UNIVERSIDADE BEIRA INTERIOR
Engenharia

Avaliação de Desempenho de Sistemas de Detecção de Intrusões Baseados em Anomalias a Nível de Contentor para Aplicações *Multi-Tenant* Usando Algoritmos de Aprendizagem Automática

(Versão Definitiva Após Defesa Pública)

Marcos Aurélio Oliveira Cavalcanti

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º Ciclo de Estudos)

Orientador: Prof. Doutor Mário Marques Freire

Covilhã, Julho de 2021

Dissertação elaborada no Instituto de Telecomunicações - Delegação da Covilhã e no Departamento de Informática da Universidade da Beira Interior e submetida à Universidade da Beira Interior para discussão em provas públicas.

Este trabalho foi financiado pela FCT/MCTES através de fundos nacionais e quando aplicável cofinanciado por fundos comunitários no âmbito do projeto UIDB/50008/2020, foi financiado pela FCT/COMPETE/FEDER através do projeto SECURIoTESIGN com número de referência POCI-01-0145-FEDER-030657 e foi suportado pela operação Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, cofinanciada pelo Fundo Europeu de Desenvolvimento Regional (FEDER) através do Programa Operacional Regional do Centro (Centro 2020), no âmbito do Sistema de Apoio à Investigação Científica e Tecnológica - Programas Integrados de ICDT.

Cofinanciado por:

Dedicatória

À minha mãe. Vossa presença durante esta trajetória tornou tudo mais fácil. Muito obrigado.

Agradecimentos

Agradeço, em primeiro lugar a Deus por ter permitido que eu alcançasse mais uma conquista em minha vida.

À minha família, pelo carinho, compreensão e apoio em todas as decisões de minha vida.

Ao Professor Doutor Mário Marques Freire, meu orientador, por quem tenho admiração e respeito, deixo aqui a minha eterna gratidão, primeiro por ter acreditado neste trabalho e segundo pelas orientações precisas dentro da linha construtivista.

Aos professores e funcionários do Departamento de Informática da Faculdade de Engenharia da Universidade Beira Interior pela boa integração, profissionalismo e apoio ao meu percurso académico na instituição.

Por fim, meus sinceros agradecimentos a todos aqueles que ou com palavras incentivadoras ou fornecendo dados e informações contribuíram para a produção desta investigação.

Resumo

A virtualização dos recursos computacionais proporcionadas pelos contentores tem ganho cada vez mais atenção e tem sido amplamente utilizada na computação em nuvem. Essa nova demanda por tecnologia de contentores vem crescendo e o uso de Docker e Kubernetes é considerável. De acordo com pesquisas recentes de tecnologia, os contentores agora são comuns. No entanto, atualmente, um dos maiores desafios surge do fato de que vários contentores, com proprietários diferentes, podem coabitar no mesmo *host*. Em ambientes *multi-tenant* baseados em contentor, as questões de segurança são uma grande preocupação. A segurança nestes ambientes é um elemento essencial para proteção dos recursos computacionais, sistemas e dados. Muitas atividades de investigação nesta área visam aprimorar os mecanismos de sistema de deteção de intrusões. O método de deteção de intrusões por anomalias é um meio de monitorização do comportamento do sistema que define o que é normal e procura por atividades que se desviem do comportamento normal pré-estabelecido, denominadas de anomalias.

Este trabalho de investigação apresenta uma avaliação de desempenho de algoritmos de aprendizagem automática na deteção de intrusões em ambiente *multi-tenant* em contentores. Na campanha experimental desta investigação foram adotadas três configurações diferentes para os classificadores: A) classificação das chamadas de sistema usando *Label Encoder* e *One Hot Encoder*, B) classificação das chamadas de sistema utilizando o algoritmo janela deslizante com *Label Encoder* e *One Hot Encoder* e C) classificação das chamadas de sistema com as técnicas janela deslizante e Bag of System Calls (BoSC).

Os melhores resultados de classificação na deteção de intrusões neste ambiente foram obtidos com os algoritmos Decision Tree e Random Forest, usando uma janela deslizante de tamanho 30 e o algoritmos BoSC, atingindo ambos uma F-Measure de 99,8%. Contudo, o algoritmo Decision Tree tem um menor tempo de execução e consome menos CPU e memória do que o algoritmo Random Forest.

Palavras-chave

Contentores, Sistema de Deteção de Intrusões, Chamadas de Sistema, Algoritmos de Aprendizagem Automática.

Abstract

The virtualization of computing resources provided by containers has gained increasing attention and has been widely used in cloud computing. This new demand for container technology is growing and the use of Docker and Kubernetes is considerable. According to recent technology research, containers are now common. However, currently, one of the biggest challenges arises from the fact that multiple containers, with different owners, can cohabit on the same host. In container-based multi-tenant environments, security issues are a major concern. Security in these environments is an essential element for protecting computing resources, systems and data. Many research activities in this area aim to improve the intrusion detection system mechanisms. The anomaly intrusion detection method is a means of monitoring the behavior of the system that defines what is normal and searches for activities that deviate from the pre-established normal behavior, called anomalies.

This investigation proposed an evaluation of the performance of machine learning algorithms in host-based intrusion detection in a multi-tenant container environment. In the experimental campaign of this investigation, three different configurations were adopted for the classifiers: A) classification of system calls using *Label Encoder* and *One Hot Encoder*, B) classification of system calls using the sliding window algorithm with *Label Encoder* and *One Hot Encoder* and C) classification of system calls with the sliding window and Bag of System Calls (BoSC) techniques.

The best classification results in intrusion detection in this environment were obtained with the Decision Tree and Random Forest algorithms, using a sliding window of size 30 and the BoSC algorithm, both reaching an F-Measure of 99.8%. However, the Decision Tree algorithm has a shorter execution time and consumes less CPU and memory than the Random Forest algorithm.

Keywords

Containers, Intrusion Detection System, System Calls, Machine Learning Algorithms.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ANN Artificial Neural Networks. 26

API Application Programming Interface. 8, 10, 11, 15

AUC Area Under the Curve. 68

BoSC Bags of System Calls. 5, 36, 59, 63, 67, 71, 72, 74

CPU Central Process Unit. 1, 12, 13, 15, 33, 55, 73

CSV Comma Separated Values. xv, xvi, 51, 54–57

DIDS Distributed Intrusion Detection System. 30

DNS Domain Name System. 11, 37

DoS Denial of Service. 11

DSRM Design Science Research Methodology. 3, 4

E/S Entrada/Saída. 1, 12, 16

FPR False Positive Rate. 37

HIDS Host-Based Intrusion Detection System. 28, 30, 36, 38

HSM Hardware Security Modules. 11

ID Identity. 55

IDPS Intrusion Detection and Prevention System. 27

IDS Intrusion Detection System. 27, 29–33, 37, 38

IP Internet Protocol. 18, 37, 45, 47

JSON JavaScript Object Notation. 11

LXC Linux Containers. 12, 15–17

MAC Mandatory Access Control. 21

MITM Man-in-the-Middle. 11

MSA Microservice Architectures. 36

MTD Moving Target Defenses. 36

NIDS Network Intrusion Detection Systems. 28–30, 38

OLTP Online Transaction Processing. 46

OS Operating System. 20

OWASP Open Web Application Security Project. 11, 36

PaaS Platform as a Service. 16

PID Process Identifier. 16, 55

RAM Random Access Memory. 73, 74

REST Representational State Transfer. 11

RNA Redes Neurais Artificiais. 26

ROC Receiver Operator Characteristic. xvi, 33, 68

SECCOMP Secure Computing Mode. 21

SEV Secure Encrypted Virtualization. 11

SGX Software Security Guard Extensions. 11, 20

SME Secure Memory Encryption. 11

SOAP Simple Object Access Protocol. 11

SQL Structured Query Language. 11, 58

SSH Secure Socket Layer. 12

STIDE Sequence Time-Delay Embedding. 36

SVM Support Vector Machine. 27

TIC Tecnologias da Informação e Comunicação. 1

TLS Transport Layer Security. 11

TPR True Positive Rate. 37

VM Virtual Machine. 1, 8, 11, 13, 14, 41

Capítulo 1

Introdução

1.1 Âmbito e Foco da Dissertação

Face à migração para o paradigma atual da computação em nuvem, os serviços de computação em nuvem estão a ser amplamente implantados em diversos setores da indústria e serviços. A computação em nuvem está a mudar rapidamente a face da infraestrutura de serviço de *Internet*, aproveitando a escala e a flexibilidade das infraestruturas físicas partilhadas fornecidas pela computação em nuvem [1]. Muitas empresas estão optando pela migração da estrutura tradicional de Tecnologia de Informação e Comunicação (TIC) para a computação em nuvem, devido as diversas vantagens apresentadas, nomeadamente, em relação ao custo do modelo. A computação em nuvem usa um modelo de negócios *pay-per-use*, no qual os trabalhos do utilizador são externalizados para centros de dados [2].

O relatório Flexera 2021 *State of the Cloud* [3] destaca as tendências em 2021 sobre a adoção das tecnologias da computação em nuvem pelo quinto ano consecutivo. A principal iniciativa consiste em otimizar o uso de recursos da nuvem (economia de custos), seguida pela migração de mais cargas de trabalho para a nuvem. Investigações recentes da *International Data Corporation* (IDC) relatam que a pandemia COVID-19 provou ser um acelerador da adoção e extensão da nuvem e continuará a impulsionar uma adoção mais rápida para a TIC centrada na nuvem [4].

Face à grande procura de serviços de computação em nuvem surgiram novas tecnologias como resposta. A virtualização de servidores é uma das respostas a esta procura. A virtualização é uma técnica amplamente utilizada por plataformas de serviços do tipo PaaS (Plataforma como Serviço), IaaS (Infraestrutura como Serviço), SaaS (Software como Serviço), fim de proporcionar um ambiente isolado, seguro e escalável para utilizadores e organizações [5]. Trata-se de um *software* que possibilita a abstração dos componentes físicos do servidor (*hardware*) para execução de diversos sistemas operativos convidados em máquinas virtuais. As máquinas virtuais (VM)s realizam imitação virtual de um servidor físico e de seus recursos (CPU, memória, E/S). Outro aspecto a considerar na virtualização é que os recursos do servidor físico são divididos pelas diversas máquinas virtuais para diferentes tipos de aplicações.

A virtualização de recursos computacionais pode ser proporcionada, também, através de um contentor. A praticidade deste recurso consiste em não depender do sistema base ao qual a aplicação é executada, permitindo a execução em simultâneo de múltiplos contentores com diferentes estruturas e aplicações e ainda partilhar o mesmo *kernel* e sistema operativo. Neste caso, a tecnologia de contentores concede a implementação de *software* de modo eficiente e em grande escala. Recentemente, a virtualização baseada em conten-

tores tem ganho cada vez mais atenção, e tem sido amplamente utilizada na computação em nuvem [6]. Um contentor é uma unidade padrão de *software* que empacota o código e todas as suas dependências para que a aplicação seja executada de forma rápida e confiável num ambiente de computação [7].

Os microsserviços apresentam um melhor desempenho em arquiteturas de *software* baseadas em contentores. Ao contrário das arquiteturas monolíticas, a separação e a divisão do trabalho permite que os serviços continuem em execução, mesmo que os outros falhem, o que mantém a aplicação como um todo mais confiável [8]. Logo, o contentor permite o isolamento da aplicação, através dos recursos presentes no *kernel* do Linux, como grupos de controlo e *namespaces* sendo considerado como um fator de **segurança** para aplicações executadas na mesma infraestrutura, em ambiente *multi-tenant*. O uso crescente da virtualização impõe requisitos de segurança rigorosos à integridade do *software* e ao isolamento da carga de trabalho da computação em nuvem [9].

Perante a exposição deste cenário, é importante salientar que a maior disponibilidade de serviços e aplicações em geral, via *Internet*, certamente provocou a explosão de utilizadores de âmbito mundial. Por isso é necessário ter em consideração o aumento significativo de fraudes e de outras atividades maliciosas neste ambiente *multi-tenant*, pois, há riscos associados a utilização indevida de contentores para atacar outros contentores, o sistema operativo *host*, ou outros *hosts* [10]. As ferramentas de *software* de deteção e de análise de intrusões são necessárias para minimizar as perdas e para que se fortaleçam as defesas contra tais atividades maliciosas. Estas ferramentas são capazes de monitorizar todos tipos de instâncias, ações e processos, para analisar e decidir se constituem ameaça para o sistema [11]. É neste último contexto que se destacam os sistemas de deteção de intrusões (*Intrusion Detection Systems*) [12, 13].

Esta dissertação enquadra-se na área de segurança da computação em nuvem, envolvendo especialmente tecnologia de contentores em ambiente *multi-tenant* e especificamente no estudo e aplicabilidade dos algoritmos de deteção de anomalias de aprendizagem automática na deteção de intrusões em contentores. Os contentores tornaram-se uma tecnologia padrão para aplicações dispostas na arquitetura de microsserviços devido à tecnologia de virtualização leve através do agrupamento e isolamento de recursos. Porém este agrupamento e isolamento de recursos não são suficientes como dispositivos de segurança em relação aos contentores maliciosos em ambientes *multi-tenant* [14].

Os contentores não foram desenvolvidos com uma camada de segurança para isolar entre recipientes mal-intencionados e potencialmente maliciosos que partilham o mesmo ambiente como se fossem um contentor honesto mas, na verdade procuram vulnerabilidades que vão desde a exploração do *kernel* e ataques aos recursos partilhados do anfitrião até a configurações erradas [5]. Considerando um contentor como um ambiente de computação distribuída gerido por um anfitrião (*host*), o foco desta investigação está na avaliação do desempenho de sistemas de deteção de intrusões baseados em *host*, em ambientes *multi-tenant* em contentores, usando algoritmos de aprendizagem automática para deteção de anomalias.

Dessa forma, a aplicação dos sistemas de deteção de intrusões nestes ambientes é de fundamental importância, pois a tecnologia de contentores está a ser cada vez mais adotado em uso individual e industrial, e como esses recipientes podem ser integrados em sistemas de **missão crítica**, a deteção em tempo real de ataques cibernéticos maliciosos torna-se um requisito operacional crítico [15].

1.2 Definição do Problema e Objetivos da Investigação

As aplicações em contentores e a arquiteturas de microsserviço também podem ser usadas para construir sistemas críticos se a execução for suportada por contentores seguros [16].

A arquitetura de microsserviço [17] é um paradigma de Engenharia de *Software* que promove a decomposição de aplicações monolíticas com muitos códigos em microsserviços e contentores múltiplos, menores e independentes. No entanto, existem preocupações de segurança devido à complexidade da configuração de segurança da arquitetura de microsserviços e ao fato de que esses recursos e estruturas podem ser compartilhados em ambientes *multi-tenant*. As aplicações que são executadas neste tipo de tecnologia compartilham bibliotecas e recursos virtuais orquestrados pelo sistema operativo. Os contentores ou aplicações maliciosas que partilham o mesmo ambiente podem atacar outros contentores, o sistema operativo do *host* e/ou outros *hosts*. Neste caso, é fundamental como solução complementar a monitorização destas atividades maliciosas através de ferramentas de deteção de intrusões.

Algumas abordagens foram publicadas e retratam o problema de deteção de intrusões a nível de contentor para aplicações *multi-tenant* [15, 18, 19]. No entanto, essas abordagens apresentam desempenho limitado. O problema abordado nesta dissertação consiste em investigar novos métodos que melhorem o desempenho de sistemas de deteção de intrusões em contentores para aplicações *multi-tenant*. O objetivo principal deste trabalho de investigação consiste em avaliar e comparar o desempenho de algoritmos de aprendizagem automática na deteção de intrusões ao nível de contentores para aplicações *multi-tenant*.

1.3 Abordagem para Resolver o Problema

A metodologia adotada no trabalho de investigação é designada por *Design Science Research Methodology* (DSRM) [20]. Esta metodologia consiste num modelo que deve fornecer orientação e sentido sobre o que esperar dos resultados de uma investigação e que é organizado num ciclo composto por seis fases interativas, utilizadas para o controlo e melhoria contínua do desenvolvimento de uma investigação através da análise dos resultados, conforme se ilustra na Figura 1.1.

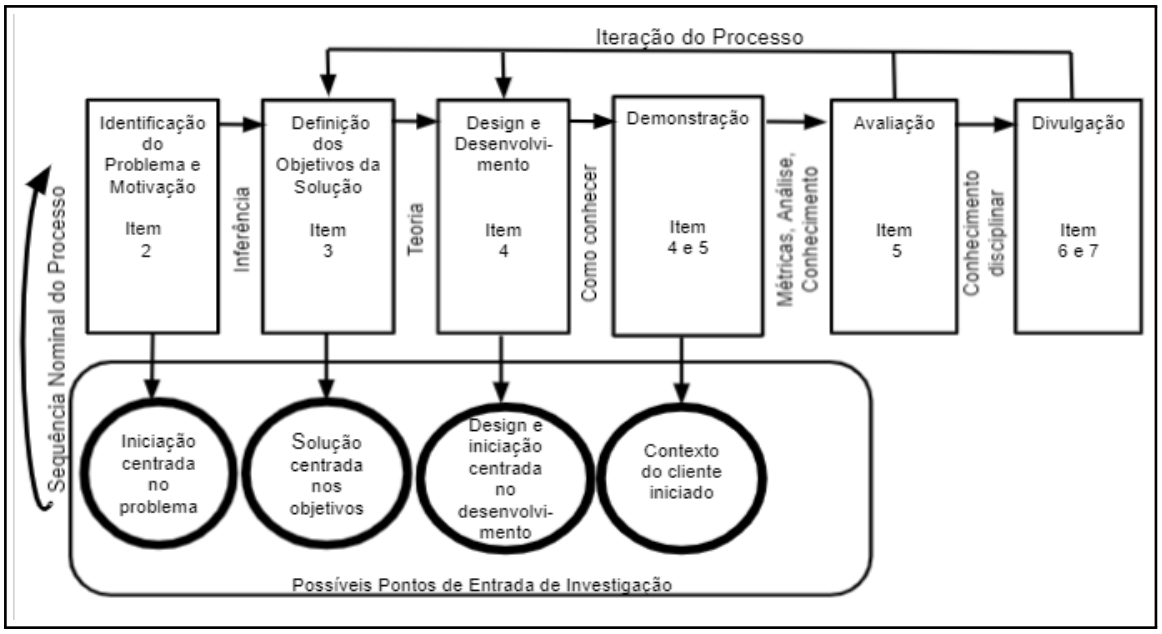


Figura 1.1: Modelo do processo DSRM (redesenhado e adaptado a partir de [20]).

A aplicação desta metodologia permite a correlação das atividades do trabalho de investigação, descritas abaixo, com cada fase interativa, ver figura 1.1, e como pré-requisito para a fase posterior.

1. *Background* e Estado da Arte: Nesta etapa foram explorados os conceitos fundamentais para um melhor entendimento do problema que se pretende investigar, associado ao estudo comparativo de soluções e tecnologias existentes.

2. Especificação do Problema e Motivação: O estudo para definição do problema a ser investigado teve como foco algumas abordagens [15, 18, 19] que retratam o problema de deteção de intrusões a nível de contentor para aplicações *multi-tenant*. A busca de um elemento adicional a este campo de investigação tem em consideração que os algoritmos de aprendizagem automática podem ser usados em sistemas de deteção de intrusões para melhorar o desempenho, precisão e robustez [21].

3. Objetivo da Investigação: O objetivo principal desta investigação foi elaborado de forma esclarecedora e realista diante da identificação do problema a que pretende-se apresentar solução e servindo de ponto norteado ao percurso da atividade investigativa.

4. Especificação da Arquitetura, Implementação e Configuração do Sistema de Deteção de Intrusões: O ambiente experimental proposto no projeto de investigação teve a sua implementação consolidada nesta investigação em concordância com as documentações e especificações técnicas presentes nos *software* que compõem o ambiente experimental para a produção e comparação de resultados perante as diferentes simulações efetivadas.

5. Avaliação da Solução a Implementar e Resultados Esperados: As métricas selecionadas no projeto de investigação são um elemento indicador para escolha entre a combinação das técnicas BoSC e janela deslizante com os classificadores de aprendizagem automática o de melhor desempenho. A evolução para um estudo comparativo entre os diversos classificadores de aprendizagem automática nas diferentes técnicas concebeu um perfil do comportamento dos mesmos na deteção de intrusões no ambiente experimental.

6. Disseminação dos Resultados: Os resultados alcançados no desenvolvimento do trabalho de investigação, conducente a esta dissertação de mestrado, proporcionaram a escrita e submissão do artigo científico: **Performance Evaluation of Container-Level Anomaly-Based Intrusion Detection Systems for Multi-Tenant Applications Using Machine Learning Algorithms**, aceite para apresentação e publicação nas atas da International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021 [22].

7. Escrita da Dissertação de Mestrado: A atividade materializou-se paralelamente a todo o trabalho prático, para que, tudo o que fosse feito fica-se devidamente documentado e registado com o intuito de possível replicação do ambiente experimental, melhoria ou trabalhos futuros.

O cronograma das atividades do trabalho de investigação definido por meio do diagrama de Gantt permitiu ter uma percepção temporal da duração de cada uma das atividades descritas anteriormente.

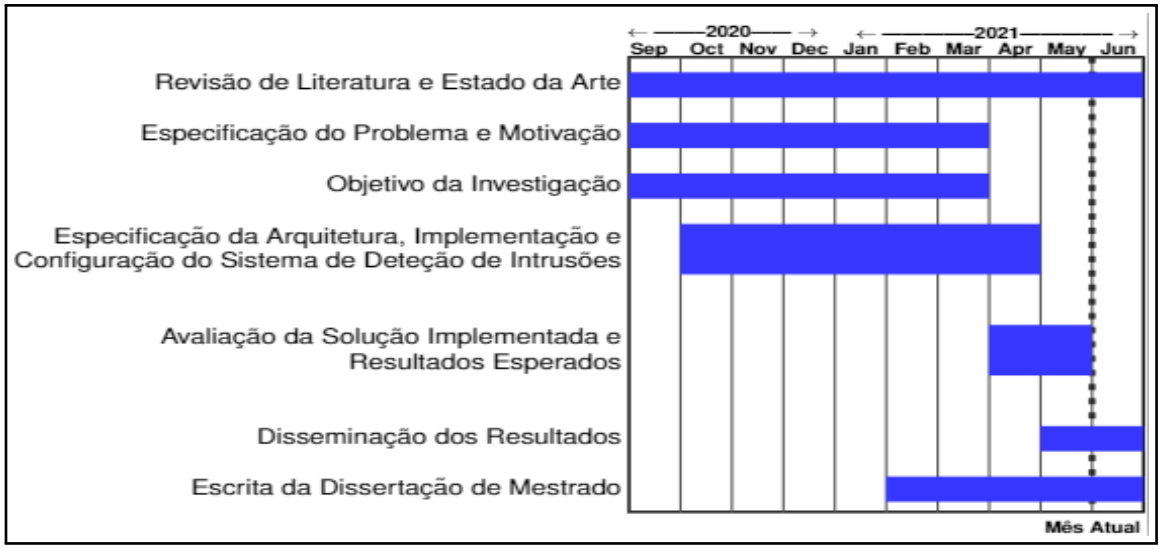


Figura 1.2: Diagrama de Gantt.

Para o primeiro semestre de 2020/2021 foram planeadas e concluídas as seguintes tarefas: revisão de literatura e estado da arte, especificação do problema e motivação e especificação, implementação e configuração da arquitetura proposta. No segundo semestre de

2020/2021 foram planeadas e concluídas as seguintes tarefas: avaliação da solução implementada e resultado esperados, disseminação dos resultados e escrita da dissertação do mestrado.

1.4 Principais Contribuições

A principal contribuição científica, resultante do trabalho de investigação relatado nesta dissertação, consiste na proposta, implementação e avaliação de desempenho de um sistema de deteção de intrusões baseado em anomalias em ambientes *multi-tenant* em contentores, considerando oito algoritmos de aprendizagem automática, disponíveis no *Scikit-learn* [23]: *AdaBoost*, *Decision Tree*, *K-Nearest Neighbors*, *Gaussian Naive Bayes*, *Multilayer Perceptron*, *Multinomial Naive Bayes*, *Random Forest* e *Support Vector Machine*. São investigadas e avaliadas três arquiteturas diferentes para o classificador: Uma baseda no *Label Encoder* e *One Hot Encoder*, outra baseada em janela deslizante com *Label Encoder* e *One Hot Encoder* e a terceira baseada no uso de janela deslizante e do algoritmo *Bag of System Calls*. Esta contribuição foi aceite para apresentação oral e pulicação nas atas da conferência The 16th International Conference on Availability, Reliability and Security [22], listada no *Core Conference Ranking* como Core B.

1.5 Organização da Dissertação

A dissertação está dividida em 5 (cinco) capítulos organizados da seguinte forma. No primeiro capítulo é explanado o âmbito e foco da dissertação, a definição do problema e objetivos da investigação, a abordagem para resolver o problema, as principais contribuições e a forma como esta dissertação está organizada.

O segundo capítulo inclui a revisão bibliográfica e a fundamentação teórica dos aspectos ligados ao tema proposto e às áreas relacionadas. Neste capítulo é realizada uma abordagem sobre a recolha e análise documental com relevância em prol do tema de investigação proposto e para uma correta compreensão do problema a que se pretende dar resposta, juntamente com a exploração de algumas soluções e tecnologias presente no mercado das tecnologias da computação em nuvem, nomeadamente tecnologias de contentores.

O terceiro capítulo relaciona os recursos de *hardware* e *software* do ambiente experimental. Neste capítulo são demonstrado os procedimentos da implementação da infraestruturas do ambiente experimental necessária para a obtenção dos resultados.

No quarto capítulo são discutidos os resultados obtidos sobre o desempenho dos algoritmos de aprendizagem automática na deteção de intrusões em ambiente *multi-tenant* em contentores.

E por fim o quinto capítulo onde são apresentados as principais considerações finais deste trabalho de investigação e sugestões e direções de pesquisas para trabalhos futuros.

Capítulo 2

Background e Estado da Arte

2.1 Introdução

Um contentor representa um modelo de virtualização que apresenta vantagens tanto para o provedor do serviço quanto para os utilizadores. Estas vantagens estão relacionadas principalmente em relação ao isolamento das aplicações, desenvolvimento e utilização da arquitetura de microsserviços. É importante salientar que a introdução da arquitetura de microsserviços aumenta a agilidade do *software*, em que as partes do *software* tornam-se unidades independentes de desenvolvimento, versão, implantação e escalonamento [24].

Para este tipo de virtualização, os contentores são considerados o padrão para implementação de microsserviços e aplicações em nuvem [25]. Logo, eles permitem que um desenvolvedor empacote uma aplicação com todas as partes de que precisa, como bibliotecas e outras dependências, e envie tudo como um pacote [26].

Os avanços recentes do sistema operativo aprimoraram seus recursos de *multi-tenant*, ou seja, a capacidade de repartir um recurso [27]. Contudo, os serviços de contentores em ambiente *multi-tenant* tem-se revelado uma alternativa bastante interessante devido ao proveito da escala e flexibilidade da infraestruturas físicas partilhadas fornecida pela computação em nuvem. Pode-se dizer que virtualização *multi-tenant* soluciona o gargalo da virtualização baseada em *hardware*, ao se concentrar na virtualização baseada em *software* [28].

Neste modelo *multi-tenant*, para manter a qualidade do serviço, principalmente em relação a **segurança**, o provedor precisa gerenciar seus locatários e as condições de seus serviços. Pois, diferentes inquilinos têm diferentes requisitos de segurança, enquanto diferentes políticas de segurança são necessárias para diferentes inquilinos [1]. Devido ao aumento de incidentes de ataques cibernéticos, a construção de sistemas de deteção de intrusões continua sendo uma prioridade para proteger a segurança dos sistemas de informação [29].

Sendo assim, este capítulo aborda conceitos para a construção de uma base sólida de conhecimento para o desenvolvimento do trabalho de investigação. Este capítulo é formado por 7 secções. A secção 2.1 refere-se a introdução do capítulo 2. A secção 2.2 apresenta uma introdução sobre microsserviços, definição, principais características, comparação com a arquitetura monolítica e segurança. A secção 2.3 é dedicada aos contentores onde é possível constatar um breve histórico, tipos, propriedades, isolamento, tecnologia, monitorização (chamadas de sistema) e segurança. A secção 2.4 descreve uma introdução sobre sistema de deteção de intrusões, métodos, recentes abordagens, tipos, sistema de

deteção de intrusões distribuídos e avaliação. A secção 2.5 explana a definição e conceitos relacionado a aprendizagem automática e suas categorias. A secção 2.6 descreve um resumo dos principais trabalhos de pesquisas pertinentes ao temo proposto e a secção 2.7 apresenta a conclusão do capítulo.

2.2 Microsserviços

Em soluções baseadas em nuvem, a arquitetura de microsserviços tem sido empregada como um padrão no desenvolvimento de *software* em comparação com a arquitetura monolítica. Na arquitetura monolítica, uma única aplicação é responsável pelo gerenciamento de todos os processos. Acessa os dados, persiste no banco de dados, exibe a *interface* de interação com o utilizador e o administrador, processa pedidos de clientes e todas as outras ações que a aplicação necessitar.

Em engenharia de *software*, a arquitetura monolítica retrata uma aplicação que é projetada sem modularidade externa, isto é, sem a preocupação de desenvolver uma aplicação que possa ser um módulo para uma outra aplicação. Afirma-se que um monolíto é considerado um *software* cujos módulos não podem ser executados independentemente [30].

Um sistema de *software* com base em microsserviços consiste em vários componentes, microsserviços, que se comunicam por meio de chamadas de procedimento remoto síncronas ou um sistema de mensagens assíncronas [31, 32].

No entanto, os microsserviços devem ser componentes independentes, implantados de forma isolada e equipados com ferramentas de persistência de memória dedicadas como por exemplo, base de dados [30].

O desenvolvimento de aplicações em nuvem levou à adoção da arquitetura de microsserviços, em oposição às monolíticas, a fim de aproveitar as características da computação em nuvem como escalabilidade, disponibilidade ou confiabilidade [33]. A crescente adoção de microsserviços na nuvem é motivada pela facilidade de implantação e atualização do *software* [34]. Por definição pode-se dizer que a arquitetura de microsserviços é um padrão de arquitetura nas quais aplicações complexas são compostas de processos pequenos e independentes que se comunicam uns com os outros utilizando APIs independentes de linguagem [35].

Dessa forma, a estruturação de *software* em unidades de computação menores, definida pela arquitetura de microsserviços, permite a alocação otimizada dos componentes do *software* em contentores adequados nas máquinas virtuais VMs, em execução na máquina *host* fornecida pela infraestrutura em nuvem [34].

