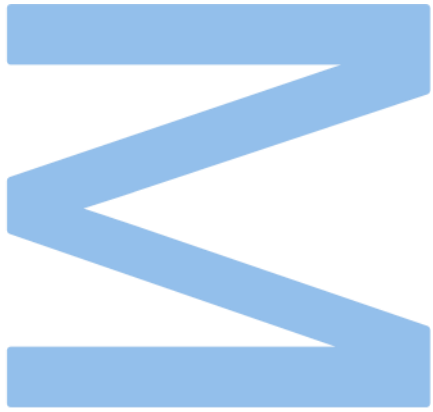
Sistemas de Deteção de Intrusão utilizando *Machine Learning*



Mara Quintas Almeida
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2022

Orientador
João Paulo da Conceição Soares, Professor Auxiliar Convidado,
Faculdade de Ciências da Universidade do Porto

Coorientador
Rolando da Silva Martins, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto






U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Declaração de Honra

Eu, Mara Quintas Almeida, inscrito(a) no Mestrado de Engenharia de Redes e Sistemas Informáticos da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Mara Quintas Almeida

Porto, 30 de Setembro de 2022

Abstract

Of the vast array of intrusion detection systems today, most are based primarily on signature detection techniques. These systems, also known as misuse detection, use acquired knowledge to detect attacks on new data. However, they are not able to detect new attacks, and a great deal of control is required over the signatures in the database, which need to be constantly updated.

With the main goal of increasing the range of detectable attacks of an existing system, JBriareos, we propose a new approach for intrusion detection, capable of detecting unknown attacks by monitoring the host. The additional component to this system, is a machine learnig model, which analyzes anomalies in the data, improving the performance of the system thus allowing to deal with 'zero-day' exploits.

For the detection of 'zero-day' attacks on modern systems, several host based intrusion detection technologies are proposed. These techniques use system calls to detect anomalies, but generally suffer from high computational cost, low detection rates, and low system modularization, which ultimately makes the usefulness of the system unfeasible. None of the systems studied presents a modular architecture that allows the intrusion detection system to be adapted to different contexts. In order to mitigate this set of problems, and to increase the range of detectable attacks in the existing system, a machine learning model for anomaly detection is coupled. Also, when combined with the modularity of our approach, it also the integration of our intrusion detection system in contexts different from those originally intended.

In the implemented machine learning model, two different approaches are used. A sequence-based approach and a frequency-based approach, each with different feature extraction processes. For the frequency approach simpler algorithms used, and a comparison of results between them is made, and for the sequence approach a deep learning model is used, due to the nature of the input data.
A major question in the study of intrusion detection using system calls is what is the optimal window value, which translates into the number of system calls required for the detection of an attack.
In this work, tests were performed with different window sizes, i.e., different sizes of system call sequences, to study the impact this has on the intrusion detection rate.

Resumo

Da vasta gama de sistemas de deteção de intrusão atuais, a grande maioria baseia-se principalmente em técnicas de deteção através de assinaturas. Estes sistemas, também conhecidos por deteção de uso indevido, utilizam conhecimento adquirido para detetar ataques. Em contrapartida, não são capazes de detetar novos ataques, e é necessário um grande controlo a nível das assinaturas que constam na base de dados, sendo necessária a atualização constante das mesmas.

Com o principal objetivo de aumentar a gama de ataques detetáveis de um sistema já existente, o JBriareos, propomos uma nova abordagem para deteção de intrusões, capaz de detetar ataques desconhecidos através da monitorização do *host*. O componente a adicionar a este sistema, trata-se de um modelo de *machine learnig*, que analisa anomalias nos dados, melhorando o desempenho do sistema permitindo assim lidar com *exploits* de 'dia zero'.

Para a deteção de ataques de 'dia zero' em sistemas modernos, são propostas várias tecnologias de deteção de intrusão baseados em *host*, utilizando o conjunto de dados ADFA-LD. Estas técnicas utilizam chamadas de sistema para detetar anomalias, mas geralmente sofrem de alto custo computacional, baixa deteção e baixa modularização do sistema, que acaba por inviabilizar a utilidade do sistema. Nenhum dos sistemas estudados apresenta uma arquitetura modular que permita a adaptação do sistema de deteção de intrusão a diferentes contextos. De forma a mitigar este conjunto de problemas, e a aumentar a gama de ataques detetáveis no sistema já existente, é acoplado um modelo de *machine learning* para deteção de anomalias, sendo também possível a integração deste sistema de deteção de intrusões em contextos diferentes dos originalmente destinados.

No modelo de *machine learning* implementado são utilizadas duas abordagens diferentes ao conjunto de dados ADFA-LD. A abordagem baseada em sequência e a abordagem baseada em frequência, cada uma com processos de extração de caraterísticas diferente. Para a abordagem de frequências são utilizados algoritmos mais simples, e é feita a comparação de resultados entre eles e para a abordagem de sequências é utilizado um modelo de *deep learning*, devido à natureza dos dados de *input*.
Uma grande questão no estudo de deteção de intrusões utilizando chamadas de sistema é qual o valor ideal da janela, que se traduz no número de chamadas de sistema necessárias para a deteção de um ataque.
Neste trabalho, foram realizados testes com diferentes tamanhos de janela, ou seja, diferentes

tamanhos de sequências de chamadas de sistema, para estudar o impacto que esta tem na taxa de deteção de intrusões.

Agradecimentos

Gostaria de agradecer em primeiro lugar aos meus orientadores de tese, João Soares e Rolando Martins, por todo o apoio, ajuda e sugestões. Gostaria também de agradecer a todos os meus amigos e colegas pelo companheirismo, inter-ajuda e apoio.
Estou também muito grata à minha família, especialmente aos meus pais, por me proporcionarem a oportunidade de formação, pelo apoio incondicional e motivação durante todos estes anos.

Conteúdo

Abstract i

Resumo iii

Agradecimentos v

Conteúdo ix

Lista de Tabelas xii

Lista de Figuras xv

Acrónimos xvii

1 Introdução 1

2 Estado da Arte 3

2.1 Sistemas de Deteção de Intrusão 3

2.1.1 Técnicas de Deteção de Intrusão 5

2.1.1.1 Assinaturas 5

2.1.1.2 Anomalias 5

2.1.2 Classificação dos Sistemas de Deteção de Intrusão 6

2.1.2.1 Sistema de Deteção de Intrusão de Rede 7

2.1.2.2 Sistema de Deteção de Intrusão de *Host* 8

2.1.2.3 Sistema de Deteção de Intrusão Híbrido 9

2.1.3 Snort . . . 9
2.1.4 Suricata . . . 10
2.1.5 OSSEC . . . 12
2.1.6 *TripeWire Open Source* . . . 13
2.1.7 Modelo Híbrido . . . 14
2.1.8 NIDS vs HIDS . . . 15
2.2 *Machine Learning* . . . 16
2.2.1 *Machine Learning* no contexto de IDS . . . 17
2.2.1.1 Coleção de Dados e Seleção de Caraterísticas . . . 18
2.2.1.2 Conjuntos de dados IDS . . . 21
2.2.1.3 Algoritmos . . . 22
2.2.1.4 *Behavior-Based Access Control* . . . 26
2.2.1.5 Modelo HIDS com CNN . . . 28
2.2.1.6 Filtro de Alarme Inteligente . . . 30
2.3 Discussão . . . 32

3 JBriareos **35**
3.1 Visão Geral . . . 35
3.2 Arquitetura . . . 35
3.3 Adição de Componente . . . 37

4 Desenho e Arquitetura **39**
4.1 Descrição do Conjunto de Dados . . . 39
4.2 Transformações ao Conjunto de Dados . . . 41
4.2.1 Abordagem por Sequência . . . 42
4.2.2 Abordagem por Frequência . . . 44
4.3 Modelo Geral . . . 45

5 Implementação dos Modelos e Resultados **47**

5.1 Sequências 49
5.2 Frequências 52
5.3 Comparação dos Resultados 61

6 Conclusão **65**

6.1 Trabalho Futuro 66

Bibliografia **67**

Lista de Tabelas

2.1 *Tradeoffs* entre SIDS e AIDS 6

2.2 *Tradeoffs* entre Sistema de Deteção de Intrusão baseado em Rede (*Network-based Intrusion Detection System*) (NIDS) e Sistema de Deteção de Intrusão baseado em *Host* (*Host-based Intrusion Detection System*) (HIDS) 16

2.3 *Tradeoffs* NIDS face a HIDS 16

2.4 Descrição das Caraterísticas dos Dados NSL-KDD 22

2.5 Comparação das Ferramentas 1 33

2.6 Comparação das Ferramentas 2 33

3.1 Comparação entre as diferentes versões Briareos 37

4.1 Descrição Geral dos Dados ADFA-LD 40

4.2 Classe dos Ataques ADFA-LD/WD 41

5.1 *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo Memória de Curto Prazo Longa (*Long Short-Term Memory*) (LSTM) 51

5.2 Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo LSTM 52

5.3 *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo K-Vizinhos mais Próximos (*K-Nearest Neighbors*) (KNN) 54

5.4 Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo KNN 54

5.5 *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo *Naïve Bayes* 56

5.6 Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo *Naïve Bayes* 56

5.7 *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo Floresta Aleatória (Random Forest) (RF) 58

5.8 Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo RF 58

5.9 *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo Máquina de Vetor de Suporte (*Support-Vector Machine*) (SVM) 60

5.10 Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo SVM 60

5.11 Tempo de Execução do Pré-Processamento de ambas as Abordagens (Dados de Validação W=7) 61

Lista de Figuras

2.1 Arquitetura Funcional de um Sistema de Deteção de Intrusão (*Intrusion Detection System*) (IDS) 4

2.2 Arquitetura Funcional de um IDS Híbrido 9

2.3 Arquitetura do Snort 10

2.4 Arquitetura Suricata de alto nível 12

2.5 Arquitetura OSSEC 13

2.6 Arquitetura *TripeWire* 14

2.7 Arquitetura Híbrida Proposta 15

2.8 Processo de Seleção de Caraterísticas 19

2.9 Taxonomia dos Algoritmos de Seleção de Caraterísticas 21

2.10 Ilustração do Funcionamento do Algoritmo *Naïve Bayes* 23

2.11 Ilustração do Funcionamento do Algoritmo K-Vizinhos mais Próximos (*K-Nearest Neighbors*) (KNN) 24

2.12 Ilustração do Funcionamento de uma Árvore de Decisão 25

2.13 Ilustração do Funcionamento do Algoritmo Máquina de Vetor de Suporte (*Support-Vector Machine*) (SVM) 25

2.14 Ilustração do Funcionamento do Algoritmo Recurrent Neural Network (*Rede Neural Recorrente*) (RNN) 26

2.15 Arquitetura do Modelo BBAC 28

2.16 Arquitetura CNN 30

2.17 Arquitetura de Alto Nível do Modelo de Filtro de Alarme Inteligente baseado em KAV 31

3.1 Arquitetura JBriareos 36

4.1 Representação de uma sub-sequência do ADFA-LD original 42

4.2 Transformação aos Dados Originais por Sequência 43

4.3 Transformação aos Dados Originais por Frequência 45

4.4 Modelo Geral do Sistema de Deteção de Intrusão baseado em *Host* (*Host-based Intrusion Detection System*) (HIDS) Proposto 46

5.1 Curvas Curva Caraterística de Operação do Receptor (*Receiver Operating Characteristic Curve*) (ROC) para Janela de 2 e 3 49

5.2 Curvas ROC para Janela de 4 e 5 50

5.3 Curvas ROC para Janela de 6 e 7 50

5.4 Curva ROC para Janela de 10 (W=10) 51

5.5 Curvas ROC para Janela de 2 e 3 - KNN 53

5.6 Curvas ROC para Janela de 4 e 5 - KNN 53

5.7 Curvas ROC para Janela de 6 e 7 - KNN 54

5.8 Curva ROC para Janela de 10 (W=10) - KNN 54

5.9 Curvas ROC para Janela de 2 e 3 -*Naïve Bayes* 55

5.10 Curvas ROC para Janela de 4 e 5 - *Naïve Bayes* 55

5.11 Curvas ROC para Janela de 6 e 7 - *Naïve Bayes* 56

5.12 Curva ROC para Janela de 10 (W=10) -*Naïve Bayes* 56

5.13 Curvas ROC para Janela de 2 e 3 -Floresta Aleatória (Random Forest) (RF) . . 57

5.14 Curvas ROC para Janela de 4 e 5 - RF 57

5.15 Curvas ROC para Janela de 6 e 7 - RF 58

5.16 Curva ROC para Janela de 10 (W=10) - RF 58

5.17 Curvas ROC para Janela de 2 e 3 -SVM 59

5.18 Curvas ROC para Janela de 4 e 5 - SVM 59

5.19 Curvas ROC para Janela de 6 e 7 - SVM 60

5.20 Curva ROC para Janela de 10 (W=10) - SVM 60

5.21 Curva de ROC dos Testes dos Algoritmos com Melhores Resultados 62

5.22 Curva de ROC Média dos Testes dos Algoritmos com Melhores Resultados 63

Acrónimos

AIDS Sistema de Deteção de Intrusão baseado em Anomalia (*Anomaly-based Intrusion Detection System*)

AUC Área sob a Curva (*Area Under the Curve*)

BBAC Controlo de Acesso baseado no Comportamento *(Behavior-Based Access Control*)

CNN Rede Neural Convolucional *(Convolutional Neural Network*)

CSRF Falsificação de Pedidos em Locais Cruzados *(Cross-Site Request Forgery)*

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

DOS Negação de Serviço (*Denied of Service*)

DT Árvore de Decisão (*Decision Tree*)

FTP Protocolo de Transferência de Ficheiros (*File Transfer Protocol*)

HIDS Sistema de Deteção de Intrusão baseado em *Host* (*Host-based Intrusion Detection System*)

HTTP Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)

IDPS Sistema de Deteção e Prevenção de Intrusão (*Intrusion Detection and Prevention System*)

IDS Sistema de Deteção de Intrusão (*Intrusion Detection System*)

IP Protocolo Internet (*Internet Protocol*)

IPS Sistema de Prevenção de Intrusão (*Intrusion Prevention System*)

JDS Sistema Distribuído JBriareos (JBriareos *Distributed System*)

JHC Componente de *host* do JBriareos (JBriareos *Host Component*)

