUIS C. E. ; LIMA, JOAO CARLOS D. ; MACHADO, ALENCAR . MLFV: Network-Aware Orchestration for Placing Chains of Virtualized Machine Learning Functions. In: GLOBECOM 2019 2019 IEEE Global Communications Conference, 2019, Waikoloa. 2019 IEEE Global Communications Conference (GLOBECOM), 2019. p. 1. **Qualis A1**.

4. SOUSA JUNIOR, E. A. ; **CORREA, J. H. G. M.** ; CERAVOLO, I. A. ; GUIMARAES, R. L. ; RIBEIRO, M. R. N. . Teoria e Prática na Virtualização de Funções de Redes em Ambiente de Nuvem. In: Paulo Ribeiro Lins Júnior. (Org.). Minicursos do SBrT 2018. 1ed.Campina Grande: Sociedade Brasileira de Telecomunicações, 2019, v. 1, p. 1-45. **Minicurso do SBrT e Capítulo de livro dos minicursos.**

5. SANTOS, RAPHAEL A. G. DOS ; LEAL-JUNIOR, ARNALDO G. ; RIBEIRO, MOISES R. N. ; BENINCA, EDUARDO A. ; MARQUES, CARLOS A. F. ; PEREIRA, LUIS ; ANTUNES, PAULO ; **CORREA, JOAO H. G. M.** ; PONTES, MARIA J. ; NETO, ANSELMO FRIZERA ; ANDRE, PAULO S. B. . Datacenter Thermal Monitoring Without Blind Spots: FBG-Based Quasi-Distributed Sensing. IEEE Sensors Journal, v. 21, p. 9869-9876, 2021.

# REFERÊNCIAS


AGGARWAL, C. C. *Data mining: the textbook*. [S.l.]: Springer, 2015. ISBN 978-3-319-14142-8.

AKAMAI. *Q3/2017 - State of the Internet Security Report*. 2017. Https://www.akamai.com/de/de/multimedia/documents/state-of-the-internet/q3-2017-state-of-the-internet-security-report.pdf. Acessado: 15-12-2017.

AKAMAI. <*https://www.akamai.com/br/pt/multimedia/documents/state-of-the-internet/soti-security-web-attacks-and-gaming-abuse-report-2019.pdf*>. 2019. Acesso: 05-10-2020.

ALHARBI, T. et al. Smart and lightweight ddos detection using nfv. In: *Proceedings of the International Conference on Compute and Data Analysis*. New York, NY, USA: ACM, 2017. (ICCDA '17), p. 220–227. ISBN 978-1-4503-5241-3. Disponível em: <http://doi.acm.org/10.1145/3093241.3093253>.

ALMEIDA, L. C. de. Ferramenta computacional para identificação e bloqueio de ataques de negação de serviço em aplicações web. Master Thesis. 2013.

AMARAL, P. et al. Machine learning in software defined networks: Data collection and traffic classification. In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2016.

AMJAD, A. et al. Detection and mitigation of ddos attack in cloud computing using machine learning algorithm. *EAI Endorsed Transactions on Scalable Information Systems*, European Alliance for Innovation (EAI), v. 6, n. 23, 2019.

ANONYMOUS. *Slowloris Tool, 2013*. [S.l.]: http://ha.ckers.org/slowloris/, 2013. Acessado: 25-11-2014.

ANTONAKAKIS, M. et al. Understanding the mirai botnet. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017. p. 1093–1110. ISBN 978-1-931971-40-9. Disponível em: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>.

ARABNEJAD, H. et al. An auto-scaling cloud controller using fuzzy q-learning - implementation in openstack. In: AIELLO, M. et al. (Ed.). *Service-Oriented and Cloud Computing*. Cham: Springer International Publishing, 2016. p. 152–167. ISBN 978-3-319-44482-6.

ARKKO, J. *Centralised Architectures in Internet Infrastructure*. [S.l.], 2019. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-arkko-arch-infrastructure-centralisation-00>.

AZMANDIAN, F. et al. Securing virtual execution environments through machine learning-based intrusion detection. p. 1–6, 2015.

BAO, N. K.; AHN, S. W.; PARK, M. An elastic-hybrid honeynet for cloud environment. *CSEIT, NCS, SPM, NeTCoM*, p. 117127, 2018.

BHARDWAJ, K.; MIRANDA, J. C.; GAVRILOVSKA, A. Towards iot-ddos prevention using edge computing. In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, 2018. Disponível em: <https://www.usenix.org/conference/hotedge18/presentation/bhardwaj>.

BHOSALE, K. S.; NENOVA, M.; ILIEV, G. The distributed denial of service attacks (ddos) prevention mechanisms on application layer. In: *2017 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*. [S.l.: s.n.], 2017. p. 136–139.