2.2.1 Arquitetura de Microsserviços

A arquitetura de microsserviços pode ser definida como uma aplicação distribuída em que todos os seus módulos são microsserviços [30]. Estes módulo por se tratarem de unidades individuais do *software* ofertam aos desenvolvedores um gerenciamento com maior eficiência. Cada módulo ou microsserviços representa um processo de negócios ou funcionalidade distinta, ou seja, cada microsserviços é parte da aplicação que possui sua própria lógica de negócios e vários adaptadores para realizar funções como acesso a banco de dados e mensagens [32]. Ilustrado na figuras 2.1 e 2.2, para comparação, as arquiteturas monolítica e de microsserviços.

Os contentores fornecem uma implementação de forma dinâmica para a arquitetura dos microsserviços, pois, a relação entre contentores e microsserviços é que a sua natureza isolada, neutra em termos de linguagem e distributiva, ajuda os desenvolvedores a empacotarem de modo fácil a aplicação e utilizá-la [31]. Neste caso a tecnologia Docker é utilizada de forma ampla na arquitetura de microsserviços [36] devido ao impacto mínimo de suas imposições sobre processamento, memória e rede [37].

É importante destacar que a computação *Serverless* oferece um modelo de serviço para desenvolver e implementar *software* eficientemente no mercado da computação em nuvem sem ter de gerir qualquer infraestrutura subjacente [38]. Este fator é fundamental para a arquitetura de microsserviços, por que, os microsserviços são uma arquitetura moderna para construir e implementar aplicações complexas usando computação sem servidor [39]. A computação sem servidor é um habilitador chave para aplicações baseadas em microsserviços e faz com que a infraestrutura seja orientada a eventos, totalmente controlada pelas necessidades de cada serviço que compõe o *software* [31].

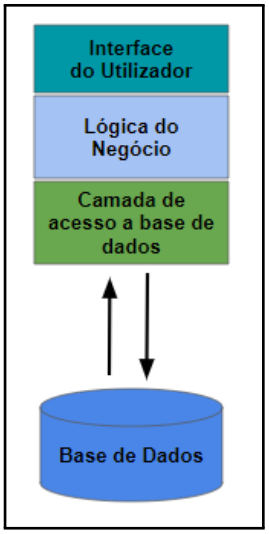


Figura 2.1: Arquitetura monolítica.

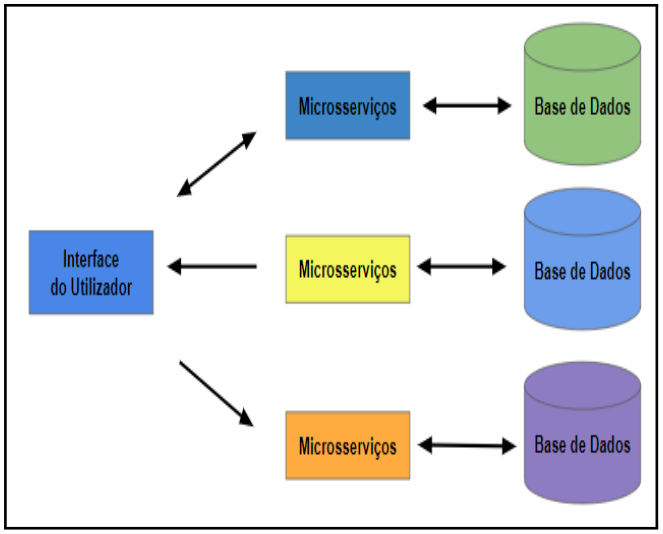


Figura 2.2: Arquitetura de microsserviços.

2.2.2 Principais Características de Microsserviços

A arquitetura de microsserviços é considerada como um novo paradigma de engenharia de *software* para o desenvolvimento de aplicações por meio da composição de pequenos

serviços [30]. Em comparação com outras arquiteturas de *software* e serviços, estas são as características distintivas de microsserviços presentes na tabela 2.1.

Tabela 2.1: Principais características de microsserviços adaptado de [30, 40].

Principais Características de Microsserviços	
Flexibilidade	Um sistema é capaz de acompanhar o ambiente de negócios em constante mudança e é capaz de suportar todas as modificações que são necessárias para uma organização permanecer competitiva no mercado;
Encapsulamento	Os serviços são, em grande medida, encapsulados e isolados dos outros e podem até ser escritos em diferentes linguagens de programação;
Composição Distribuída	Microsserviços baseados em outros microsserviços e se comunicam pela rede;
Modularidade	Um sistema é composto de componentes isolados onde cada componente contribui para o comportamento geral do sistema, em vez de ter um único componente que oferece funcionalidade total. Microsserviços tenderão a oferecer APIs de granulação mais fina que se prestam a reutilização flexível;
Evolução	Um sistema deve permanecer sustentável enquanto evolui constantemente e adiciona novos recursos;
Serviço de Redes	Os serviços são acessíveis pela rede e a reutilização acontece entrando em contato com o serviço, em vez de instalar e vincular a uma biblioteca.

2.2.3 Segurança em Microsserviços

"Em qualquer sistema distribuído, a segurança se torna uma grande preocupação" [30]. A essência de microsserviços é que eles são, ou compõem para formar, sistemas distribuídos altamente modulares, reutilizáveis por meio de uma API exposta em rede [40]. A segurança em microsserviços está relacionada a diversos fatores, características e camadas deste tipo de arquitetura para a implementação da aplicação.

Para a consolidação dos objetivos de segurança: confidencialidade e integridade de dados, autenticação de entidade e mensagem, autorização e disponibilidade do sistema, em sistemas distribuídos, as questões de segurança, neste tipo de arquitetura de microsserviços, devem ser levantadas o quanto antes possível e envolve todas as fases do desenvolvimento do projeto. Pois, isso implica que os microsserviços herdam vantagens e desvantagens dos sistemas distribuídos e dos serviços da *Web* inclusive em relação a segurança [40].

Para ilustrar a complexidade do problema, na tabela 2.2, dividiu-se as questões de segurança de microsserviços em categorias ou camadas: *hardware*, virtualização, *cloud*, comunicação, serviço e orquestração [40, 41].

Tabela 2.2: Problemas de segurança de microsserviços adaptado de [40, 41].

Tipos de Camadas		
Camadas	**Ameaças**	**Mitigação**
Cloud	A computação em nuvem traz uma infinidade de questões de segurança[42]; incluindo controle ilimitado do provedor de nuvem sobre tudo o que executa.	*Reverse sandboxes* protegem enclaves de código e dados de qualquer ataque remoto, incluindo ataques de sistema operativo e *hypervisor*. Exemplos: SGX, SME/SEV.
Virtualização	*Deploy* afeta a segurança; processos do sistema operativo oferecem pouca separação de outros serviços no mesmo sistema; contentores e VMs oferecem mais proteção contra serviços comprometidos. Os ataques: *sandbox escape*, comprometimento do hipervisor e ataques de compartilhamento de memória, o uso de imagens maliciosas e vulneráveis.	Preferir opções de *deploy* com isolamento mais forte; configurações seguras como nenhum acesso à biblioteca compartilhada e nenhum cache de *hardware* compartilhado; verificação da origem e integridade da imagem; atualizações de software oportunas e princípio do menor privilégio.
Hardware	*Bugs* e *backdoors* de *hardware* podem ser introduzidos no momento da fabricação.	Projetar próprio *hardware*; diversificação de *hardware* e uso de módulos de segurança de *hardware*, Hardware Security Modules (HSM).
Comunicação	Ataques na pilha de rede e protocolos; ataques contra protocolos específicos para o estilo de integração de serviço (SOAP, RESTFul Web Services[43][44]). Os ataques: espionagem (sniffing), identity spoofing, session hijacking, DoS e MITM; TLS: Heartbled e POODLE.	Utilizar protocolos de segurança padrão e verificados, como padrões de segurança TLS ou JSON.
Serviço e Aplicação	Injeção de SQL, autenticação interrompida e controle de acesso, exposição de dados confidenciais, Cross-Site Scripting (XSS), desserialização insegura, configuração incorreta de segurança geral.	Análise de código estático/dinâmico, revisão manual de código. Prática básica de engenharia de software: validação de entrada, tratamento de erros, APIs claras e bem documentadas. Proteção dos dados (encriptação). Conhecer os 10 riscos de segurança para aplicação web publicados pela OWASP.
Orquestração	A descoberta do serviço[45][46] fornece um ponto central semelhante ao DNS-like para a localização de serviços. Ataques: serviço de descoberta comprometedor e registo de nós maliciosos dentro do sistema, redirecionando a comunicação para eles.	A proteção da plataforma de orquestração e seus componentes é crítica, e não é uma área bem investigada. Uma implementação segura de componentes de descoberta de serviços e registo é importante.

2.3 Contentores

Historicamente, o isolamento de uma aplicação em um contentor teve seu marco inicial na década 70, mais precisamente em 1979. Desde de então vários eventos e tecnologia foram desenvolvidas até os dias atuais. Em ordem cronológica pode-se relacionar os seguintes eventos:

- Em 1979, início do isolamento do processo, uma característica principal do contentor, com a introdução do comando "*chroot*" ao sistema operativo Unix[1]. O comando "*chroot*" do sistema operativo Unix é uma operação que muda o diretório *root* do processo corrente e de seus processos filhos [47];

- Em 2000, surge o FreeBSD Jail[2] um recurso para criação de múltiplos ambientes virtuais nativo do FreeBSD. A introdução desta tecnologia ao mercado reforça a noção da virtualização ao nível do sistema operativo. O FreeBSD Jail possibilita a criação de um diretório, um endereço de rede e um *hostname* para essa máquina virtual, que pode ser acessada pela console do servidor onde está instalada ou por SSH [48];

- Em 2001, tem-se a contribuição do Linux-VServer[3], uma tecnologia similar ao FreeBSD Jail, que é uma implementação de servidor virtual privado que foi criada adicionando recursos de virtualização em nível de sistema operativo ao *kernel* Linux;

- Em 2004, a Oracle, lança o Solaris Container [49] combinando o controle de recursos do sistema operativo e o isolamento de processo. No ano de 2005, surge OpenVZ [50], projeto comunitário suportado pela Parallels, Inc., uma tecnologia de virtualização de nível de sistema operativo baseada no *kernel* e no sistema operativo do Linux;

- E em 2006, a Google, apresenta ao mercado tecnológico o *Process Container* projetado para ser usado para limitar, contabilizar e isolar o uso de recursos (CPU, memória, E/S, rede) de um grupo de processos. O *Process Container*, em 2008, foi renomeado *ControlGroups* (*cgroups*) e adiconado ao *kernel* do Linux apartir da versão 2.6.24;

- Também, em 2008, é que surge primeira implementação do gerenciador de contentor Linux, denominada de LXC [51], com *cgroups* e *namespaces*, recursos presente no *kernel* do Linux. Os *namespaces* providenciam o isolamento de processo em grupos, enquanto o *cgroups* é responsável por coordenar o uso de recursos para cada contentor dentro do sistema principal. O LXC é de baixo nível, muito flexível e abrange quase todos os recursos suportados pelo *kernel* [51].

- Mais tarde, em 2011 a Cloud Foundry[4] lança no mercado o Warden uma implementação de contentor que isola ambiente em qualquer sistema operativo.

[1] `https://www.opengroup.org/membership/forums/platform/unix`
[2] `https://www.freebsd.org/doc/handbook/jails.html`
[3] `http://www.linux-vserver.org/Welcome_to_Linux-VServer.org`
[4] `https://www.altoros.com/blog/cloud-foundry-containers-warden-docker-and-garden/`

- E finalmente, em 2013, a DotCloud[5] apresenta ao mercado um projeto interno conhecido como Docker para facilitar o uso do LXC Linux. Docker realmente facilitou o desenvolvimento e implementação da aplicação, o que resultou na adoção massiva de contentores por provedores de serviços em nuvem.

Por definição, um contentor é um método de virtualização que permite aos utilizadores implementar aplicações em qualquer ambiente de forma rápida e eficiente. A virtualização baseada em contentor surgiu como uma alternativa leve para VMs [52]. Esta vantagem em relação as VMs é devido a que muitos contentores podem partilhar o mesmo *kernel* do sistema operativo em vez de ter uma cópia dedicada para cada um, como nas VMs [52], fornecendo um mecanismo de virtualização leve. Os contentores são chamados de tecnologia de virtualização leve devido à forma como operam na virtualização no nível do sistema, onde a camada de virtualização é executada como uma aplicação dentro do sistema operativo [27].

Como resultado este modelo de virtualização têm-se custos mínimos de tempo de inicialização e desligamento, baixos requisitos de recursos para cada imagem e alta escalabilidade. Assim, os contentores são projetados para funcionar com a quantidade mínima de recursos necessários para executar a tarefa do sistema operativo básico e poucos *software* adicionados serão suficientes [53]. É importante salientar que esta tecnologia tem como base os *namespaces* e *cgroups* do Linux. O que facilita associar um conjunto de processo e recursos, como memória ou CPU e segregar de outro processos e recursos. Logo, como exemplo de comparação entre contentores e VMs tem-se que, um contentor pode iniciar em 50 milissegundos, enquanto uma VM pode levar de 30 a 40 segundos para iniciar [54]. Na Figura 2.3 existe uma arquitetura de contentor simples, em comparação a uma arquitetura simples de VM.

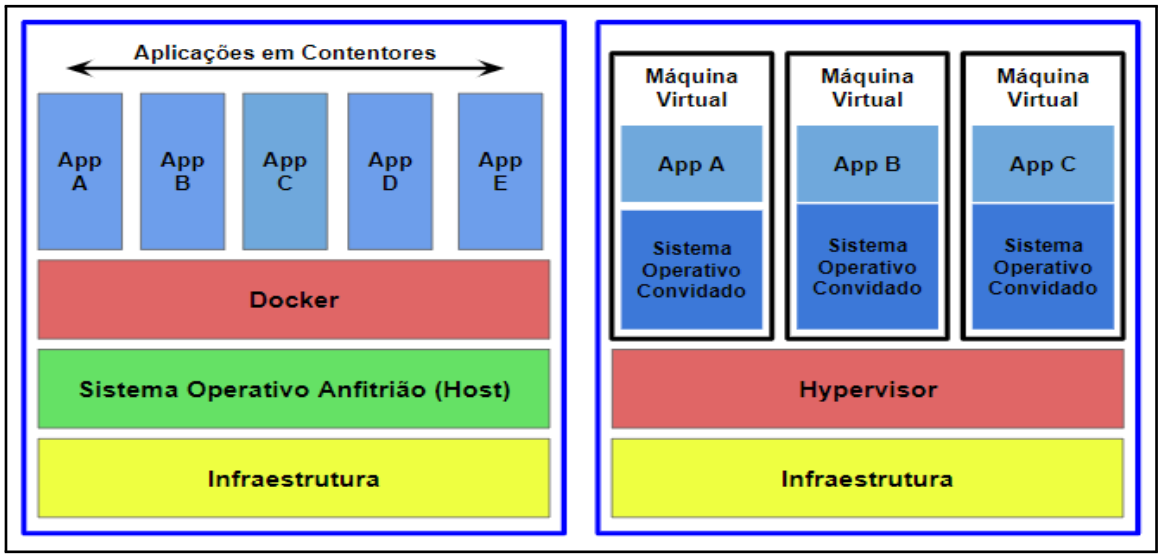


Figura 2.3: Comparação entre contentor e *hypervisor*: (a) Contentor; (b) Máquina Virtual (redesenhado a partir de [7]).

[5] https://www.docker.com/docker-news-and-press/dotcloud-inc-now-docker-inc

2.3.1 Tipos de Contentores

Os contentores estão classificados de acordo com a sua utilização. E podem ser divididos em **contentores de sistema** ou **contentores de aplicação**. Um contentor de sistema pode executar um sistema operativo, semelhante a como um sistema operativo seria executado encapsulado em uma VM. Já os contentores de aplicação permitem que o utilizador crie e execute um contentor separado para várias aplicações independentes ou vários serviços que constituem uma única aplicação.

Contentor de Sistema

Os contentores de sistema exercem uma função semelhante às VMs, pois partilham o *kernel* do sistema operativo *host* e proporcionam o isolamento do espaço do utilizador. Esta tecnologia é construída com os recursos do *cgroups* e *namespaces*, podendo executar diferentes serviços como se fosse um servidor físico. E ainda suporta a instalação de diferentes bibliotecas, linguagens e bases de dados similar a qualquer sistema operativo. Na prática, uma das grandes vantagens dos contentores de sistema é viabilizar o mesmo ambiente para desenvolvimento e produção. Como o grau de abstração do *hardware* chega ao nível da aplicação, o mesmo contentor testado pelo desenvolvedor será utilizado no servidor. Como exemplo desta tecnologia pode-se citar: LXC[6] e OpenVZ[7].

Contentor de Aplicação

Enquanto os contentores do sistema operativo foram desenvolvidos para executar vários processos e serviços, os contentores de aplicação são desenvolvidos para empacotar, isolar e executar um único serviço, ou seja, encapsulam um ambiente de execução leve para a aplicação. Oferecem grandes vantagem para a arquitetura de microsserviços, em contrapartida a arquitetura monolítica, porque permitem criar um contentor separado para cada componente da aplicação e fornecem maior controlo, segurança e restrição de processo. Microsserviços e contentores são assuntos intimamente relacionados, onde os contentores são considerados a opção de implantação padronizada para microsserviços [25]. Os contentores de aplicação são alternativas importantes para as VMs e têm vários benefícios sobre elas, especialmente em desempenho e tamanho [52]. Pois, executar cada microsserviço em uma VM separada não é eficiente porque as VMs são pesadas em comparação com os contentores, isto é, cada VM requer seu próprio sistema operativo [55]. Como exemplo prático de tecnologias de contentor de aplicação pode-se citar: Docker[8] e Rocket[9]. Na tabela 2.3 tem-se um resumo da compraração entre os tipos de contentores e seus exemplos prático.

[6] `https://linuxcontainers.org/`
[7] `https://openvz.org/`
[8] `https://www.docker.com/`
[9] `https://rocket.readthedocs.io/en/latest/README/`

Tabela 2.3: Comparação entre tipos de contentores adaptado de [56].

Tipos de Contentores	
Contentores de Sistema	**Contentores de Aplicação**
Utilizado como sistema operativo.	Destinado a funcionar para um único serviço.
Nenhum sistema de arquivos em camadas por padrão.	Sistema de arquivo em camadas.
Construídos em *cgroups*, *namespaces*, isolamento de recursos de processo nativo.	Construído com base em tecnologia de contentor de sistema operativo.
Exemplos: LXC, OpenVZ, Linux VServer, BSD Jails e Solaris Zones.	Exemplos: Docker e Rocket.

2.3.2 Isolamento e Alocação de Recursos em Contentores

Os ***namespaces*** e ***cgroups*** são caractersitcas presente no *kernel* do Linux e são responsável por dois conceitos fundamentais para a construção dos contentores: o **isolamento** e a **limitação de recursos**. Tecnologia como Docker e LXC, contam com os *namespaces* do *kernel* e grupos de controle (*cgroups*) para isolar a aplicação em execução dentro do contentor [15].

Os ***namespaces*** são um dos principais recursos do *kernel* do Linux e são responsáveis por fazerem a separação entre os recursos do *kernel*. Garantem que um processo só pode ver o conjunto especificado de recursos alocados. São um aspecto fundamental dos contentores no Linux. Um *namespace* envolve um recurso do sistema global em uma abstração que faz com que pareça aos processos dentro do *namespace* que eles têm sua própria instância isolada do recurso global [57].

Os ***cgroups*** permitem o controlo sobre quais recursos do *host* são alocados para os contentores e quando são alocados. Os grupos de controlo, geralmente chamados de *cgroups*, são um recurso do *kernel* do Linux que permite que os processos sejam organizados em grupos cujo a utilização de vários tipos de recursos pode ser limitado e monitorado [58]. Isto é, *cgroups* permitem restringir e medir as alocações de recursos para cada grupo de processos.

Utilizando *cgroups*, pode-se alocar recursos como CPU, rede, memória, largura de banda da rede, disco e prioridade. Os mesmos, portanto, pode-se garantir que cada máquina virtual tenha exatamente aqueles recursos que são realmente necessários [59]. Para finalizar há diversos tipos de *namespaces* disponíveis nas representações Linux que são apresentados na tabela 2.4 a seguir. A segunda coluna da tabela mostra o valor do sinalizador que é usado para especificar o tipo de *namespace* em várias APIs. E a terceira coluna é um resumo dos recursos isolados pelo tipo de *namespace* [57].

Tabela 2.4: Tipos de *namespaces* (adaptado de [57]).

Tipos de *Namespaces*		
Namespace	***Flag***	**Descrição**
cgroup	*CLONE_NEWCGROUP*	Diretório raiz.
IPC (*InterProcess Communication*)	*CLONE_NEWIPC*	Gerenciando o acesso a recursos IPC. Sistema V IPC, filas de mensagens POSIX.
Network	*CLONE_NEWNET*	Gerenciando interfaces de rede. Dispositivos de rede, pilhas, portas, etc.
Mount	*CLONE_NEWNS*	Pontos de montagem do sistema de arquivos.
PID (*Process ID*)	*CLONE_NEWPID*	IDs de processo. Isolamento do processo.
Time	*CLONE_NEWTIME*	*Clocks* de inicialização e monotônicos.
User	*CLONE_NEWUSER*	IDs de usuário e grupos.
UTS (*Unix Timesharing System*)	*CLONE_NEWUTS*	Isolando identificadores de kernel e versão. Nome do host e nome de domínio NIS.

2.3.3 Tecnologia de Contentores

A nova geração de virtualização são os contentores [60]. Os contentores são técnicas de virtualização baseadas em *namespaces* e *cgroups* e são adequados para gerenciamento de uma aplicação no modelo de serviço em nuvem PaaS [27]. Nesta secção do trabalho será apresentado as principais tecnologia de contentores.

Contentores Linux - LXC

LXC foi a primeira técnica de contentor amplamente utilizada [60]. LXC é uma tecnologia de virtualização que atualmente funciona usando recursos do *kernel* do Linux, *cgroups* e *namespaces*. Os contentores Linux fornecem uma virtualização leve por meio do agrupamento de recursos como processos, arquivos e dispositivos em espaços isolados [14].

Onde, o *cgroups*, descrito na secção 2.3.2, é usado como o gerenciamento de recursos no LXC, inclui: contagem de núcleos, limite de memória, limite de E/S de disco, etc. E o *namespaces*, descrito na secção 2.3.2, é usados para isolar os contentores em processo, inclui: PID (filho PID / PID aninhado), sistema de arquivos (*chroot*), etc [61]. Além deste recursos, para fornecer o serviço, o LXC conta com mais os seguintes recursos presente no *kernel*:

- **AppArmor**[10]: é um sistema de Controle de Acesso Mandatório (MAC - *Mandatory Access Control*) construído sobre a *interface* LSM (*Linux Security Modules*) do Linux. O modelo de segurança do AppArmor é vincular atributos de controle de acesso a programas, e não a utilizadores [62];
- **SELinux**[11]: *Security-Enhanced Linux* é uma arquitetura de segurança para sistemas Linux que fornece uma variedade de políticas de segurança por meio do LSM (*Linux Security Modules*). Na prática, o *kernel* consulta o SELinux antes de cada chamada do sistema para saber se o processo está autorizado a fazer a operação dada [63];
- **Seccomp**[12]: é um recurso do *kernel* Linux que opera em estado de computação segura do processo [64];
- **Chroots**[13]: é uma operação que altera o diretório raiz aparente para o processo em execução atual e seus filhos [47, 65];
- **Capacidades de kernel**: fornecem controlo refinado sobre as permissões de superusuário, permitindo que o uso do usuário *root* seja evitado [66].

Para finalizar, há um servidor de imagens de sistema para LXC e LXD[14] denominado Linux Containers - Image server[15], onde é possível criar e publicar imagens no repositório público.

Docker

Docker é considerado uma tecnologias de contentores leve, ou seja de fácil execução, devido ao fato de que permitem aos utilizadores implantar aplicações em qualquer ambiente de forma mais rápida e eficiente do que usando máquinas virtuais. O Docker armazena imagem por meio do Docker *registry* e oferece um registro público de serviços e aplicações com dependências completas por meio do Docker *Hub*[16] como uma plataforma de código aberto [60, 67].

O Docker estende o LXC com mais *kernel* e recursos baseados em aplicação para gerenciar dados, processos e isolamento avançado dentro do sistema operativo *host* e utiliza uma arquitetura cliente-servidor [60]. Na arquitetura Docker, o Docker *client* se comunica com o Docker *daemon*, que faz o trabalho de construir, executar e distribuir contentores Docker [67]. O Docker *daemon* é residente no segundo plano do módulo, que inclui principalmente duas partes: servidor Docker e mecanismo Docker [68]. O Docker *client* e o *daemon* podem ser executados no mesmo sistema ou pode conectar um Docker client a um *daemon* remoto do Docker [67].

[10] `https://debian-handbook.info/browse/pt-BR/stable/sect.apparmor.html`
[11] `https://wiki.archlinux.org/index.php/SELinux`
[12] `https://man7.org/linux/man-pages/man2/seccomp.2.html`
[13] `https://man7.org/linux/man-pages/man2/chroot.2.html`
[14] LXD é um sistema gerenciador de contentores. `https://linuxcontainers.org/lxd/introduction/`
[15] `https://uk.images.linuxcontainers.org/`
[16] `https://hub.docker.com/`

Na criação de contentores Docker há elementos, denominados de objetos, nas quais podem-se destacar:

- **Imagens**: é um *template* que será utilizado pelo contentor. Nelas são definidos quais as bibliotecas e configurações estarão presentes no contentor. Uma imagem é um modelo somente de leitura com instruções para criar um contentor [67];
- **Contentores**: um contentor no contexto do Docker. É um ambiente isolado entre o sistema operativo e outros contentores, a nível de processos, incluindo sistemas de arquivos e recursos de rede. Um contentor é uma instância executável de uma imagem [67];
- **Serviços**: permitem dimensionar contentores em vários *daemons* Docker. Um serviço permite definir o estado desejado, como o número de réplicas do serviço que devem estar disponíveis a qualquer momento [67].

OpenVZ

Open Virtuozzo (OpenVZ) é uma solução que implementa uma técnica de virtualização a nível de sistema operativo que virtualiza servidores na camada do sistema operativo (*kernel*) [69]. É uma solução gratuita e de código aberto. Cada contentor é executado como um servidor autônomo; um contentor pode ser reiniciado de forma independente e ter acesso *root*, utilizadores, endereços IP, memória, processos, arquivos, aplicação, bibliotecas do sistema e arquivos de configuração [50].

rkt

Rkt é um mecanismo de contentor, que tem como função empacotar aplicação tornando contentores portáteis para simplificar a implantação do ambiente [70]. Representa uma abordagem nativa de *pod*[17], um ambiente de execução plugável e uma área de superfície bem definida que o torna ideal para integração com outros sistemas [72]. A arquitetura do Rkt significa que cada *pod* é executado diretamente no modelo de processo Unix clássico, ou seja, não há *daemon* central, em um ambiente autocontido e isolado [72].

Estas e outras tecnologias de contentores são listadas na página *online Top 10 Best Container Software In 2021* [73].

2.3.4 Chamadas de Sistema

As chamadas de sistema (*System Calls*) oferecem uma estrutura de monitorização dos contentores de forma confiável e tecnologicamente possível. São mecanismo de rastreio

[17]Os *pods* são os menores objetos implantáveis mais básicos no Kubernetes e representa uma única instância de um processo em execução em seu cluster [71].

simples e poderosos, isto é, são funções usadas por uma aplicação para solicitar a execução de serviços ao núcleo do sistema operativo. Caso uma aplicação se comporte de uma forma que não se espera, pode verificar pelas suas chamadas de sistema para obter uma reposta do que aconteceu exatamente durante a sua execução, porque, as chamadas de sistema são funcionalidades robustas, uma vez que refletem o comportamento do processo [74]. Logo, ao monitorizar as chamadas de sistema entre o contentor e o núcleo do sistema operativo, pode-se aprender sobre comportamento do recipiente de forma a detetar qualquer mudança de comportamento, o que pode retratar uma tentativa de intrusão contra o recipiente [15]. Dentre as ferramentas de monitorização das chamadas de sistema destacam-se: *strace* e *sysdig*.

Strace

É uma ferramenta útil de diagnóstico, instrução e depuração. Em uso mais simples, *strace* executa o comando especificado e intercepta e regista as chamadas de sistema que são invocadas por um processo e os sinais que são recebidos por um processo [75]. A execução do *strace* tem como base um recurso do *kernel* do Linux conhecido como *ptrace* [76].

O recurso *ptrace* fornece um meio pelo qual um processo pode observar e controlar a execução de outro processo e examinar e alterar a memória e os registos dos traços de rastreamento. É usado principalmente para implementar a depuração de pontos de interrupção e rastreamento de chamadas de sistema [77].

A depuração de pontos de interrupção e rastreamento de chamadas de sistema do mecanismo de funcionamento do *ptrace* pode proporcionar um impacto no desempenho do processo de coleta de rastreamento principalmente em ambientes de produção ao utilizar o *strace* .

Sysdig

A ferramenta de monitorização *sysdig* de código aberto é ideal para solução de problemas do sistema, análise e exploração. É utilizada para capturar, filtrar e decodificar as chamadas de sistema e outros eventos do sistema operativo. Esta ferramenta inclui uma linguagem específica de filtros e de pequenos *scripts* que analisam o fluxo de eventos do *sysdig* para realizar ações úteis, denominado de *Chisels* [78].

A combinação de diversas ferramentas de monitorização *strace*, *tcpdump*[18], *htop*[19], *iftop*[20] e *lsof*[21] é parte da infraestrutura *sysdig*. Devido a sua arquitetura exclusiva permite uma inspeção profunda em contentores oferecendo suporte nativo para todas as tecnologias de contentores Linux, incluindo Docker, LXC [79]. Na tabela 2.5 são destacadas as principais características entre as ferramentas *strace* e *sysdig*.

[18] `https://www.tcpdump.org/manpages/tcpdump.1.html`
[19] `https://htop.dev/`
[20] `http://www.ex-parrot.com/pdw/iftop/`
[21] `https://github.com/lsof-org/lsof`

Tabela 2.5: Comparação entre *strace* e *sysdig*.

Sysdig* vs *Strace			
Ferramentas	**Chamadas de Sistema**	**Foco Principal**	**Suporte**
Strace	sim	Processo	OS
Sysdig	sim	Contentor	OS, Docker, LXC

2.3.5 Segurança em Contentores

A segurança é o grande desafio na execução de um serviço em ambiente virtual, principalmente no sistema *multi-tenant* [80]. As vulnerabilidades apresentadas pelos contentores variam da explorações de *kernel* a ataques aos recursos compartilhados do *host* Linux a configurações erradas, canais laterais e vazamento de dados [5].

A segurança em ambiente *multi-tenant* envolve diversos componentes das tecnologias dos contentores como imagens, registos, orquestradores, contentores e sistemas operativos *hosts*. Existem dois tipos de riscos a considerar: Primeiro, a imagem ou contentor comprometido. E segundo o uso indevido de um contentor para atacar outros contentor e o sistema operativo *host* [10].