JMS Servidor Gestor JBriareos (JBriareos *Manager Server*)

KNN K-Vizinhos mais Próximos (*K-Nearest Neighbors*)

LSTM Memória de Curto Prazo Longa (*Long Short-Term Memory*)

ML Aprendizagem Máquina (*Machine Learning*)

MLP *Perceptron* Multicamadas (*Multilayer Perceptron*)

NIDS Sistema de Deteção de Intrusão baseado em Rede (*Network-based Intrusion Detection System*)

NSM Monitorização da Segurança da Rede (*Network Security Monitoring*)

OCSP Profissional Certificado de Segurança Ofensiva (*Offensive Security Certified Professional*)

OISF *Open Information Security Foundation*)

OSSEC *Open Source HIDS SECurity*

RF Floresta Aleatória (Random Forest)

RNN Recurrent Neural Network (*Rede Neural Recorrente*)

ROC Curva Caraterística de Operação do Receptor (*Receiver Operating Characteristic Curve*)

R2L *Remote to Local*

SGDM Descida Estocástica Gradiente com Momento (*Stochastic Gradient Descent with Momentum*)

SIDS Sistema de Deteção de Intrusão baseado em Assinatura (*Signature-based Intrusion Detection System*)

SMTP Protocolo simples de transferência de correio (*Simple Mail Transfer Protocol*)

SSH *Shell* Segura (*Secure Shell*)

SSL Camada de *Sockets* Seguras (*Secure Sockets Layer*)

SVM Máquina de Vetor de Suporte (*Support-Vector Machine*)

TCP Protocolo de Controlo de Transmissão (*Transmission Control Protocol*)

UDP Protocolo de Datagramas do Utilizador (*User Datagram Protocol*)

U2R Utilizador para a Raiz (*User to Root*)

XMPP Protocolo de Mensagens Extensíveis e Presença (*Extensible Messaging and Presence Protocol*)

XSS *Script* entre sites (*Cross-Site Scripting*)

Capítulo 1

Introdução

De acordo com a evolução do recurso, tanto da Internet como de computadores à qual assistimos, também aumenta a atividade e a habilidade dos agentes mal-intencionados.
A utilização de *firewalls* é uma boa forma de proteger os serviços online contra ataques. No entanto, confiar numa única linha de defesa não é uma boa prática.
Deteção de Intrusões, seja por software ou hardware, permite detetar e impedir possíveis comportamentos anómalos, incluindo tentativas de intrusão por parte de agentes mal-intencionados, de executar com sucesso. Para tal, tipicamente estes monitorizam interações com os sistemas, inspecionando pacotes de rede de entrada e saída. No entanto, o aumento de conhecimento e a habilidade de agentes mal-intencionados obriga a melhorar e fortalecer estes sistemas, de forma a lidar com os crescentes níveis de segurança exigidos.

De forma a mitigar ataques, principalmente desconhecidos, neste projeto pretendemos melhorar um sistema de Sistema de Deteção de Intrusão (*Intrusion Detection System*) (IDS) existente (JBriareos), acoplando um modelo de Aprendizagem Máquina (*Machine Learning*) (ML) para melhorar o seu desempenho, permitindo assim lidar com *exploits* de 'dia zero'.
Os modelos de ML permitem melhorar a eficácia de deteção de anomalias dos IDS's. No entanto, apenas funcionam para os contextos em que foram treinados.
Este trabalho, é focado no desenvolvimento de modelos cujo objetivo é detetar ataques a partir de chamadas de sistema. O conjunto de dados utilizado foi o ADFA-LD, composto por sequências de vários tipos de ataques, com padrões de ataque diferentes, e por sequências de atividade inofensiva. Para a criação do modelo propriamente dito, os dados foram analisados de forma aprofundada, juntamente com a literatura acerca dos mesmos, para chegarmos a uma conclusão de quais seriam as transformações ideais dos dados para o melhor aproveitamento da informação disponível, que impacto tem a escolha do valor da janela para a análise das chamadas de sistema, e que vantagens e desvantagens traz essa escolha.

Assim, são utilizadas duas abordagens completamente diferentes. Uma abordagem analisa a disposição das chamadas de sistema, tendo em conta a ordem em que acontecem e que chamadas de sistema são realizadas, numa janela regulada pelo utilizador. A abordagem seguinte analisa a frequência que determinadas chamadas de sistema constam numa determinada janela, regulada

também pelo utilizador.
Estas duas técnicas utilizam os dados originais transformados como *input*, possuindo um método de extração de caraterísticas diferente. Os algoritmos utilizados para criação do modelo de deteção, são também distintos. Na técnica baseada na sequência das chamadas de sistema é utilizado um modelo de *deep learning*, pois como é necessário o processamento das chamadas de sistema em si, tendo em conta a sua ordem e o posicionamento da mesmas, é necessário um algoritmo mais complexo. No caso da técnica baseada nas frequências de chamadas de sistema, foram utilizados algoritmos mais simples, como o Máquina de Vetor de Suporte (*Support-Vector Machine*) (SVM), K-Vizinhos mais Próximos (*K-Nearest Neighbors*) (KNN), e Floresta Aleatória (Random Forest) (RF).
A utilização de cada uma das abordagens traz vantagens e desvantagens, e devem ser consideradas de acordo com a função que queremos que o sistema desempenhe.

No Capítulo 2 é feita uma contextualização dos sistemas de deteção de intrusões, das diferentes técnicas que podem ser utilizadas, e a classificação destes sistemas em relação ao ambiente onde atuam. Ainda neste Capítulo são abordados alguns exemplos de IDS que já existem, alguns dos quais já com a utilização de algoritmos de *machine learning*. O Capítulo 3 é uma *overview* do *background* deste projeto, ou seja, a arquitetura do JBriareos, e a adição do componente que irá trazer-lhe vantagens no que diz respeito à gama de ataques detetáveis. Os dois Capítulos seguintes dizem respeito ao desenho e arquitetura do modelo implementado, bem como os algoritmos utilizados e a criação do modelo de deteção. De seguida, no Capítulo 6 são apresentados os resultados dos diferentes modelos e no Capítulo 7 a conclusão e respetivo trabalho futuro.

Capítulo 2

Estado da Arte

2.1 Sistemas de Deteção de Intrusão

Os Sistemas de Deteção de Intrusão começaram a ser estudados em 1980, tendo sido primitivamente referidos num artigo seminal de J.P. *Anderson* [1]. No final da década de 80 foram propostos métodos realistas de certos tipos de sistemas de deteção de intrusões, e em 1990 com o crescimento explosivo da Internet, ganharam mais relevância. Atualmente são sistemas bastante sofisticados, tanto a nível comercial como público.

Um IDS é um conjunto de componentes, de *software* ou *hardware* projetado para detetar e prevenir qualquer atividade maliciosa ou ataque contra o sistema ou rede, com o principal objetivo de proteger o sistema em questão [2]. Esta tecnologia monitoriza o tráfego e as interações do sistema a diferentes níveis de abstração, utilizando uma camada de defesa que protege a rede e os *hosts*. A principal diferença em relação a um Sistema de Deteção e Prevenção de Intrusão (*Intrusion Detection and Prevention System*) (IDPS) é que este tem a capacidade de detetar o ataque e impedir que o ataque chegue ao destino, enquanto a base de um IDS apenas deteta atividade maliciosa e gera alertas.

A arquitetura funcional de um IDS, apresentada na Figura 2.1, é composta por quatro componentes principais: componente de captura de eventos (E *box*), tal como o nome indica, interceta as interações do sistema capturando eventos. O evento capturado é enviado para o componente de análise (A *box*), que é responsável por analisar a informação recolhida e verificar se existe algum vetor de ataque ou anomalia. Caso seja detetada uma ação maliciosa, o componente de contra medida (C *box*) é responsável pela reação ao ataque detetado (por exemplo, geração de alarme; ações de proteção, como reconfiguração de *firewalls*; ações de sobrevivência, como isolamento do sistema atacado e ativação dos alternativos). Todos os eventos recolhidos por E *box* ou produzidos por A *box* são guardados em D *box* [3].

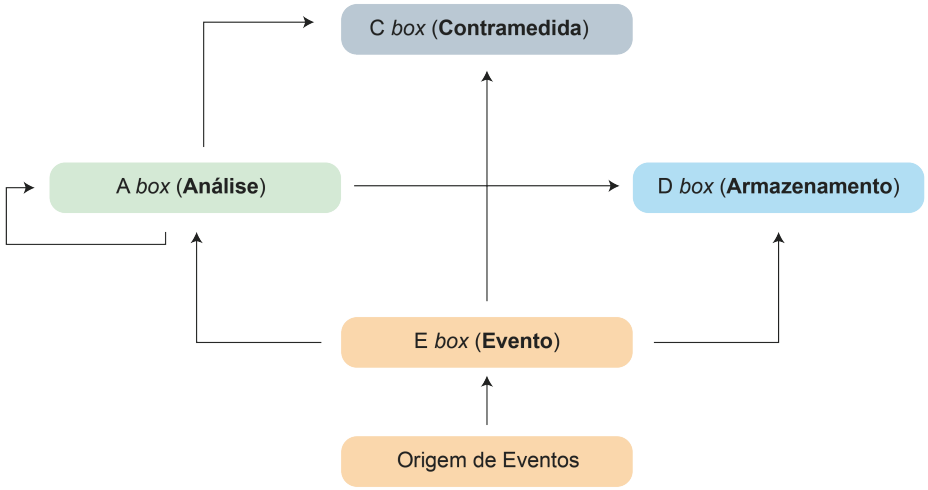


Figura 2.1: Arquitetura Funcional de um IDS

A utilização de um IDS é complexa por diversas razões, das quais a mais comum é a ocorrência de erros na deteção. No caso de ser detetada erradamente uma intrusão, diz-se que ocorre um falso positivo, pelo contrário, quando o IDS não deteta uma intrusão, diz-se que ocorreu um falso negativo. Estas taxas podem comprometer um IDS, pois muitos falsos positivos desacreditam o sistema, e muitos falsos negativos tornam o IDS desnecessário e enganador. Portanto é crucial que estas duas taxas tenham o menor valor possível, e que haja um equilíbrio entre elas. Um dos fatores que influencia o valor das taxas de falsos positivos e falsos negativos é a técnica que é usada na deteção.

Consoante a função que o IDS desempenha, existem diferentes técnicas de deteção de intrusão: assinatura e anomalia.
A deteção baseada em assinatura baseia-se em técnicas de correspondência de padrões para encontrar um ataque conhecido, e a deteção baseada em anomalia identifica eventos que levantam suspeitas por diferirem significativamente da maioria dos dados, ou seja, deteta desvios à normalidade no comportamento do sistema computacional sob análise.

Vários sistemas de deteção de intrusões têm sido propostos refletindo as necessidades relativamente a ameaças e intrusões. Estes sistemas distinguem-se pela sua arquitetura, abordagem tomada, podendo ser caraterizados por *Host-based* (i.e., executam apenas nos hosts) ou *Network-based* (i.e., executam nos vários dispositivos de redes, ex: *routers*), e mecanismos de deteção/prevenção de anomalias, como por exemplo, a utilização de modelos de *machine learning*.

2.1.1 Técnicas de Deteção de Intrusão

Os sistemas de deteção de intrusão podem ser classificados com base na técnica de deteção que é utilizada para o reconhecimento das ações maliciosas. As técnicas principais de IDS são a assinatura, também conhecido por deteção de uso indevido, que funciona consoante o conhecimento, e a anomalia que é baseado em comportamento.

2.1.1.1 Assinaturas

Um Sistema de Deteção de Intrusão baseado em Assinatura (*Signature-based Intrusion Detection System*) (SIDS), baseia-se em técnicas de correspondência de padrões para encontrar um ataque conhecido, através de regras já existentes na base de dados. Estes sistemas são também conhecidos como deteção com base em conhecimento ou deteção de uso indevido.

No SIDS, são utilizados métodos de correspondência para encontrar uma intrusão anterior, ou seja, quando uma assinatura de intrusão coincide com a assinatura de uma intrusão já registada e existente na base de dados de assinaturas, é acionado um sinal de alarme.
Para este método de deteção, os registos do *host* são inspecionados para encontrar sequências de comandos, *syscalls* ou ações que tenham sido previamente identificadas como *malware* [4].

A ideia principal é construir uma base de dados de assinaturas intrusas e comparar o atual conjunto de atividades com as assinaturas existentes e levantar um alarme se for encontrada uma correspondência. No caso de ser encontrada uma nova assinatura que não faz parte das regras existentes na base de dados atual, é adicionada essa nova assinatura/regra aos dados.
Uma assinatura que pode indicar um SYN *flooding attack* é, por exemplo, a ausência excessiva de segmentos ACK.

2.1.1.2 Anomalias

Um Sistema de Deteção de Intrusão baseado em Anomalia (*Anomaly-based Intrusion Detection System*) (AIDS) funciona baseado no comportamento procurando atividades invulgares que se desviem das médias estatísticas de atividades anteriormente vistas. Ou seja, as ações captadas são comparadas com a base de conhecimento de comportamento adquirido até ao momento, e caso se desvie do perfil conhecido como normal, então é motivo de alerta. Por exemplo, se um utilizador entrar sempre na rede a partir da Califórnia e aceder a ficheiros de engenharia, se o mesmo utilizador entrar a partir de Pequim e olhar para ficheiros .hr, isto é um motivo para alarme [4].

O modelo de comportamento normal e anormal pode ser criado através de algoritmos de *machine learning*, utilizando dados de treino. Qualquer desvio significativo entre o comportamento observado e o modelo é considerado uma anomalia, o que pode ser interpretado como uma intrusão. O pressuposto para esta técnica é que o comportamento malicioso difere do comportamento

típico do utilizador. Os comportamentos de utilizadores anómalos que são diferentes dos comportamentos padrão são classificados como intrusões.