BODDY, S. *Spaceballs Security: The Top Attacked Usernames and Passwords*. 2018. Disponível em: <https://www.f5.com/labs/articles/threat-intelligence/spaceballs-security--the-top-attacked-usernames-and-passwords>.

BOERO, L.; MARCHESE, M.; ZAPPATORE, S. Support vector machine meets software defined networking in ids domain. In: *2017 29th International Teletraffic Congress (ITC 29)*. [S.l.: s.n.], 2017. v. 3, p. 25–30.

BOUTABA, R. et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *JISA*, v. 9, n. 1, p. 16, Jun 2018.

BREIMAN, L.; CUTLER, A. Random forests. *Mach. Learn*, v. 45, 2001. Disponível em: <https://www.stat.berkeley.edu/~breiman/RandomForests/>.

BUCHANAN, B. *Snort Analyser*. 2014. Acessado: 10-04-2019. Disponível em: <https://asecuritysite.com/forensics/snort?fname=hping_syn.pcap&rulesname=rulesdos.rules>.

BUCZAK, A. L.; GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, v. 18, n. 2, p. 1153–1176, Secondquarter 2016. ISSN 1553-877X.

CARDOSO, D. G. *Dimensionamento Vertical Automático de Recursos em Nuvens Computacionais*. 66 f. Dissertação (Mestrado em Informática) — Universidade Federal do Espírito Santo, Vitória-ES, 2019.

CORRÊA, J. H. et al. Ml-based ddos detection and identification using native cloud telemetry macroscopic monitoring. *Journal of Network and Systems Management*, Springer, v. 29, n. 2, p. 1–28, 2021.

CORRÊA, J. H. et al. *On Machine Learning DoS Attack Identification from Cloud Computing Telemetry*. 2019. Disponível em: <https://arxiv.org/pdf/1904.06211.pdf>.

CORRÊA, J. H. et al. Selectivity and autoscaling as complementary defenses for DDoS protection to cloud services. In: *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. Coimbra, Portugal: [s.n.], 2019.

CRISTOFARO, E. D. et al. Paying for likes? understanding facebook like fraud using honeypots. In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2014. (IMC '14), p. 129–136. ISBN 9781450332132. Disponível em: <https://doi.org/10.1145/2663716.2663729>.

DANTAS, Y. G.; NIGAM, V.; FONSECA, I. E. A selective defense for application layer ddos attacks. In: *2014 IEEE Joint Intelligence and Security Informatics Conference*. [S.l.: s.n.], 2014. p. 75–82.

DIMOLIANIS, M.; PAVLIDIS, A.; MAGLARIS, V. Syn flood attack detection and mitigation using machine learning traffic classification and programmable data plane filtering. In: *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. [S.l.: s.n.], 2021. p. 126–133.

DOCUMENTATION, O. *OpenStack Docs: Ceilometer*. 2019. <https://docs.openstack.org/ceilometer/rocky>. Acessado:11-06-2019.

DOSHI, R.; APTHORPE, N.; FEAMSTER, N. Machine learning ddos detection for consumer internet of things devices. In: *2018 IEEE Security and Privacy Workshops (SPW)*. [S.l.: s.n.], 2018. p. 29–35.

FADLULLAH, Z. et al. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys Tutorials*, PP, n. 99, p. 1–1, 2017. ISSN 1553-877X.

FAN, W. et al. Enabling an anatomic view to investigate honeypot systems: A survey. *IEEE Systems Journal*, v. 12, n. 4, p. 3906–3919, 2018.

FAYAZ, S. K. et al. Bohatei: Flexible and elastic ddos defense. In: *24th {USENIX} Security Symposium ({USENIX} Security 15)*. [S.l.: s.n.], 2015. p. 817–832.

FILHO, E. de A. *Iniciação à lógica matemática*. [S.l.]: NBL Editora, 2002.

FIORINO, L. et al. Reprodutibilidade e extensibilidade de datasets de rede: um estudo da replicação de traces de pacotes. In: *Submetido ao XI Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2021.

GOGUNSKA, K. et al. On the cost of measuring traffic in a virtualized environment. In: *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. [S.l.: s.n.], 2018. p. 1–6.

HAN, W. et al. Honeymix: Toward sdn-based intelligent honeynet. In: *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. New York, NY, USA: Association for Computing Machinery, 2016. (SDN-NFV Security '16), p. 1–6. ISBN 9781450340786. Disponível em: <https://doi.org/10.1145/2876019.2876022>.

HE, Z.; ZHANG, T.; LEE, R. B. Machine learning based ddos attack detection from source side in cloud. In: *2017 IEEE 4th International CSCloud*. [S.l.: s.n.], 2017. p. 114–120.