Em continuação a temática em questão existem diversos artigos de investigação que abordam a questão da segurança num ambiente de contentores. Um resumo com foco nas principais abordagens destes artigos em relação as metodologias de segurança para contentores e suas ameaças será apresentado.

Abed et al. [15] citam que existem três cenários de ataques possível no ambiente *multi-tenant*. Primeiro, uma fonte de ataque tem origem fora do *host*, atacando o *kernel* do *host* e/ou os contentores. Segundo, outra fonte de ataque vem de outros contentores que residem no mesmo *host* e atacam contentores vizinhos. E por último uma terceira classe de ataque ocorre quando um contentor ataca o *kernel* do *host*.

Sultan et al. [52] apresentam soluções de segurança de contentores na nuvem baseadas em *software* e *hardware*. É apresentado quatro situações generalizado: (I) proteger um contentor de uma aplicação dentro dele, (II) proteção entre contentores, (III) proteger o *host* de contentores e (IV) proteger contentores de um *host* malicioso ou semi-honesto. Logo, para as três primeiras situações utilizam soluções em *software* e contam com os recursos do *kernel* Linux e módulos de segurança do Linux. O quarto caso depende de soluções baseadas em *hardware*, como módulo de plataforma confiável, (*Trusted Platform Module* -TPMs), e suporte de plataforma confiável (por exemplo, Intel SGX).

Mattetti et al. [14] descrevem a existência de dois tipos principais de mecanismos de proteção que podem ser aplicados a ambientes de contentores: mecanismos de proteção de

segurança (AppArmor e SELinux) e sistemas de deteção de intrusões baseados em *host*. Em sua pesquisa a aplicação de ambos mecanismos de proteção apresenta um certo nível de dificuldade devido ao agrupamento dos processos e aplicação no *host* e dentro do contentor. Neste mesmo trabalho é exposto dois tipos de ameaças: ameaças no *host* e ameaças às cargas de trabalho. As ameaças ao *host*, são as que colocam em risco todo o sistema ao sair de contentores e obtém privilégios de administradores, permitindo a execução arbitrária de código e a propagação de ataques a outros componentes na mesma rede. No caso das ameaças às cargas de trabalho dos contentores, são as que colocam em risco as cargas de trabalho que são executadas dentro dos contentores e os principais perigos são o vazamento de dados confidenciais do cliente e danos às suas cargas de trabalho ou faturamento.

Bui et al. [5] elaboram um estudo sobre o nível de segurança da tecnologia de contentores Docker. O estudo considera duas áreas: (1) a segurança interna do Docker e (2) como o Docker interage com os recursos de segurança do *kernel* do Linux, como SELinux e AppArmor, para proteger o sistema *host*. Durante a análise verificou que a tecnologia Docker fornece um alto nível de isolamento e limitação de recursos para seus contentores utilizado as configurações do *namespaces*, *cgroups* e seu sistema de arquivos *copy-on-write*, mesmo com a configuração padrão. Por fim, seu artigo científico demonstra que os contentores do Docker são seguros, mesmo com a configuração padrão. E que o nível de segurança dos contentores Docker também pode ser aumentado se o operador os executar como ”sem privilégios” e permitir soluções de proteção adicionais no *kernel* do Linux, como AppArmor ou SELinux.

Para finalizar, Xin et al. [81] verificaram-se a segurança em contentores Linux através de um conjunto de dados com 223 *exploits*[22]. Desde 88 *exploits*, do conjunto de dados, foram utilizado para avaliar os mecanismos de segurança de contentores Linux. Descobriu-se que os *exploits* de escalonamento de privilégios podem escapar com sucesso para fora do contentor e comprometer o *host*. Sob a proteção dos mecanismos de segurança, ainda existem 11 *exploits* que podem quebrar o isolamento do contentor. Observou-se ainda, em destaque:

1. os mecanismos de segurança do *kernel*, como Capability, SECCOMP e MAC, desempenham um papel mais importante na prevenção de ataques de escalonamento de privilégios do que os mecanismos de isolamento do contentor, ou seja, *namespace* e *cgroup*;

2. cada mecanismo de segurança do *kernel* incluindo *namespace*, *cgroup*, SECCOMP, Capabilities e MAC restringe as permissões do *kernel* de ângulos diferentes de maneira fragmentada, enquanto os enquadramentos entre eles são confusos e complicados. A configuração inadequada desses mecanismos de segurança pode reduzir sua capacidade de proteção do contentor.

[22] É uma parte de um *software*, um fragmento de dados ou uma sequência de comandos que tomam vantagem de um defeito, falha ou vulnerabilidade [82].

3. Os mecanismos de segurança dos contentores dependem da segurança do *kernel*, enquanto a relação de interdependência e influência mútua requer configurações cuidadosas para derrotar eficazmente os ataques de escalonamento de privilégios.

2.4 Sistema de Detecção de Intrusões

A *Internet* representa um fonte valiosa de informações relacionadas a diversos setores da sociedade. Estas informações atrai a atenção de grupos ou agente maliciosos que tentam acessar ou comprometer a integridade, confidencialidade ou disponibilidade destes recursos. Com o recente interesse e o progresso no desenvolvimento de tecnologias de *Internet* e comunicação na última década, a segurança de rede surgiu como um domínio de pesquisa vital [83].

Muitas investigações demonstram que a invasão da rede (*Internet*) registou um aumento consistente e levou ao roubo de privacidade pessoal e se tornou uma das principais plataformas de ataque nos últimos anos [84]. Ilustrado na Figura 2.4, no Relatório de Cibersegurança em Portugal, observa-se o aumento do número de incidentes registados e de vulnerabilidades identificadas.

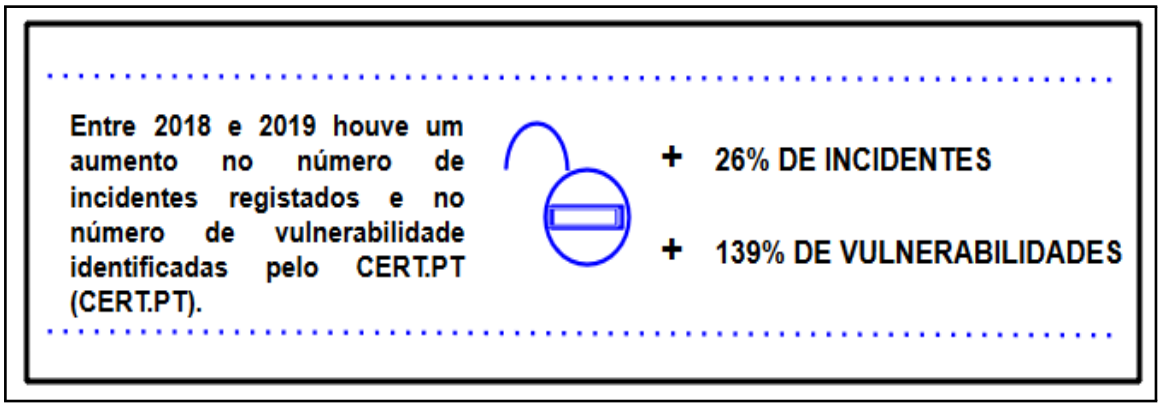


Figura 2.4: Relatório de cibersegurança em Portugal (redesenhado a partir de [85]).

O desvio do comportamento do utilizador é considerado uma intrusão [86]. Monitorizar os eventos e informar ao utilizador a ocorrência dos mesmos em tempo real é um das atividades dos sistemas de deteção de intrusões. A deteção de intrusões é considerado um mecanismos de segurança para identificar o comportamento do utilizador autorizado ou não autorizado no sistema ou na rede [86]. Os mesmos fornecem a capacidade de identificar violações de segurança em um sistema [87]. Portanto, sistemas de deteção de intrusões são projetados para reconhecer tentativas de intrusões, bloquear ataques e produzir alertas que podem ser analisados posteriormente [88].

2.4.1 Métodos de Deteção de Intrusões: por Assinaturas e por Anomalias

A deteção de intrusões é o processo de monitorizar os eventos que ocorrem em um sistema de computador ou rede e analisá-los em busca de sinais de possíveis atividades maliciosas [11]. Há tradicionalmente duas metodologias de deteção de intrusões, cada uma tem vantagens e desvantagens respectivas e ambas estão disponíveis para os sistemas baseados em *hosts* e/ou redes. A deteção de intrusões é um mecanismo de segurança representativo que abrange ferramentas, técnicas e estratégias para reconhecer ameaças em evolução, contribuindo assim para uma estrutura de computação segura e confiável [89]. É importante destacar que devido à natureza complementar dessas duas abordagens, muitos sistemas tentam combinar as duas técnicas, denominado **Deteção Híbrida** [90]. No entanto, os métodos atuais usados para esses sistemas incluem o uso de deteção por assinaturas ou por anomalias [87]. A tabela 2.6 apresenta uma lista de diversos trabalhos publicados em *surveys* em relação a deteção de intrusões.

Tabela 2.6: Listas de trabalhos publicados em *surveys*.

Trabalhos Publicados em *Surveys*
Nadeem et al. [91]
Weller-Fahy et al. [92]
Sabahi et al. [93]
Bharathy et al. [94]
Saxena et al. [95]
Butun et al. [96]
Tidjon et al. [97]
Benkhelifa et al. [98]
Hindy et al. [99]
Uikey et al. [100]
Lakshminarayana et al. [101]
Salo et al.[102]
Nisioti et al. [103]
Chiba et al. [104]
Subaira et al. [105]

Deteção por Assinaturas

A técnica de deteção por assinaturas, também conhecida como "*Knowledge-Based Intrusion Detection*", efetua uma busca em todo tráfego de rede à procura de padrões similares ou iguais aos padrões previamente conhecidos. Esses padrões são denominados de assinaturas e incluem registos específicos de arquivos de *log* ou cabeçalhos e informações internas de pacotes, trafegando na rede, identificados como ameaça e armazenados em uma base de dados.

A deteção é baseada em assinaturas de ataques conhecidos e regras definidas por um administrador [106]. Como exemplo prático pode-se citar os sistemas de deteção e prevenção de intrusões como Snort[23], IBM X-Force[24] e Suricata[25] , todos sistemas são baseados em assinaturas que monitoram o comportamento de um sistema e o compara com um padrão predefinido de comportamento aceitável [107].

Os algoritmos para este tipo de método são usados com base em uma abordagem de deteção de intrusões que utiliza algoritmos baseados em regras. Esses algoritmos definem os padrões de ataques conhecidos e, em seguida, compará-los com novos eventos, com o objetivo de identificar tentativas de intrusões. Assim, esses sistemas podem classificar ataques conhecidos comparando a atividade observada com padrões armazenados, mas não podem identificar novos ataques [106].

Tabela 2.7: Vantagens e desvantagens da deteção por assinaturas adaptado de [108].

Deteção por Assinatura	
Vantagens	Em relação à deteção por anomalias, são muito mais eficientes, gerando um número menor de alarmes falsos. Contra medidas podem ser imediatas e redução na quantidade de informação a ser tratadas.
Desvantagens	Somente identifica ataques conhecidos, ou seja, compara os dados analisados com as assinaturas armazenadas em uma base de dados, necessita de uma constante atualização no conjunto de assinaturas. Como grande parte das assinaturas são muito especificas, novos ataques e variações de ataques conhecido dificilmente são detectados.

Deteção por Anomalias

A deteção de anomalias é considerada uma técnica aplicada em diversos contextos não só limitado à segurança de sistemas, mas também em estatística, finanças e análise de fraude [109]. Neste paradigma é necessário definir o que é o comportamento normal e buscar por atividades que se desviam do comportamento normal pré-estabelecido, denominada de anomalia. Anomalia é um evento que causa uma alteração em relação ao perfil padrão do sistema [110]. A deteção por anomalias pressupõe que seja possível criar um modelo de atividade normal do sistema [106].

As técnicas de deteção de anomalias podem ser classificadas em um dos três modos a seguir: Primeiro, a **deteção de anomalias supervisionada** processa dados rotulados em uma base de dados que apresenta um *label* para cada amostra, como "normal" e "anormal". Segundo, no caso da **deteção de anomalias semi-supervisionada** a abordagem típica é construir um modelo para a classe correspondente ao comportamento normal e usar o modelo para identificar anomalias nos dados de teste. E por último a **deteção de anomalias não supervisionadas** que detecta anomalias com dados não rotulados,

[23] https://www.snort.org/
[24] https://www.ibm.com/security/xforce
[25] https://suricata-ids.org/

assumindo que a proporção das instâncias "normais" seja predominante na base de amostras [109].

Logo, a deteção por anomalias analisa estatisticamente o comportamento da rede ou do utilizador. Os sistemas de deteção de intrusões baseados em anomalias procuram desvios de comportamentos normais [106]. Não utiliza uma base de dados, mas sim um padrão de comportamento da rede ou do utilizador. É também conhecido como "*Behavior-Based Intrusion Detection*".

Tabela 2.8: Vantagens e desvantagens da deteção por anomalias adaptado de [108].

Deteção por Anomalias	
Vantagens	Capaz de detectar ataques desconhecidos. Produz um conjunto de dados que podem ser utilizado na definição de assinaturas para detentores baseados em assinaturas.
Desvantagens	Produz um grande número de alarmes falsos devido ao comportamento imprevisível de utilizadores e sistemas. Requer sessões de coleta de amostra de dados do sistema, de modo a caracterizar os padrões de comportamento normais.

2.4.2 Algoritmos de Deteção de Anomalias

Esta subsecção relata diversos algoritmos de deteção de anomalias que podem ser utilizados para construção de uma nova abordagem para os sistemas de deteção de intrusões.

- **AdaBoost**: proposto em 1995 por Freund e Schapire como um algoritmo eficiente do campo de aprendizagem do conjunto [111]. Combina iterativamente os classificadores fracos, tendo em conta uma distribuição de peso nas amostras de treino, de modo a que mais peso seja atribuído a amostras mal classificadas pelas iterações anteriores [112];
- **Algoritmo C4.5**: publicado em 1987, tendo como desenvolvedor John Ross Quinlan. É um algoritmo utilizado para criar árvores de decisão utilizadas para classificação e são portanto conhecidas como classificadores estatísticos. É considerado uma extensão do algoritmo anterior de Quinlan's ID3 [113]. C4.5 constrói árvores de decisão a partir de um conjunto de dados de formação da mesma forma que ID3, usando o conceito de entropia da informação [114];
- **Bags of System Calls (BoSC)**: pode considera-se que a análise das chamadas de sistema entre o contentor e o *kernel* do *host*, tem como resultado o padrão de comportamento do contentor. Qualquer mudança do padrão de comportamento, pode significar em uma tentativa de intrusão contra o contentor. A técnica *Bag of System Calls* (BoSC) é uma das abordagens básicas para deteção de anomalias utilizando chamadas de sistema [15]. Esta técnica é uma deteção de anomalias baseada em

frequência inteira. É um recurso definido como uma lista ordenada Xi = (c1, c2, c3,..., cm) onde m = | Σ | e Cj é o número de ocorrência da chamada de sistema Sj na sequência de entrada Zi [115];

- **Hidden Markov Models (HMM)**: é um método probabilístico utilizado em muitas aplicações, como reconhecimento de voz e diagnósticos médicos e oferece uma estrutura matemática sólida que pode efetivamente formar uma base teórica robusta para a modelagem de problemas [116]. Pode ser aplicado em deteção de intrusões, em que a sequência de eventos é considerada anômala se a probabilidade de ele pertencer ao perfil definido está abaixo de um determinado limite definido pelo utilizador [117];

- **K-Nearest Neighbors (KNN)**: o algoritmo de agrupamento *K-means* foi proposto por Macqueen em 1967. Esta é uma das técnicas de particionamento mais frequentemente usadas. *K-means* usa *K-centroids* para definir *k-clusters* [118]. Exclusivamente, quando um novo evento é processado, os K vizinhos mais próximos a ele são calculados e o rótulo mais frequente entre eles é atribuído ao novo evento [119];

- **Naïve Bayes (NB)**: é um classificador probabilístico baseado no Teorema de Bayes com independência entre recursos. É um algoritmo simples e escalonável que pode ser eficaz e eficiente em conjuntos de dados maiores [120]. Assim, este método permite classificar novos eventos com base nos anteriores, resultando na capacidade de identificar possíveis intrusões com determinada probabilidade [121];

- **Random Forest**: criado por Tin Kam Ho [122]. São um tipo de métodos de conjuntos de classificação e regressão que constroem algumas árvores de decisão aleatórias idênticas e fazem previsões mediando os resultados de árvores individuais [123];

- **Redes Neurais Artificiais (RNA)**: (*Artificial Neural Networks*) ANNs, são definidas como sistemas de processamento paralelo e distribuído construídos com um grande número de processadores simples e enormemente conectados [124]. As RNAs aumentam a capacidade dos sistemas de previsão de intrusão de generalizar os dados e classificá-los em normais ou anormais [125];

- **Repeated Incremental Pruning to Produce Error Reduction (RIPPER)**: é um classificador baseado em regras amplamente usado para classificação binária e multiclasse [126]. O algoritmo foi projetado por Cohen em 1995 e progride por quatro fases: growth, pruning, optimization e selection [127];

- **Sequence Time-Delay Embedding (STIDE)**: esta técnica define o comportamento normal usando uma base de dados de sequências curtas, cada um com tamanho k. Inicialmente introduzida por Forrest e Longsta [128]. Para construir a base de dados, eles deslizam uma janela de tamanho k + 1 sobre o rastreamento de chamadas de sistema e para armazenar as sequências de chamadas de sistema. É

técnica simples e eficiente, pode-se perceber que, mantendo as informações do pedido das chamadas, o tamanho da base de dados pode crescer linearmente com o número de chamadas de sistema no rastreamento [15];

- **Support Vector Machine (SVM)**: são sistemas de aprendizagem que usam um espaço de hipóteses de funções lineares em um espaço de recursos de alta dimensão, treinados com um algoritmo de aprendizagem da teoria de otimização. O algoritmo SVM é baseado na ideia de um classificador hiperplano, ou separabilidade linear [129]. Este método classificador pode ser usado para detetar anomalias, pois usa o conhecimento prévio, adquirido com o treinamento, para dividir eventos em classes diferentes separados por um plano superior, que é definido por um número de vetores de suporte usados para definir limites [130].

2.4.3 Tipos de Sistema de Deteção de Intrusões: Baseados em *Host* e Baseados em Redes

A deteção de intrusões permite que as organizações protejam seus sistemas das ameaças que vêm com o aumento da conectividade de rede e da dependência de sistemas de informação [131]. Um sistema de deteção de intrusões é um *software* que analisa sistemas locais (máquinas ou sistemas operativos) ou suportes de comunicação (protocolos da rede de comunicação) em busca de atividades maliciosas, denominada intrusões. Há, basicamente, duas classificações de IDS que são IDS ativo e IDS passivo.

O IDS ativo também conhecido como sistema de deteção e prevenção de intrusões, (*Intrusion Detection and Prevention System*). O IDPS é desenvolvido para bloquear ataques mecanicamente desacreditados, sem interferência de um operador [132]. O IDPS tem o benefício de fornecer ação disciplinar em tempo real em resposta ao ataque [133].

No caso do IDS passivo é um sistema planeado para monitorizar e investigar a atividade de tráfego de rede e alertar um operativo para potenciais susceptibilidades e ataques, não é hábil para executar qualquer função defensiva ou correctiva por conta própria [133].

Pode-se dizer que um sistema de deteção de intrusões começa onde o *firewall* termina [132]. É usado para a proteção da infraestrutura de rede e computadores contra atividades maliciosas e usos não autorizados [134]. Quando uma atividade não autorizada é detetada, o sitema de deteção de intrusões produz uma notificação ou alerta que informa aos administradores do sistema sobre possíveis ataques. Estas atividades não autorizados são intrusos que são classificadas como: externos e internos.

O intruso externo é um utilizador não autorizado, não autenticado do mundo externo, enquanto o intruso interno é um utilizador autorizado que tenta acessar recursos de dados não autenticados do sistema [132].

Dessa forma, um sistema de deteção de intrusões é uma ferramenta que pode lidar com

ataques originados de instalações de *Internet* ou *Intranet* [135]. Por fim, a estrutura de deteção de intrusões vem em dois formatos principais: um sistema de deteção de intrusões baseado em *host* e um sistema de deteção de intrusões baseado em rede [89].

Sistema de Deteção de Intrusões baseado em *Host*

Um sistema de deteção de intrusões baseado em *host*, HIDS, é capaz de controlar e analisar as partes internas de um sistema de computação, analisando dados tais como arquivos de *log*, bem como os pacotes de rede e suas *interfaces*, chamada de sistema e dados de equipamentos como *firewall* e roteadores.

O HIDS tem como função analisar os principais arquivos do sistema, comportamentos do processo, utilização incomum de recursos, acesso não autorizado, etc [134]. Em alguns casos tem integração com o *syslog*. Os sistema de deteção de intrusões baseados em *host* operam com informações coletadas em um sistema de computador individual [131]. Entretanto, este ponto de vantagem permite que os sistema de deteção de intrusões baseados em *host* analisem atividades com grande confiabilidade e precisão, determinando exatamente quais processos e utilizadores estão envolvidos em um determinado ataque ao sistema operativo [131].

No entanto, a principal desvantagem é não poder ser centralizado, isto é, cada computador precisa de um sistema HIDS instalado, consumindo recursos e adicionando latência ao tráfego de rede. Em comparação ao NIDS os sistemas HIDS são mais susceptíveis a ataques que os NIDS por estarem dentro do *host* e geralmente possuírem privilégios de administrador [131]. Ilustrado na figura 2.5 tem-se um exemplo de rede utilizado um HIDS.

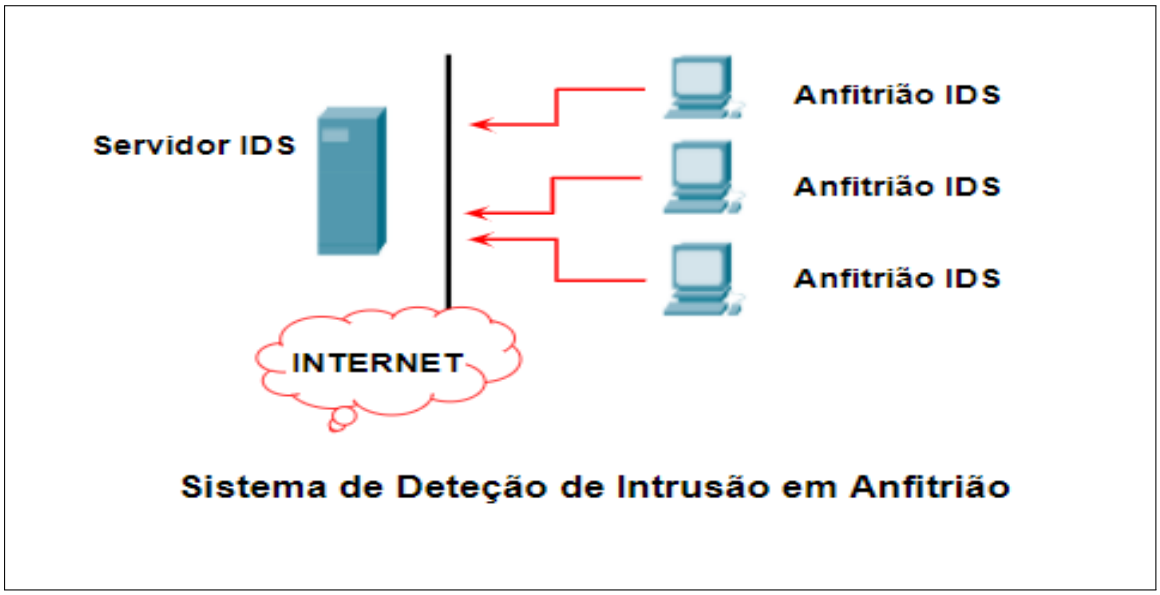


Figura 2.5: Sistema de deteção de intrusões baseado em *host* (redesenhado a partir de [95]).

Sistema de Deteção de Intrusões baseado em Rede

A maior parte dos sistemas comerciais de deteção de intrusões é baseada em rede. Estes IDSs detetam ataques capturando e analisando pacotes de rede. Ouvindo em um segmento de rede ou *switch*, um IDS baseado em rede pode monitorizar o tráfego de rede que afeta vários *hosts* que estão conectados ao segmento de rede, protegendo assim esses *hosts* [131].
Em seu funcionamneto um IDSs baseados em rede geralmente consistem em um conjunto de sensores ou *hosts* de propósito único colocados em vários pontos de uma rede. Essas unidades monitorizam o tráfego de rede, realizando análises locais desse tráfego e relatando ataques a um console de gerenciamento central. Como os sensores estão limitados à execução do IDS, eles podem ser protegidos com mais facilidade contra ataques. Muitos desses sensores são projetados para funcionar no modo ”furtivo”, a fim de tornar mais difícil para um invasor determinar sua presença e localização [131].

Existem vantagens e desvantagens em relação ao uso do NIDS. Em relação as vantagens destacam-se: alguns IDSs baseados em rede bem posicionados podem monitorizar uma grande rede; a implantação de IDSs baseados em rede tem pouco impacto sobre um rede existente e IDSs baseados em rede podem ser muito seguros contra ataques e até mesmo tornado invisível para muitos atacantes. A principal desvantagem de um sistema NIDS reside no fato de que ele não pode analisar o tráfego criptografado, como é o caso de VPNs, e pode ter limitações para analisar redes com grande fluxo de dados. Além disso, a maior parte dos sistemas NIDS não pode dizer se um ataque foi bem-sucedido ou não; eles só podem discernir que um ataque foi iniciado. Isso significa que, depois que um IDS baseado em rede encontra um ataque, os administradores devem investigar manualmente cada *host* atacado para determinar se ele foi realmente invadido [131]. Como exemplo tem-se ilustrado na figura 2.6 uma rede configurada com o NIDS.

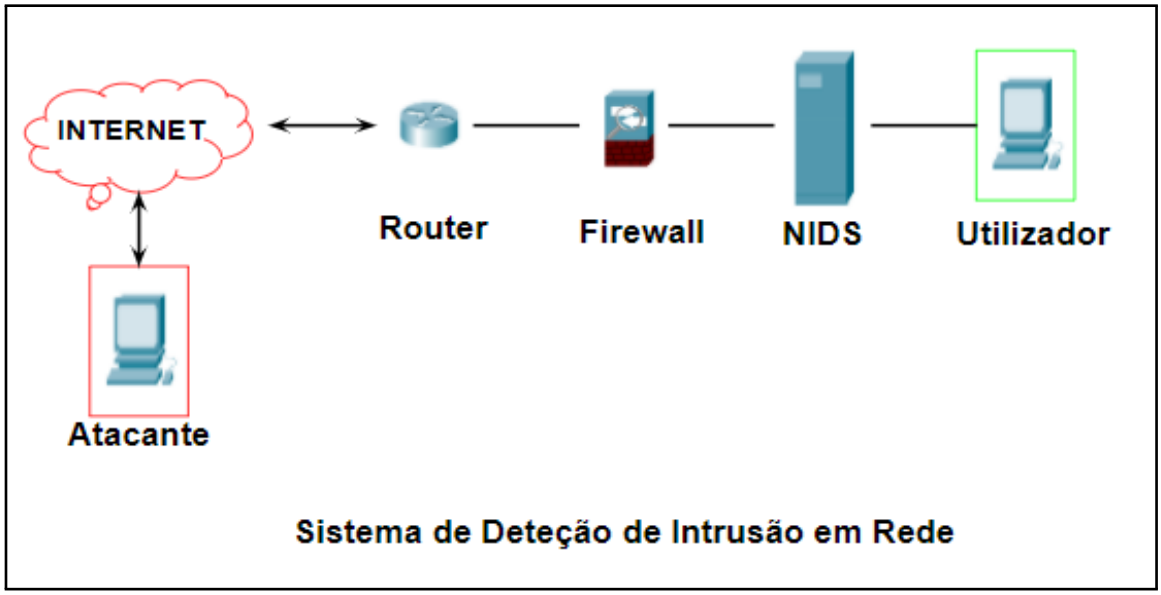


Figura 2.6: Sistema de deteção de intrusões baseado em rede (redesenhado a partir de [95]).

2.4.4 Sistema de Deteção de Intrusões Distribuído

Os sistemas híbridos de deteção de intrusões também são conhecidos como DIDS[136]. São usado em grandes redes, este sistema de deteção de intrusões distribuída consiste em vários IDS que podem ser baseados em rede ou baseado em *host* [137]. É composto por vários métodos de deteção de intrusões (por exemplo: HIDS, NIDS, etc.), todos os quais se comunicam entre si, ou com um servidor central que permite a monitorização da rede [138].

As vantagens e desvantagens do NIDS e HIDS também são aplicáveis a sistemas de deteção de intrusões híbridos. Porém, o DIDS trabalha com maior carga de dados do que os outros dois sistemas [136]. No ambiente em nuvem, DIDS pode operar no *host* do utilizador, máquinas ou no *backend* do servidor de processamento [137]. Dessa forma, o DIDS com manuseio de dados eficiente e *design* elegante parece ser capaz de deteção de intrusões do que os outros dois sistemas [136].

2.4.5 Avaliação de Sistema de Deteção de Intrusões

Para análise e comparação do desempenho dos algoritmos de deteção de anomalias de aprendizagem automática na deteção de intrusões propõem-se a utilização de uma abordagem de avaliação composta de cargas de trabalho, métricas e metodologias de medição.

Cargas de Trabalho

As cargas de trabalho são utilizadas para avaliação de um IDS em simulação de um cenário real de ataques. São divididas em cargas de trabalho **maliciosas** e **benignas**. E podem ser usados separadamente ou em conjunto, **mistas**, para medir a capacidade de um IDS, ou sua cobertura de ataque. Em relação ao formato as cargas de trabalho são classificadas em: **formato executável** e **formato de rastreamento**. No caso do **formato executável** são utilizadas para teste ao vivo de um IDS tendo como vantagem a semelhança de uma carga de trabalho real monitorada por um IDS durante a operação. Quanto ao **formato de rastreamento**, estas cargas de trabalho são geradas pela gravação de uma execução ao vivo de cargas de trabalho para reprodução posterior. É necessário o uso de ferramentas de rastreamento e reprodução [139]. Exemplo: tcpdump [140] e tcpplay [141].

No sentido de otimização para a geração de cargas de trabalho **benignas** há dois métodos a discutir: o uso de ***drivers* de cargas de trabalho** e abordagens de **geração manual**. O uso de ***drivers* de cargas de trabalho** é considerado uma prática comum para gerar cargas de trabalho benignas puras artificiais com características diferentes para testar o sistema com diferentes tipos de interação. Exemplo: IOzone [142] e mysqlslap [143]. Na abordagem de **geração manual** de cargas de trabalho, estas são aplicadas ao contexto de

avaliação de IDSs baseados em *host*, pois abrangem a execução de tarefas de utilizadores do sistema real conhecidas por utilizarem recursos específicos do sistema [139].