O uso de cada uma das técnicas traz vantagens e desvantagens, que devem ser consideradas na implementação de um sistema de deteção de intrusão de acordo com a sua função, do local de implantação, do tipo de dados de *input*, etc.
A deteção baseada em assinatura é mais simples e possui uma taxa de falsos positivos baixa, visto que a atividade em análise é comparada com as regras da base de dados já conhecidas, sendo mais difícil haver erros na previsão de atividade normal. Em contrapartida, esta técnica poderá ter uma taxa de falsos negativos mais elevada, visto que não é capaz de detetar novos ataques. Em comparação, a deteção de anomalia, geralmente possui uma taxa de falsos negativos mais baixa, pois deteta ataques desconhecidos, no entanto a sua implementação é mais complexa. O sistema ideal seria a conjugação das duas técnicas, que detetasse novos ataques, e que possuísse uma taxa de falsos positivos baixa, no entanto o processo de deteção iria ser mais complexo e demorado.
Na Tabela 2.1 são apontados pontos fortes e pontos fracos do uso de cada uma das técnicas.

	IDS baseado em Assinatura	**IDS baseado em Anomalia**
Pontos Fortes	**i)** Simples e eficaz na deteção de ataques conhecidos; **ii)** Taxa de alarmes falsos baixa; **iii)** Funciona bem contra ataques com padrão comportamental fixo [2].	**i)** Deteta ataques desconhecidos, apontando novas vulnerabilidades; **ii)** Sem necessidade de conhecimento prévio das falhas de segurança; **iii)** Pode usar algoritmos de ML para treinar os dados; **iv)** Pode usar teste estatístico para identificar intrusão; **v)** Pode diminuir a taxa de alarmes falsos para ataques desconhecidos [2].
Pontos Fracos	**i)** Não deteta ataques desconhecidos; **ii)** Recolha de assinaturas deve ser continuamente atualizada; **iii)** Eficiência do sistema é determinada pela velocidade de criação das novas assinaturas [2].	**i)** Taxa alta de falsos negativos devido ao algoritmo de deteção amplamente treinado; **ii)** Menos eficaz em ambiente dinâmico; **iii)** Necessita de grandes conjuntos de treino para caraterizar padrões normais [2].

Tabela 2.1: *Tradeoffs* entre Assinaturas e Anomalias

2.1.2 Classificação dos Sistemas de Deteção de Intrusão

Na secção anterior categorizamos um IDS com base nos métodos utilizados para identificar intrusões. Um IDS também pode ser classificado com base na origem dos dados de *input* utilizados para detetar atividades anormais.
Em termos de origem de dados, existem três tipos de tecnologias IDS, nomeadamente IDS com

base no *host* - Sistema de Deteção de Intrusão baseado em *Host* (*Host-based Intrusion Detection System*) (HIDS), IDS com base na rede - Sistema de Deteção de Intrusão baseado em Rede (*Network-based Intrusion Detection System*) (NIDS) e uma abordagem híbrida, onde é combinada a deteção em *host* e em rede.

Um NIDS monitoriza e analisa o tráfego de um segmento de rede, sem ter acesso à informação dos *hosts*. Um HIDS inspeciona dados provenientes do sistema *host*, ou seja, é um agente que controla se a política de segurança do sistema foi derrubada, seja internamente ou externamente. Os NIDS concentram-se mais no abuso de vulnerabilidades enquanto os HIDS em torno do abuso de privilégios.

2.1.2.1 Sistema de Deteção de Intrusão de Rede

Um Sistema de Deteção de Intrusão baseado em Rede inspeciona e monitoriza um segmento de rede com o objetivo de registar e detetar possíveis ações maliciosas. A informação que é analisada por um NIDS pode ser baseada em pacotes, baseada em fluxo ou baseada em sessão. Dependendo da técnica que é usada para a deteção, assinatura ou anomalia, os dados são tratados de formas diferentes. Na deteção baseada em fluxo e baseada em sessão é usada uma abordagem maioritariamente de análise de anomalias, enquanto na abordagem que segue a análise de pacotes são analisadas as assinaturas.

Na deteção de pacotes, estes sistemas analisam o cabeçalho e os dados da carga útil, sendo capazes de bloquear intrusões através da identificação de assinaturas no tráfego e depois rejeitar ou permitir pacotes de acordo com as regras previamente criadas.

Na deteção de fluxo, os pacotes são agrupados por período de tempo, podendo ser aplicada uma estratégia de engenharia de caraterísticas ou modelos de *deep learning*, que fazem parte de ML, mas com algumas especificidades no seu funcionamento. No primeiro caso, são utilizados vetores de caraterísticas com o uso de algoritmos de *machine learning* pouco profundos (algoritmos mais simples que não fazem parte do grupo de modelos de *deep learning*). No segundo caso, os algoritmos de *deep learning* são usados para extrair caraterísticas dos dados e posteriormente os modelos de *machine learning* pouco profundos são utilizados para classificação.

A deteção a partir de dados de sessão, baseia-se num processo de interação entre dois terminais. Uma sessão normalmente é dividida num tuplo de cinco (Protocolo Internet (*Internet Protocol*) (IP) do cliente, porta do cliente, IP do servidor, porta do servidor e protocolo). Através desta informação são obtidas métricas importantes para análise, a partir de caraterísticas estatísticas, onde a informação estatística é utilizada para compor vetores de caraterísticas adequados para modelos de *machine learning* pouco profundos e caraterísticas de sequência. Na análise de sequências, existe uma ordem rigorosa de pacotes, e são obtidas informações detalhadas sobre a interação da sessão [5].

Normalmente o NIDS está implantado numa zona estratégica do segmento da rede de maneira a intercetar o máximo de informação para análise e posteriormente enviar para os nós da rede,

de maneira a antecipa-los evitando um ataque semelhante.

2.1.2.2 Sistema de Deteção de Intrusão de *Host*

Um Sistema de Deteção baseado em *Host* recolhe eventos da própria máquina, e monitoriza o estado dos vários componentes, seja *hardware*, sistema de ficheiros, servidores, atividade de aplicações, *logs* do sistema, servidores, etc. A recolha de eventos em máquinas HIDS também é capaz de monitorizar tráfego de rede, mas apenas na máquina em que o HIDS está implantado.

A deteção baseada em *logs* do sistema, incluem os registos de atividade do sistema operativo, dos programas de aplicação, chamadas de sistema, registos de alerta e registos de acesso. O tratamento desta informação pode ser feito de várias formas distintas, como: combinação de algoritmos de *machine learning* com regras, em que é usado por exemplo as regras para gerar alertas e ML para filtrar esses alertas, diminuindo a taxa de falsos positivos [6]. Outra técnica possível é o uso de engenharia de caraterísticas, que é extração de caraterísticas de acordo com o conhecimento do domínio, e posteriormente são identificados comportamentos anormais de acordo com as caraterísticas extraídas. Também é possível uma análise de texto, os *logs* são considerados texto simples, e são usadas palavras-chave para identificar anomalias [7].

Os HIDS analisam os ficheiros de registo de auditoria para identificar e detetar processos maliciosos do sistema. No entanto, analisar uma grande quantidade de dados para distinguir processos normais e processos maliciosos, é necessário muito tempo de computação e muitos recursos.

Diversas investigações introduziram métodos para resolver este problema, uma delas foi a de Marteau [8], introduziu uma nova medida de correspondência em dados sequenciais simbólicos para detetar ataques desconhecidos. No método proposto, o autor concentrou-se nas sequências de chamadas de sistema, utilizando o algoritmo de cobertura de sequências para a deteção de intrusão (SC4ID). O algoritmo SC4ID é baseado na cobertura ótima de uma sequência por uma série de sub-sequências extraídas de um conjunto pré-definido de sequências.

Subba et al. [9] introduziram um método para melhorar a eficiência do cálculo em HIDS. O método proposto transformou a chamada do sistema num vetor de dimensão n e depois reduziu a dimensão dos vetores das caraterísticas de entrada através de um processo de redução de dimensão. Os vetores de caraterísticas são posteriormente analisados por vários classificadores de machine learning (*Naive Bayes*, *Perceptron* Multicamadas (*Multilayer Perceptron*) (MLP), C4.5 *Decision Tree*, e SVM) para identificar ações maliciosas. Para avaliar o modelo proposto, foi utilizado o conjunto de dados ADFA-LD.

Na secção seguinte são abordados dois sistemas do tipo HIDS: OSSEC e *TripeWire Open Source*.

2.1.2.3 Sistema de Deteção de Intrusão Híbrido

Um Sistema de Deteção de Intrusão Híbrido combina dois tipos ou mais de IDS para alcançar vantagens e completar uma deteção tão precisa como Double Guard [10], que utiliza IDS baseado em *host* e IDS baseado em rede.
Com a combinação de uma deteção baseada em rede e deteção baseada em *host*, o nível de proteção do sistema a proteger aumenta consideravelmente. Não só fica protegido o *host*, como o segmento de rede em que este está ligado. No entanto, existem sempre aspetos negativos. Um dos aspetos negativos de um sistema híbrido é o tempo de análise dos dados.
A Figura 2.2 representa a arquitetura funcional de um IDS baseado em rede e baseado em *host*.

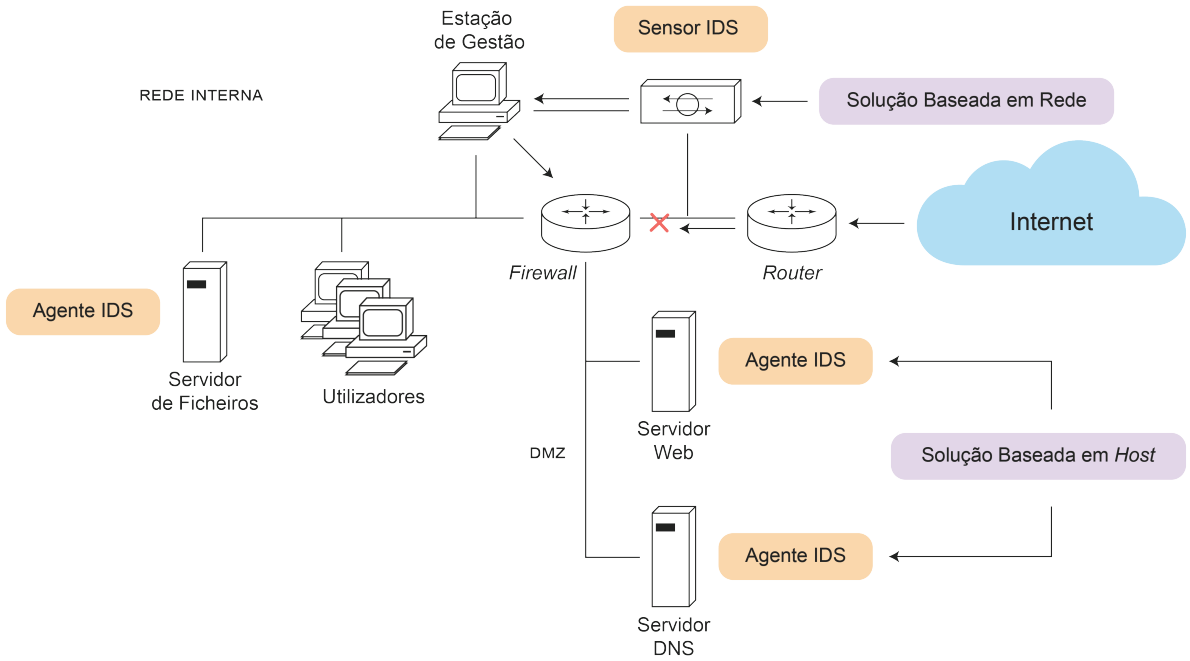


Figura 2.2: Arquitetura Funcional de um IDS Híbrido

2.1.3 Snort

Snort é um NIDS em C, de código aberto, criado em 1998 por *Martin Roesch* e adquirido pela Cisco em 2013. Este sistema é simples, flexível e intuitivo, capaz de realizar análises de tráfego em tempo real e registar pacotes em redes IP, utilizando um sistema baseado em regras, com o objetivo de evitar e prevenir intrusões [3][11].

O Snort fornece um serviço baseado em assinaturas no qual os clientes têm acesso a regras e filtros, melhorando a segurança oferecida por ele e tornando-o mais fácil de gerir, no entanto também permite ao utilizador criar as suas próprias regras [12].

A arquitetura do Snort, apresentada na Figura 2.3, contém quatro componentes principais: *sniffing*, pré-processamento de pacotes, deteção de intrusão e alerta e registo, e pode ser

configurado em vários modos de operação executando apenas certas tarefas. Pode simplesmente exibir no ecrã os pacotes capturados (modo *sniffer*), utilizar o modo *packet logger*, que regista os pacotes capturados no disco, ou realizar a deteção e análise do tráfego na rede, denominado por modo NIDS [13].
No entanto, quando funciona de maneira integral, começa por efetuar a captura dos pacotes, e descodifica-los para obter a informação que vai ser analisada. Depois de desfragmentar o pacote, vai comparar a informação recolhida com as regras que possui na sua base de dados. As regras podem desencadear ações, tais como alertar, registar, deixar passar e rejeitar o pacote analisado. Este sistema é um IDS por *default*, no entanto existe a possibilidade de cumprir ações de prevenção, acoplando o *NFQueues* ou o DAQ, que tomam medidas caso seja detetada uma ação maliciosa [14] [15].

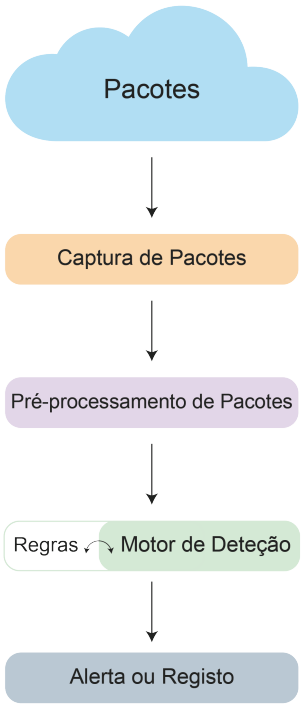


Figura 2.3: Arquitetura do Snort

2.1.4 Suricata

Suricata é um NIDS, implementado em C e Rust, de código aberto e foi lançado em 2010 pela *Open Information Security Foundation*) (OISF).
Este sistema combina deteção de intrusão (IDS), IPS, Monitorização da Segurança da Rede (*Network Security Monitoring*) (NSM) e processamento PCAP, com o objetivo de identificar e interromper ações maliciosas [16]. Para isso, utiliza métodos baseados em regras na inspeção dos pacotes, e pode utilizar as mesmas regras do Snort. No entanto, existem muitas outras funcionalidades como suporte de *scripts* lua, facilita a combinação de várias regras, o que melhora a capacidade de deteção de vetores de ataque mais complexos.