HELPSEC. *Malware-infected home routers used to launch DDoS attacks*. [S.l.]: http://www.helpsec.net/malware-infected-home-routers-used-to-launch-ddos-attacks, 2016. Acessado: 23-03-2016.

HETTICH, S.; BAY, S. D. *KDD Cup 1999 Data*. 1999. Irvine, CA: University of California, Department of Information and Computer Science. Acessado: 30-09-2021. Disponível em: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

HUANG, K.; SIEGEL, M.; MADNICK, S. Systematically understanding the cyber attack business: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 4, jul. 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3199674>.

JIA, Y. et al. Flowguard: An intelligent edge defense mechanism against iot ddos attacks. *IEEE Internet of Things Journal*, v. 7, n. 10, p. 9552–9562, 2020.

JYOTHI, V. et al. Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks. In: *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. [S.l.: s.n.], 2016. p. 587–588. ISSN 2380-6923.

KLAINE, P. V. et al. A survey of machine learning techniques applied to self organizing cellular networks. *IEEE Communications Surveys Tutorials*, PP, n. 99, p. 1–1, 2017.

KUPREEV, O.; BADOVSKAYA, E.; GUTNIKOV, A. *DDoS attacks in Q3 2019*. 2019. <https://securelist.com/ddos-report-q3-2019/94958/>. Acessado: 30-01-2020.

LAI, S.-F. et al. Design and implementation of cloud security defense system with software defined networking technologies. In: *2016 International Conference on Information and Communication Technology Convergence (ICTC)*. [S.l.: s.n.], 2016. p. 292–297.

LIEBERGELD, S.; LANGE, M.; BORGAONKAR, R. Cellpot: a concept for next generation cellular network honeypots. *Internet Society*, p. 1–6, 2014.

MA, L. et al. Research on sql injection attack and prevention technology based on web. In: *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. [S.l.: s.n.], 2019. p. 176–179.

MAURICIO, L. A. et al. Uma arquitetura de virtualização de funções de rede para proteção automática e eficiente contra ataques. In: SBC. *Anais do XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS-SBRC 2017)*. [S.l.], 2017. v. 22, n. 1/2017.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, USA, 2011.

MENDONÇA, G. et al. Uma abordagem leve para detecção de ddos a partir de roteadores domésticos. In: *37th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC)*. Porto Alegre, RS, Brazil: SBC, 2019. p. 834–847. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/7406>.

MIAO, R. et al. The dark menace: Characterizing network-based attacks in the cloud. In: *Proceedings of the 2015 Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2015. (IMC '15), p. 169–182. ISBN 9781450338486. Disponível em: <https://doi.org/10.1145/2815675.2815707>.

MISHRA, A. et al. Classification based machine learning for detection of ddos attack in cloud computing. In: IEEE. *2021 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2021. p. 1–4.

MYSORE, R. N. et al. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 39, n. 4, p. 39–50, ago. 2009. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/1594977.1592575>.

NGUYEN, T. T. T.; ARMITAGE, G. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, v. 10, n. 4, p. 56–76, Fourth 2008. ISSN 1553-877X.

NICHOLS, N. et al. Identification of program signatures from cloud computing system telemetry data. In: *2016 IEEE SSCI*. [S.l.: s.n.], 2016. p. 1–5.

OPENSTACK, P. *Openstack Documentation*. 2019. Acessado: 11-06-2019. Disponível em: <http://docs.openstack.org/>.

OPENSTACK, P. *Openstack Software*. 2020. Acessado: 20-08-2020. Disponível em: <https://www.openstack.org/software>.

PHAN, T. V.; PARK, M. Efficient distributed denial-of-service attack defense in sdn-based cloud. *IEEE Access*, v. 7, p. 18701–18714, 2019. ISSN 2169-3536.

PROJECT, S. T. *SNORT Users Manual*. 2020. Disponível em: <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/>.

QU, C.; CALHEIROS, R. N.; BUYYA, R. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 4, jul. 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3148149>.

REDHAT. *IaaS x PaaS x SaaS*. 2021. Acessado: 08-10-2021. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-computing/iaas-vs-paas-vs-saas>.

RUHL; JOHNJG12. *Snort rule to detect http flood*. 2015. Acessado: 10-04-2019. Disponível em: <https://stackoverflow.com/questions/28395370/snort-rule-to-detect-http-flood>.

SABAT, S. *Big Data Know How <http://bigdataknowhow.weebly.com/>*. 2016.