O uso de **base de dados de exploração** e **técnicas de injeção de vulnerabilidade e ataque** são um alternativa de modelo de atividade realista, para gerar cargas de trabalho **maliciosas**. Para a primeira empregam-se ferramentas de teste de penetração como uma base de dados de exploração prontamente disponível. Exemplo: Metasploit [144]. Na segunda, **técnicas de injeção de vulnerabilidade e ataque**, permite o teste de IDS ao vivo, primeiro injetando artificialmente o código vulnerável explorável em uma plataforma de destino e, em seguida, atacando a plataforma. É útil para avaliar IDSs baseados em *host* e em rede [139].

Os traços de produção do mundo real submetem um IDS em teste a uma carga de trabalho conforme observado durante a operação em um ambiente de produção real. Existem dois métodos para gerar cargas de trabalho **benigna**, **maliciosa** ou **mista** na forma de traços: **aquisição de traços** e **geração de traços** [139].

Na **aquisição de traços**, há duas possibilidades. Primeiro; coletar traços de um sistema real em produção de organizações ou instituição. Segundo, encontrar traços que podem ser usados para treinar e testar os IDSs, no entanto, esses traços podem está desatualizados [139].

No caso da **geração de traços**, tem-se também duas formas principais: **ambiente de teste** e o uso de ***honeypots***. A utilização de um **ambiente de teste** é definida como um cenário para realizar ataques, a fim de realizar a coleta de traços, esta técnica pode gerar traços que não são representativos de interações realistas. No caso dos ***honeypots*** estimulam a operação de sistemas reais, onde é possível coletar a interação dos atacantes que interagem com eles, fornecendo os traços que são necessários para avaliar os IDSs [139]. Na tabela 2.9 encontra-se um resumo das principais características das cargas de trabalho.

Tabela 2.9: Resumo das principais características de cargas de trabalho.

Categoria	**Tipos de cargas de trabalho**
Conteúdo	Benigna
	Maliciosa
	Mista
Formato	Executável
	Traço

Métricas

Em relação as métricas de avaliação do IDS, destaca-se duas categorias: **métricas relacionadas ao desempenho** e **relacionadas à segurança**. As **métricas relacionadas ao desempenho**, quantificam as propriedades não funcionais de um IDS em teste, como capacidade e consumo de recursos. No caso das **métricas relacionadas à segurança**, estas quantificam a precisão da deteção de ataques de um IDS. E dividem-se em **métricas básicas** e **compostas** [139].

As **métricas básicas** quantificam várias propriedades de deteção de ataques individuais. Embora sejam quantificadas individualmente, essas propriedades precisam ser analisadas em conjunto para caracterizar com precisão a eficiência de deteção de ataques de um IDS [139]. As métricas básicas são derivadas de uma matriz de confusão. A maior parte das métricas das ferramentas de classificação binária baseia-se nas medidas brutas obtidas pelo cálculo de uma matriz de confusão [145]. Essas matrizes são as combinações das possíveis situações finais em que determinada amostra, quando verificada se pode encontrar no final da fase de classificação e agrupamento. A tabela 2.10 exibe as quatro categorias de classificação originadas por um IDS.

Para a descrição das métricas, considerar o seguinte:

- TP (true positive) são os valores verdadeiros positivos;
- TN (true negative) são os valores verdadeiros negativos;
- FP (false positive) são os valores falsos positivos;
- FN (false negative) são os valores falsos negativos.

Tabela 2.10: Matriz de confusão.

		Valor Predito	
		Benigno	Malicioso
Real	Benigno	Verdadeiro Negativo (TN)	Falso Positivo (FP)
	Malicioso	Falso Negativo (FN)	Verdadeiro Positivo (TP)

Para situações de aplicações críticas ao negócio, *Recall* fornece ao utilizador informações sobre a ferramenta que apresenta melhor desempenho considerando a deteção de intrusões e *Precision* como métrica desempatador [146]. A métrica de *F-Measure* é a média harmónica entre a *Precision* e *Recall*. Como essa medida é uma média, ela oferece uma visão mais precisa da eficiência do classificador do que simplesmente *Precision* ou *Recall*.

As métricas *Recall*, *Precision* e *F-Measure* aplicadas a esta investigação serão descritas a seguir.

Recall é a proporção de casos positivos corretamente classificados como positivos [146] e é definida por:

$$Recall = \frac{TP}{TP + FN}$$

Precision é a proporção de classificações positivas verdadeiras, dado o conjunto de todas as classificações positivas [146] e é definida por:

$$Precision = \frac{TP}{TP + FP}$$

F-Measure representa a média harmónica de Precision e Recall [146] e é definida por:

$$F - Measure = 2 * \frac{prec * recall}{prec + recall} = \frac{2 * TP}{2 * TP + FN + FP}$$

As **métricas compostas** identificam os pontos operacionais ideais de um IDS. Nesta situação a combinação das métricas básicas é usada para produzir valores ótimos de taxa de deteção de verdadeiros e falsos positivo ou para comparar vários IDSs. No mesmo sentido, uma curva ROC (*Receiver Operating Characteristic*) representa vários pontos operacionais sendo útil para identificar um ponto operacional ideal ou para comparar vários IDSs [139].

Metodologia de Medição

A metodologia de medição está relacionada com a especificação das propriedades IDS e das cargas de trabalho empregadas e métricas para avaliar as propriedades. Há nove propriedades de um IDS que são comummente consideradas na prática. São elas: precisão de deteção de ataque, cobertura de ataque, resistência a técnicas de evasão, velocidade de deteção e relatório de ataque, consumo de CPU, consumo de memória, consumo de rede, sobrecarga de desempenho e capacidade de processamento de carga de trabalho. Para esta investigação destacam-se apenas duas: **Precisão de deteção de ataque** e **Cobertura de ataque**, pois estas propriedades são avaliadas para IDSs de todos os tipos. Assim, pode-se definir a **Precisão de deteção de ataque** como a precisão da deteção de ataque de um IDS na presença de cargas de trabalho mistas e a **Cobertura de ataque** como a precisão da deteção de ataques de um IDS na presença de ataques sem qualquer atividade benigna de fundo [139].

2.5 Aprendizagem Automática

O recurso computacional da aprendizagem automática tem como objetivo a criação de modelos analíticos com base em um conjunto de dados amostral. É uma técnica de análise de dados [147]. Os modelos de aprendizagem automática fazem uso de funções estatísticas para descrever os relacionamentos entre os dados.

No entanto, aprendizagem automática é um campo de estudo da inteligência artificial cuja a ideia central é de que sistemas podem aprender com dados, identificar padrões e tomar decisões com menos intervenções de humanos. Segundo, Xiuquan Li et al. [148] a inteligência artificial é a tecnologia que reproduz os processos de inteligência humana por máquinas, especialmente sistemas informáticos.

A aprendizagem automática pode ser descrito como um processo que obtém informações úteis a partir de dados, alcançados através do desenvolvimento de algoritmos de previsão fiáveis. Estes algoritmos podem ser muito poderosos na otimização, mas o seu sucesso depende principalmente da condição e do tamanho dos dados recolhidos. Os algoritmos de aprendizagem automática podem ser divididos em três categorias: aprendizagem supervisionada, aprendizagem sem supervisão e aprendizagem por reforço.

Aprendizagem Supervisionada

Os problemas de aprendizagem supervisionada podem ser divididos em classificação e regressão. No geral, os dados de entrada são chamados de dados de formação e têm um rótulo associado ou resultado é conhecido, isto é, exemplos rotulados como uma entrada com saída preferencial são usados para algoritmos de treinamento [147].

O algoritmo em problemas de classificação agrupa uma amostra em um domínio discreto, identificando-a em uma classe dentre um conjunto finito de classes possíveis. O algoritmo em problemas de regressão classifica uma amostra em um domínio contínuo, o que significa que estar tentando mapear variáveis de entrada para alguma função contínua.

Dentre as técnicas para resolver problemas de aprendizado supervisionado estão regressão linear, regressão logística, redes neurais artificiais, máquina de suporte vetorial, árvores de decisão, k-Nearest Neighbors e Naive Bayes.

Nesta investigação enquadra-se os seguintes algoritmos de classificação de aprendizagem automática do *Scikit-learn* [23]: AdaBoost, Decision Tree, Gaussian Naive Bayes, K-Nearest Neighbors, Multi-layer Perceptron, Multinomial Naive Bayes, Random Forest e Support Vector Machine.

Aprendizagem não Supervisionada

Neste tipo de aprendizagem nenhum tipo de etiqueta é informado ao algoritmo de aprendizado, deixando-o sozinho para encontrar relações na estrutura de entrada fornecida [149], ou seja, descobrir alguma estrutura dentro dos dados e explorar os dados são os dois

objetivos principais da aprendizagem não supervisionada [147]. Os algoritmos de aprendizagem não supervisionados incluem redes neurais artificiais, Expectativa-Maximização, clusterização k-médias, máquina de suporte vetorial, Clusterização Hierárquica.

Juntamente com as duas categorias acima mencionadas, há outro campo chamado **aprendizagem semi-supervisionada**, que contém conjuntos de dados com alguns pontos de dados rotulados, além de dados predominantemente não rotulados, isto é, usa menos quantidades de dados rotulados e grandes quantidades de dados não rotulados [150].

Aprendizagem por Reforço

Na aprendizagem por reforço, não é recebido pelo algoritmo nenhum conjunto de dados rotulado. Em vez disso, a informação é recolhida após interagir com o ambiente dinâmico através de diferentes ações. O algoritmo é recompensado após cada ação, daí o seu objetivo é maximizar esta recompensa média esperada onde a ação se tornaria ideal [149].

Método de tentativa e erro é usado neste tipo de aprendizagem em que as ações geram as melhores recompensas. Classificação, regressão e previsão são usadas. Agente, ambiente e ações são os três principais componentes usados neste tipo de aprendizagem onde o objetivo é que o agente selecione aquelas ações que exploram a recompensa previsível. Ao aplicar uma boa política, o agente é capaz de atingir a meta muito mais rápido [147]. Encontra-se na tabela 2.11 um resumo com as três principais abordagens de aprendizagem automática.

Tabela 2.11: Principais abordagens de aprendizagem automática.

Categorias	Descrição
Supervisionada	O modelo é treinado em um conjunto de dados e as previsões são feitas com novas entradas.
não Supervisionada	Um padrão é obtido dos dados após serem explorados.
Reforço	O modelo interage com o ambiente, toma decisões e aprende com suas ações.

2.6 Trabalhos Relacionados

Esta secção descreve um resumo dos trabalhos relacionados com a temática da investigação, com foco nos pontos principais de cada um deles. Estes trabalhos de investigação são como pontos de referência para entendimento do problema em questão à nível de complementação e comparação, apontam o que já foi executado de importante, definem conceitos e quais os resultados obtidos.

Uma investigação desenvolvida em, Abed et al. [15], apresentam um estudo sobre sistema de deteção de intrusões baseado em *host* por anomalias em tempo real que pode ser usado para detetar as atividades maliciosas contra aplicações em contentores Linux. A

proposta é um HIDS que combina a técnica *sliding window* com a técnica *Bag of system calls* (BoSC). O sistema de deteção de intrusões demonstrado usa pacotes de chamadas de sistema monitorizadas a partir do *kernel* do *host* para aprender o comportamento de uma aplicação em execução em um contentor Linux e determinar o comportamento anômalo do contentor. Os resultados obtidos mostram uma taxa de deteção de 100% e uma baixa taxa de falsos positivos de 0,58%.

Flora et al. [18] apresentam uma análise preliminar da viabilidade para a deteção de intrusões por anomalias, focando nas tecnologias Docker e LXC em ambiente *multi-tenant*. O artigo apresenta uma arquitetura de análise e coleta de chamadas de sistema. São implementados os algoritmos STIDE e BoSC e é estudado o processo de treinamento dos algoritmos para diferentes situações com tamanhos de janela variados de 3 a 6 chamadas de sistema em múltiplos cenários realistas. Os resultados obtidos mostram um estado de aprendizagem estável para STIDE com tamanhos de janela entre 3 e 4, e variando de 3 a 6 para BoSC.

Srinivasan et al. [19] implementaram um modelo de identificação de anomalias em tempo real que utiliza n-gramas de chamadas de sistema e a probabilidade da sua ocorrência em contentores Docker. O rastreamento é processado usando o Estimador de Máxima Verossimilhança (MLE) e Simple Good Turing (SGT) para fornecer uma estimativa melhor de valores de sequências de chamadas de sistema. Neste trabalho foi utilizado o conjunto de dados da Universidade do Novo México (UNM) para validar a abordagem com acurácia entre 87% e 97%.

Mattetti et al. [14] desenvolveram um *framework*, denominado LiCShield, que representa o primeiro passo na implementação dessa plataforma como "*Security as a Service*" para contentor Linux permitindo prevenir ataques e garantir a segurança dos mesmos. Esta estrutura é fundamentada no conceito de que aplicações e cargas de trabalho em nuvem são testadas em um ambiente de pré-produção. Neste artigo ainda é proposto um modelo de co-implantação do LiCShield com HIDS, que é estimado para fornecer a melhor proteção com sobrecarga de desempenho mínima.

Torkura et al. [151] elaboraram uma metodologia de análise de risco que consiste em técnicas com alcance em várias camadas que avaliam a arquiteturas de microsserviço, MSA, tendo como resultado métricas de risco de segurança. Estas métricas, ou seja, o risco de segurança é computado usando dois modelos de risco: OWASP *Risk Rating Methodology* (ORRM) e *Common Vulnerability Score System* (CVSS). As técnicas, *Moving Target Defenses* (MTD), transformam componentes especificados do sistema para criar incerteza para os atacantes, reduzindo assim a probabilidade de ataques bem-sucedidos, ou seja, capacidade de ataques. Na avaliação prática é comprovado que os microsserviços são adequados para a contratação de MTD, e demonstra a eficiência que mais de 70% da superfície de ataque são randomizadas, melhorando assim a segurança.

Stavridou et al. [152] encontraram uma base teórica sólida para técnicas de tolerância a intrusões. Tendo como destaque fatores como redundância e diversidade aplicados ao sis-

temas. A ideia chave, deste artigo é tendo encontrado intrusões, fazer com que o sistema aja para minimizar seu efeito. Como sugestão final é aconselhado o desenvolvimento de sistemas com arquiteturas de *software* que permitam operações contínuas (embora degradadas) após invasões e suporte à resposta medida às invasões encontradas.

Por fim, Huang et al. [153] disponibilizam uma análise detalhada sobre os mecanismos de segurança existentes no contentor Docker e as suas ameaças. Apresenta as técnicas correspondentes de deteção de ameaças para imagens e instâncias. A proposta de deteção considera dois aspectos de segurança: segurança de imagem do Docker e segurança da instância do contentor. No caso, da imagem Docker, detetaram as vulnerabilidades e exposições de aplicações pré-instalados pela base de dados CVE (*Common Vulnerabilities and Exposures*). Utilizaram a base de dados de assinatura malicioso para alcançar o objetivo de descobrir arquivos maliciosos. Um algoritmo de aprendizagem de máquina foi necessário para prevenir ataques desconhecidos. No caso, da instância executaram a instância do contentor, monitorando dois aspectos: uso de recursos de computador e solicitações de IP/DNS.

Na tabela 2.12 é possível estabelecer uma comparação entre os trabalhos relacionados com a temática da dissertação e esta dissertação.

Tabela 2.12: Comparação entre os trabalhos relacionados. (√: consta, —: não consta)

Trabalhos Relacionados				
Documento	**Tecnologia LXC**	**Tecnologia Docker**	**Implementação IDS**	**Métricas**
Este Trabalho	—	√	√	Precision, Recall e F-Measure
Abed et al.[15]	√	—	√	TPR e FPR
Flora et al.[18]	√	√	√	Precision, Recall, F-Measure e FPR
Huang et al.[153]	—	√	—	—
Mattetti et al.[14]	√	—	√	—
Srinivasan et al.[19]	—	√	√	Accuracy, Recall e FPR

2.7 Conclusão

Este capítulo torna-se em uma fonte de informação relevante para o aprofundamento do trabalho de investigação, pois, contém as principais definições que servem de base e orientação para a consolidação, entendimento e desenvolvimento deste trabalho.

A arquitetura de microsserviços é considerada um padrão de adoção, em relação a monolítica, devido as suas características como escalabilidade, disponibilidade e confiabilidade. Em relação a segurança foi exposto a complexidade, as principais ameaças e possíveis mitigação em relação as diversas camadas que compõem o sistema para este tipo de arquitetura: *cloud*, comunicação, *hardware*, virtualização, serviço/aplicação e orquestração.

Na secção seguinte, foi descrito um histórico da origem da tecnologia de contentores até os dias atuais. Explicou-se com detalhes os tipos de contentores. Na próxima subsecção têm-se os conceitos fundamentais para a implementação dos contentores como o isolamento e limitação de recursos. Logo após, é apresentado as principais tecnologias de contentores presentes no mercado de computação em nuvem e suas características.

Em continuação descreve-se a definição de deteção de intrusões para uma maior compreensão do problema, sua metodologias e seu impactos em cibersegurança. As metodologias de deteção de intrusões são divididas em deteção por assinaturas e por anomalias, seguidas de suas vantagens e desvantagens. Depois, são apresentado recentes abordagens dos principais algoritmos de deteção de anomalias para os sistemas de deteção de intrusões.

Encontra-se, também neste capítulo, a definição de um sistema de deteção de intrusões como um sistema capaz de detetar atividades maliciosas em *software* ou redes de computadores. São dividido em IDS ativo e IDS passivo. Esta estrutura do sistema de deteção de intrusões podem ser classificado em dois formatos principais: os HIDS que detetam ataques processando informações a nível de aplicação e os NIDS que detetam intrusões processando informações da rede.

Um método de avaliação composto com cargas de trabalho, métricas e metodologia de avaliação torna-se necessário para exploração do comportamento de um sistema de deteção de intrusões e apresentação de resultados concretos da investigação. De forma resumida, tem-se o registo do conceito de aprendizagem automática e suas categorias que permite um melhor aprofundamento do tema da investigação e de sua complexidade. Por fim, é escrito, de forma resumida, os principais artigos científicos relevantes para o trabalho e pertinentes a temática da investigação.

Capítulo 3

Implementação do Ambiente Experimental e Método de Classificação

3.1 Introdução

A implementação do ambiente experimental constitui uma etapa fundamental para a execução do trabalho de investigação e obtenção dos resultados. Este capítulo descreve as características e configurações dos recursos de *hardware* e *software* usados no trabalho de investigação conducente a esta dissertação. Este capítulo é composto por seis secções. A presente secção 3.1 é dedicada à introdução do capítulo 3. A secção 3.2 apresenta de forma detalhada o ambiente experimental. A secção 3.3 explana sobre a instalação dos contentores com a tecnologia Docker e configuração dos servidores MySQL. A secção 3.4 descreve as principais ações para simulação dos comportamentos benigno e malicioso e coleta de dados. A secção 3.5 refere-se a implementação para o processo de tratamento de dados. A secção 3.6 regista as atividades para a configuração dos classificadores de aprendizagem automática e finalmente a secção 3.7 a apresentaa a conclusão do capítulo.

3.2 Ambiente Experimental

Esta secção encontra-se dividida em duas subsecções, uma dedicada à caracterização do ambiente experimental, que fornece uma perspetiva geral de todos os componentes deste ambiente, e a outra, dedicada à especificação da infraestrutura, na qual são apresentados detalhes sobre as características de *hardware* e *software* usados no âmbiente experimental.

3.2.1 Caracterização do Ambiente Experimental

Tendo como referências os trabalhos anteriores [15, 18], o ambiente experimental é constituido por uma máquina virtual com o sistema operativo Ubuntu na versão 18.04.5 LTS (Bionic Beaver). O ambiente de virtualização foi implementado através do contentor Docker na versão 20.10.03, pois, em tempo de execução, uma instância de microsserviço pode ser configurada para ser executada como um processo num contentor [32] e devido ao mesmo ser considerado líder entre todas as tecnologias de contentores para a nuvem [154]. A figura 3.1 ilustra o contexto geral do ambiente experimental, o prompt de comando do Linux como a interface do utilizador e a implementação do TPC-C *Benchmark* para execução e manipulação das bases de dados.

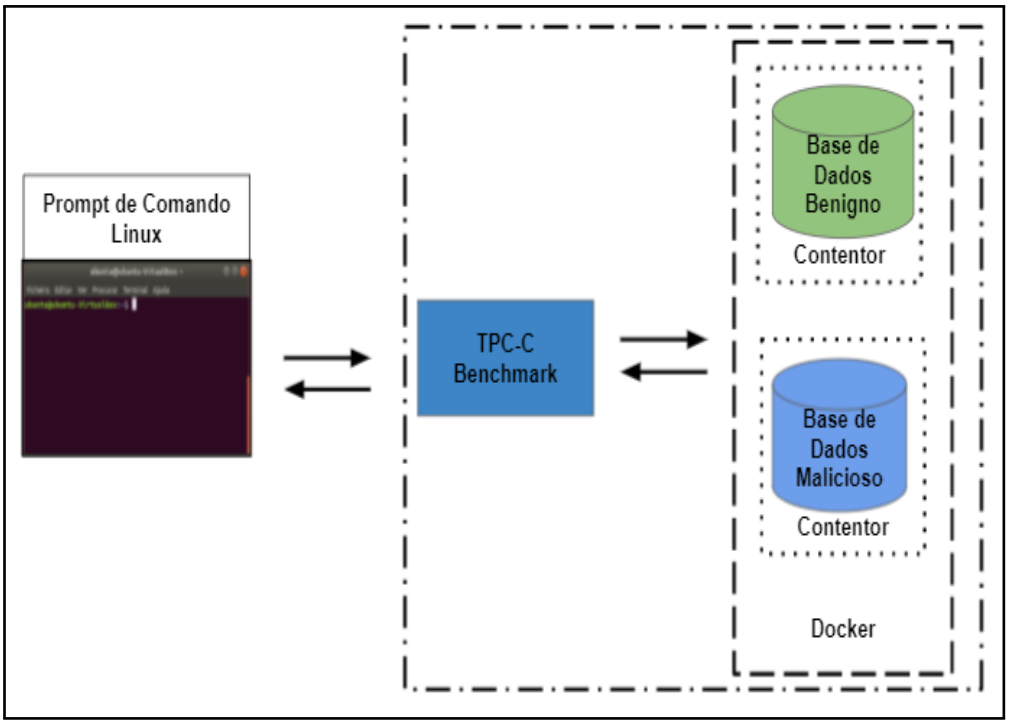


Figura 3.1: Contexto geral do ambiente experimental.

A figura 3.2 ilustra em destaque o contentor benigno e malicioso e o contexto geral de todos os componentes do sistema e suas definições para obtenção dos dados sobre os comportamentos e utilização dos mesmos para avaliação do desempenho dos classificadores.

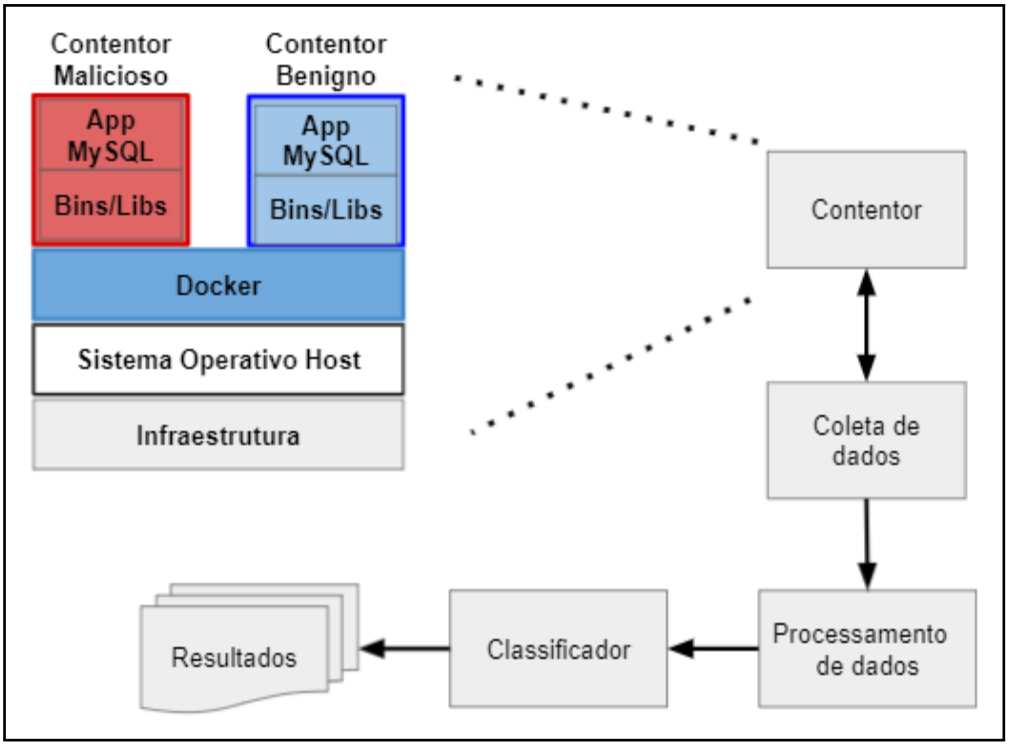


Figura 3.2: Perspetiva geral do ambiente experimental.

Esta VM foi criada em um host com Windows 10 Enterprise usando VirtualBox. De acordo com a figura 3.2, a arquitetura para o sistema de deteção de intrusões, proposta nesta investigação, é formada pelos seguintes componentes:

- **Contentor**: Os recipientes, com as características apresentadas na figura 3.2, têm como função a obtenção de um ambiente *multi-tenant* com os serviços em execução e geração de cargas de trabalho benignas e maliciosas;

- **Coleta de Dados**: A monitorização de *system calls* é uma técnica amplamente utilizada para detetar um comportamento suspeito de uma aplicação [155], pelo que deverão ser coletadas as interações entre processos e o *kernel* do Linux, que incluem as chamadas de sistema (*system calls*);

- **Processamento de Dados**: É uma implementação que envolve a limpeza, seleção e adequação dos dados;

- **Classificador**: O classificador tem como objetivo a análise dos dados a fim de produzir um perfil do contentor ou os resultados de uma análise de rastreamento.

3.2.2 Especificação da Infraestrutura

A especificação da infraestrutura do ambiente experimental concebe um valor fundamental para se ter uma melhor compreensão do seu funcionamento e capacidade de adquirir resultados representativos para a investigação. Neste caso, a tabela 3.1 refere-se as características do *hardware*. E a tabela 3.2 são descritas as tecnologias, em termo de *software*, utilizadas para a implementação do ambiente experimental.

Tabela 3.1: Características *hardware*.

Hardware	
Definição	**Características**
Dispositivo	DESKTOP-MP70HAF
Procesador	Intel Core i7-7700 3.6Ghz
Memória RAM	16GB
Disco Rígido	930GB
Sistema Operativo	Windows 10 Enterprise

Tabela 3.2: Características *software*.

Softwares	
Definição	**Características**
Software de Virtualização	VirtualBox
Máquina Virtual	2 cores, 4GB of RAM
Sistema Operativo	ubuntu-18.04.5 LTS (Bionic Beaver)
Contentor	Docker versão 20.10.03
Imagem Contentor Benigno	MySQL versão 8.0.23
Imagem Contentor Malicioso	MySQl versão 5.6.27
Ferramentas de monitorização (system calls)	sysdig
Implementação TPC-C Benchmark	`https://github.com/Percona-Lab/tpcc-mysql`

3.3 Contentores Docker

3.3.1 Instalação do *Docker Engine* no Ubuntu

Os pré-requisitos para a instalação do *Docker Engine* segue os procedimentos que constam na documentação oficial[1] do *software*. Pois, nesta etapa e nas demais houve a preocupação de consultar e efetuar os procedimentos de instalação de acordo com a documentação. A figura 3.3 exemplifica a imagem de teste e execução do contentor para verificar se tudo está funcionando como deveria. E a figura 3.4 a versão Docker instalada.

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:95ddb6c31407e84e91a986b004aee40975cb0bda14b5949f6faac5d2deadb4b9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figura 3.3: Imagem de teste Docker *hello-world*.

[1] `https://docs.docker.com/engine/install/ubuntu/`

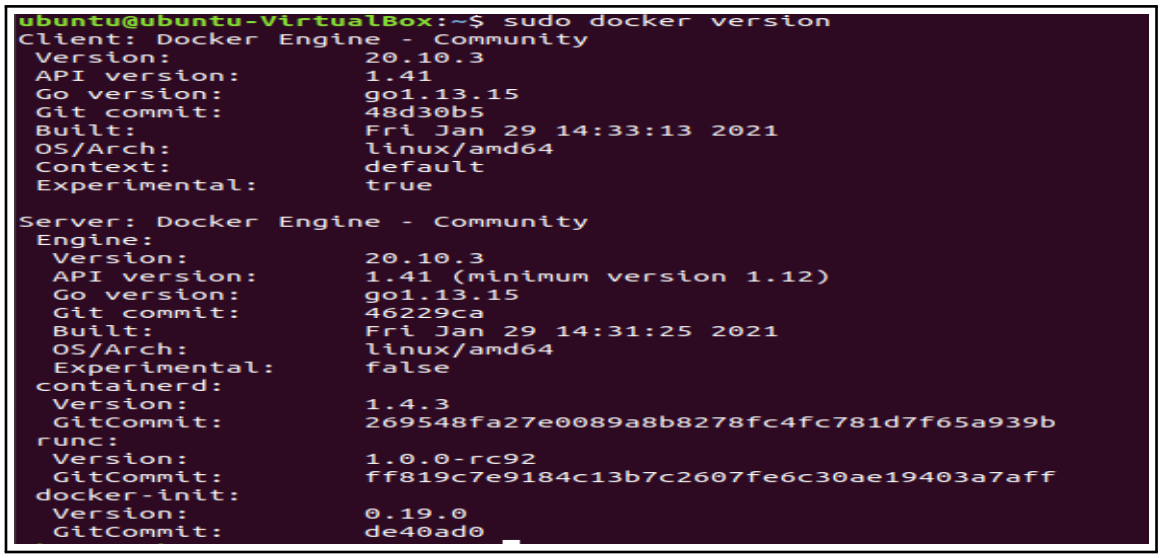


Figura 3.4: Versão Docker instalada.

3.3.2 Imagem do MySQL

Docker Hub[2] é o repositório oficial de imagens de contentor *Docker* com uma variedade para localizar, partilhar e fazer o *download* de imagens. O comando *docker pull*[3] tem a função de fazer *download* da imagem. A figura 3.5 ilustra download da imagem MySQL[4] versão 8.0.23 e a figura 3.6 a imagem MySQL versão 5.6.27. Por fim, na figura 3.7, lista todas as imagens instaladas.