O Suricata é baseado principalmente no Snort, mas para além de ser mais moderno, possui maior escalabilidade, precisão e eficiência [17].

Com a utilização do Suricata, é possível analisar e interpretar tráfego na camada de aplicação, para extrair ficheiros HTTP, certificados Camada de *Sockets* Seguras (*Secure Sockets Layer*) (SSL) e solicitações do Sistema de Nomes de Domínio (*Domain Name System*) (DNS), aumentando a probabilidade de registar pacotes e extrair executáveis potencialmente maliciosos, após procurar por assinaturas conhecidas. Outras caraterísticas relevantes, são o suporte para IPv4 e IPv6, GeoIP e reputação IP [18].

O facto de também suportar *multithreading* é importante, pois aumenta o desempenho na inspeção dos pacotes e permite maior rendimento e desempenho principalmente em redes com elevado volume de tráfego [18].
O Suricata utiliza como técnica de deteção um método baseado em assinaturas, por isso, detetar novos ataques ou *malwares* não faz parte das suas caraterísticas.

A arquitetura deste sistema, apresentada na Figura 2.4, é constituída por quatro componentes principais: aquisição de pacotes, descodificação de pacotes, deteção com *pipeline* e *output*. Durante a fase de recolha dos pacotes, é feito também o pré-processamento, ao contrário do que acontece no Snort em que são dois componentes isolados. De seguida, é efetuada a descodificação da informação de forma separada, permitindo a inspeção do fluxo de dados na camada de aplicação. Após a descodificação, para a deteção de assinaturas, são permitidas cadeias de strings múltiplas numa única *pipeline*. O resultado é então exibido no ecrã ou armazenado na base de dados.
Podemos concluir, que o Suricata adota uma abordagem mais estratificada da leitura dos eventos e uma melhor análise da informação devido a sua arquitetura [19].

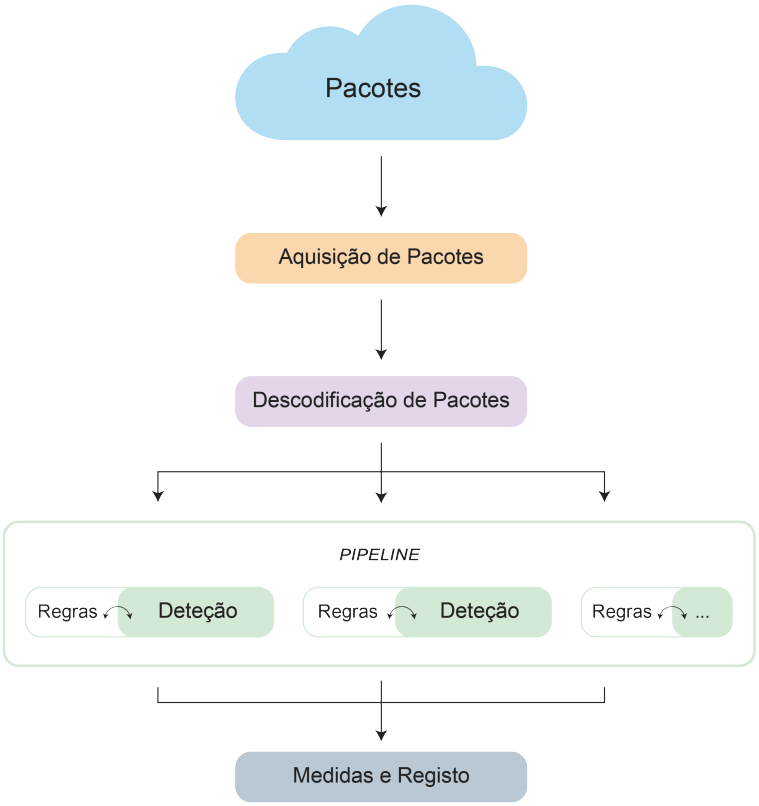


Figura 2.4: Arquitetura Suricata de alto nível

2.1.5 OSSEC

Open Source HIDS SECurity (OSSEC), é um HIDS de código aberto, escrito em C, criado por Daniel B. Cid e atualmente pertence à Trend Micro. Este HIDS, é capaz de realizar análises de registos em tempo real, analisar a integridade dos ficheiros da máquina, detetar *rootkits*, fornecer respostas ativas através de regras de *firewall* e integração de ferramentas de terceiros consoante os problemas que surgem.
O OSSEC é uma solução multiplataforma, no entanto o seu gestor funciona apenas em sistemas UNIX [20].

A sua arquitetura do OSSEC, apresentada na Figura 2.5, é composta por dois componentes principais: gestor e agentes. O gestor é um servidor centralizado e é responsável pela receção, processamento e correlação da informação recolhida pelos agentes. Assim, mantém os registos das máquinas duma determinada rede centralizados, simplificando a sua gestão. É neste componente que é armazenada as bases de dados de verificação da integridade dos ficheiros, as regras, os registos, eventos, entradas de auditoria dos ficheiros, e as principais configurações dos agentes. O facto do gestor estar centralizado, protege as máquinas onde está implantado o HIDS, pois se alguma máquina for comprometida, a informação que o gestor possui não é comprometida nem adulterada. Além disso, todas as comunicações entre os agentes e o gestor são encriptadas.
Os agentes estão implantados nas máquinas a proteger e são responsáveis por recolher informação

em tempo real sobre o sistema e enviar a informação para o gestor [21].

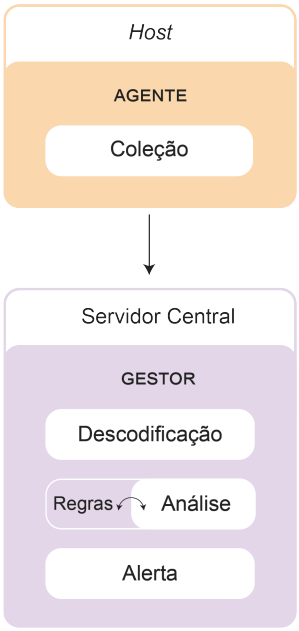


Figura 2.5: Arquitetura OSSEC

2.1.6 *TripeWire Open Source*

O *Open Source Tripwire* é um HIDS público simples e é útil para detetar alterações de ficheiros em sistemas operativos. Esta ferramenta foi criada com base em código originalmente contribuído pela *Tripwire*, Inc. em 2000. Inicialmente foi desenvolvido para sistemas UNIX, mas atualmente existem versões para outros sistemas operativos [3][22].

O Tripwire é capaz de monitorizar os ficheiros, controlando se vários atributos-chave permanecem inalterados, como por exemplo: datas de criação e id de *inodes*, números de ligações para *inodes*, instante e conteúdo da última modificação efetuada, lista de permissões de acesso, incluindo os bits *setuid* e *setgid*.

O controlo da alteração é feito através do estado atual do sistema de ficheiros com um ficheiro de política previamente definida pelo utilizador. É detetada uma potencial intrusão se ocorrerem quaisquer alterações suspeitas.
Esta ferramenta é baseada em comportamento, deteta alterações não previstas de atributos de ficheiros, no entanto não é capaz de indicar as causas e possíveis consequências dessas mesmas alterações. Outra caraterística é não ser reativo, pois apenas produz relatórios das alterações detetadas [3].

A arquitetura deste sistema é representada na Figura 2.6. É constituída por componentes de análise, onde são detetadas e verificadas alterações não autorizadas. Caso as alterações sejam

permitidas, é atualizado o ficheiro de políticas, e caso não sejam permitidas, são tomadas as medidas de segurança adequadas pelo componente de alerta.

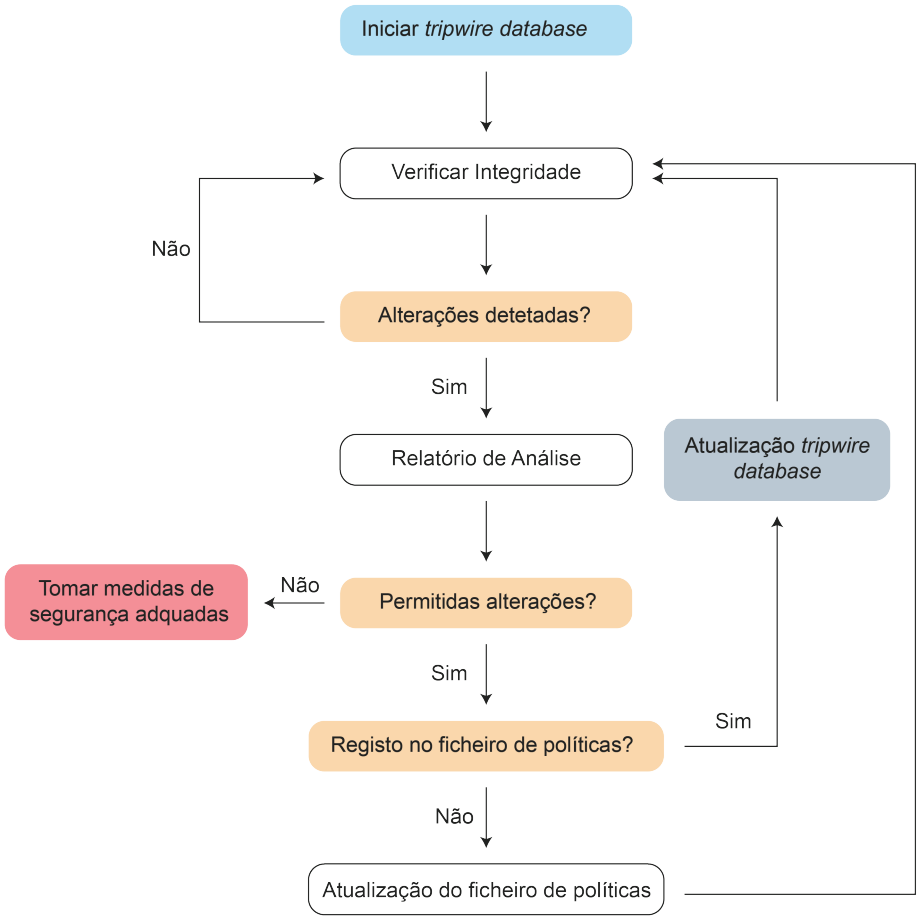


Figura 2.6: Arquitetura *TripeWire*

2.1.7 Modelo Híbrido

Zhenqi Wang e Dankai Zhang [23] desenvolveram um sistema de deteção baseado em rede e baseado em *host* com agentes. A ideia principal é a utilização de componentes de controlo e comunicação onde vai ser cruzada a informação vinda do HIDS e do NIDS.

O centro de controlo é utilizado principalmente para armazenar regras e atualizá-las, determinar se aconteceu algum ataque, comunicações de rede para trocar informações e gerar alertas. Este componente é constituído pelo sistema mestre e um *backup* do sistema mestre. Se surgir alguma falha do sistema principal, o *backup* completa a tarefa que estava a ser executada.

Tanto o HIDS como o NIDS funcionam individualmente, e comunicam com o centro de controlo através do módulo de comunicação, onde é cruzada a informação vinda dos dois sistemas. O modelo de gestão de utilizadores consiste em passar a autenticação de identidade, e gerir todo o sistema, incluindo a atualização da base de regras, gestão dos registos do sistema e análise.

A arquitetura híbrida proposta por Zhenqi Wang e Dankai Zhang é representada pela figura 2.7.

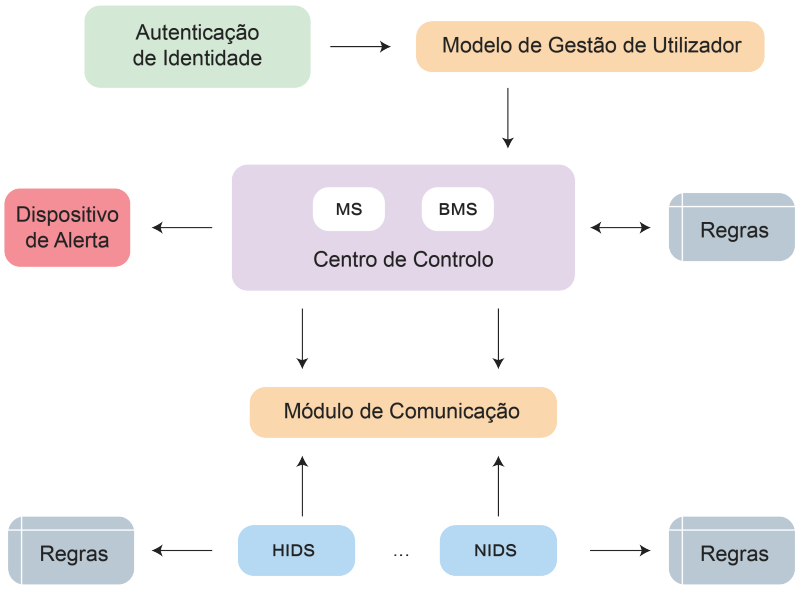


Figura 2.7: Arquitetura Híbrida Proposta

2.1.8 NIDS vs HIDS

A utilização de um NIDS face a um HIDS depende do ambiente em questão e das condições desse ambiente a proteger, pois ambas as soluções são viáveis quando se trata de deteção de intrusões. Estas duas opções têm *tradeoffs* que devem ser analisados e ponderados para a decisão de qual sistema usar.

Um HIDS é capaz de detetar ataques dentro de tráfego encriptado, e protegem mesmo estando *offline*. Para além de não necessitar de largura de banda adicional para funcionar, uma vez que opera ao nível do *host*.
A principal vantagem dos HIDS é a capacidade de inspeção de comportamentos anormais dentro dos *hosts*, tais como alterações de ficheiros e acesso não autorizado. Em termos de compatibilidade entre plataformas, os NIDS são melhores do que os HIDS, uma vez que podem proteger qualquer *host*, independentemente do seu sistema operativo.
Em algumas situações, implantarmos vários HIDS pode trazer mais benefícios do que ter um único NIDS, porque um NIDS é um ponto único de falha.
No entanto a solução mais viável é utilizar tanto os NIDS como os HIDS, pois utilizam abordagens diferentes, e os pontos fortes de um dos sistemas pode mitigar os pontos fracos do outro sistema.