Sabrine, H.; Abderrahmane, B.; Fouzi, S. Comparative study of security methods against ddos attacks in cloud computing environment. In: *2019 International Symposium on Networks, Computers and Communications (ISNCC)*. [S.l.: s.n.], 2019. p. 1–5.

SAHAY, R. et al. Aroma: An sdn based autonomic ddos mitigation framework. *Computers and Security*, v. 70, p. 482 – 499, 2017. ISSN 0167-4048. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167404817301499>.

SANTOS, P. B. et al. Monitoramento de recursos para aplicações de rob ótica em espaços inteligentes baseados em uma nuvem openstack. In: SBC. *Anais do XVI Workshop em Clouds e Aplicações (WCGA-SBRC 2018)*. [S.l.], 2018. v. 16.

SHAMELI-SENDI, A. et al. Taxonomy of distributed denial of service mitigation approaches for cloud computing. *JNCA*, v. 58, p. 165 – 179, 2015. ISSN 1084-8045.

SHEA, R.; LIU, J. Understanding the impact of denial of service attacks on virtual machines. In: *Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on*. [S.l.: s.n.], 2012. p. 1–9.

SHOREY, T. et al. Performance comparison and analysis of slowloris, goldeneye and xerxes ddos attack tools. In: *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. [S.l.: s.n.], 2018. p. 318–322.

SINGH, S. et al. Detecting different attack instances of ddos vulnerabilities on edge network of fog computing using gaussian naive bayesian classifier. In: *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. [S.l.: s.n.], 2020. p. 1–6.

SOLANAS, M.; HERNANDEZ-CASTRO, J.; DUTTA, D. Detecting fraudulent activity in a cloud using privacy-friendly data aggregates. *arXiv preprint arXiv:1411.6721*, 2014.

SOMANI, G. et al. Ddos attacks in cloud computing: Issues, taxonomy, and future directions. *Computer Communications*, v. 107, p. 30 – 48, 2017. ISSN 0140-3664. Disponível em: <http://www.sciencedirect.com/science/article/\\pii/S0140366417303791>.

SURESH, M.; ANITHA, R. Evaluating machine learning algorithms for detecting ddos attacks. In: SPRINGER. *International Conference on Network Security and Applications*. [S.l.], 2011. p. 441–452.

TANENBAUM, A. S. *Redes de Computadores*. 4a edição. ed. [S.l.]: Elsevier, 2003. ISBN 85-352-1185-3.

TANENBAUM, A. S.; BOS, H. *Sistemas Operacionais Modernos*. 4a edição. ed. [S.l.]: Pearson Education do Brasil, 2016. ISBN 978-85-4301-818-8.

TANG, P. et al. Efficient auto-scaling approach in the telco cloud using self-learning algorithm. In: *2015 IEEE Global Communications Conference (GLOBECOM)*. [S.l.: s.n.], 2015. p. 1–6.

TELESCOPE, U. N. *CAIDA Resource Catalog*. 2021. Acessado: 30-09-2021. Disponível em: <https://catalog.caida.org/details/dataset/telescope_codered_worm>.

TORRES, C. F.; STEICHEN, M.; STATE, R. The art of the scam: Demystifying honeypots in ethereum smart contracts. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019. p. 1591–1607. ISBN 978-1-939133-06-9. Disponível em: <https://www.usenix.org/conference/usenixsecurity19/presentation/ferreira>.

URIAS, V. E.; STOUT, W. M. S.; LIN, H. W. Gathering threat intelligence through computer network deception. In: *2016 IEEE Symposium on Technologies for Homeland Security (HST)*. [S.l.: s.n.], 2016. p. 1–6.

VERISIGN. *What Is a DDoS Attack?* 2018. Https://blog.verisign.com/security/ddos-protection/q2-2018-ddos-trends-report-52-percent-of-attacks-employed-multiple-attack-types/. Acessado: 11-03-2019.

YI, S.; LI, C.; LI, Q. A survey of fog computing: Concepts, applications and issues. In: *Proceedings of the 2015 Workshop on Mobile Big Data*. New York, NY, USA: Association for Computing Machinery, 2015. (Mobidata '15), p. 37–42. ISBN 9781450335249. Disponível em: <https://doi.org/10.1145/2757384.2757397>.

ZARGAR, S. T.; JOSHI, J.; TIPPER, D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, v. 15, n. 4, p. 2046–2069, 2013.

ZEKRI, M. et al. Ddos attack detection using machine learning techniques in cloud computing environments. In: *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*. [S.l.: s.n.], 2017. p. 1–7.

ÖZÇELIK, M.; CHALABIANLOO, N.; GüR, G. Software-defined edge defense against iot-based ddos. In: *2017 IEEE International Conference on Computer and Information Technology (CIT)*. [S.l.: s.n.], 2017. p. 308–313.