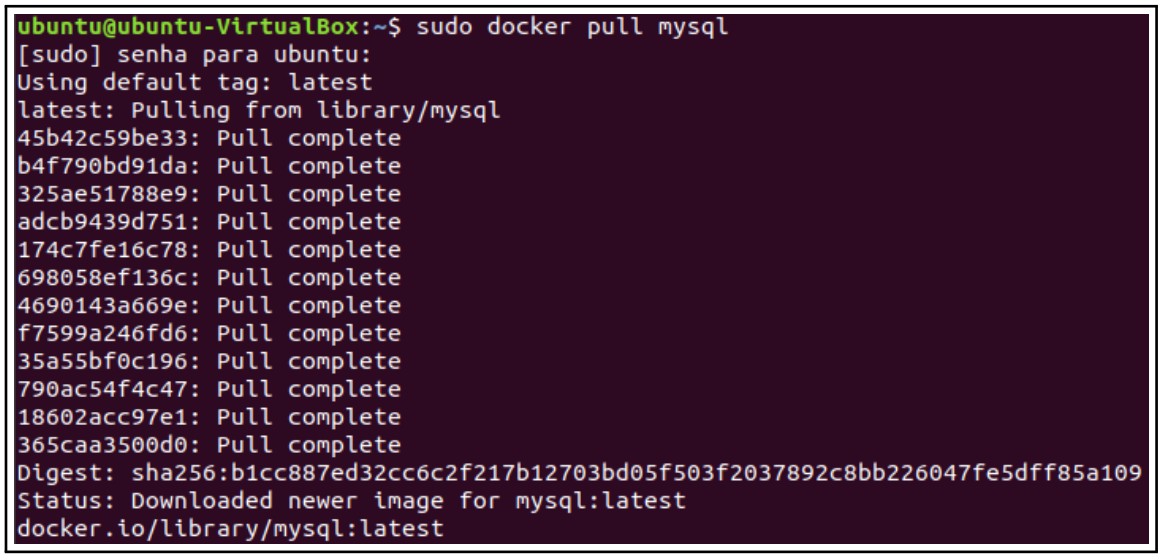


Figura 3.5: Imagem MySQL versão: 8.0.23.

[2] https://hub.docker.com/
[3] https://docs.docker.com/engine/reference/commandline/pull/
[4] https://hub.docker.com/_/mysql

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker pull mysql:5.6.27
5.6.27: Pulling from library/mysql
45b42c59be33: Already exists
b4f790bd91da: Already exists
325ae51788e9: Already exists
adcb9439d751: Already exists
174c7fe16c78: Already exists
698058ef136c: Already exists
4690143a669e: Already exists
66676c1ab9b3: Pull complete
25ebf78a38b6: Pull complete
349a839d5e27: Pull complete
40b03e3e5980: Pull complete
Digest: sha256:853105ad984a9fe87dd109be6756e1fbdba8b003b303d88ac0dda6b455f36556
Status: Downloaded newer image for mysql:5.6.27
docker.io/library/mysql:5.6.27
```

Figura 3.6: Imagem MySQL versão: 5.6.27.

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
mysql         5.6.27    5f47254ca581   6 days ago       449MB
mysql         latest    2933adc350f3   6 days ago       546MB
hello-world   latest    bf756fb1ae65   13 months ago    13.3kB
```

Figura 3.7: Listas das imagens instaladas.

De acordo com a documentação *Docker*, os volumes[5] são os mecanismos para persistir dados gerados e usados pelos contentores. Neste caso, nas figuras 3.8 e 3.9 abaixo, observa-se a execução do comando *docker run* com a *flag* -v para criar um contentor e partilhar os dados de uma pasta do contentor com o host e a *flag* - -*name* para especificar o contentor. Para facilitar a identificação dos contentores em relação ao comportamento, os mesmos foram simplesmente denominados de benigno e malicioso.

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker run -e MYSQL_ROOT_PASSWORD=12345  -v /home/ubuntu/Desktop/benigno:/var/lib/mysql  --name benigno -d mysql
081dc724d89aad7c3decf300332a9fc4203de097e66bfa11fd686973dda064b5
```

Figura 3.8: Execução do contentor benigno.

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker run -e MYSQL_ROOT_PASSWORD=123456 -v /home/ubuntu/Desktop/malicioso:/var/lib/mysql --name malicioso -d mys
ql:5.6.27
bc11fcd1cec797e2963666401e6ff20d2463bce73f7213ef1e939aafaba22b2e
```

Figura 3.9: Execução do contentor malicioso.

Na figura 3.10 pode-se observar os contentores em execução e a figura 3.11 seus respectivos ficheiros dos volumes gerados no sistema operativo *host*.

[5] https://docs.docker.com/storage/volumes/

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                  NAMES
bc11fcd1cec7   mysql5.6.27 "docker-entrypoint.s…"   14 seconds ago   Up 11 seconds   3306/tcp, 33060/tcp    malicioso
081dc724d89a   mysql       "docker-entrypoint.s…"   3 minutes ago    Up 3 minutes    3306/tcp, 33060/tcp    benigno
```

Figura 3.10: Contentores em execução.

Figura 3.11: Ficheiros dos contentores.

Em relação ao endereço IP, a documentação oficial Docker [36] esclarece que, por padrão, o contentor recebe um endereço IP para cada rede Docker à qual se conecta. O comando *docker inspect* fornece informações detalhadas sobre os objetos *Docker* inclusive o endereço IP. Neste caso, constam nas figuras 3.12 e 3.13, o *IPAddress* e o *Pid* dos contentores em execução no ambiente experimental para acesso, interação e acompanhamento dos mesmos.

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker inspect benigno | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.2",
                    "IPAddress": "172.17.0.2",
ubuntu@ubuntu-VirtualBox:~$ sudo docker inspect benigno | grep Pid
            "Pid": 16911,
            "PidMode": "",
            "PidsLimit": null,
```

Figura 3.12: *IPAddress* e *Pid* do contentor benigno.

```
ubuntu@ubuntu-VirtualBox:~$ sudo docker inspect malicioso | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.3",
                    "IPAddress": "172.17.0.3",
ubuntu@ubuntu-VirtualBox:~$ sudo docker inspect malicioso | grep Pid
            "Pid": 17214,
            "PidMode": "",
            "PidsLimit": null,
```

Figura 3.13: *IPAddress* e *Pid* do contentor malicioso.

3.3.3 Descrição do Conjunto de Dados

Para popular as bases de dados dos contentores benigno e malicioso utilizou-se o TPC-C *Benchmark* [156] que é um *benchmark* de processamento de transações *on-line*. O TPC-C *Benchmark* é uma carga de trabalho, composta por um conjunto de cinco transações de leitura e escrita, que simula as atividades encontradas em um ambiente de processamento de transações em tempo real (OLTP). O mesmo tem como objetivo dispor dados relevantes de desempenho que auxiliem os utilizadores a comparar dois ou mais sistemas quanto a sua capacidade de processar transações. A base de dados é composta por nove tipos de tabelas com uma ampla variedade de tamanhos de registos e populações. A figura 3.14 mostra as tabelas e os seus respectivos relacionamentos:

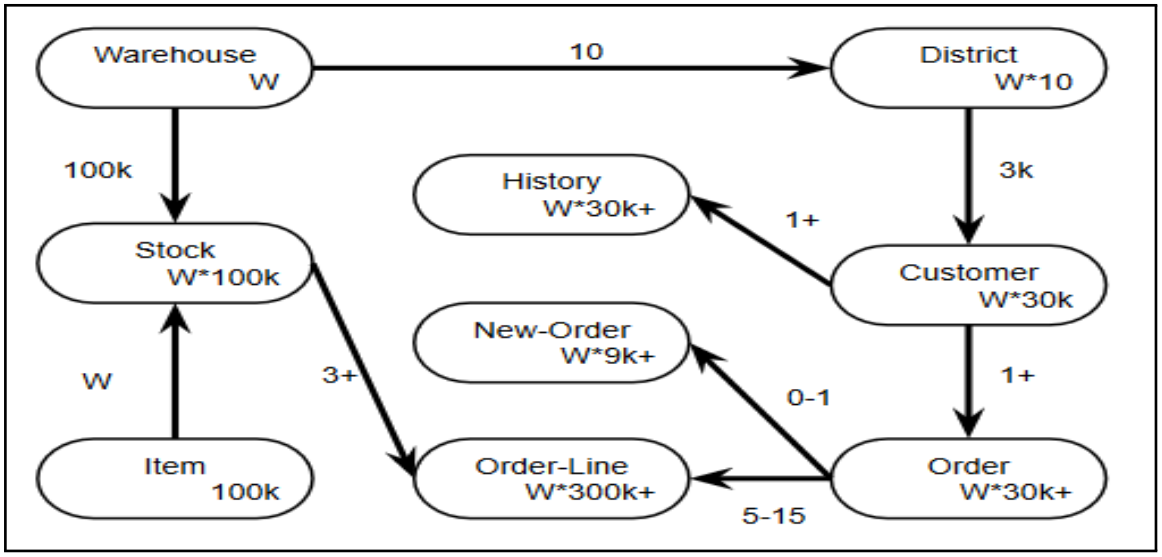


Figura 3.14: Tipos de tabelas (figura redesenhada a partir de [157]).

As características do TPC-C *Benchmark* são definidas como um sistema de processamento de transações *on-line* usado para registar as atividades de uma empresa distribuidora. Esta empresa encontra-se distribuída por um número de armazéns (warehouses), distri-

tos (districts) e estoque para estes armazéns, bem como itens e pedidos para esses item. O número de armazéns é o parâmetro configurável chave para determinar a escala de execução do *benchmark*. Por fim, a base de dados escolhida para o ambiente experimental desta investigação foi selecionada com 100 armazéns (warehouses). Na figura 3.15 observa-se o tamanho final da base de dados e sua tabelas disponibilizado no MySQL *workbench-community_8.0*[6].

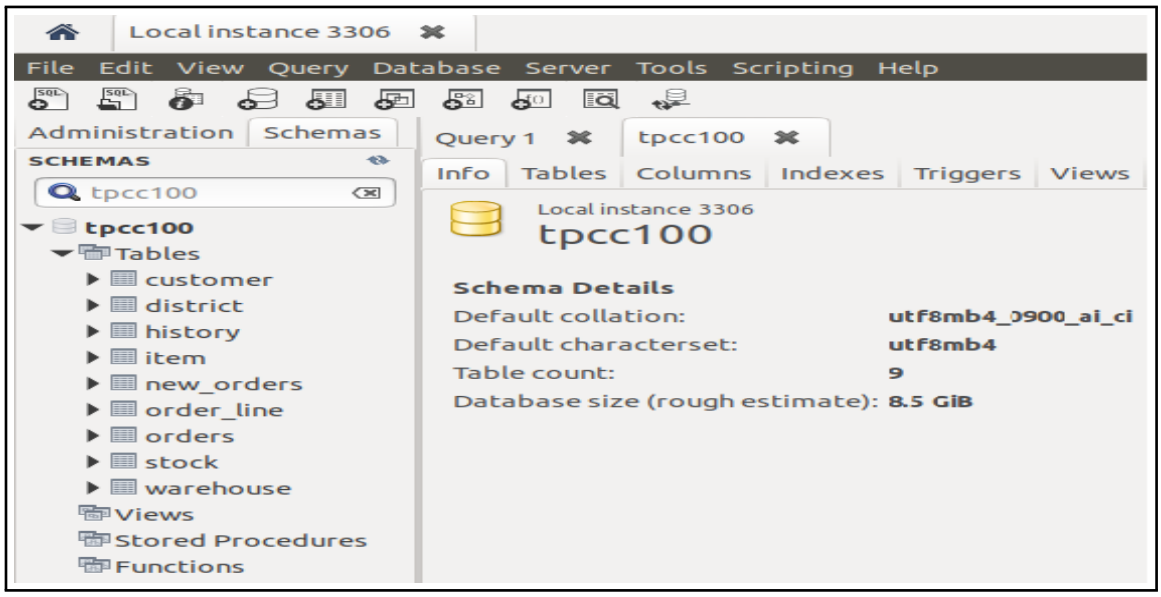


Figura 3.15: Tamanho da base de dados e tabelas.

3.3.4 Carregamento dos Dados nos Contentores

Uma implementação [158] foi instalada e configurada para carregamento dos dados do TPC-C *Benchmark* nos contentores. A implementação permite o carregamento das bases de dados e a execução das mesmas para fins de comparação de desempenho (Benchmarking). O comando para execução desta atividade possue cinco variáveis que representam informações sobre a base de dados (imagem Docker) e o contentor na qual a mesma foi instalada. As informações IP do contentor, base de dados, utilizador, senha e quantidades de armazéns são repassadas em dois momentos distintos através do *prompt* de comando do sistema operativo Linux. A figura 3.16 mostra o carregamento da base de dados no contentor benigno. A figura 3.17 no contentor malicioso.

[6] `https://dev.mysql.com/downloads/workbench/`

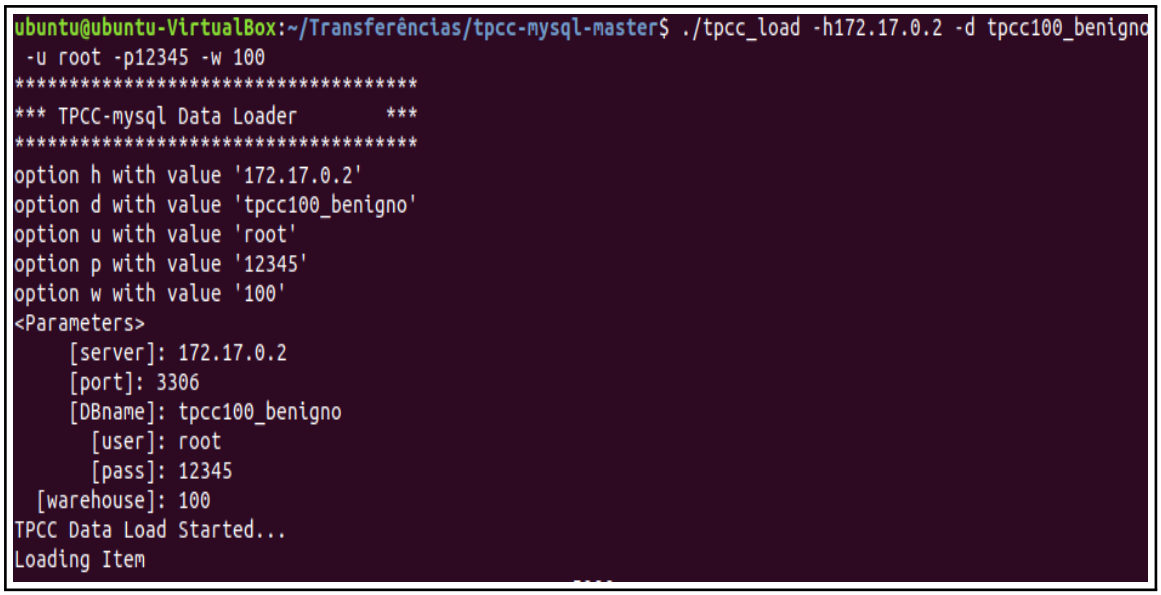


Figura 3.16: Carregamento do contentor benigno.

Figura 3.17: Carregamento do contentor malicioso.

Para a execução de rotinas e comandos dentro do contentor em execução usa o comando *docker exec* seguido do nome do contentor. Com este comando pode-se verificar a base de dados e suas tabelas em cada contentor. Nas figuras 3.18, o contentor benigno e na figura 3.19 o contentor malicioso.

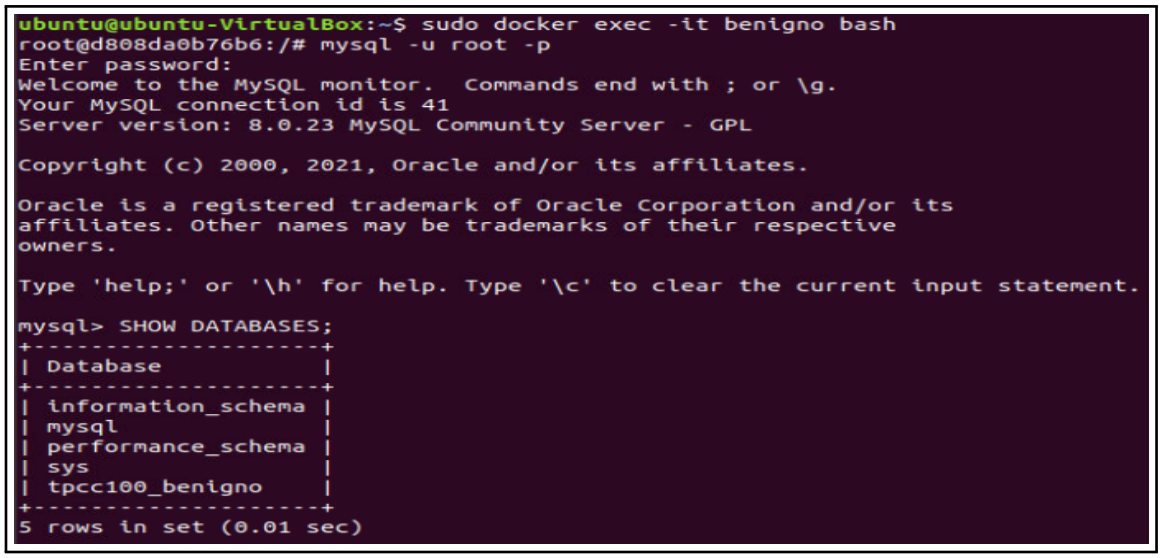


Figura 3.18: Base de dados do contentor benigno.

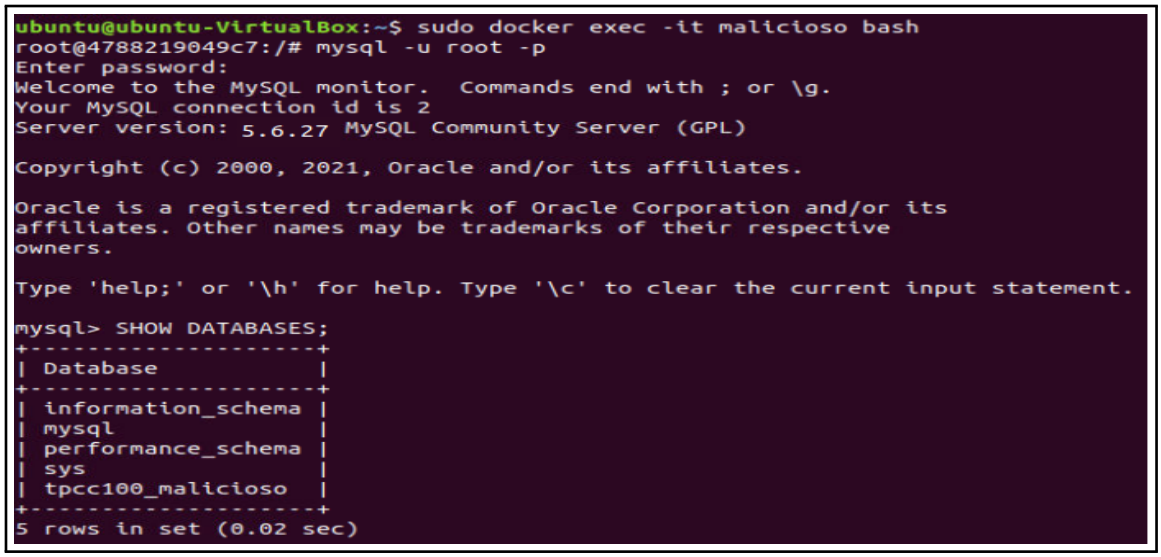


Figura 3.19: Base de dados do contentor malicioso.

A ferramenta de monitorização para acompanhar e analisar as estruturas das base de dados foi MySQL *Workbench-community_8.0*. O *dashboard* do MySQL *Workbench-community_8.0* é uma ferramenta visual que facilita a administração e desenvolvimento de uma base de dados. Esta ferramenta, no ambiente experimental, permite verificar os indicadores da base de dados, após o carregamento da base de dados TPC-C *Benchmark*, e onde também é possível constatar o tamanho das mesmas e suas tabelas nos contentores benigno e malicioso representados nas figuras 3.20 e 3.21.

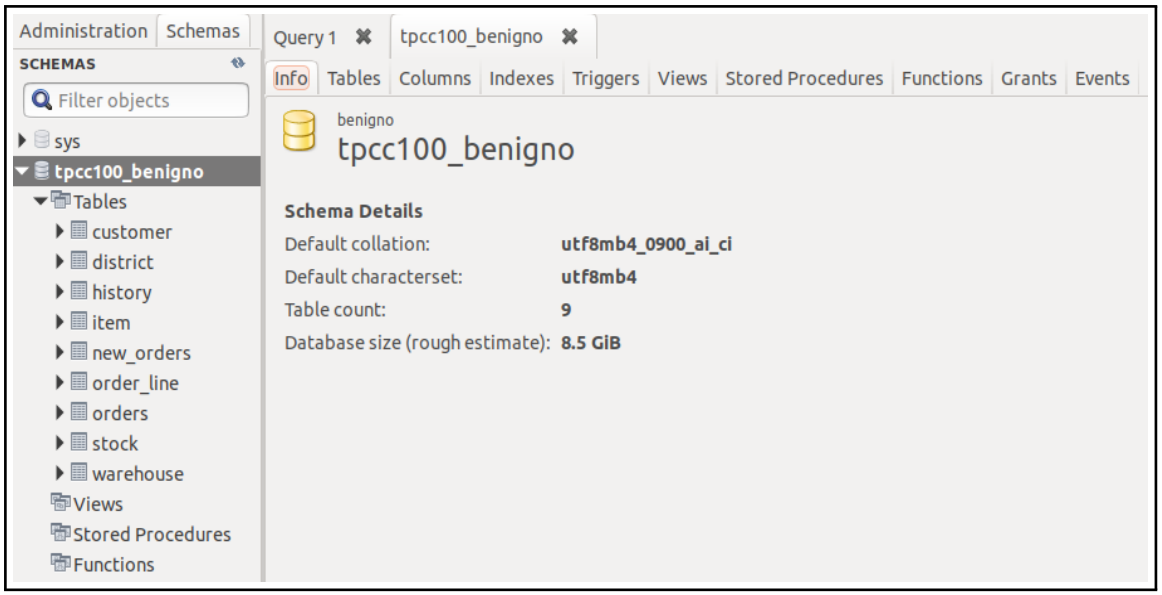


Figura 3.20: Tamanho da base de dados do contentor benigno.

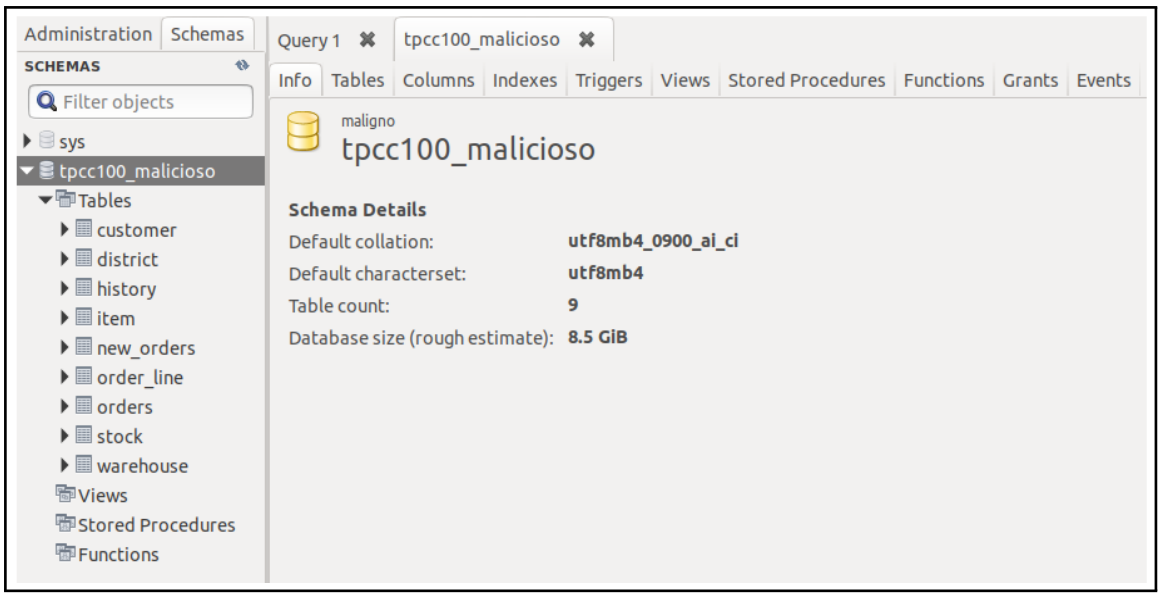


Figura 3.21: Tamanho da base de dados do contentor malicioso.

3.4 Coleta de Dados

Os dados definidos para a caracterização dos comportamentos benigno e malicioso dos contentores são as chamadas de sistema (*systemcalls*). Uma chamada de sistema é o recurso na qual uma aplicação ou processo solicita uma serviço ao *kernel* do sistema operativo. Na seleção em relação ao comportamento dos contentores e a coleta das respectivas chamadas de sistema utilizou uma ferramenta para solução de problemas, análise e exploração do sistema denominada *sysdig*. Para maiores esclarecimento sobre esta ferramenta ver subsecção 2.3.4.

3.4.1 Coleta do Comportamento do Contentor Benigno

A coleta das chamadas de sistema que representam o comportamento benigno realizou-se perante a seguinte ordem: primeiro, o contentor necessitou ser submetido a implementação das operações do TPC-C *Benchmark* que executa várias transações *on-line* e garante uma variação nas rotinas e interações normais para a base de dados `tpcc100_benigno`. Segundo, a ferramenta de monitorização *sysdig* foi o recurso utilizado para a escrita e leitura das chamadas de sistema e geração do ficheiro relacionado a este comportamento.

Como exemplo, na figura 3.22, o comando executa um série de transações de inserções e atualizações na base de dados do contentor benigno com 100 armazéns, 32 conexões e taxa de transferência a cada 10 s em um período total de 7 540 s.

```
ubuntu@ubuntu-VirtualBox:~/Transferências/tpcc-mysql-master$ ./tpcc_start -h172.17.0.2 -P 3306 -d tpcc10
0_benigno -u root -p12345 -w 100 -c 32 -r 10 -l 7540
****************************************
*** ###easy### TPC-C Load Generator ***
****************************************
option h with value '172.17.0.2'
option P with value '3306'
option d with value 'tpcc100_benigno'
option u with value 'root'
option p with value '12345'
option w with value '100'
option c with value '32'
option r with value '10'
option l with value '7540'
<Parameters>
     [server]: 172.17.0.2
     [port]: 3306
     [DBname]: tpcc100_benigno
       [user]: root
       [pass]: 12345
  [warehouse]: 100
 [connection]: 32
     [rampup]: 10 (sec.)
    [measure]: 7540 (sec.)

RAMP-UP TIME.(10 sec.)

MEASURING START.
```

Figura 3.22: Execução do *benchmark* no contentor benigno.

Paralelamente ao comando anterior, a ferramenta *sysdig* composta pela *flag* -w, para escrita, nome do ficheiro, `benigno_traces.scap` e filtro, `container.name=benigno`, coleta as chamadas de sistema. Figura 3.23

```
ubuntu@ubuntu-VirtualBox:~$ sudo sysdig -w benigno_traces.scap  container.name=benigno
```

Figura 3.23: Comando *sysdig* para escrita.

Após a finalização da implementação do TPC-C *Benchmark* e escrita das chamadas de sistema segue-se a leitura das mesmas com as seguintes configurações na ferramenta *sysdig*, *flag* -r, para leitura, nome do ficheiro `benigno_traces.scap` e o redirecionamento para um ficheiro CSV, `benigno_traces.csv`.

```
ubuntu@ubuntu-VirtualBox:~$ sudo sysdig -r benigno_traces.scap > benigno_traces.csv
```

Figura 3.24: Comando *sysdig* para leitura.

Durante o período de simulação do comportamento benigno armazenou e coletou um total de 23 774 746 chamadas de sistema no tempo de execução de 2 h 05 min 40 s, exposto na tabela 3.3.

Tabela 3.3: Total de chamadas de sistema comportamento benigno.

Tempo de execução	**Plataforma**	**Total de chamadas de sistema**	**Tamanho (GB)**
2 h 05 min 40 s	Docker	23 774 746	3,19

3.4.2 Coleta do Comportamento do Contentor Malicioso

A simulação do comportamento malicioso do contentor foi constituída pela exploração das vulnerabilidades apresentadas na imagem MySQL versão:5.6.27 seguida da implementação TPC-C *Benchmark*. Uma consulta as bases de dados da CVE[7] (Common Vulnerabilities and Exposures) e Exploit Database (EDB)[8] se fez necessária para determinar as vulnerabilidades, isto é *exploits*, que são definidos como um pedaço de software, um pedaço de dados ou uma sequência de comandos que tiram proveito de uma falha de segurança ou vulnerabilidade em uma aplicação ou sistema, aplicadas ao ambiente de teste. A lista de vulnerabilidades está descrita na tabela 3.4 composta do ID CVE, tipo de acesso, tipo de vulnerabilidade e *score*.

Tabela 3.4: Lista de vulnerabilidades.

Lista de Vulnerabilidades			
ID CVE	**Vulnerabilidade**	**Tipo de acesso**	***Score***
2012-2122 [159]	*Authentication Bypass*	Remoto	5,1
2013-1861 [160]	*DoS Overflow*	Remoto	5,0
2016-5616 [161]	*System User Privilege Escalation / Race Condition*	Local	4,4
2017-3599 [162]	*Integer Overflow*	Remoto	7,8

De forma similar ao comportamento benigno a coleta do comportamento malicioso iniciou-se com a execução da implementação de operações da bases de dados do TPC-C *Benchmark* seguida de umas das vulnerabilidades. Na tabela 3.5 são apresentados o total de chamadas de sistemas coletadas do comportamento malicioso, tempo de execução e tamanho em (GB). O tempo de execução de cada uma das vulnerabilidades foi distribuído conforme exposto na tabela 3.6 .

[7] `https://cve.mitre.org/`
[8] `https://www.exploit-db.com/`

Tabela 3.5: Total de chamadas de sistema comportamento malicioso.

Tempo de execução	Plataforma	Total de chamadas de sistema	Tamanho (GB)
2 h 05 min 40 s	Docker	18 923 927	2,909

Tabela 3.6: Total de chamadas de sistema coletadas por vulnerabilidade.