As principais diferenças entre os sistemas de deteção baseado em *host* e sistemas baseado em rede são os locais de análise de deteção, e o local de implantação do sistema. Em relação a

HIDS são implantados no próprio *host* a proteger, e no caso de NIDS num segmento de rede da própria rede. Os dados de input do modelo de deteção dizem respeito à informação do ambiente onde está o sistema está implantado, e por conseguinte a atividade maliciosa detetada também é diferente.
Na tabela 2.3 são apresentados os *tradeoffs* da utilização de HIDS e de NIDS.

Tabela 2.2: *Tradeoffs* entre NIDS e HIDS

	NIDS	**HIDS**
Deteção	**i)** Aplicações no *host*; **ii)** Funcionamento do sistema operativo e a sua própria rede.	**i)** Aplicações na camada de rede; **ii)** Aplicações na camada de transporte; **iii)** Aplicações na camada TCP/IP.
Dados de Input	**i)** Pacotes de tráfego de rede; **ii)** Eventos; **iii)** Perfis de utilizadores [7].	**i)** Configuração do sistema; **ii)** Atividade da aplicação; **iii)** *Logs*; **iv)** Comandos do sistema; **v)** Atividade de rede do próprio *host*; **vi)** Processos a correr; **vii)** Acesso a ficheiros; **viii)** Registos de segurança de modificação [7].
Atividade Maliciosa Detetada	**i)** Ataque TCP SYN; **ii)** Pacotes fragmentados; **iii)** XSS; **iv)** CSRF [7].	**i)** Roube de chaves; **ii)** Roubo de identidade; **iii)** Acesso não autorizado; **iv)** *Spamming*; **v)** Processos maliciosos; **vi)** Atividade de *botnets*; **vii)** Utilização de *spyware* [7].

Tabela 2.3: *Tradeoffs* NIDS face a HIDS

2.2 *Machine Learning*

Machine Learning é uma técnica de inteligência artificial de extração de conhecimento a partir de conjuntos de dados. Os modelos de *machine learning* compreendem um conjunto de regras e métodos que são aplicados para encontrar padrões de dados interessantes, para reconhecer ou prever comportamentos.

De acordo com Mitchell [24], *machine learning* é definido da seguinte forma: "Diz-se que um programa de computador aprende com a experiência E em relação a alguma classe de tarefas T e

desempenho P, se o seu desempenho nas tarefas T, medido como P, melhora com a experiência E".

Os modelos de ML são utilizados em vários contextos, entre eles no contexto de deteção de intrusões. Estes modelos, em comparação com outros métodos, são capazes de aumentar consideravelmente a taxa de deteção.

2.2.1 *Machine Learning* no contexto de IDS

No contexto de IDS, a utilização de modelos de ML faz sentido quando diz respeito a AIDS, devido ao facto de serem analisadas anomalias que fogem do padrão da atividade normal e são técnicas que têm sido aplicadas de forma muito ativa nesta área.
Muitos dos sistemas de deteção de intrusão sofrem de taxas elevadas de falsos positivos, gerando muitos alertas inúteis e consequentemente pode fazer com que um ataque prejudicial seja ignorado. Outro ponto negativo de muitos IDS's é o facto de não possibilitarem o reconhecimento de ataques desconhecidos. Assim, muitos investigadores concentraram-se no desenvolvimento de IDS com taxas de deteção mais elevadas e taxas reduzidas de falsos alarmes [5][9].
O objetivo da utilização de técnicas de *machine learning* no contexto de IDS, é elevar os níveis de precisão na deteção, e diminuir a exigência de conhecimento por parte dos humanos.

Vários algoritmos e técnicas, tais como algoritmos de agrupamento, redes neurais, regras de associação, árvores de decisão, algoritmos genéticos, e métodos de vizinhos mais próximos, foram aplicados para descobrir o conhecimento de padrões em conjuntos de dados de intrusão [25][26][27][28].
Dependendo do funcionamento do algoritmo, juntamente com as caraterísticas dos dados que queremos analisar, são eleitos os algoritmos mais pertinentes para o contexto em questão e é definido então o modelo a utilizar.
Estes modelos podem atingir níveis de deteção muito satisfatórios quando existem dados de treino suficientes, e possuem generalidade razoável para detetar tanto variantes de ataque como novos ataques.

Os algoritmos de ML podem ser aplicados a dados que possuem um atributo de classe, ou a dados que não possuem classe. Para isto, existem também algoritmos que seguem métodos supervisionados ou então métodos não supervisionados.
A aprendizagem supervisionada requer dados de *input* que estejam etiquetados durante a fase de treino. Estes dados de treino são frequentemente etiquetados por um cientista de dados na fase de pré-processamento, antes de serem utilizados para treinar e testar o modelo. Assim, o modelo aprende a relação entre os dados de *input* e a classe a que pertence, podendo posteriormente classificar conjuntos de dados novos e prever resultados. Um lado negativo deste método é o tempo que tem de ser gasto na etiquetagem dos dados.
Pelo contrário, a aprendizagem sem supervisão utiliza dados sem etiquetas. É frequentemente usada para identificar padrões e tendências, ou para agrupar dados que tenham semelhanças entre si. Este método é frequentemente uma abordagem utilizada numa fase exploratória,

para compreensão do conjunto de dados. Contudo, o desempenho da deteção de métodos de aprendizagem não supervisionada é geralmente inferior ao dos métodos de aprendizagem supervisionada.

Uma das etapas mais importantes e estudadas para o sucesso dos modelos aplicados aos conjuntos de dados, é a qualidade dos dados, como por exemplo, os atributos que estes possuem. O tipos de atributos e métodos de extração de caraterísticas diferem entre os diferentes elementos de dados, fazendo com que os modelos de ML mais apropriados também sejam diferentes.

Nas secções seguintes são introduzidos conjuntos de dados de referência para IDS, algoritmos de ML frequentemente utilizados em IDS e são apresentados exemplos de modelos para sistemas de deteção de intrusão desenvolvidos com *machine learning*.

2.2.1.1 Coleção de Dados e Seleção de Caraterísticas

Grande parte do tempo quando trabalhamos com *Machine Learning* é gasto na preparação e no pré-processamento dos dados, o que inclui a recolha, limpeza, análise, visualização, e engenharia de características.

Geralmente, podemos considerar três fases na recolha de dados: aquisição de dados, etiquetagem de dados e melhoramento de dados já existentes. As técnicas de aquisição de dados podem ser utilizadas para descobrir, aumentar ou gerar novos conjuntos de dados. A etiquetagem de dados pode ser aplicada a um conjunto de dados já existe, contudo é necessário acrescentar etiquetas a entradas individuais. Finalmente, podemos melhorar os dados existentes ou utilizar modelos já treinados. Apesar desta distinção, estes três métodos podem ser utilizados simultaneamente [29].

A engenharia das caraterísticas é definida como a aplicação de transformações ás nossas caraterísticas num conjunto de dados para gerar novas ou alterar as existentes. Este processo é normalmente uma tarefa manual, *ad-hoc*. É necessário muito conhecimento de domínio para compreender que transformações aplicar aos dados.

Após este processo de geração e transformação, podemos aplicar a seleção de caraterísticas para reduzir a nossa dimensionalidade através da remoção de possíveis variáveis relativamente irrelevantes. Dados de alta dimensionalidade podem aumentar significativamente os requisitos de armazenamento de memória e os custos computacionais para a análise de dados. Daí vem a seleção das características. Nesta fase, como o nome sugere, são selecionadas as caraterísticas que queremos utilizar para treinar o nosso modelo. Pode parecer contra-intuitivo diminuir a quantidade de informação dada para o treino, mas por vezes o *tradeoff* no desempenho trás aspetos muito positivos [30].

Proposta de Taxonomia de Classificação para Algoritmos de Seleção de Caraterísticas

Sofiane Maza e Mohamed Touahria [30], apresentam uma nova taxonomia de classificação para algoritmos de seleção de caraterísticas para reduzir a dimensão dos dados selecionando atributos de interesse sem redundância (redundância mínima) e irrelevância (relevância máxima). Estes objetivos dão-nos uma boa compreensão dos dados, reduzindo o ruído, evitam problemas de *overfitting*, e permitem selecionar o melhor subconjunto de caraterísticas, que possuem uma relação altamente relevante com a classe a que pertencem.

A figura 2.8 ilustra o processo mais utilizado para seleção de caraterísticas [31]. A medida de avaliação (etapa de avaliação) e o método de pesquisa (geração de subconjuntos) são as etapas mais relevantes, visto que influenciam na eficácia da seleção de caraterísticas.
O modelo de funcionamento começa com o conjunto de dados e são gerados os subconjuntos de dados para de seguida serem avaliados segundo um critério de paragem. Para passar à fase de validação dos resultados, é necessário que os subconjuntos gerados sejam significativamente bons e representativos dos dados na totalidade. Caso não se verifique, são gerados novos subconjuntos.

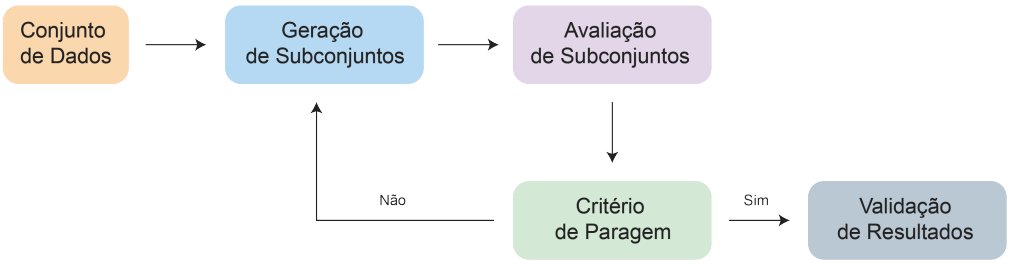


Figura 2.8: Processo de Seleção de Caraterísticas

Este processo segue um mecanismo de deteção que se concentra em três categorias tais como: incremental, decremental, e seleção aleatória [31].

- Incremental: A seleção das caraterísticas é feita de forma incremental, iniciando com um conjunto vazio de caraterísticas. Em cada iteração o processo acrescenta uma nova caraterística ao conjunto, sendo selecionada dependendo da métrica de avaliação. No final do processo, o conjunto de caraterísticas contém as melhores caraterísticas que representam o melhor subconjunto de caraterísticas.
- Decremental: A seleção das caraterísticas é feita de forma decremental, iniciando com o conjunto de todas de caraterísticas. Em cada iteração o processo elimina uma das caraterística do conjunto, sendo selecionada dependendo da métrica de avaliação. No final do processo, o conjunto de caraterísticas contém as melhores caraterísticas que representam o melhor subconjunto de caraterísticas.
- Seleção Aleatória: Em cada interação, é selecionado um grupo de características como subconjunto e é avaliado usando métricas de avaliação. No final do processo, é obtido o

melhor subconjunto entre todos os subconjuntos selecionados. A forma de selecionar o grupo de caraterísticas depende das técnicas que são utilizadas pela abordagem proposta.

A seleção de caraterísticas foi classificada em três grupos: embalamento, filtro e abordagem híbrida [31].

- Embalamento: Utiliza os algoritmos de aprendizagem/classificação para selecionar as melhores caraterísticas, como uma ferramenta de avaliação. Utiliza a taxa de precisão como um *feedback* para determinar a eficácia dos subconjuntos de caraterísticas.
- Abordagem de filtro: Utiliza os dados estatísticos de aprendizagem como medida para avaliar os atributos independentemente do algoritmo de aprendizagem/classificação. Os métodos de embalamento alcançam melhor desempenho de classificação do que o método de filtragem, no entanto são mais dispendiosos computacionalmente. Por outro lado, a abordagem de filtro é atribuída quando queremos processar dados de alta dimensão.
- Abordagem híbrida: É a combinação de métodos de embalamento e de filtragem, que surgiu para cobrir os inconvenientes das outras abordagens e beneficiar das vantagens.

Uma nova taxonomia de classificação foi proposta para algoritmos de seleção de caraterísticas. Nesta taxonomia, são apresentadas as caraterísticas de cada algoritmo em função da sua técnica de seleção, mecanismo de seleção, solução de subconjunto único/multi-solução, conjunto de dados, abordagem de seleção e os classificadores para avaliar o desempenho.
A figura 2.9 ilustra a taxonomia global dos algoritmos de seleção de caraterísticas.

Os algoritmos são classificados em cinco abordagens de acordo com as técnicas que são integradas com eles, que são Algoritmos Determinísticos, Padrões Inteligentes, Redes Neurais, Conjunto *Fuzzy* & *Rough* e Enxame Inteligente.
Como foi dito anteriormente, os algoritmos de seleção de características que utilizam a abordagem de filtro têm menor complexidade computacional do que a abordagem de invólucro que é considerada mais elevada, porque um filtro é baseado na distância, informação, dependência e medidas de consistência, ao contrário da abordagem baseada em embalamento que é baseado na taxa de erro dos classificadores.

Além disso, é integrado o aspeto de solução única e multi-solução nesta classificação, o que significa que o algoritmo tem um subconjunto de caraterísticas como uma solução ou multi-subconjunto.

Para a classificação e avaliação do subconjunto de caraterísticas são utilizados algoritmos classificadores como Árvore de Decisão (*Decision Tree*) (DT), SVM, K-Vizinhos mais Próximos (*K-Nearest Neighbors*) (KNN), MLP.

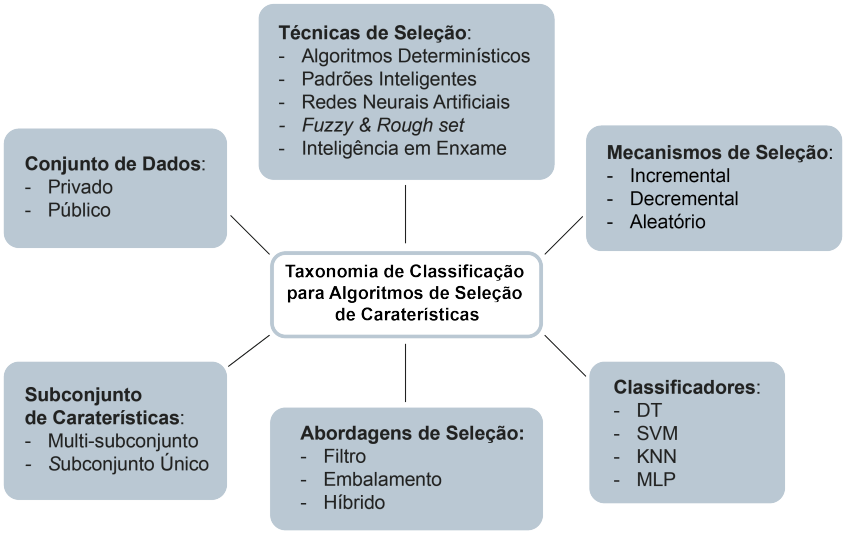


Figura 2.9: Taxonomia dos Algoritmos de Seleção de Caraterísticas

2.2.1.2 Conjuntos de dados IDS

Diferentes conjuntos de dados públicos e privados são integrados como dados de referência para a avaliação de IDS's. Os conjuntos de dados privados e auto produzidos são gerados para evitar conjuntos de dados de treino incompletos, no entanto ainda são pouco acessíveis e é difícil garantir a sua eficiência.