Vulnerabilidade	Tempo de execução	Plataforma	Total de chamadas de sistema	Tamanho (MB)
2012-2122	40 min 10 s	Docker	5 243 875	792
2013-1861	20 min 10 s	Docker	4 194 300	648
2016-5616	30 min 10 s	Docker	5 242 875	812
2017-3599	35 min 10 s	Docker	4 242 875	657

Da mesma forma para o comportamento benigno na figura 3.25 tem-se, como exemplo, o comando que executa um série de transações de inserções e atualizações na base de dados do contentor malicioso `tpcc100_malicioso` com 100 armazéns, 32 conexões e transações a cada 10 s em um período de 1200 s. Este comando foi carregado para cada uma das vulnerabilidades da imagem MySQL versão:5.6.27 alterando o tempo de execução.

```
ubuntu@ubuntu-VirtualBox:~/Transferências/tpcc-mysql-master$ ./tpcc_start -h172.17.0.3  -P 3306 -d tpcc1
00_malicioso -u root -p123456 -w 100 -c 32 -r 10 -l 1200
****************************************
*** ###easy### TPC-C Load Generator ***
****************************************
option h with value '172.17.0.3'
option P with value '3306'
option d with value 'tpcc100_malicioso'
option u with value 'root'
option p with value '123456'
option w with value '100'
option c with value '32'
option r with value '10'
option l with value '1200'
<Parameters>
     [server]: 172.17.0.3
     [port]: 3306
     [DBname]: tpcc100_malicioso
       [user]: root
       [pass]: 123456
  [warehouse]: 100
 [connection]: 32
     [rampup]: 10 (sec.)
    [measure]: 1200 (sec.)

RAMP-UP TIME.(10 sec.)

MEASURING START.
```

Figura 3.25: Execução do *benchmark* no contentor malicioso.

Paralelamente ao comando anterior, a ferramenta *sysdig* composta pela *flag* -w, para escrita, nome do ficheiro binário, `malicioso_traces_1.scap` e filtro, `container.name = malicioso`, coleta as chamadas de sistema para a primeira vulnerabilidade. Este procedimento foi extendido para as demais vulnerabilidades apenas alterando o nome do ficheiro. Figura 3.26.

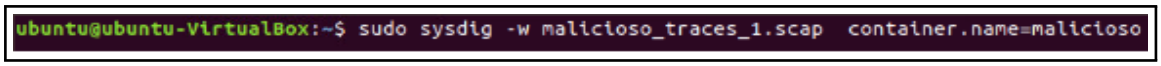


Figura 3.26: Comando *sysdig* para escrita.

Após o procedimento de escrita presente na figura 3.27 o comando da *sysdig* com a *flag* -r, para leitura, nome do ficheiro `malicioso_traces_1.scap` e o redirecionamento para um ficheiro CSV, `malicioso_traces_1.csv` empregou-se após cada finalização da implementação do TPC-C *Benchmark* seguida de sua vulnerabilidade para o processo de coleta dos dados.

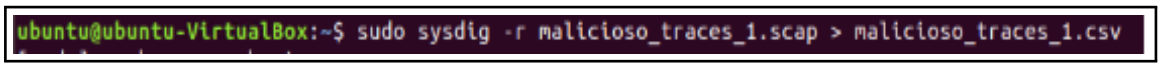


Figura 3.27: Comando sysdig para leitura.

Dessa forma, o processo de coleta de dados para criação de um conjunto de dados composto dos comportamentos benigno e malicioso dos contentores, representados pelas suas respectivas chamadas de sistema, operando em um ambiente com cargas de trabalho mista e realista teve como resultado uma base de dados para treino e teste dos algoritmos de aprendizagem automática neste ambiente . Ilustrado na tabela 3.7 o total de chamadas de sistema coletadas dos comportamentos benigno e malicioso.

Tabela 3.7: Total de chamadas de sistemas.

Total de chamadas de sistema	**Tamanho (GB)**
47 904 546	6,904

3.5 Processamento de dados

O processamento de dados busca aprimorar a qualidade dos dados coletados dos comportamentos benigno e malicioso dos contentores. Nesta etapa, baseada no modelo de Extração do Conhecimento em Bases de Dados (*knowledge-discovery in databases*) [163] ocorre o pré-processamento dos dados que engloba as atividades de limpeza dos dados, seleção de atributos como também correções de informações ausentes ou inconsistentes de forma a não comprometer a qualidade dos modelos.

A ferramenta *sysdig* fornece um recurso para criação de ficheiro de extensão SCAP, ficheiro binário, contendo as chamadas de sistema. Este recurso permite o armazenamento das chamadas de sistema para posterior análise. No caso especifico desta investigação este recurso foi usado durante o procedimento de coleta e leitura com um redirecionamento para um ficheiro de extensão CSV (explicado na secção 3.4). Na figura 3.28, um exemplo de um ficheiro CSV com as chamadas de sistema em estado bruto.

```
104327 19:05:08.751479622 1 mysqld (9094) > pwrite fd=11(<f>/var/lib/mysql/#ib_16384_1.dblwr) size=16384 pos=5292032
104328 19:05:08.751484227 1 mysqld (9094) > switch next=20498 pgft_maj=0 pgft_min=32 vm_size=1824820 vm_rss=583492 vm_swap=6788
104329 19:05:08.753491252 0 mysqld (4494) < fsync
104330 19:05:08.753498704 0 mysqld (4494) > futex addr=FFFF9534D4A2AD34 op=385(FUTEX_CLOCK_REALTIME|FUTEX_PRIVATE_FLAG|FUTEX_WAKE) val=-117420838441473
104331 19:05:08.753504481 0 mysqld (4494) < futex res=1
104332 19:05:08.753506696 0 mysqld (4494) > fsync
104333 19:05:08.753605601 0 mysqld (4494) > switch next=4489(mysqld) pgft_maj=0 pgft_min=1 vm_size=1824820 vm_rss=583492 vm_swap=6788
104334 19:05:08.753607554 0 mysqld (4489) < fsync
104335 19:05:08.753623491 0 mysqld (4489) > switch next=20383(sysdig) pgft_maj=0 pgft_min=2 vm_size=1824820 vm_rss=583492 vm_swap=6788
104336 19:05:08.753627117 1 mysqld (4493) < futex res=0
104337 19:05:08.753629956 1 mysqld (4493) > futex addr=FFFFFFFF9C0BA4E0 op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE) val=-1676958719
104338 19:05:08.753630767 1 mysqld (4493) < futex res=0
104339 19:05:08.753632802 1 mysqld (4493) > futex addr=FFFFFFFF9C0BA4B0 op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE) val=-1676958465
104340 19:05:08.753636692 1 mysqld (4493) < futex res=1
104341 19:05:08.753753627 1 mysqld (4493) > futex addr=FFFFAF9900000030 op=128(FUTEX_PRIVATE_FLAG) val=-88403311853568
104342 19:05:08.753761598 1 mysqld (4493) > switch next=20527(mysqld) pgft_maj=0 pgft_min=1 vm_size=1824820 vm_rss=583492 vm_swap=6788
104343 19:05:08.753763470 1 mysqld (20527) < futex res=0
104344 19:05:08.753765145 1 mysqld (20527) > futex addr=FFFFFFFF9C0BA460 op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE) val=-1676958719
104345 19:05:08.753765808 1 mysqld (20527) < futex res=0
104346 19:05:08.753814123 1 mysqld (20527) > write fd=34 size=7859
104347 19:05:08.753860110 1 mysqld (20527) > write fd=34 size=8192
```

Figura 3.28: Ficheiro CSV com as chamadas de sistema.

Em continuação realizou-se o carregamento dos ficheiros CSV dos comportamentos benigno e malicioso dos contentores para a separação dos dados obtidos em concordâncias com a sua respectiva coluna. De acordo com a documentação da ferramenta *sysdig*[9] por padrão, a mesma imprime as informações de cada evento capturado em uma única linha com o seguinte formato:

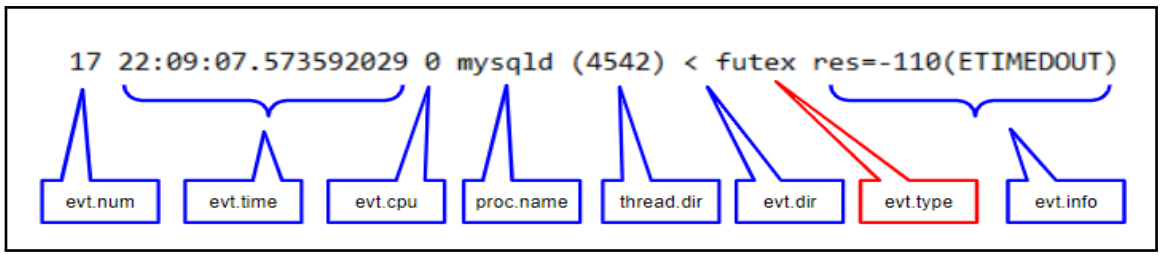


Figura 3.29: Formato da informação *sysdig*.

onde:

- *evt.num* é o número do evento incremental;
- *evt.time* é a data/hora do evento;
- *evt.cpu* é o número da CPU onde o evento foi capturado;
- *proc.name* é o nome do processo que gerou o evento;
- *thread.tid* é o ID que corresponde ao PID para processos de thread único;
- *evt.dir* é a direção do evento, “>” para inserir eventos e “<” para eventos de saída;

[9] `https://man7.org/linux/man-pages/man8/sysdig.8.html`

- *evt.type* é o nome do evento, isto é, as chamadas de sistema em destaque na figura 3.29;
- *evt.info* é a lista de argumentos do evento.

Logo, a organização do conjunto de dados levou em conta esta mesma estrutura para definição das colunas dos ficheiros CSV como *Features* e para construção da base de dados composta pelos comportamentos benigno e malicioso.

	evt_num	evt_time	evt_cpu	proc_name	thread_tid	evt_dir	evt_type	evt_args	label	comportamento
0	3	22:09:07.561630141	0	mysqld	(4429)	entrada	futex	addr=80op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE)va...	0	benigno
1	4	22:09:07.561630932	0	mysqld	(4429)	saida	futex	res=0	0	benigno
2	5	22:09:07.561634434	0	mysqld	(4429)	entrada	futex	addr=D4op=128(FUTEX_PRIVATE_FLAG)val=0	0	benigno
3	6	22:09:07.561645660	0	mysqld	(4429)	entrada	switch	next=0(<NA>)pgft_maj=0pgft_min=1vm_size=207161...	0	benigno
4	7	22:09:07.562099701	0	mysqld	(4428)	saida	futex	res=-110(ETIMEDOUT)	0	benigno
5	8	22:09:07.562102337	0	mysqld	(4428)	entrada	futex	addr=0op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE)val...	0	benigno
6	9	22:09:07.562102819	0	mysqld	(4428)	saida	futex	res=0	0	benigno
7	10	22:09:07.562104678	0	mysqld	(4428)	entrada	futex	addr=54op=128(FUTEX_PRIVATE_FLAG)val=0	0	benigno
8	11	22:09:07.562107484	0	mysqld	(4428)	entrada	switch	next=0(<NA>)pgft_maj=0pgft_min=1vm_size=207161...	0	benigno
9	12	22:09:07.563081614	2	mysqld	(4427)	saida	futex	res=-110(ETIMEDOUT)	0	benigno
10	13	22:09:07.563084000	2	mysqld	(4427)	entrada	futex	addr=20op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE)va...	0	benigno

Figura 3.30: Resulatdo final do ficheiro CSV do comportamento benigno.

O pré-processamento dos ficheiros CSV ocorreu através de uma implementação em Python[10] com a biblioteca Pandas[11] e o parâmetro *chunksize*[12] para iteração com grandes conjunto de dados. A implementação foi responsável pela organização e tratamento das informações. Ilustrado na figura 3.30 o resultado final do ficheiro CSV do comportamento benigno para interação com os algoritmos de aprendizagem automática. Nesta figura, destaca-se as seguintes modificações:

- na coluna *evt_dir* a alteração do sinal de “>” (maior que) para a palavra “entrada” e do sinal de “<” (menor que) para a palavra “saida”;
- adição da coluna *label* e seu respectivo valor, 0 (zero) para benigno;
- e adição da coluna comportamento.

Em particular com o contentor malicioso para uma melhor compreensão do desempenho dos algoritmos de aprendizagem automática em relação as vulnerabilidades da imagem MySQL versão: 5.6.27 e aproveitamento das informações coletadas fez-se a identificação da coluna comportamento em concordância com as respectivas vulnerabilidades desta imagem, conforme tabela 3.8.

[10] https://www.python.org/
[11] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html
[12] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Tabela 3.8: Classificação do comportamento malicioso.

Comportamento Malicioso		
ID CVE	**Vulnerabilidade**	**Comportamento**
2012-2122	*Authentication Bypass*	malicioso_1
2013-1861	*DoS Overflow*	malicioso_2
2016-5616	*System User Privilege Escalation / Race Condition*	malicioso_3
2017-3599	*Integer Overflow*	malicioso_4

Na figura 3.31 observase, um exemplo do resultado final do ficheiro CSV do comportamento malicioso. Destaca-se as seguintes alterações:

- na coluna *evt_dir* a alteração do sinal de “>” (maior que) para a palavra “entrada” e do sinal de “<” (menor que) para a palavra “saida”;
- adição da coluna *label* e seu respectivo valor, 1(um) para malicioso;
- e adição da coluna comportamento em concordância com sua vulnerabilidade conforme a tabela 3.8.

	evt.num	evt.time	evt.cpu	proc.name	thread.tid	evt.dir	evt.type	evt.args	label	comportamento
0	3	17:32:33.455197664	0	mysqld	(4947)	entrada	io_getevents		1	malicioso_4
1	4	17:32:33.455202278	0	mysqld	(4947)	entrada	switch	next=4953(mysqld)pgft_maj=0pgft_min=1vm_size=1...	1	malicioso_4
2	5	17:32:33.455203791	0	mysqld	(4953)	saida	io_getevents		1	malicioso_4
3	6	17:32:33.455205354	0	mysqld	(4953)	entrada	io_getevents		1	malicioso_4
4	7	17:32:33.455206713	0	mysqld	(4953)	entrada	switch	next=4950(mysqld)pgft_maj=0pgft_min=2vm_size=1...	1	malicioso_4
5	8	17:32:33.455208055	0	mysqld	(4950)	saida	io_getevents		1	malicioso_4
6	9	17:32:33.455209842	0	mysqld	(4950)	entrada	io_getevents		1	malicioso_4
7	10	17:32:33.455211052	0	mysqld	(4950)	entrada	switch	next=4949(mysqld)pgft_maj=0pgft_min=8vm_size=1...	1	malicioso_4
8	11	17:32:33.455212447	0	mysqld	(4949)	saida	io_getevents		1	malicioso_4
9	12	17:32:33.455213895	0	mysqld	(4949)	entrada	io_getevents		1	malicioso_4

Figura 3.31: Resulatdo final do ficheiro CSV do comportamento malicioso.

Logo, o processo tratamento de dados representa uma fase importante para que os mesmos estejam de acordo como os modelos que serão construídos pelos algoritmos de aprendizagem automática. Contudo, para facilitar a manipulação e disponibilização através dos algoritmos de aprendizagem automática o conjunto de dados foi armazenados em um base de dados, denominada *db_contentor*, com a tecnologia de gerenciamento de base de dados PostgreSQL[13]. Figura 3.32.

[13] `https://www.postgresql.org/`

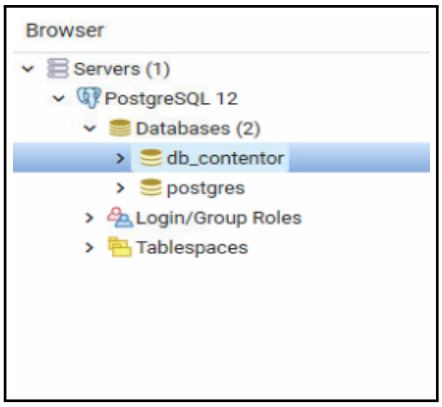


Figura 3.32: Base de dados `db_contentor`.

Como parte da proposta de investigação a geração de uma base de dados contendo dados coletados dos contentores neste ambiente experimental está disponível *online*[14] e pode ser utilizada para validação e trabalhos futuros com a mesma temática desta investigação.

3.6 Classificador

A seleção das tecnologias envolvidas na implementação dos classificadores de aprendizagem automática e posterior avaliação do desempenho dos mesmos usando uma base de dados de informações sobre os comportamentos benigno e malicioso do contentor teve o objetivo de construir um cenário de programação local com conexão com a base de dados `db_contentor`. Para a construção deste cenário foram empregados as seguintes tecnologias para a realização desta investigação, com uma breve descrição das mesmas, têm-se:

- A linguagem de programação *Python*[15] é uma linguagem e plataforma completa que pode usar tanto para pesquisa e desenvolvimento. Há também os módulos e bibliotecas para escolher e que fornecem várias maneiras de fazer cada tarefa como por exemplo o tratamento dos dados;
- A biblioteca de mapeamento objeto-relacional SQL em código aberto desenvolvida para a linguagem de programação *Python*, *SQLAlchemy*[16], oferece um conjunto de ferramentas completo de padrões de persistência projetado para acesso eficiente e de alto desempenho a base de dados;
- A biblioteca *Pandas*[17] para exploração, análise e manipulação de dados. É um pacote *Python* para ciências de dados que facilita a manipulação dos mesmos com uma estrutura poderosa e flexível.
- A biblioteca *NumPy*[18] usada para se trabalhar com computação numérica e realizar cálculos em *arrays* multidimensionais;

[14] `https://github.com/marcoscavalcant/db_container`
[15] `https://www.python.org/`
[16] `https://www.sqlalchemy.org/`
[17] `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html`
[18] `https://numpy.org/`

- A biblioteca *Scikit-learn* fornece algoritmos de aprendizagem não supervisionados e supervisionados para a linguagem de programação *Python* [23].

As técnicas Bag of System Calls (BoSC) e janela deslizante foram implementadas em *Python* de acordo com suas definições. A técnica BoSC é definida como uma lista ordenada (c1, c2, c3,..., cm) onde m é o número total de chamadas de sistema distintas e cj é o número de ocorrências da chamada de sistema, sj , na sequência de entrada Zj [115]. Na figura 3.33 é exibido um exemplo de um saco de chamadas de sistema para um total de 31 (trinta e um) chamadas de sistema distintas na sequência de entrada n=10. Observa-se a soma de todas as entradas no *array* é igual ao tamanho da janela.

```
[0, 5, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0]
```

Figura 3.33: Resultado com a técnica BoSC.

E por fim, a técnica da janela deslizante com tamanhos que variam entre 5, 10, 15, 20, 25 e 30 que são deslizados sobre os ficheiros que contêm as chamadas de sistema e as chamadas dentro da janela são registadas [128]. Ilustrado na figura 3.34 um exemplo da aplicação da técnica janela deslizante n=5.

```
[['futex', 'futex', 'futex', 'switch', 'futex'],
 ['futex', 'futex', 'switch', 'futex', 'futex']]
```

Figura 3.34: Resultado com a técnica janela deslizante n=5.

3.6.1 Importação das Bibliotecas

A princípio esta etapa possui um fundamental importância para o correto funcionamento da implementação, pois as bibliotecas são funções pré-escritas que atendem as demandas especificas de uma linguagem de programação. Figura 3.35.

```
import pandas as pd
import numpy as np
import sqlalchemy
from sklearn import metrics
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Figura 3.35: Importação das bibliotecas.

3.6.2 Conexão com a Base de Dados

Esta conexão faz-se necessária devido ao grande volume de dados armazenados (mais de 6 GB) em relação aos comportamentos dos contentores e com intuito de facilitar a manipulação e aproveitamento das características dos mesmos pela implementação *Python* oferecendo assim liberdade na configuração do conjunto de dados para treino e teste dos algoritmos de aprendizagem automática.

```
engine = sqlalchemy.create_engine("postgresql://mar:1234@localhost:5432/contentor")
con = engine.connect()
print(engine.table_names())
```

Figura 3.36: Conexão com a base de dados.

3.6.3 Conjunto de Dados

O conjunto de dados escolhido para treinar e testar os algoritmos de aprendizagem automática é, na verdade, um subconjunto dos dados armazenados. Houve a preocupação de escolher uma quantidade de dados com equilíbrio entre amostras das diferentes classes, ou seja, um conjunto de dados balanceado. O conjunto de dados têm 200.000 instâncias, ou dados, nos quais 100.000 registos de chamadas de sistema do comportamento benigno e outros 100.000 registos de chamadas de sistema do comportamento malicioso divididos igualmente (25.000 registos) entre as quatro vulnerabilidades. Ilustrado na figura 3.37.

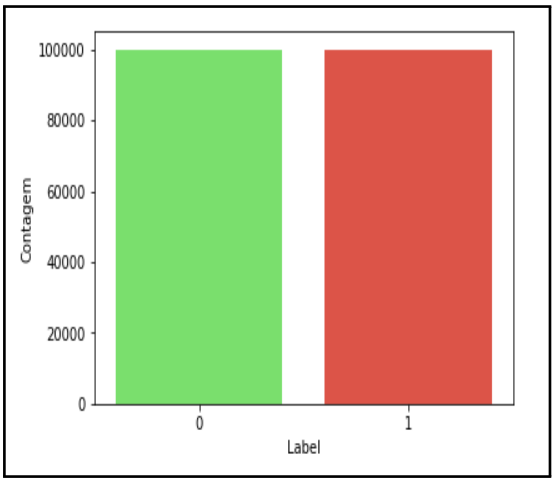


Figura 3.37: Contagem dos dados.

A partir deste conjunto de dados, apenas os atributos, *features*, necessário para treinar e testar os algoritmos de aprendizagem automática foram selecionados. Conforme se pode

ver na figura 3.38. Neste caso, *evt_type*, isto é, as chamadas de sistema e *label*. Esta decisão é importante para minimizar o custo de processamento e eliminar o ruído do modelo, conforme se ilustra na figura 3.39.

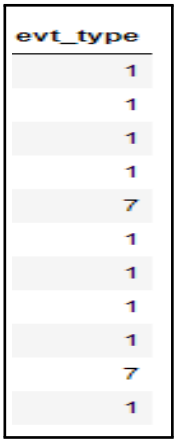
t_num	evt_time	evt_cpu	proc_name	thread_tid	evt_dir	evt_type	evt_info	label	comportamento
3	19:04:36.230453636	0	mysqld	(4493)	entrada	futex	addr=E0op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE)va...	0	benigno
4	19:04:36.230454708	0	mysqld	(4493)	saida	futex	res=0	0	benigno
5	19:04:36.230459330	0	mysqld	(4493)	entrada	futex	addr=30op=128(FUTEX_PRIVATE_FLAG)val=0 ...	0	benigno
6	19:04:36.230475421	0	mysqld	(4493)	entrada	switch	next=4495(mysqld)pgft_maj=0pgft_min=1vm_size=1...	0	benigno
7	19:04:36.230476786	0	mysqld	(4495)	saida	futex	res=-110(ETIMEDOUT)	0	benigno
3	15:45:12.895890230	0	mysqld	(4961)	entrada	select		1	malicioso_3
4	15:45:12.895895591	0	mysqld	(4961)	entrada	switch	next=26817(sysdig)pgft_maj=0pgft_min=386vm_siz...	1	malicioso_3
5	15:45:12.970343844	0	mysqld	(4958)	saida	futex	res=-110(ETIMEDOUT)	1	malicioso_3
6	15:45:12.970349887	0	mysqld	(4958)	entrada	futex	addr=40op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE)va...	1	malicioso_3
7	15:45:12.970350318	0	mysqld	(4958)	saida	futex	res=0	1	malicioso_3
8	15:45:12.970353410	0	mysqld	(4958)	entrada	futex	addr=84op=137(FUTEX_PRIVATE_FLAG|FUTEX_WAIT_BI...	1	malicioso_3
9	15:45:12.970357581	0	mysqld	(4958)	entrada	switch	next=3301(llvmpipe-0)pgft_maj=0pgft_min=0vm_si...	1	malicioso_3
10	15:45:12.990022535	0	mysqld	(4956)	saida	io_getevents		1	malicioso_3
11	15:45:12.990028765	0	mysqld	(4956)	entrada	io_getevents		1	malicioso_3
12	15:45:12.990036640	0	mysqld	(4956)	entrada	switch	next=26817(sysdig)pgft_maj=0pgft_min=1vm_size=...	1	malicioso_3
435	15:45:13.107639311	0	mysqld	(4959)	saida	futex	res=-110(ETIMEDOUT)	1	malicioso_3

Figura 3.38: Visão parcial do conjunto de dados.

evt_type	label
switch	0
futex	0
futex	0
futex	0
futex	0
futex	0
switch	0
futex	0
futex	0
futex	0
futex	0

Figura 3.39: Estratificação do conjunto de dados.

Para converter as variáveis categóricas da coluna *evt_type* em dados em um formato acessível, isto é, dados numéricos compreensíveis para construção dos modelos de classificação utilizou-se o *Label Encoder* e *One Hot Encoder* do *Scikit-learn*. Figura 3.40 e 3.41.



evt_type
1
1
1
1
7
1
1
1
1
7
1

Figura 3.40: *Label Encoder* no conjunto de dados.

```
array([[0., 1., 0., 1., 1., 0.],
       [0., 1., 0., 1., 0., 0.],
       [0., 0., 1., 1., 0., 0.],
       [0., 1., 0., 1., 1., 0.],
       [0., 1., 0., 1., 0., 0.],
       [0., 0., 1., 1., 0., 0.],
       [0., 1., 0., 1., 1., 0.],
       [0., 1., 0., 1., 0., 0.],
       [0., 0., 1., 1., 0., 0.],
       [1., 0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0., 0.]])
```

Figura 3.41: *One Hot Encoder* no conjunto de dados.

Após abordagem de codificação da coluna *evt_type* dividiu-se o conjunto de dados em 60% para treino e 40% para teste, figura 3.42, e procedeu a implementação, conforme documentação e no formato padrão do *Scikit-learn* [23], dos algoritmos de aprendizagem automática: AdaBoost, Decision Tree, Gaussian Naive Bayes, K-Nearest Neighbors, Multi-layer Perceptron, Multinomial Naive Bayes, Random Forest e Support Vector Machine para obtenção dos resultados.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.4,random_state=109)
```

Figura 3.42: Divisão do conjunto de dados.

A figura 3.43 ilustra como exemplo a implementação do classificador AdaBoost.

```
from sklearn.ensemble import AdaBoostClassifier
classifier_AB = AdaBoostClassifier(n_estimators=50, random_state=40)
classifier_AB.fit(X_train, y_train)
y_pred_AB = classifier_AB.predict(X_test)
print(classification_report(y_test, y_pred_AB,digits=3))
```

Figura 3.43: Implementação do algoritmo AdaBoost.

3.6.4 Arquitetura do Classificador

A arquitetura do classificador para obtenção dos resultados e a aplicação das técnicas Bag of System Calls (BoSC) e janelas deslizantes com algoritmos de aprendizagem automática é ilustrada na figura 3.44. O classificador pode operar em três configurações diferentes: A) classificação das chamadas de sistema usando o *Label Encoder* e *One Hot Encoder*, B)

classificação das chamadas de sistema utilizando o algoritmo janela deslizante com *Label Encoder* e *One Hot Encoder* e C) classificação das chamadas de sistema com as técnicas janela deslizante e BoSC.

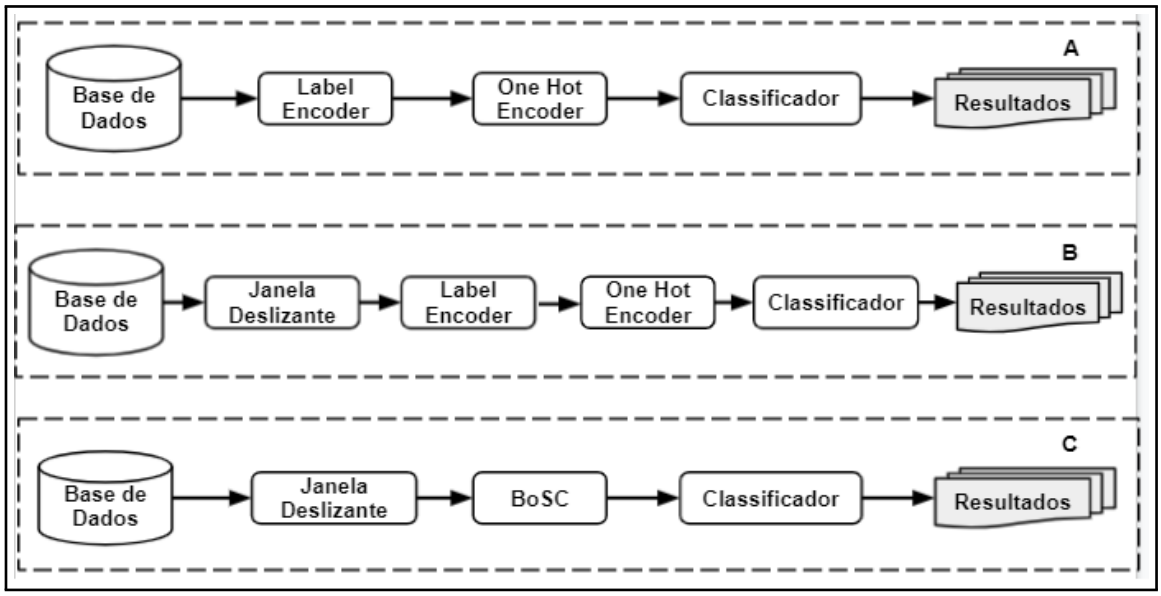


Figura 3.44: Arquitetura do classificador com três configurações possíveis usando: A) *Label Encoder* e *One Hot Encoder*, B) Janela deslizante com *Label Encoder* e *One Hot Encoder* e C) Janela deslizante com *Bag of System Call Algorithm* (BoSC).

3.6.5 Otimização do Classificador

As configurações dos algoritmos de aprendizagem automática aplicados nesta investigação conforme documentação oficial [23] estão ilustradas na Tabela 3.9. No caso dos Algoritmos AdaBoost, K-Nearest Neighbors, Random Forest e Support Vector Machine, alguns parâmetros específicos, `n_neighbors`, `C` e `n_estimators`, foram monitorados com a base de dados mencionada na subsecção 3.6.3 em relação às métricas *Precision* e *Recall* a fim de selecionar valores que ofereçam uma possível otimização destes classificadores.

O intervalo selecionado para realizar uma análise de otimização dos classificadores K-Nearest Neighbors e Support Vector Machine é de 1 a 40. Essa monitorização é realizada diretamente nos classificadores com o conjunto de dados selecionado e visa determinar valores para os parâmetros mencionados que sinalizam redução nos custos de processamento e otimização do desempenho do classificador.

No caso dos algoritmos AdaBoost e Random Forest, o intervalo foi definido de acordo com suas configurações que constam na documentação padrão [23]. De 1 a 50 para o algoritmo AdaBoost e de 1 a 100 para o algoritmo Random Forest.

Tabela 3.9: Configuração dos classificadores (baseado em [23]).