Primeiramente, o mais importante para um conjunto de dados ter uma aplicabilidade positiva, é serem gerados com base em infraestruturas de computação modernas e conter conhecimento atualizado capazes de refletir caraterísticas recentes de atividade computacional normal e maliciosa.
Em segundo lugar, a dimensionalidade do conjunto de dados é por vezes um obstáculo para tarefas computacionais. É importante que o processamento dos dados não seja demasiado demorado e que não contenha atributos em demasia e que não sejam importantes para a deteção. Ou seja, é bastante relevante que haja uma seleção de atributos prévia, antes do uso propriamente dito do conjunto de dados. Para isso, existem muitos métodos, algoritmos e ferramentas para minimizar o tamanho dos dados, como foi referido da secção anterior.

KDDcup99 e NSL-KDD [32][33]

Um dos conjuntos de dados mais aprofundados e utilizados em estudos de IDS é o KDDcup99, e o NSL-KDD, que é obtido pela remoção das instâncias redundantes e duplicadas do KDDcup99. O KDDcup99 é uma coleção de cinco milhões de registos de ligações de sete semanas de tráfego de rede e é distribuído em duas versões, o conjunto completo de dados com 4898431 registos e o subconjunto de 10% com 494307 registos. Este conjunto de dados contém 41 caraterísticas (32 contínuas e 9 atributos nominais) com classe alvo. As 41 caraterísticas foram distribuídas em quatro categorias (básico, conteúdo e tráfego baseado no tempo ou tráfego baseado no *host*).

A tabela 2.4 contém algumas descrições exemplo de caraterísticas existentes neste conjunto de dados.

Número	Nome	Tipo	Categoria	Descrição
1	Duração	Contínua	1	Duração da ligação
2	*Hot*	Contínua	2	Numero de indicadores *hot*
3	Contagem	Contínua	3	Número de ligações ao mesmo host nos últimos dois segundos
4	Contagem dst host	Contínua	4	Contagem das ligações ao mesmo host

Tabela 2.4: Descrição das Caraterísticas dos Dados NSL-KDD

Cada instância do KDDcup99 e NSL-KDD é classificada por atividade normal ou de ataque. Este conjunto de dados tem 24 tipos de ataques que foram categorizados em 4 classes, nomeadamente: Negação de Serviço (*Denied of Service*) (DOS), ataque de Sonda, Utilizador para a Raiz (*User to Root*) (U2R) e *Remote to Local* (R2L).

A etapa de pré-processamento permite para preparar o conjunto de dados para a experimentação antes de qualquer estudo ou construção de modelos. É realizado para eliminar todos os problemas relativos ao tamanho, informação incompleta, e registo de duplicação. Para isso, existem várias técnicas de redução e normalização com o objetivo de preparar o conjunto de dados para as etapas de treino e teste [30].

2.2.1.3 Algoritmos

Após a passagem pelas fases descritas nas secções anteriores, como a recolha de dados e respetiva limpeza, feita a engenharia de caraterísticas, segue-se a fase de treinamento. Para esta fase, são utilizados algoritmos de classificação, onde, no caso de se tratar de dados supervisionados, o classificador aprende as classes para que, na fase de teste possa prever corretamente dados novos. Existem vários algoritmos que podem ser utilizados, consoante a natureza dos dados transformados.

Naïve Bayes

Naïve Bayes é um tipo de rede *Bayesiana*. É uma técnica baseada em probabilidade condicional e na hipótese de independência de atributos. Para cada amostra, o classificador calcula as probabilidades condicionais para diferentes classes. Este método assume o valor de cada caraterística como independente e considera a correlação entre as caraterísticas. Este modelo inclui duas probabilidades, uma é a probabilidade condicional e outra é a probabilidade de classe. A probabilidade de classe é determinada dividindo a frequência de cada instância de classe pelo total de instâncias. A probabilidade condicional é a razão entre a ocorrência de cada atributo para uma determinada classe e a ocorrência de amostras para essa classe [34].
Este algoritmo é robusto ao ruído e é capaz de aprender de forma incremental, no entanto, não tem um bom desempenho em dados relacionados com os atributos.

A figura 2.10 ilustra o funcionamento geral do algoritmo *Naïve Bayes*.

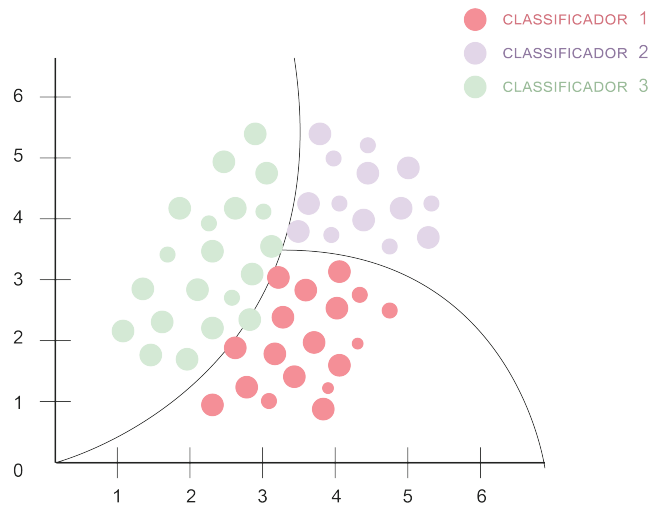


Figura 2.10: Ilustração do Funcionamento do Algoritmo *Naïve Bayes*

K-Nearest Neighbor

A ideia central do KNN é baseada na hipótese múltipla. Se a maioria dos vizinhos de uma amostra pertence à mesma classe, a amostra tem uma elevada probabilidade de pertencer à classe. Assim, o resultado da classificação está apenas relacionado com os vizinhos mais próximos do topo da escala. O parâmetro k influencia o desempenho dos modelos KNN e diz respeito ao número de vizinhos que é considerado. Quanto menor for o k, mais complexo é o modelo e maior é o risco de sobreajustamento. Pelo contrário, quanto maior for o k, mais simples é o modelo. Este algoritmo pode ser aplicado a dados com algum ruído, é adequado a dados não lineares e treina rapidamente. No entanto possui baixa precisão sobre a classe minoritária, e é sensível ao parâmetro k [5].
O funcionamento da divisão das classes feita pelo algoritmo KNN está ilustrada na figura 2.11

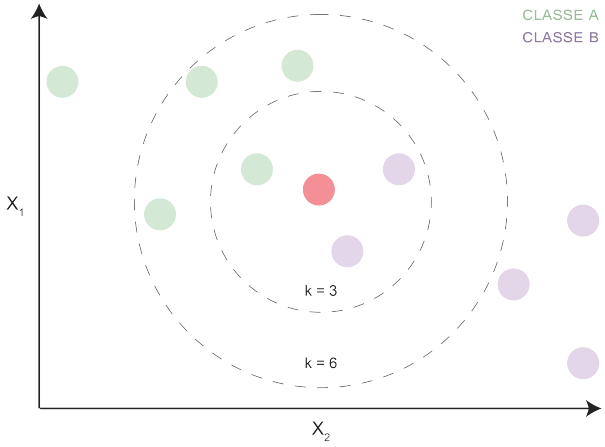


Figura 2.11: Ilustração do Funcionamento do Algoritmo KNN

Árvores de Decisão

Este algoritmo aprende e modela um conjunto de dados em problemas de classificação. O processo de aprendizagem inclui a seleção de características, geração de árvores, e poda de árvores. Classifica dados novos de acordo com o que aprendeu com os dados anterior utilizando um critério bem definido na seleção das melhores caraterísticas de cada árvore de nós durante a sua construção [5].

Um modelo de árvore de decisão tem um nó de raiz ligado a diferentes nós como dados de atributo que decidem o caminho para cada nó. As decisões são tomadas por comparação de dados anteriores e marcadas como folhas. Uma abordagem comum de árvore de decisão é o algoritmo C4.5 [35].

Um dos pontos positivos é a seleção automática de caraterísticas, e outro é possuir uma interpretação forte por ser semelhante a uma árvore. Os lados negativos do uso deste modelo é a tendência dos resultados da classificação para a classe maioritária e ignorar a correlação dos dados.

A figura 2.12 ilustra o funcionamento geral do algoritmo *DT*.

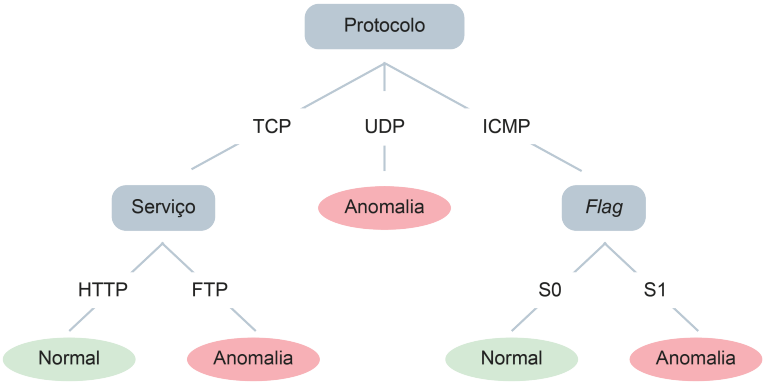


Figura 2.12: Ilustração do Funcionamento de uma Árvore de Decisão

Máquina de Vetores de Suporte

O SVM é um dos algoritmos de ML supervisionados mais usados. O algoritmo faz a separação dos dados em classes pelo hiperplano que maximiza a margem entre todas as classes de ataque. Normalmente o classificador é utilizado para a classificação binária, ou seja, classifica o conjunto de treino em duas classes diferentes [36].
É importante referir que este modelo não tem um bom desempenho em grandes dados ou múltiplas tarefas de classificação e é sensível aos parâmetros da função do kernel.
A figura 2.13 ilustra o funcionamento geral do algoritmo SVM.

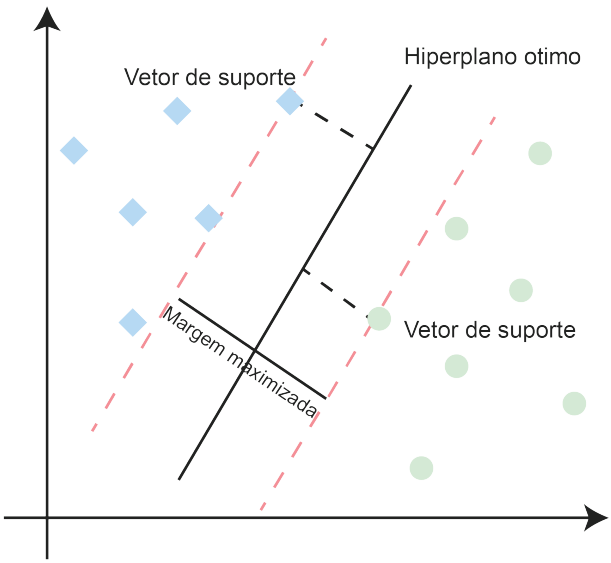


Figura 2.13: Ilustração do Funcionamento do Algoritmo SVM

Rede Neural Recorrente

Os modelos Recurrent Neural Network (*Rede Neural Recorrente*) (RNN) fazem parte dos algoritmos de *deep learning*. Estes algoritmos pertencem a um ramo de *machine learning* e são algoritmos que tentam modelar abstrações de alto nível de dados usando um grafo profundo com várias camadas de processamento, compostas de várias transformações lineares e não lineares.
O modelo RNN tem uma principal diferença introduzida pela arquitetura de redes neurais, que é uma ligação cíclica de neurónios, que permite inferência de condições anteriores dos mesmos. Esta caraterística permite a um nó da rede recordar o seu estado anterior. O RNN é especialmente útil na área do processamento de linguagem e vídeo, onde o contexto da sequência de dados é altamente relevante para a estrutura dos dados de input [37]. Um exemplo de RNN são as redes Memória de Curto Prazo Longa (*Long Short-Term Memory*) (LSTM), que são adequadas para classificar, processar e fazer previsões com base em dados de séries temporais.
Um ponto negativo deste tipo de algoritmos, é o facto de não serem tão interpretáveis e ocuparem muito tempo de processamento.
A figura 2.14 ilustra o funcionamento geral do algoritmo RNN.

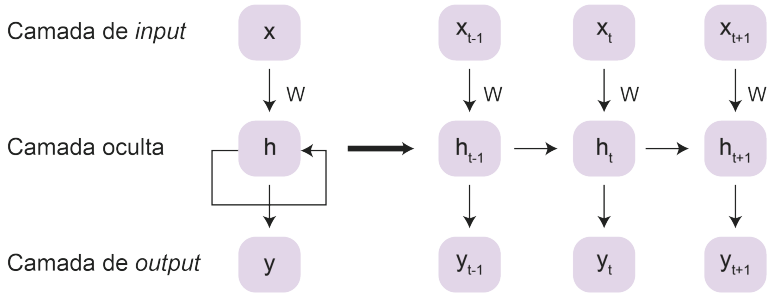


Figura 2.14: Ilustração do Funcionamento do Algoritmo RNN

2.2.1.4 *Behavior-Based Access Control*

O sistema Controlo de Acesso baseado no Comportamento *(Behavior-Based Access Control)* (BBAC) [6] foi desenvolvido com o objetivo de ultrapassar limitações a nível empresarial, no que diz respeito à fiabilidade da informação dada aos consumidores de documentos ou produtores sem reputação.