Classificador	Configuração
AdaBoost	n_estimators=50, random_state=40
Decision Tree	criterion='gini', splitter='best', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, min_impurity_decrease=0.0, ccp_alpha=0.0
Gaussian Naive Bayes	priors=None, var_smoothing=1e-09
K-Nearest Neighbors	algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=3, p=2, weights='uniform'
Multi-layer Perceptron	solver='lbfgs', alpha=1e-5, max_iter=2000, hidden_layer_sizes=(5, 2), random_state=1
Multinomial Naive Bayes	(alpha=1.0, fit_prior=True, class_prior=None)
Random Forest	n_estimators=100, random_state=40
Support Vector Machine	C=1.0, kernel='linear', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None

Na figura 3.45 o parâmetro `C` com valor maior que 1 observa-se um crescimento para as métricas *Precision* entre 65% a 79% e *Recall* entre 67% a 85%. A partir do parâmetro `C` maior que 5 os valores para as métricas permanecem constantes. Assim, com valor do parâmetro `C` igual a 1 pode-se obter aproximadamente medidas registadas para a métrica *Precision* de 80% e *Recall* de 84%.

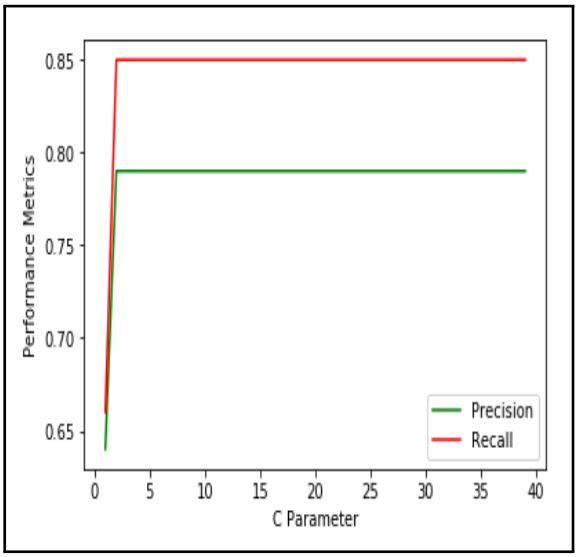


Figura 3.45: Parâmetro `C`

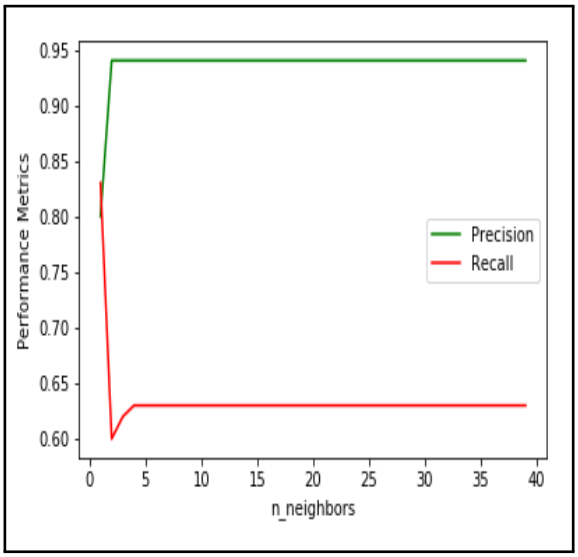


Figura 3.46: Parâmetro `n_neighbors`.

No caso da figura 3.46, com o valor de aproximadamente `n_neighbors` maior que 2, um ligeiro crescimento é notado com um valor acima de 60% para a métrica *Recall*. Com valores de `n_neighbors` acima de 5, pequenas variações podem ser vistas nos valores obtidos pela métrica *Recall* em um intervalo de 60% a 63%.

Para a métrica *Precision* para valores de aproximadamente `n_neighbors` maiores que 2 mostra uma diminuição nos valores variando de 95% a 90%. Deste ponto em diante, o gráfico mostra pequenas variações dentro desta faixa. Portanto, com um valor `n_neighbors` igual ou menor que 3, valores para métricas *Recall* acima de 83% e *Precision* acima de 93% devem ser obtidos.

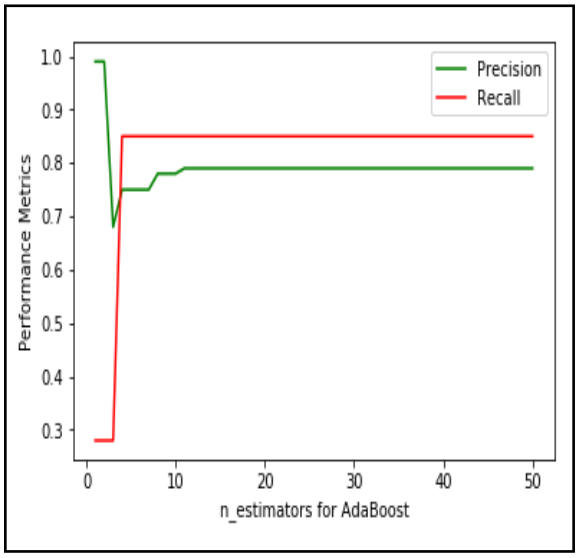


Figura 3.47: Parâmetro `n_estimators` para AdaBoost.

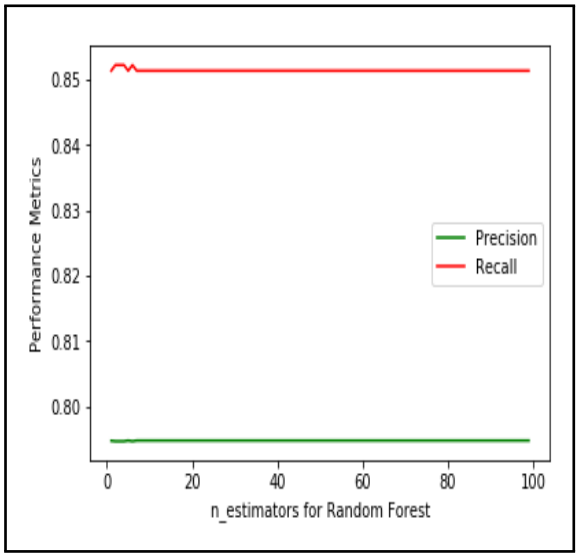


Figura 3.48: Parâmetro `n_estimators` para Random Forest.

O gráfico da figura 3.47 ilustra a evolução do parâmetro `n_estimators` do algoritmo AdaBoost. De acordo com a documentação *Scikit-learn* [23], o parâmetro `n_estimators` é definido como o número máximo de estimadores em que o reforço é terminado. No caso de um ajuste perfeito, o processo de aprendizagem é interrompido precocemente.

Neste caso, existe um ponto de intersecção entre a *Precision* e a *Recall* para o valor de `n_estimators` = 2,5. E que no intervalo de aproximadamente 2,5 < `n_estimators` < 5, um valor para *Precision* acima de 75% e *Recall* acima de 86% pode ser alcançado. A partir de `n_estimators` acima de 5, as medições de *Precision* e de *Recall* tornam-se constantes com *Precision* acima de 70% e *Recall* acima de 85%. Assim, um valor de `n_estimators` = 50 pode alcançar valores para a *Recall* acima de 85% e para *Precision* acima de 75%.

O gráfico da figura 3.48 mostra o desempenho do parâmetro `n_estimators` que é usado para controlar o número de árvores no processo pelo algoritmo Random Forest em relação à *Precision* e *Recall*.

Assumindo 0<`n_estimators`<5, a *Recall* demonstra um comportamento crescente com valores entre 85% e 90%. E a *Precision* exibe um resultado constante abaixo de 80%. Portanto, para um valor de `n_estimators` = 100, acredita-se que possa atender as medidas

para as métricas de *Recall* acima de 90% e *Precision* acima de 80%.

3.7 Conclusão

Neste capítulo foram expostos de forma detalhada e objetiva os procedimentos e tecnologias utilizadas para implementação do ambiente experimental e obtenção do resultados da investigação.

De início é exposto uma visão geral do ambiente experimental composto pelos componente contentor e implementações para coleta de dados, processamento de dados e classificador e explicado suas funcionalidade e relacionamentos entre eles. Também de forma esclarecedora são divulgados as características técnicas de *hardware* e *software* utilizadas neste ambiente experimental com o intuito de oferecer uma melhor compreensão desta implementação.

Seguindo a ordem dos componentes da arquitetura do ambiente experimental é detalhada a instalação da tecnologia *Docker* , conforme documentação, como também todos procedimentos para *download* e configuração das imagens dos servidores MySQL versão:8.0.23 e versão:5.6.27. No tocante ao carregamento dos dados dos contentores benigno e malicioso utilizou-se um implementação do TPC-C *Benchmark* que é definida como um sistema de transações *on-line* usado para registar as atividades de um empresa atacadista.

Esta mesma implementação foi responsável pela simulação do comportamento benigno malicioso. É importante destacar que no caso do comportamento malicioso precisou adicionar a implementação do TPC-C *Benchmark* vulnerabilidades presentes na imagem MySQL versão:5.0.27. Estas vulnerabilidades, *exploits*, foram selecionadas em consulta as bases de dados *Common Vulnerabilities and Exposures* (CVE) e *Exploit Database* (EDB). Para a coletas das chamadas de sistema em relação ao comportamento dos contentores fez-se uso da ferramenta de diagnóstico *sysdig* que possuem um módulo para escrita, leitura e armazenamento das chamadas de sistema.

Por fim, o tratamento dos dados contou com uma implementação na linguagem de programação *Python* para realizar a adequação dos ficheiros com as chamadas de sistema para formato acessível para manipulação destes ficheiros e construção de um conjunto de dados para treino e teste dos algoritmos de aprendizagem automática e obtenção dos resultados através do classificador.

Capítulo 4

Avaliação do Desempenho do Sistema de Detecção de Intrusões a Nível do Contentor

4.1 Introdução

Neste capítulo são expostos e discutidos os resultados do desempenho dos algoritmos de aprendizagem automática definidos para uso nesta investigação em relação ao conjunto de dados obtidos sobre os comportamentos benigno e malicioso dos contentores. Os resultados são analisados e comparados, através de tabelas e gráficos, de modo individual ou em grupo entre os classificadores de forma a selecionar os mais interessantes e com o intuito de verificar e destacar o impacto do conjunto de dados no desempenho dos diferentes classificadores. É composto por seis secções. A presente secção 4.1 é dedicada à introdução do capítulo. A secção 4.2 descreve os resultados com aplicação dos codificadores *Label Encoder* e *One Hot Encoder* do *Scikit-learn*. A secção 4.3 explana sobre os resultados alcançados com a técnica janela deslizante. A secção 4.4 apresenta os resultados com as técnicas janela deslizante e BoSC. A secção 4.5 descreve o uso dos recursos e por fim a secção 4.6 a conclusão do capítulo.

4.2 Resultados com *Label Encoder* e *One Hot Encoder*

Os resultados contidos na tabela 4.1 devem-se à aplicação dos codificadores *Label Encoder* e *One Hot Encoder* do *Scikit-learn* que são utilizados para converter dados categóricos (chamadas de sistema) em dados numéricos.

Os comentários sobre os resultados alcançados pelos classificadores são analisados individualmente por métrica. Em relação a métrica *Precision*, os resultados estão entre 54,7% a 93,6% em destaque o valor acima de 54,7% para o algoritmo Guassian Naive Bayes. O algoritmo K-Nearest Neighbors com resultado acima de 93,6%. E por fim com intervalo de resultado entre 75,1% a 79,5% para os demais classificadores.

No caso da métrica *Recall* os valores estão dentro de uma faixa de intervalo de 62,4% a 99,7%. Em evidência para o classificador Guassian Naive Bayes com valor de 99,7%. O algoritmo K-Nearest Neighbors com resultado de 62,4%. E os demais classificadores com valores acima de 85,0%.

Os resultados expostos para a métrica *F-Measure* estão no intervalo de 70,7% a 83,2%. O algoritmo Guassian Naive Bayes com valor acima de 70,0% e algoritmo K-Nearest Neighbors com resultado acima de 74,0%. Os demais classificadores com valores acima de 82,0%.

Tabela 4.1: Resultados com *Label Encoder* e *One Hot Encoder*.

	Métricas		
Classificador	Precision	Recall	F-Measure
AdaBoost	0,794	0,852	0,822
Decision Tree	0,794	0,851	0,822
Gaussian Naive Bayes	0,547	0,997	0,707
K-Nearest Neighbors	0,936	0,624	0,749
Multi-layer Perceptron	0,751	0,853	0,799
Multinomial Naive Bayes	0,794	0,851	0,822
Random Forest	0,794	0,851	0,822
Support Vector Machine	0,795	0,855	0,832

Logo, com os valores presentes na tabela 4.1, é possível verificar que o melhor resultado pertence ao classificador Support Vector Machine com 83,2% para *F-Measure*. A figura 4.1, curva ROC, fornece uma visão clara para medir e comparar o desempenho dos algoritmos de classificação. Neste caso destaca-se os classificadores AbaBoost, Decision Tree, Multinomial Naive Bayes e Random Forest com AUC igual a 89,2%.

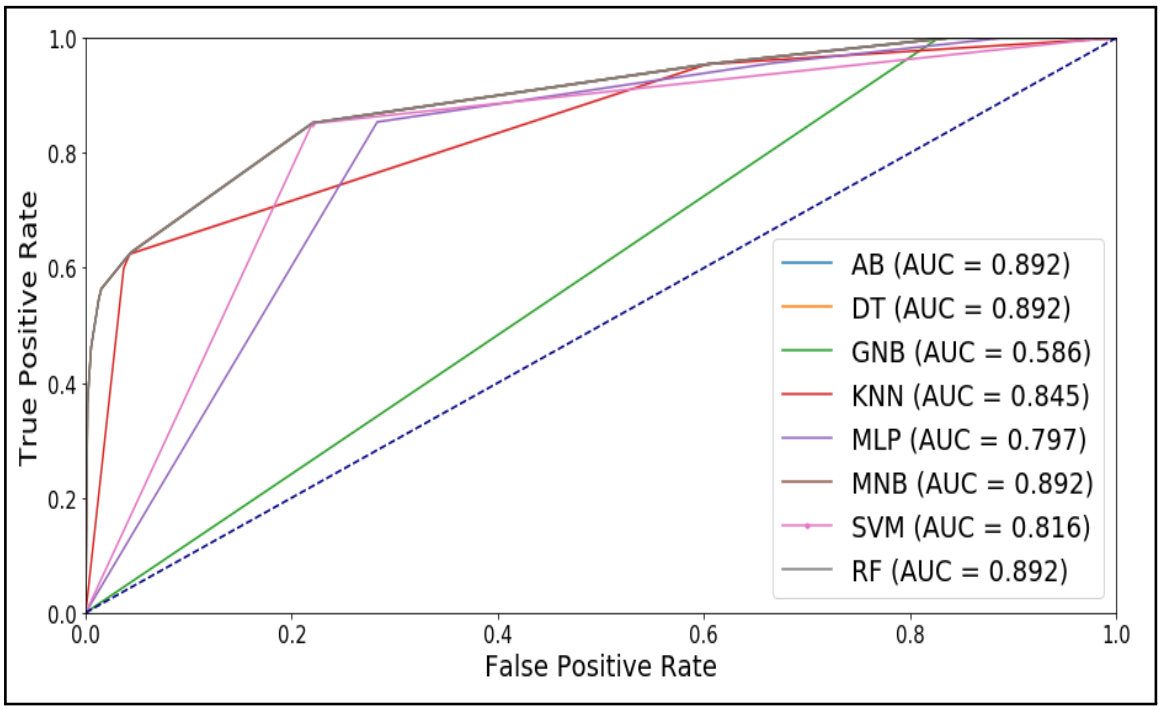


Figura 4.1: Curva ROC para algoritmos de aprendizagem automática com *Label Encoder* e *One Hot Encoder*.

4.3 Resultados com a técnica janela deslizante

A comparação dos resultados presentes na tabela 4.2 avalia o desempenho dos algoritmos de aprendizagem automática com a técnica janela deslizante com tamanhos 5, 10, 15, 20, 25 e 30. A análise dos resultados desta tabela será realizado por cada classificador.

Tabela 4.2: Resultados com a técnica janela deslizante.

Classificador	Métricas	Tamanho da Janela Deslizante					
		5	10	15	20	25	30
AdaBoost	Precision	0,968	0,973	0,977	0,981	0,979	0,981
	Recall	0,902	0,940	0,960	0,964	0,973	0,976
	F-Measure	0,934	0,956	0,968	0,972	0,976	0,979
Decision Tree	Precision	0,972	0,975	0,978	0,982	0,983	0,985
	Recall	0,934	0,968	0,976	0,980	0,982	0,984
	F-Measure	0,952	0,972	0,977	0,981	0,983	0,985
Guassian Naive Bayes	Precision	0,613	0,654	0,682	0,709	0,732	0,751
	Recall	0,998	1,000	1,000	1,000	1,000	1,000
	F-Measure	0,759	0,791	0,811	0,830	0,845	0,858
K-Nearest Neighbors	Precision	0,958	0,978	0,985	0,992	0,994	0,996
	Recall	0,939	0,959	0,966	0,966	0,966	0,964
	F-Measure	0,958	0,969	0,976	0,978	0,980	0,980
Multi-layer Perceptron	Precision	0,968	0,981	0,988	0,992	0,993	0,993
	Recall	0,931	0,967	0,980	0,986	0,989	0,990
	F-Measure	0,949	0,974	0,984	0,989	0,991	0,991
Multinomial Naive Bayes	Precision	0,963	0,975	0,980	0,987	0,988	0,989
	Recall	0,896	0,937	0,951	0,962	0,968	0,971
	F-Measure	0,928	0,956	0,965	0,974	0,978	0,980
Random Forest	Precision	0,971	0,981	0,988	0,993	0,994	0,996
	Recall	0,937	0,975	0,984	0,986	0,989	0,991
	F-Measure	0,954	0,978	0,986	0,989	0,992	0,994
Support Vector Machine	Precision	0,960	0,976	0,987	0,990	0,992	0,994
	Recall	0,923	0,961	0,978	0,984	0,989	0,992
	F-Measure	0,941	0,969	0,982	0,987	0,990	0,993

No caso do AdaBoost nota-se que a *Precision* atingiu resultados entre 96,8% a 98,1%. Para a *Recall* percebe-se um aumento entre medidas percentuais de 90,2% a 97,6%. E a *F-Measure* com resultados entre 93,4% a 97,9%.

O classificador Decision Tree apresentou resultados acima dos 93,0% para todos os tamanhos de janelas deslizantes e métricas. A *Precision* com resultados acima de 98,0% para as janelas deslizantes 20, 25 e 30. A métrica *Recall* com resultados acima de 98,1% para as janelas deslizantes 25 e 30. E a *F-Measure* com resultados entre 95,2% a 98,5%.

Os resultados alcançados pelo Guassian Naive Bayes estão no intervalo de 61,3% a 100%. A *Precision* com resultados na faixa de 61,3% a 75,1%. A *Recall* com resultados de 100% para as janelas deslizantes de tamanhos 10, 15, 20, 25 e 30. E, finalmente, *F-Measure* com resultados acima dos 80% para as janelas de tamanhos 15, 20, 25 e 30.

No caso do algoritmo K-Nearest Neighbors, os resultados são superiores a 93,0% para todos os tamanhos de janelas deslizantes. A *Precision* com resultados superior a 99,2% para os tamanhos de janelas 25 e 30. A métrica *Recall* com resultados superiores a 95,9% para as janelas deslizantes de tamanhos 15, 20, 25 e 30. E a medida *F-Measure* com resultados igual a 98% para as janelas de tamanhos 25 e 30.

O Multi-layer Perceptron mostra resultados acima dos 93,1%. A *Precision* com resultados superiores a 98,8% para as janelas de tamanhos 20, 25 e 30. E a *Recall* com resultados acima de 98% para os tamanhos de janela 20, 25 e 30. E *F-Measure* com resultados acima de 98,9% para os tamanhos de janelas 25 e 30.

Para o Multinomial Naive Bayes os resultados estão acima dos 89%. A *Precision* com valores acima de 98% para os tamanhos de janelas 20, 25 e 30. A *Recall* com resultado acima de 97% para o tamanho de janela 30. E a métrica *F-Measure* com resultados acima de 97% para os tamanhos de janelas 20, 25 e 30.

No caso do Random Forest os resultados obtidos com a técnica janela deslizante são superiores a 93%. A *Precision* com resultados acima de 99% para as janelas de tamanhos 20, 25 e 30. A *Recall* com resultado acima de 98,4% para os tamanhos de janelas 20, 25 e 30. E a métrica *F-Measure* com resultados acima de 99% para os tamanhos de janelas 25 e 30.

O Support Vector Machine apresenta resultados acima dos 92,3%, em todos os tamanhos da técnica janela deslizante. A *Precision* com valores acima de 99% para as janelas de tamanhos 25 e 30. A *Recall* com resultados acima de 97,8% para os tamanhos de janelas 20, 25 e 30. E finalmente, a *F-Measure* com resultados acima de 98,2% para as janelas de tamanhos 20, 25 e 30.

Portanto, nesta configuração percebe-se uma evolução dos valores apresentados pelas métricas para os diferentes tamanhos das janelas deslizantes. Assim, com os valores presentes na tabela 4.2, é possível verificar que o melhor resultado pertence ao classificador Random Forest com 99,4% para *F-Measure* com tamanho da janela deslizante igual a 30.

4.4 Resultados com a técnica janela deslizante e BoSC

Em continuação a exposição dos resultados na tabela 4.3 são exibidos as medidas obtidas na avaliação do desempenho dos algoritmos de aprendizagem automática com as técnicas janela deslizante e BoSC. A análise dos resultados desta tabela será realizado por cada classificador.

Tabela 4.3: Resultados com a técnica janela deslizante e BoSC.

Classificador	Métricas	Janelas Deslizantes e BoSC					
		5	10	15	20	25	30
AdaBoost	Precision	0,966	0,976	0,989	0,990	0,992	0,996
	Recall	0,921	0,968	0,976	0,985	0,991	0,991
	F-Measure	0,943	0,972	0,982	0,987	0,992	0,994
Decision Tree	Precision	0,977	0,988	0,992	0,996	0,997	0,999
	Recall	0,920	0,966	0,984	0,990	0,994	0,996
	F-Measure	0,948	0,977	0,988	0,993	0,996	0,998
Guassian Naive Bayes	Precision	0,606	0,654	0,689	0,714	0,732	0,751
	Recall	0,999	0,998	0,999	1,000	1,000	1,000
	F-Measure	0,754	0,790	0,816	0,833	0,845	0,858
K-Nearest Neighbors	Precision	0,964	0,984	0,990	0,995	0,996	0,998
	Recall	0,922	0,964	0,982	0,989	0,993	0,997
	F-Measure	0,943	0,974	0,986	0,992	0,995	0,997
Multi-layer Perceptron	Precision	0,969	0,980	0,990	0,992	0,993	0,997
	Recall	0,924	0,965	0,978	0,984	0,990	0,994
	F-Measure	0,946	0,972	0,984	0,988	0,992	0,996
Multinomial Naive Bayes	Precision	0,963	0,976	0,981	0,987	0,989	0,989
	Recall	0,897	0,938	0,952	0,962	0,968	0,971
	F-Measure	0,929	0,956	0,966	0,975	0,978	0,980
Random Forest	Precision	0,977	0,988	0,993	0,996	0,998	0,999
	Recall	0,921	0,967	0,985	0,991	0,995	0,997
	F-Measure	0,948	0,977	0,989	0,993	0,996	0,998
Support Vector Machine	Precision	0,968	0,974	0,988	0,991	0,993	0,995
	Recall	0,913	0,965	0,978	0,983	0,989	0,993
	F-Measure	0,940	0,969	0,983	0,987	0,991	0,994

O algoritmo AdaBoost apresenta resultados na faixa dos 92,1% a 99,6%. As métricas *Pre-*

cision, *Recall* e *F-Measure* com resultados acima dos 99% para as janelas deslizantes 25 e 30.

O classificador Decision Tree demonstra resultados acima dos 99%. A métrica *Precision* com resultados acima dos 99% para as janelas deslizantes 15, 20, 25 e 30. A *Recall* com resultados superior a 99% para as janelas de tamanhos 25 e 30. E *F-Measure* com resultados acima de 98,8% para as janelas de tamanhos 20, 25 e 30.

Para o algoritmo Guassian Naive Bayes os resultados estão no intervalo de 60,6% a 100%. A *Precision* com resultados acima de 71,4% para os tamanhos de janelas deslizantes 25 e 30. A *Recall* com resultados iguais a 100% para os tamanhos de janelas deslizantes 20, 25 e 30. E finalmente, a *F-Measure* com valores acima a 83,3% para as janelas de tamanhos 25 e 30.

No caso do K-Nearest Neighbors os resultados estão no intervalo de 92,2% a 99,8%. A métrica *Precision* com valores acima de 99,0% para as janelas deslizantes de tamanhos 20, 25 e 30. A *Recall* e *F-Measure* com resultados acima de 99,0% para as janelas deslizantes de tamanhos 25 e 30.

O classificador Multi-layer Perceptron mostra resultados no intervalo de 92,4% a 99,7%. A *Precision* com resultados no intervalo de 96,9% a 99,7%. A *Recall* com resultado de 99,4% para a janela de tamanho 30. E a *F-Measure* com resultados acima de 98,8% para as janelas de tamanhos 25 e 30.

Os resultados apresentados pelo Multinomial Naive Bayes estão no intervalo de 89,7% a 98,9%. A *Precision* com resultados acima de 98,7% para as janelas deslizantes de tamanhos 25 e 30. A *Recall* com valores de 97,1% para o tamanho de janela deslizante 30. E *F-Measure* com resultados acima de 97,5% para as janelas de tamanhos 25 e 30.

Os resultados expostos pelo classificador Random Forest situam-se no intervalo de 92,1% a 99,9%. A *Precision* com resultados de 99,9% para o tamanho de janela deslizante 30. A *Recall* e *F-Measure* com resultados acima de 98,9% para os tamanhos de janelas deslizantes 20, 25 e 30.

O algoritmo Support Vector Machine exibe resultados no intervalo de 91,3% a 99,5%. A *Precision* com valores acima de 98,8% para as janelas deslizantes de tamanhos 20, 25 e 30. A *Recall* com resultados acima 98,3% para as janelas deslizantes de tamanhos 25 e 30. E finalmente, a métrica *F-Measure* com resultados acima de 98,7% para as janelas de tamanhos 25 e 30.

Neste configuração, em evidência têm-se que os melhores resultados pertencem aos classificadores Decision Tree e Random Forest com valor de 99,8% para *F-Measure* com janela deslizante tamanho igual a 30 e BoSC.

4.5 Uso de Recursos

A tabela 4.4 mostra os recursos utilizados para cada classificador no que diz respeito a CPU, RAM, e tempo de execução nos processos de treino e teste com a base de dados descrita na subsecção 3.6.3 em relação a três situação distintas em que os classificadores obtiveram um melhor desempenho. O comando GNU/Linux/`usr/bin/time` fornece os tempos de consumo e execução da CPU. A utilização da RAM foi recolhida utilizando o *software* psrecord [164].

Observa-se a partir do conjunto de medidas que com a utilização dos codificadores *Label Encoder* e *One Hot Encoder* o classificador Multi-layer Perceptron teve um bom desempenho de execução com custo de processo e consumo de RAM de 745 MB e tempo de execução 7 s atingindo os valores para *Precision* de 75,1%, *Recall* 85,3% e *F-Measure* de 79,9%. Seguido do AdaBoost com custo de processo e consumo de RAM de 873 MB e tempo de execução de 9,5 s atingindo os valores para *Precision* de 79,4%, *Recall* de 85,2% e *F-Measure* de 82,2%.

Tabela 4.4: Recurso computacional.

	Label Encoder One Hot Encoder			Janela Deslizante n = 30			Janela Deslizante n=30 + BoSC		
Classificador	CPU (%)	RAM (MB)	Tempo (s)	CPU (%)	RAM (MB)	Tempo (s)	CPU (%)	RAM (MB)	Tempo (s)
AdaBoost	89	873	9,5	93	1929	120	87	794	48
Decision Tree	79	985	11,3	98	1863	480	89	913	39
Gaussian Naive Bayes	73	1083	8,1	79	1689	58	85	893	44
K-Nearest Neighbors	83	596	526	93	1754	2820	91	1147	1550
Multi-layer Perceptron	86	745	7	96	1756	384	92	1094	244
Multinomial Naive Bayes	92	957	10	77	1505	55	82	674	45
Random Forest	97	953	18	98	1609	59	98	1163	135
Support Vector Machine	89	754	1200	98	879	2845	98	893	300

Com a técnica janela deslizante n=30 o classificador Multinomial Naive Bayes destaca-se com um custo de processo e consumo de RAM de 1505 MB e um tempo de execução de 55

s com valores para *Precision* de 98,1%, *Recall* 97,1% e *F-Measure* de 98,0%. É importante também mencionar o desempenho do Random Forest com custo de processo e consumo de RAM de 1609 MB e tempo de execução de 59 s e valores para *Precision* de 99,6%, *Recall* de 99,1% e *F-Measure* de 99,4%.

No caso da técnicas de janela deslizante n = 30 e BoSC, o classificador Decision Tree tem um custo de processo e consumo de RAM de 913 MB com um tempo de execução de 39 s para os valores de *Precision* com 99,9%, *Recall* com 99,6% e *F-Measure* de 99,8%. Seguido pelo Guassian Naive Bayes com custo de processo e consumo de RAM de 893 MB e tempo de execução de 44 s com valores de *Precision* com 75,1%, *Recall* de 100% e *F-Measure* de 85,8%.

4.6 Conclusão

Neste capítulo foi possível analisar e discutir os resultados alcançados na aplicação e viabilidade dos classificadores de aprendizagem automática na deteção de intrusões por anomalias em um ambiente *multi-tenant* em contentor, através de um conjunto de dados composto pelas chamadas de sistema.

Os gráficos e tabelas expostos ilustram os resultados dos experimentos realizados para avaliação do desempenho dos algoritmos de aprendizagem automática na deteção de anomalias. Os resultados obtidos foram analisados seguindo os critérios pré-estabelecidos na investigação e permitiram a obtenção das características de atuação dos classificadores na deteção de anomalias e identificação que os algoritmos Decision Tree com tempo de execução de 39 s e Random Forest com tempo de execução de 135 s ambos com *F-Measure* de 99,8% na arquitetura janela deslizante n=30 e BoSC apresentam o melhor desempenho.

É importante destacar que todos os valores obtidos são sob a tecnologia de contentores *Docker*. Acredita-se que a quantidade e qualidade dos dados coletados, ou seja, às características do conjunto de dados exerçam influências no modo como os classificadores atuam na deteção de anomalias. Prova disso, pode ser justificada pela variação das medidas alcançadas pelas métricas *Precision*, *Recall* e *F-Measure* nos diferentes modelos de classificação e cenários.

Dessa forma, pelo que foi exposto, considera-se que os resultados obtidos sobre o desempenho dos classificadores validam sua aplicabilidade na deteção de anomalias em um conjunto de dados formado pelas chamadas de sistema.