A técnica de deteção usada é baseada em anomalia acoplado a um modelo de *machine learning* e é classificado como *host-based*. O BBAC é implantado em cada *host*, monitoriza o tráfego de rede que sai e entra do próprio *host* e controla informação do seu sistema de ficheiros. Para o BBAC desempenhar a sua função tem de recolher informação de fluxo de rede, por exemplo, TCP e Protocolo de Datagramas do Utilizador (*User Datagram Protocol*) (UDP); Protocolos de transporte, por exemplo, Protocolo de Transferência de Hipertexto (*Hypertext*

Transfer Protocol) (HTTP), Protocolo de Mensagens Extensíveis e Presença (*Extensible Messaging and Presence Protocol*) (XMPP), e Protocolo simples de transferência de correio (*Simple Mail Transfer Protocol*) (SMTP); Registos de auditoria, por exemplo, produzidos por servidores Web, Profissional Certificado de Segurança Ofensiva (*Offensive Security Certified Professional*) (OCSP), e DNS; Conteúdo a nível da aplicação, por exemplo, mensagens de correio eletrónico e chat, documentos PDF, e páginas wiki.

O BBAC funciona através da realização de cálculos precisos da fiabilidade dos utilizadores, por exemplo, padrões de comportamento e de utilização, bem como documentos, por exemplo, proveniência e dependências de dados do fluxo de trabalho.

Este sistema recorre a uma combinação de técnicas de *machine learning* supervisionadas e não supervisionadas para controlar e classificar se certas entidades, tais como *hosts*, utilizadores ou documentos, estão a demonstrar algum comportamento suspeito. O pressuposto é que as mudanças no comportamento observável (por exemplo, mudanças nos padrões de tráfego de saída) são indicativas da execução de ataques (por exemplo, dados internos expostos ou propagação de um ataque).

BBAC combina *clustering* para segmentar dados em grupos que têm comportamento semelhante, com SVM para construir um classificador por *cluster* para detetar mudanças de comportamento. O objetivo do *clustering* é a criação de *clusters* baseados em aspetos de comportamento, que segundo o estudo melhora as taxas de classificação, resultando em menos falsos positivos. BBAC utiliza o algoritmo K-Means para formar os *clusters* iniciais e uma implementação de um protótipo de uma etapa adicional que utiliza árvores de decisão.

A BBAC utiliza o algoritmo SVM para classificar os dados como normais e anormais. A particularidade é a construção de vários classificadores SVM, por exemplo, classificadores SVM para comportamentos de ligação TCP, comportamentos de pedidos HTTP, e comportamentos de texto, numa tentativa de aumentar a precisão do processo de deteção.

O uso de *Decision Trees* é motivado pela dificuldade de um operador humano perceber a representação dos classificadores e a compreensão dos vários *clusters*. Para melhorar a capacidade de compreensão do *clustering*, são extraídas regras simples que representam os diferentes *clusters* identificados pelo algoritmo K-Means, e depois essas regras são usadas para atribuir um *host* a um grupo. As regras são representadas em árvores de decisão, onde são aprendidas automaticamente as condições do ramo (em termos de limiares de caraterísticas), sob as quais um *host* pertence a um *cluster* e não a outro qualquer. Cada nó intermédio da árvore é um teste de um atributo e cada ramo é um possível resultado do teste. Os nós de folha contém etiquetas de classe. A utilização de uma árvore de decisão é semelhante a seguir um diagrama de fluxo de controlo.
A figura 2.15 representa a arquitetura do BBAC proposto.

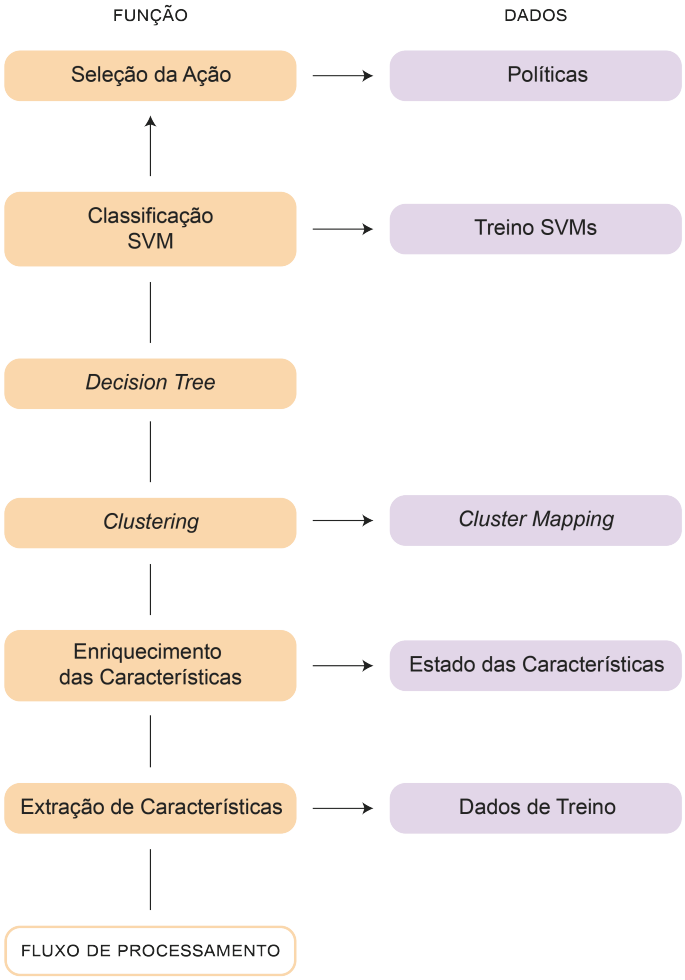


Figura 2.15: Arquitetura do Modelo BBAC

2.2.1.5 Modelo HIDS com CNN

Nam Nhat Tran, Ruhul Sarker e Jiankun Hu [38], propuseram um modelo de HIDS, baseado em anomalia com ML. O algoritmo usado para a deteção de intrusões foi o Rede Neural Convolucional *(Convolutional Neural Network)* (CNN). Os conjuntos de dados utilizados neste estudo foram os NGIDS-DS e o ADFA-LD.

Para dados de *input* foram reunidas frações de chamadas ao sistema com um comprimento de uma janela deslizante. O tamanho das sequências foi assumido como um heurística, devido ao facto de não haver regra geral nem estudos suficientes que cheguem a um valor correto concreto. Devido ao facto dos dados originais não estarem num formato compatível com o algoritmo CNN, foram transformados numa matriz 4-D. As duas primeiras dimensões referem-se ao tamanho da matriz, ou seja, o número de linhas, que é igual a um, e o número de colunas, que é igual ao tamanho da janela deslizante. A terceira dimensão da matriz é uma, equivalendo a ser o canal na imagem RGB. A quarta dimensão é o número da observação no conjunto de dados. Por razões de simplicidade, economia de tempo e devido à propriedade dos dados de entrada, a arquitetura

CNN foi apenas implantada com apenas uma camada oculta, como podemos observar a seguir.

A camada de *input* tem um tamanho de [1 9 1] ou [1 5 1], onde 9 e 5 são os dois melhores tamanhos de janela, segundo este estudo, determinado de forma heurística. Com estes dados de *input*, o tamanho dos filtros (kernels) é também [1 x] com x ajustado em conformidade. Aplicando cada filtro respetivamente a cada entrada de *input*, é executado o produto entre um valor de entrada X com o peso do filtro W, e depois é adicionado um bias B ao resultado, obtendo um valor escalar Y. O filtro é aplicado de forma deslizante ao longo do *input*, produzindo uma cadeia de valor escalar com a mesma altura dos dados de *input*, mas com largura menor. No passo seguinte, essas cadeias de número serão alimentadas através de uma camada ReLU, a fim de introduzir não linearidades no modelo. Um ReLU mantém os valores positivos e substitui os valores negativos por zero. Em seguida, uma camada de *pooling* é aplicada aos dados, resultando numa saída escalar, como se pode observar na figura seguinte. Posteriormente, na camada *fully connenected* cada elemento é ligado a todos os elementos da camada anterior, de maneira a efetuar uma classificação baseada em caraterísticas extraídas destas camadas anteriores. Nesta fase, uma camada de *dropout* com uma *drop rate* variável é também utilizada para resolver situações de excesso de *overfitting*. Finalmente, é apresentada a camada *softmax*, que desempenha um papel como meio para converter os resultados da classificação em probabilidades. O processo de treino foi conduzido com o algoritmo Descida Estocástica Gradiente com Momento (*Stochastic Gradient Descent with Momentum*) (SGDM). Com o SGDM, o tamanho do mini *batch* é também um parâmetro ajustável, que foi escolhido com base no desempenho otimizado em detrimento do tempo de treino. Dois métodos de regularização L1 e L2 foram também aplicados para melhorar os resultados experimentais. Os processos de treino e teste foram conduzidos repetidamente através de diferentes conjuntos de parâmetros. Os dois melhores conjuntos de resultados são alcançados com janelas de 5 e 9.

A taxa de deteção e a taxa de falsos positivos, seriam possivelmente melhoradas através da aplicação de um modelo CNN com mais camadas intermédias. Como a rede neural é fundamentalmente sobre a extração de caraterísticas, um modelo complexo produziria possivelmente mais caraterísticas únicas, o que por sua vez poderia certamente melhorar os resultados da classificação. O lado negativo de trabalhar com algoritmos de redes neurais complexas, é o facto dos resultados não serem de fácil interpretação.
A figura 2.16 representa a arquitetura do modelo proposto.

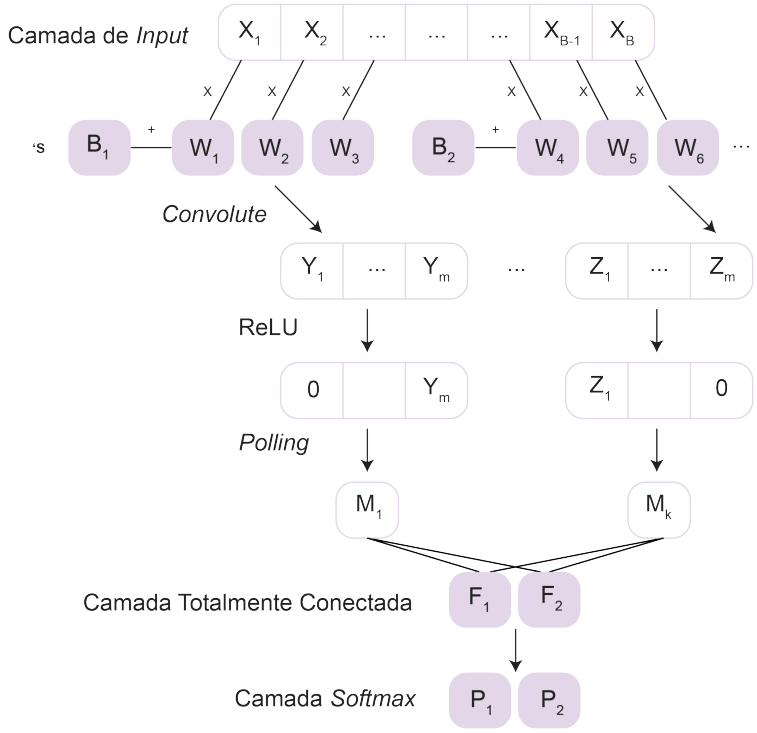


Figura 2.16: Arquitetura CNN

2.2.1.6 Filtro de Alarme Inteligente

Meng, Li e Kwok [27] desenvolveram o design de um filtro de alarmes inteligente utilizando o algoritmo KNN.

A motivação para o desenvolvimento deste modelo foi o facto de existir uma grande quantidade de sistemas com taxas de falsos positivos e falsos negativos. Sendo a principal inovação, o desenvolvimento de um método de verificação de alertas baseado no conhecimento e a utilização de um filtro de alarmes inteligente baseado no classificador KNN para filtrar os alarmes indesejados.

Este modelo é direcionado para controlo de dados provenientes de rede, e é baseado em anomalia e assinatura.

Na figura seguinte está ilustrada a arquitetura de alto nível do filtro de alarme inteligente baseado em KAV, que consiste em três componentes principais: base de dados de alarme, medida de classificação e filtro de alarme. O componente da base de dados de alarmes é responsável pelo armazenamento dos alarmes classificados e pelo treino do classificador KNN. A componente de medida de classificação é responsável pela classificação dos alarmes NIDS de entrada em *clusters* apropriados para a respetiva etiquetagem. O componente do filtro de alarme é utilizado para reduzir os alarmes NIDS indesejados em termos do limiar de classificação. O conhecimento especializado é muito importante no filtro, que desempenha duas funções principais: classificar

os alarmes no processo de treino, e decidir o limiar de classificação.

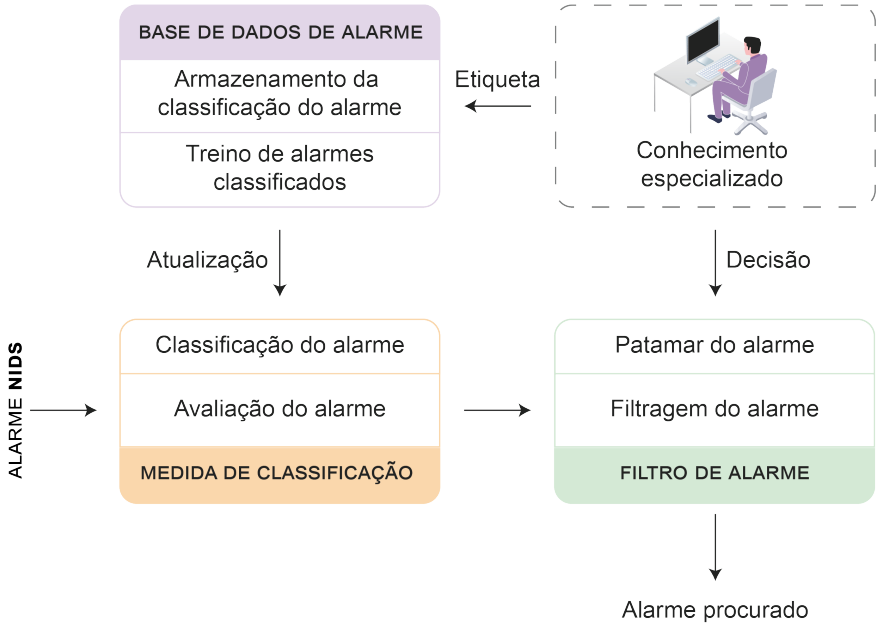


Figura 2.17: Arquitetura de Alto Nível do Modelo de Filtro de Alarme Inteligente baseado em KAV

Na implantação real, há duas fases principais no filtro: fase de preparação e fase de filtração. Na fase de preparação, um administrador de segurança pode rotular um número limitado de alarmes NIDS (por exemplo, 100 alarmes classificados) e armazená-los na componente da base de dados de alarmes. O filtro de alarme pode então treinar o classificador KNN para estabelecer um modelo KNN com base nestas instâncias rotuladas. Além disso, o administrador deve decidir o limiar de classificação no componente do filtro de alarme.