Capítulo 5

Conclusão e Trabalho Futuro

5.1 Principais Conclusões

De acordo com o decorrer da investigação, chega-se à conclusão de que os algoritmos de aprendizagem automática demonstram uma evolução do desempenho na deteção de anomalias em ambientes *multi-tenant* em contentores. E que as chamadas de sistema podem ser utilizadas para a construção de um conjunto de dados significativo do comportamento das aplicações nestes ambientes.

O objetivo principal deste trabalho de investigação consiste em avaliar e comparar o desempenho dos algoritmos de aprendizagem automática na deteção de intrusões baseada em *host* em ambientes *multi-tenant* em contentores. No início abordou-se os conceitos fundamentais que formaram uma base de conhecimento para que fosse possível entender e propor soluções para esta investigação. O estudo para a elaboração do estado da arte e a contextualização das teorias relacionadas com a presente investigação foram de fundamental importância para o planeamento de um plano de trabalho e especificamente para implementação do ambiente experimental.

Dos conceitos estudados, destacam-se a arquitetura de microsserviços, contentores, isolamento e alocação de recursos, sistema de deteção de intrusões, tecnologia de contentores e aprendizagem automática. Os conceitos e definições relacionadas aos dados estudados são importante na contextualização do problema pois, contribuíram para a construção de uma visão ampla e diversificada sobre a complexidades de implementação do sistema de deteção de intrusões e novos métodos de segurança para o ambiente *multi-tenant* em contentores como também para a determinação do conjunto de dados significativo em relação aos comportamentos benigno e malicioso das aplicações.

No desenvolvimento do ambiente experimental houve a preocupação de definir os padrões de instalação e configuração das tecnologias utilizadas em concordância com a documentação das mesmas. Os componentes do ambiente experimental foram instalados e configurados preservando seus relacionamentos e funcionalidades definidos na arquitetura para o sistema de deteção de intrusões em contentores desta investigação. Devido a metodologia de investigação adotada no presente trabalho os problemas e dificuldade na implementação do ambiente experimental foram superados após a avaliação dos resultados em um ciclo de melhoria contínua até atingir os objetivos propostos.

Os melhores resultados de classificação na deteção de intrusões neste ambiente foram obtidos com os algoritmos Decision Tree e Random Forest, usando uma janela deslizante de tamanho 30 e o algoritmos BoSC, atingindo ambos uma F-Measure de 99,8%. Contudo,

o algoritmo Decision Tree tem um menor tempo de execução e consome menos CPU e memória do que o algoritmo Random Forest.

No geral, percebe-se que a investigação tem um valor adicional de contribuição para a área de segurança da computação em nuvem. Porque, além do *background* e estado da arte, com a implementação do ambiente experimental foi possível obter resultados concretos, significativos e representativos do desempenho dos algoritmos de classificação de aprendizagem automática na deteção de intrusões em contentores.

5.2 Sugestões para Trabalho Futuro

Considera-se que a presente investigação atingiu o objetivo proposto. Porém, a abordagem utilizada merece reflexões, extensões e melhorias na forma de trabalhos futuros, os quais são especificados a seguir:

- Desenvolver uma aplicação para automatização e utilização dos modelos de classificação para a deteção de anomalias em tempo real;
- Implementar na arquitetura proposta outras tecnologias de contentores com o intuito de verificar e realizar um comparativo das abordagens;
- Incluir novas vulnerabilidades com o intuito de aumentar a representatividade dos resultados;
- Integrar os modelos de classificação de aprendizagem automática com outros sistemas de deteção de intrusões;
- Explorar na arquitetura proposta outras modalidades de deteção de anomalias, semi supervisionada e não supervisionada e realizar um comparativo das abordagens.

Bibliografia

[1] Z. Chen, W. Dong, H. Li, P. Zhang, X. Chen, and J. Cao, "Collaborative network security in multi-tenant data center for cloud computing," *Tsinghua Science and Technology*, vol. 19, no. 1, pp. 82–94, 2014. 1, 7

[2] G. C. F. A. Kai Hwang, Jack Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2011. 1

[3] T. Luxner. Cloud computing trends: 2021 state of the cloud report , year =2021,. Accessed: 22.01.2021. [Online]. Available: https://www.flexera.com/blog/cloud/cloud-computing-trends-2021-state-of-the-cloud-report 1

[4] International Data Corporation (IDC). (2020) Cloud adoption and opportunities will continue to expand leading to a $1 trillion market in 2024, according to idc. Accessed: 20.10.2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS46934120 1

[5] T. Bui, "Analysis of docker security," *ArXiv*, vol. abs/1501.02967, 2015. [Online]. Available: https://arxiv.org/abs/1501.02967 1, 2, 20, 21

[6] L. Cai, Y. Qi, W. Wei, and J. Li, "Improving resource usages of containers through auto-tuning container resource parameters," *IEEE Access*, vol. 7, pp. 108 530–108 541, 2019. 2

[7] Docker. (2020) What is a container? a standardized unit of software. Accessed: 22.10.2020. [Online]. Available: https://www.docker.com/resources/what-container 2, 13

[8] Google. (2020) Contentores no google. uma maneira melhor de desenvolver e implantar aplicações. Accessed: 22.10.2020. [Online]. Available: https://cloud.google.com/containers?hl=pt-br 2

[9] D. Xu, C. Fu, G. Li, D. Zou, H. Zhang, and X. Liu, "Virtualization of the encryption card for trust access in cloud computing," *IEEE Access*, vol. 5, pp. 20 652–20 667, 2017. 2

[10] M. P. Souppaya, J. Morello, and K. A. Scarfone, *Application container security guide. Technical Report NIST SP 800-190.* Gaithersburg, MD, USA: National Institute of Standards Technology, 2017. 2, 20

[11] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," National Institute of Standards and Technology, Tech. Rep., 2012. 2, 23

[12] U. Oktay, M. A. Aydin, and O. K. Sahingoz, "A circular chain intrusion detection for cloud computing based on improved adjointvm approach," in *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2013, pp. 201–206. 2

[13] Y. Bai and H. Kobayashi, "Intrusion detection systems: technology and development," in *17th International Conference on Advanced Information Networking and Applications, 2003. AINA 2003.*, 2003, pp. 710–715. 2

[14] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev, and L. Foschini, "Securing the infrastructure and the workloads of linux containers," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 559–567. [Online]. Available: https://ieeexplore.ieee.org/document/7346869 2, 16, 20, 36, 37

[15] A. S. Abed, C. Clancy, and D. S. Levy, "Intrusion detection system for applications using linux containers," *Lecture Notes in Computer Science*, p. 123–135, 2015. [Online]. Available: https://arxiv.org/abs/1611.03056 3, 4, 15, 19, 20, 25, 27, 35, 37, 39

[16] C. Fetzer, "Building critical applications using microservices," *IEEE Security Privacy*, vol. 14, no. 6, pp. 86–89, 2016. 3

[17] S. Newman, *Building Microservices*, 1st ed. O'Reilly Media, Inc., 2015. 3

[18] J. Flora and N. Antunes, "Studying the applicability of intrusion detection to multi-tenant container environments," in *2019 15th European Dependable Computing Conference (EDCC)*, 2019, pp. 133–136. 3, 4, 36, 37, 39

[19] S. Srinivasan, A. Kumar, M. Mahajan, D. Sitaram, and S. Gupta, *Probabilistic Real-Time Intrusion Detection System for Docker Containers: 6th International Symposium, SSCC 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers*, 01 2019, pp. 336–347. 3, 4, 36, 37

[20] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: https://doi.org/10.2753/MIS0742-1222240302 3, 4

[21] A. Nagaraja and T. S. Kumar, "An extensive survey on intrusion detection-past, present, future," in *Proceedings of the Fourth International Conference on Engineering MIS 2018*, ser. ICEMIS '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3234698.3234743 4

[22] M. M. F. Marcos A. O. Cavalcanti, Pedro R. M. Inácio, "Performance evaluation of container-level anomaly-based intrusion detection systems for multi-tenant applications using machine learning algorithms," *The 16th International Conference on Availability, Reliability and Security Proceedings, (ARES 2021)*. 5, 6

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in

Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 6, 34, 59, 62, 63, 64, 65

[24] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018. 7

[25] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "Sgx-aware container orchestration for heterogeneous clusters," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 730–741. 7, 14

[26] S. Kumar Pentyala, "Emergency communication system with docker containers, osm and rsync," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, 2017, pp. 1064–1069. 7

[27] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015. 7, 13, 16

[28] B. Siddhisena, L. Warusawithana, and M. Mendis, "Next generation multi-tenant virtualization cloud computing platform," in *13th International Conference on Advanced Communication Technology (ICACT2011)*, 2011, pp. 405–410. 7

[29] I. Ioniță and L. Ioniță, "An agent-based approach for building an intrusion detection system," in *2013 RoEduNet International Conference 12th Edition: Networking in Education and Research*, 2013, pp. 1–6. 7

[30] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," 2017. 8, 9, 10

[31] B. Jambunathan and K. Yoganathan, "Architecture decision on using microservices or serverless functions with containers," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, 2018, pp. 1–7. 8, 9

[32] R. Chandramouli, "Security strategies for microservices-based application systems," in *NIST SP 800-204*. National Institute of Standards and Technology (NIST). U.S. Department of Commerce., 2019. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-204/final 8, 9, 39

[33] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2*, ser. CLOSER 2016. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2016, p. 137–146. [Online]. Available: https://doi.org/10.5220/0005785501370146 8

[34] C. Esposito, A. Castiglione, and K. R. Choo, "Challenges in delivering software in the cloud as microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 10–14, 2016. 8

[35] D. S. Linthicum, “Practical use of microservices in moving workloads to the cloud,” *IEEE Cloud Computing*, vol. 3, no. 5, pp. 6–9, 2016. 8

[36] Docker. (2020) Get started with docker. we help developers and development teams build and ship apps. Accessed: 22.10.2020. [Online]. Available: https://www.docker.com/ 9, 45

[37] N. Kratzke, “About microservices, containers and their underestimated impact on network performance,” *ArXiv*, vol. abs/1710.04049, 2015. [Online]. Available: https://arxiv.org/abs/1710.04049 9

[38] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, “A mixed-method empirical study of function-as-a-service software development in industrial practice,” *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019. 9

[39] A. Alvarado. (2020) Serverless vs. faas: A beginner’s guide. Accessed: 26.12.2020. [Online]. Available: https://www.liquidweb.com/kb/serverless-vs-faas-a-beginners-guide/ 9

[40] T. Yarygina and A. H. Bagge, “Overcoming security challenges in microservice architectures,” in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2018, pp. 11–20. 10, 11

[41] A. Hannousse and S. Yahiouche, “Securing microservices and microservice architectures: A systematic mapping study,” *ArXiv*, vol. abs/2003.07262, 2020. [Online]. Available: https://arxiv.org/abs/2003.07262 10, 11

[42] H. Takabi, J. B. D. Joshi, and G. Ahn, “Security and privacy challenges in cloud computing environments,” *IEEE Security Privacy*, vol. 8, no. 6, pp. 24–31, 2010. 11

[43] C. Pautasso, O. Zimmermann, and F. Leymann, “Restful web services vs. ”big”’ web services: Making the right architectural decision,” in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 805–814. [Online]. Available: https://doi.org/10.1145/1367497.1367606 11

[44] T. Yarygina, “Restful is not secure,” in *International Conference on Applications and Techniques in Information Security*. Springer, 2017, pp. 141–153. [Online]. Available: https://doi.org/10.1007/978-981-10-5421-1_12 11

[45] F. Montesi and J. Weber, “Circuit breakers, discovery, and api gateways in microservices,” *arXiv preprint arXiv:1609.05830*, 2016. [Online]. Available: https://arxiv.org/abs/1609.05830 11

[46] C. Richardson and F. Smith, “Microservices: from design to deployment,” *Nginx Inc*, pp. 24–31, 2016. 11

[47] Wikipédia, a enciclopédia livre. . (1979) chroot. Accessed: 07.11.2020. [Online]. Available: https://pt.wikipedia.org/wiki/Chroot 12, 17

[48] ——. (2000) Freebsd. Accessed: 06.11.2020. [Online]. Available: https://pt.wikipedia.org/wiki/FreeBSD_jail 12

[49] Sun Microsystems. (2005) Oracle solaris containers. Accessed: 07.11.2020. [Online]. Available: https://www.oracle.com/solaris/technologies/solaris-containers.html 12

[50] OpenVZ. (2005) Open source container-based virtualization for linux. Accessed: 07.11.2020. [Online]. Available: https://openvz.org/ 12, 18

[51] Kernel: Virtuozzo, IBM, Google, Eric Biederman and others Userspace: Daniel Lezcano, Serge Hallyn, Stéphane Graber and others. (2008) Infraestrutura para projetos de contentores. Accessed: 07.11.2020. [Online]. Available: https://linuxcontainers.org/pt_br/ 12

[52] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019. 13, 14, 20

[53] N. Saleh and M. Mashaly, "A dynamic simulation environment for container-based cloud data centers using containercloudsim," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, 2019, pp. 332–336. 13

[54] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 48–56, 2017. 13

[55] B. Golden,. (2016) 3 reasons why you should always run microservices apps in containers. Accessed: 09.11.2020. [Online]. Available: https://techbeacon.com/app-dev-testing/3-reasons-why-you-should-always-run-microservices-apps-containers 14

[56] Gabor Nagy. (2015) Operating system containers vs. application containers. Accessed: 09.11.2020. [Online]. Available: https://blog.risingstack.com/operating-system-containers-vs-application-containers/ 15

[57] Michael Kerrisk. (2020) namespaces(7) — linux manual page. Accessed: 11.11.2020. [Online]. Available: https://man7.org/linux/man-pages/man7/namespaces.7.html 15, 16

[58] ——. (2020) cgroups(7) — linux manual page. Accessed: 14.11.2020. [Online]. Available: https://man7.org/linux/man-pages/man7/cgroups.7.html 15

[59] S. Singh and N. Singh, "Containers docker: Emerging roles future of cloud technology," in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, 2016, pp. 804–807. 15

[60] Á.Kovács, “Comparison of different linux containers,” in *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, 2017, pp. 47–51. 16, 17

[61] R. Rizki, A. Rakhmatsyah, and M. A. Nugroho, “Performance analysis of container-based hadoop cluster: Openvz and lxc,” in *2016 4th International Conference on Information and Communication Technology (ICoICT)*, 2016, pp. 1–4. 16

[62] Debian. (2019) O manual do administrador debian. introdução ao apparmor. Accessed: 18.11.2020. [Online]. Available: https://debian-handbook.info/browse/pt-BR/stable/sect.apparmor.html 17

[63] ArchLinux. (2020) Selinux. Accessed: 18.11.2020. [Online]. Available: https://wiki.archlinux.org/index.php/SELinux 17

[64] Michael Kerrisk. (2020) seccomp(2) — linux manual page. Accessed: 18.11.2020. [Online]. Available: https://man7.org/linux/man-pages/man2/seccomp.2.html 17

[65] ——. (2020) chroot(2) — linux manual page. Accessed: 18.11.2020. [Online]. Available: https://man7.org/linux/man-pages/man2/chroot.2.html 17

[66] ArchLinux. (2020) Capabilities. Accessed: 18.11.2020. [Online]. Available: https://wiki.archlinux.org/index.php/capabilities 17

[67] Docker. (2020) Docker overview. Accessed: 18.11.2020. [Online]. Available: https://docs.docker.com/get-started/overview/ 17, 18

[68] X. Xie, P. Wang, and Q. Wang, “The performance comparison of native and containers for the cloud,” in *2018 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2018, pp. 378–381. 17

[69] M. Ahmed, S. Zahda, and M. Abbas, “Server consolidation using openvz: Performance evaluation,” in *2008 11th International Conference on Computer and Information Technology*, 2008, pp. 341–346. 18

[70] X. Xie, P. Wang, and Q. Wang, “The performance analysis of docker and rkt based on kubernetes,” in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2017, pp. 2137–2141. 18

[71] Google Cloud. (2020) Pod. Accessed: 18.11.2020. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/pod 18

[72] CoreOS. (2020) rkt, a security-minded, standards-based container engine. Accessed: 18.11.2020. [Online]. Available: https://coreos.com/rkt/ 18

[73] SoftwareTestingHelp. (2021, Mar.) Top 10 best container software in 2021. [Online]. Available: https://www.softwaretestinghelp.com/container-software/ 18

[74] K. A. Asmitha and P. Vinod, “A machine learning approach for linux malware detection,” in *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014, pp. 825–830. 19

[75] MichaelKerrisk. strace(1) — linux manual page, year =2008,. Accessed: 22.02.2021. [Online]. Available: https://man7.org/linux/man-pages/man1/strace.1.html 19

[76] Wikipédia. strace , year =2020,. Accessed: 22.02.2021. [Online]. Available: https://pt.wikipedia.org/wiki/Strace 19

[77] MichaelKerrisk. ptrace(2) — linux manual page, year =2008,. Accessed: 22.02.2021. [Online]. Available: https://man7.org/linux/man-pages/man2/ptrace.2.html 19

[78] ——. sysdig(8) — linux manual page year =2020,. Accessed: 22.02.2021. [Online]. Available: https://man7.org/linux/man-pages/man8/sysdig.8.html 19

[79] S. Dhar. Sysdig overview year =2017,. Accessed: 22.02.2021. [Online]. Available: https://github.com/draios/sysdig/wiki/Sysdig-Overview 19

[80] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “Cloudscale: Elastic resource scaling for multi-tenant cloud systems,” in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC ’11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2038916.2038921 20

[81] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, “A measurement study on linux container security: Attacks and countermeasures,” in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 418–429. [Online]. Available: https://doi.org/10.1145/3274694.3274720 21

[82] Wikipédia, a enciclopédia livre. (2019) Exploit-segurança de computadores. Accessed: 08.12.2020. [Online]. Available: https://pt.wikipedia.org/wiki/Exploit 21

[83] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system: A systematic study of machine learning and deep learning approaches,” *Transactions on Emerging Telecommunications Technologies*, vol. n/a, no. n/a, p. e4150. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4150 22

[84] J. V. Anand Sukumar, I. Pranav, M. Neetish, and J. Narayanan, “Network intrusion detection using improved genetic k-means algorithm,” in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2018, pp. 2441–2446. 22

[85] Centro Nacional de Cibersegurança (CNCS). (2020) Relatório riscos conflitos 2020. Accessed: 28.10.2020. [Online]. Available: https://www.cncs.gov.pt/content/files/relatorio_riscos.conflitos2020__observatoriociberseguranca_cncs.pdf 22

[86] Z. S. Malek, B. Trivedi, and A. Shah, "User behavior pattern -signature based intrusion detection," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2020, pp. 549–552. 22

[87] J. Ng, D. Joshi, and S. M. Banik, "Applying data mining techniques to intrusion detection," in *2015 12th International Conference on Information Technology - New Generations*, 2015, pp. 800–801. 22, 23

[88] J. Sanchez, "Inefficiency of ids static anomaly detectors in real-world networks," *Future Internet*, vol. 7, pp. 94–109, 05 2015. 22

[89] V. Anand, "Intrusion detection: Tools, techniques and strategies," in *Proceedings of the 42nd Annual ACM SIGUCCS Conference on User Services*, ser. SIGUCCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 69–73. [Online]. Available: https://doi.org/10.1145/2661172.2661186 23, 28

[90] T. Verwoerd and R. Hunt, "Intrusion detection techniques and approaches," *Computer Communications*, vol. 25, no. 15, pp. 1356 – 1365, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366402000373 23

[91] A. Nadeem and M. P. Howarth, "A survey of manet intrusion detection prevention approaches for network layer attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2027–2045, 2013. 23

[92] D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, "A survey of distance and similarity measures used within network intrusion anomaly detection," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 70–91, 2015. 23

[93] F. Sabahi and A. Movaghar, "Intrusion detection: A survey," in *2008 Third International Conference on Systems and Networks Communications*, 2008, pp. 23–26. 23

[94] A. M. V. Bharathy, N. Umapathi, and S. Prabaharan, "An elaborate comprehensive survey on recent developments in behaviour based intrusion detection systems," in *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, 2019, pp. 1–5. 23

[95] A. K. Saxena, S. Sinha, and P. Shukla, "General study of intrusion detection system and survey of agent based intrusion detection system," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017, pp. 471–421. 23, 28, 29

[96] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 266–282, 2014. 23

[97] L. N. Tidjon, M. Frappier, and A. Mammar, "Intrusion detection systems: A cross-domain overview," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3639–3681, 2019. 23

[98] E. Benkhelifa, T. Welsh, and W. Hamouda, "A critical review of practices and challenges in intrusion detection systems for iot: Toward universal and resilient systems," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3496–3509, 2018. 23

[99] H. Hindy, D. Brosset, E. Bayne, A. K. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, "A taxonomy of network threats and the effect of current datasets on intrusion detection systems," *IEEE Access*, vol. 8, pp. 104 650–104 675, 2020. 23

[100] R. Uikey and M. Gyanchandani, "Survey on classification techniques applied to intrusion detection system and its comparative analysis," in *2019 International Conference on Communication and Electronics Systems (ICCES)*, 2019, pp. 1451–1456. 23

[101] D. H. Lakshminarayana, J. Philips, and N. Tabrizi, "A survey of intrusion detection techniques," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, pp. 1122–1129. 23

[102] F. Salo, M. Injadat, A. B. Nassif, A. Shami, and A. Essex, "Data mining techniques in intrusion detection systems: A systematic literature review," *IEEE Access*, vol. 6, pp. 56 046–56 058, 2018. 23

[103] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos, "From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3369–3388, 2018. 23

[104] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, "A survey of intrusion detection systems for cloud computing environment," in *2016 International Conference on Engineering MIS (ICEMIS)*, 2016, pp. 1–13. 23

[105] A. S. Subaira and P. Anitha, "Efficient classification mechanism for network intrusion detection system based on data mining techniques: A survey," in *2014 IEEE 8th International Conference on Intelligent Systems and Control (ISCO)*, 2014, pp. 274–280. 23

[106] A. Warzyński and G. Kołaczek, "Intrusion detection systems vulnerability on adversarial examples," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*, 2018, pp. 1–4. 24, 25

[107] S. More, M. Matthews, A. Joshi, and T. Finin, "A knowledge-based approach to intrusion detection modeling," in *2012 IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 75–81. 24

[108] M. A. Aydın, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Computers Electrical Engineering*, vol. 35, no. 3, pp. 517 – 526, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790609000020 24, 25

[109] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882 24, 25

[110] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," ser. CCS '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 251–261. [Online]. Available: https://doi.org/10.1145/948109.948144 24

[111] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997. 25

[112] Z. Dezhen and Y. Kai, "Genetic algorithm based optimization for adaboost," in *2008 International Conference on Computer Science and Software Engineering*, vol. 1, 2008, pp. 1044–1047. 25

[113] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. 25

[114] Zhu Xiaoliang, Yan Hongcan, Wang Jian, and Wu Shangzhuo, "Research and application of the improved algorithm c4.5 on decision tree," in *2009 International Conference on Test and Measurement*, vol. 2, 2009, pp. 184–187. 25

[115] Dae-Ki Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005, pp. 118–125. 26, 59

[116] M. Abdlhamed, K. Kifayat, Q. Shi, and W. Hurst, "A system for intrusion prediction in cloud computing," in *Proceedings of the International Conference on Internet of Things and Cloud Computing*, ser. ICC '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2896387.2896420 26

[117] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*, 1999, pp. 133–145. 26

[118] M. A. Jabbar, R. Aluvalu, and S. S. S. Reddy, "Cluster based ensemble classification for intrusion detection system," in *Proceedings of the 9th International Conference on Machine Learning and Computing*, ser. ICMLC 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 253–257. [Online]. Available: https://doi.org/10.1145/3055635.3056595 26

[119] Y. Liao and V. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Comput. Secur.*, vol. 21, pp. 439–448, 2002. 26

[120] O. Minawi, J. Whelan, A. Almehmadi, and K. El-Khatib, "Machine learning-based intrusion detection system for controller area networks," in *Proceedings of the 10th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, ser. DIVANet '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 41–47. [Online]. Available: https://doi.org/10.1145/3416014.3424581 26

[121] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, pp. 11 994–12 000, 12 2009. 26

[122] Tin Kam Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1. 26

[123] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001. 26

[124] G. Poojitha, K. N. Kumar, and P. J. Reddy, "Intrusion detection using artificial neural network," in *2010 Second International conference on Computing, Communication and Networking Technologies*, 2010, pp. 1–7. 26

[125] C. Modi, D. Patel, H. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013. 26

[126] A. Govada, V. S. Thomas, I. Samal, and S. K. Sahay, "Distributed multi-class rule based classification using ripper," in *2016 IEEE International Conference on Computer and Information Technology (CIT)*, 2016, pp. 303–309. 26

[127] P.-N. Tan, M. Steinback, and V. Kumar, *Introduction to Data Mining*, 01 2006. 26

[128] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 120–128. 26, 59

[129] L. Khan, M. Awad, and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering," *The VLDB Journal*, vol. 16, no. 4, p. 507–521, Oct. 2007. [Online]. Available: https://doi.org/10.1007/s00778-006-0002-5 27

[130] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of svm and ann for intrusion detection," *Computers Operations Research*, vol. 32, pp. 2617–2634, 10 2005. 27

[131] R. Bace and P. Mell, "Special publication on intrusion detection systems," in *NIST Pubs 800-31*. National Institute of Standards and Technology (NIST). U.S. Department of Commerce., 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-31.pdf 27, 28, 29

[132] S. A. Maske and T. J. Parvat, "Advanced anomaly intrusion detection technique for host based system using system call patterns," in *2016 International Conference on Inventive Computation Technologies (ICICT)*, vol. 2, 2016, pp. 1–4. 27

[133] O. Al-Jarrah and A. Arafat, "Network intrusion detection system using attack behavior classification," in *2014 5th International Conference on Information and Communication Systems (ICICS)*, 2014, pp. 1–6. 27

[134] M. Kumar and A. K. Singh, "Distributed intrusion detection system using blockchain and cloud computing infrastructure," in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, 2020, pp. 248–252. 27, 28

[135] I. Dutt, S. Borah, and I. K. Maitra, "Immune system based intrusion detection system (is-ids): A proposed model," *IEEE Access*, vol. 8, pp. 34 929–34 941, 2020. 28

[136] N. A. Premathilaka, A. C. Aponso, and N. Krishnarajah, "Review on state of art intrusion detection systems designed for the cloud computing paradigm," in *2013 47th International Carnahan Conference on Security Technology (ICCST)*, 2013, pp. 1–6. 30

[137] H. B. Baraka and H. Tianfield, "Intrusion detection system for cloud environment," in *Proceedings of the 7th International Conference on Security of Information and Networks*, ser. SIN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 399–404. [Online]. Available: https://doi.org/10.1145/2659651.2659682 30

[138] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of network and computer applications*, vol. 36, no. 1, pp. 42–57, 2013. 30

[139] A. Milenkoski, B. Payne, N. Antunes, M. Vieira, S. Kounev, A. Avritzer, and M. Luft, "Evaluation of intrusion detection systems in virtualized environments using attack injection," vol. 9404, 10 2015. 30, 31, 32, 33

[140] T. T. team. (1988) Tcpdump/libpcap public repository. Accessed: 22.01.2021. [Online]. Available: https://www.tcpdump.org/ 30

[141] S. by AppNeta. (2013) Tcpreplay - pcap editing and replaying utilities. Accessed: 22.01.2021. [Online]. Available: https://tcpreplay.appneta.com/ 30

[142] W. Norcott. (2007) Iozone. Accessed: 22.01.2021. [Online]. Available: http://www.iozone.org/ 30

[143] Oracle. (1995) mysqlslap — a load emulation client. Accessed: 22.01.2021. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html 30

[144] R. LLC. (2020) The world's most used penetration testing framework knowled. Accessed: 22.01.2021. [Online]. Available: https://www.metasploit.com/ 31

[145] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote sensing of Environment*, vol. 62, no. 1, pp. 77–89, 1997. 32

[146] N. Antunes and M. Vieira, "On the metrics for benchmarking vulnerability detection tools," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 505–516. 32, 33

[147] A. Halimaa A. and K. Sundarakantham, "Machine learning based intrusion detection system," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2019, pp. 916–920. 34, 35

[148] Xiuquan Li and Tao Zhang, "An exploration on artificial intelligence application: From security, privacy and ethic perspective," in *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2017, pp. 416–420. 34

[149] Wikipédia. Aprendizado de máquina year =2020,. Accessed: 22.02.2021. [Online]. Available: https://pt.wikipedia.org/wiki/Aprendizado_de_máquina 34, 35

[150] A. Phadke, M. Kulkarni, P. Bhawalkar, and R. Bhattad, "A review of machine learning methodologies for network intrusion detection," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, 2019, pp. 272–275. 35

[151] K. A. Torkura, M. I. H. Sukmana, A. V. D. M. Kayem, F. Cheng, and C. Meinel, "A cyber risk based moving target defense mechanism for microservice architectures," in *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 2018, pp. 932–939. 36

[152] V. Stavridou, B. Dutertre, R. A. Riemenschneider, and H. Saidi, "Intrusion tolerant software architectures," in *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, vol. 2, 2001, pp. 230–241 vol.2. 36

[153] D. Huang, H. Cui, S. Wen, and C. Huang, “Security analysis and threats detection techniques on docker container,” in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019, pp. 1214–1220. 37

[154] T. Combe, A. Martin, and R. Di Pietro, “To docker or not to docker: A security perspective,” *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016. 39

[155] M. Rajagopalan, M. A. Hiltunen, T. Jim, and R. D. Schlichting, “System call monitoring using authenticated system calls,” *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 216–229, 2006. 41

[156] A. S. Francois Raab, Walt Kohler. (1992, Mar.) Overview of the tpc-c benchmark. the order-entry benchmark. [Online]. Available: http://www.tpc.org/tpcc/detail5.asp 46

[157] C. Levine. Sigmod ’97 industrial session 5. Accessed: 22.02.2021. [Online]. Available: http://www.tpc.org/information/sessions/sigmod/ 46

[158] Percona-Lab_tpcc-mysql. (2017). [Online]. Available: https://github.com/Percona-Lab/tpcc-mysql 47

[159] Mysql - authentication bypass Available: https://www.exploit-db.com/exploits/19092 52

[160] Mysql / mariadb - geometry query denial of service 22.02.2021. [Online]. Available: https://www.exploit-db.com/exploits/38392 52

[161] Mysql / mariadb / perconadb 5.5.x/5.6.x/5.7.x - ’mysql’ system user privilege escalation / race condition Available: https://www.exploit-db.com/exploits/40678 52

[162] Mysql - integer overflow https://www.exploit-db.com/exploits/41954 52

[163] S. R. Guruvayur and R. Suchithra, “A detailed study on machine learning techniques for data mining,” in *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, 2017, pp. 1187–1192. 54

[164] psrecord. (2021, Mar.) Record the cpu and memory activity of a process. [Online]. Available: https://github.com/astrofrog/psrecord 73