Na fase de filtração o componente rotula cada alarme de entrada baseado no classificador KNN. De seguida, estes alarmes rotulados são enviados para o componente do filtro de alarme, que é responsável pela realização da filtragem do alarme de acordo com o limiar de classificação pré-determinado. Por exemplo, se um alarme for classificado abaixo do limiar de classificação, então este alarme será filtrado como um alarme não desejado.

Foi realizada uma avaliação, com três experiências para investigar o desempenho do filtro de alarme com um conjunto de dados real e num ambiente de rede, no entanto com a utilização do Snort para a geração dos alertas. Os resultados das experiências indicam que o filtro de alarme pode alcançar bons resultados de filtragem com uma boa utilização de CPU e redução a carga de trabalho de um analista de segurança através de uma filtragem eficaz de muitos alarmes indesejados.

2.3 Discussão

Nesta secção iremos comparar as abordagens e soluções utilizadas pelos trabalhos descritos anteriormente, relativamente a tecnologias que não usam algoritmos de *machine learning* acoplados, e nas secções seguintes, relativamente a tecnologias desenvolvidas baseadas em algoritmos de *machine learning*.

Na primeira tabela é possível ver que grande maioria das tecnologias apresentadas seguem uma abordagem de assinatura, pois são baseadas em regras pré-estabelecidas. Já na segunda tabela, a grande maioria dos modelos apresentados fazem a deteção da atividade maliciosa através de anomalias detetadas e não através de assinaturas. Geralmente, a deteção baseada em anomalia é feita utilizando algoritmos de *machine learning*.

É de notar que no caso do modelo de filtro de alarme inteligente, a deteção é realizada recorrendo a assinaturas e a anomalias, ou seja, é utilizada informação de conhecimento e de uso indevido, respetivamente.

Os modelos que seguem uma abordagem baseada em algoritmos de ML são capazes de alcançar melhores taxas de falsos negativos, devido ao facto de ser possível a deteção de ataques desconhecidos. Assim, modelos de ML permitem melhorar a eficácia de deteção de anomalias dos IDSs. No entanto, apenas funcionam para os contextos em que foram treinados.
Nenhum dos sistemas estudados apresenta uma arquitetura modular que permita a adaptação do IDSs a diferentes contextos. Ou seja, a integração destes IDSs em contextos diferentes dos originalmente destinados poderá resultar num decréscimo significativo do seu desempenho, até mesmo inviabilizar a utilidade do mesmo.
Por este motivo, a modularização do sistema é uma das principais contribuições da arquitetura proposta para este projeto, pois permite que haja uma base comum que pode ser adaptado para diferentes contextos, alterando, por exemplo, o modelo de ML usado ou adicionando modelos específicos para determinados contextos (ex: IoT, Cloud).

Entre o tipo de modelos apresentados que recorrem a algoritmos de ML, apesar da abordagem ser a mesma, existem algumas diferenças entre eles.
Os próprios sistemas não estão implantados nem a proteger na mesma zona, diferindo assim, no tipo de dados que analisam e nos algoritmos escolhidos para a classificação.
O modelo BBAC é baseado em *host* e controla o tráfego de rede do *host* onde está implantado, sendo por isso o tipo de informação a analisar, como, informação do sistema de ficheiros do *host*, informação de fluxo de rede, por exemplo, protocolos de transporte utilizados. Como é analisada bastante informação em simultâneo, recorre a vários algoritmos para organizar a informação e classificá-la. Primeiramente é utilizado um algoritmo de *clustering* para dividir os dados por grau de semelhança, e de seguida recorre ao algoritmo SVM para construir um classificador para cada *cluster* criado.
Em contraste, o modelo HIDS com CNN, apesar de utilizar também *machine learning* para deteção das anomalias, utiliza dados mais específicos e isolados, com apenas um atributo,

chamadas de sistema. Por se tratar de informação em que a ordem das chamadas importa, foi utilizando um algoritmo mais complexo, de maior dificuldade de interpretação, o CNN.
Por fim, o terceiro exemplo de sistema descrito, segue uma abordagem particular. Trata-se de um filtro de alarme que utiliza *machine learning*, sendo a principal inovação um método de verificação de alertas baseado em conhecimento combinado com um filtro desses alertas, utilizando o algoritmo KNN. Para explorar o impacto do valor de K, foram construídos conjuntos de dados de alarme utilizando o conjunto de dados DARPA99 [39] juntamente com o SNORT para gerar os alertas e serem armazenados para análise. Este conjunto de dados utiliza dados de tráfego de rede, sendo assim, um HIDS. Dos exemplos de sistemas apresentados neste documento, é o único que segue uma abordagem híbrida, no que diz respeito ás técnicas de deteção, ou seja, utiliza assinaturas e anomalias.
No que diz respeito ás taxas de deteção de cada um dos sistemas, o sistema que apresenta melhor performance é o BBAC, devido ao tipo de dados que utiliza, juntamente com a combinação de modelos para a classificação. O sistema HIDS com CNN, apresenta taxas de deteção mais baixas, devido ao facto de utilizar dados de chamadas de sistema que tem caraterísticas bastante particulares em relação ao dados utilizados nos outros dois sistemas, e utilizar um algoritmo mais complexo de compreensão. Poderá melhorar a sua performance aumentando o número de camadas ocultas do algoritmo, no entanto diminui o nível de interpretação dos resultados.

	Snort	**Suricata**	**OSSEC**	***Tripe Wire***
Tipo de Sistema	Rede	Rede	*Host*	*Host*
Deteção de Ataques	X	X	X	X
Prevenção de Ataques	X	X	X	
Baseado em Assinatura	X	X	X	
Baseado em Anomalia				X
Machine Learning				

Tabela 2.5: Comparação das Ferramentas 1

	Híbrido	**BBCA**	**HIDS baseado em CNN**	**Filtro de Alarme Inteligente**
Tipo de Sistema	Rede/*Host*	*Host*	*Host*	Rede
Deteção de Ataques	X	X	X	X
Prevenção de Ataques	X	X	X	
Baseado em Assinatura	X			X
Baseado em Anomalia		X	X	X
Machine Learning		X	X	X

Tabela 2.6: Comparação das Ferramentas 2

Capítulo 3

JBriareos

3.1 Visão Geral

O JBriareos é um sistema de deteção e prevenção de intrusões, baseado em *host*, escrito em Java. Este sistema é uma evolução do Briareos, que foi desenvolvido em Python. O objetivo de migrar o Briareos para Java foi melhorar o seu desempenho de análise de pacotes, para reduzir o uso de recursos no *host*. Além disso, na versão do JBriareos foi concluída a implementação de um servidor *manager* que fornece uma forma centralizada de gerir configurações e propagar regras criadas entre instâncias em execução no mesmo domínio administrativo, de maneira a que essa informação não tenha de estar presente em cada *host*.

O principal objetivo do JBriareos foi a implementação de um IDPS com uma arquitetura modular e escalável, capaz de descobrir vetores de ataques desconhecidos e partilhar informação adquirida entre instâncias. Este sistema analisa o tráfego de rede e monitoriza o comportamento do próprio *host*, a fim de detetar ataques. Quando é detetado um ataque, com o objetivo de prevenir que outro ataque igual aconteça cria regras locais.

3.2 Arquitetura

A arquitetura apresentada na figura 3.1 é constituída por 3 componentes principais: Componente de *host* do JBriareos (JBriareos *Host Component*) (JHC), Sistema Distribuído JBriareos (JBriareos *Distributed System*) (JDS) e Servidor Gestor JBriareos (JBriareos *Manager Server*) (JMS). O JHC é constituído pelo *network interceptor*, responsável por capturar o tráfego e pelo *processing engine*, responsável por analisá-lo. Os *hosts* executam instâncias JHC que protegem o próprio *host* através da utilização de um servidor *manager* que propaga as regras de *iptables* que foram previamente criadas durante a análise do tráfego da rede, pelas instâncias.
Para garantir segurança entre a comunicação entre o JHC e o JMS, é necessária a existência de um certificado ZMQ por parte dos JHC. Caso não se verifique a existência deste certificado, então o JHC executa com as suas próprias configurações. Com a implementação de um JMS,

os ficheiros de configuração das *pipelines* estão centralizadas, tornando possível a partilha das regras dos iptables criadas por um componente a outras instâncias da rede.
O facto de ser um componente centralizado é um ponto menos positivo, pois é um ponto único de falha, comprometendo a partilha das regras e configurações das *pipelines*. Para resolver esta vulnerabilidade, o JMS poderia ser replicado entre os nós e ser usado um algoritmo de consenso.

A utilização de certificados ZMQ para as comunicações entre os JHC e o JMS faz com que só as regras de confiança sejam propagadas através da rede e implantadas nos *hosts*.

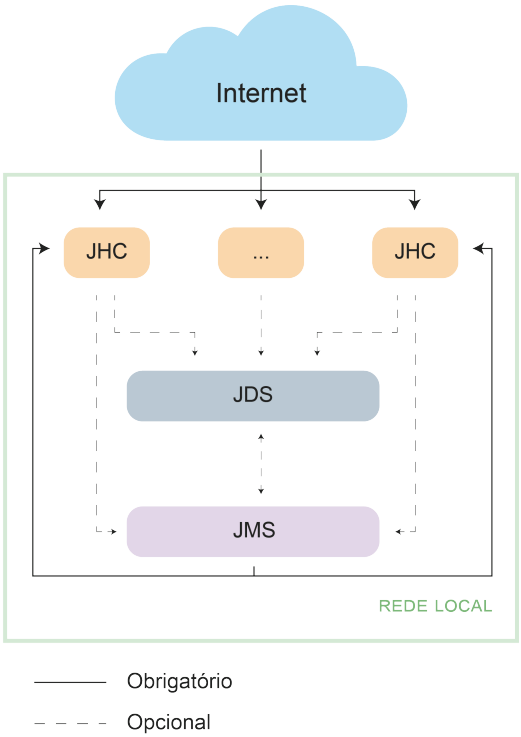


Figura 3.1: Arquitetura JBriareos

O principal objetivo da existência do JDS é reduzir a carga de trabalho dos JHC, pois é capaz de descarregar tarefas. Para isso é constituído por 4 componentes: *Broker*, *worker clusters*, *workers* e *worker manager*.

Os *workers* são os componentes que efetuam a análise de um pacote. São *docker containers* e têm acesso ao JMS, a fim de poderem publicar novas regras, no caso de ser detetado um ataque. Desta forma, a primeira vez que um ataque é detetado, pode não ser evitado.

Os *clusters* são máquinas que contem *workers*. Estão em contacto permanente com o *worker manager*, que recolhe as métricas de utilização dos *workers* presentes no anterior. Essa informação é então utilizada para iniciar ou parar os *conteiners* dos *workers*. Quando iniciados, um *cluster* reconstrói a imagem do *docker* dos *workers* para lhes fornecer o JMS e o certificado do *cluster*, para que possam solicitar configurações de *pipelines* ao JMS e publicar regras recentemente criadas.

O *worker manager* é a parte do *broker* responsável pela recolha de informações sobre os vários *clusters* e os *workers* que neles trabalham. Deve ser capaz de solicitar um *cluster* para iniciar ou parar instâncias de *workers*, dependendo da métrica de desempenho desse *cluster*.

O *broker* é responsável por inicializar o *worker manager* e por distribuir a carga de trabalho entre os diferentes *workers*, após receber dados de uma instância JHC.

3.3 Adição de Componente

A principal adição ao JBriareos é a componente de deteção de chamadas ao sistema através de modelos de *machine learning*, aumentanto assim, a gama de ataques detetados, incluindo a deteção de anomalias na atividade do *host* em que o componente JHC está implantado.
Assim, conseguimos a construção de um sistema, com uma ampla gama de ataques detetáveis, incluindo ataques desconhecidos, tanto ao nível da rede, como ao nível do *host*.
A tabela seguinte mostra os parâmetros adicionados, com esta nova abordagem, ao sistema já existente.

	JBriareos	**JBriareos com Modelo de ML**
Tipo de Sistema	*Host*	*Host*
Deteção de Ataques	x	x
Prevenção de Ataques	x	x
Baseado em Assinatura	x	x
Baseado em Anomalia		x
Machine Learning		x

Tabela 3.1: Comparação entre as diferentes versões Briareos

Com o acréscimo do novo componente é esperado que o desempenho do sistema baixe, devido à adição do tempo de execução para a deteção das anomalias. No entanto, o principal objetivo é também o aumento da deteção de novos ataques, compensando assim a perda de performance.
A adição do modelo de ML implica o desenho da arquitetura do próprio modelo para receber os dados de treino, escolha dos dados a serem utilizados e as transformações aos dados. No Capítulo seguinte é descrito o desenho e a arquitetura do modelo implementado.

Capítulo 4

Desenho e Arquitetura

4.1 Descrição do Conjunto de Dados

Os investigadores da Academia das Forças de Defesa Australianas criaram dois conjuntos de dados (ADFA-LD e ADFA-WD) como conjuntos de dados públicos que representam a estrutura de ataques modernos.
Os conjuntos de dados contém registos dos sistemas operativos Linux e Windows e foram criados a partir da avaliação do sistema HIDS baseado em chamadas de sistema. O sistema operativo utilizado no *host* para construir o ADFA-LD foi o Ubuntu Linux versão 11.04.
Algumas das instâncias de ataque no ADFA-LD foram derivadas do novo *malware* de 'dia zero', tornando este conjunto de dados adequado para realçar as diferenças entre as abordagens de SIDS e AIDS na deteção de intrusão.

Com a utilização deste conjunto de dados, abrangemos uma gama de ataques detetáveis a partir do *host*. Através de chamadas ao sistema é possível identificar várias ações maliciosas, envolvidas em diferentes ataques distintos e que reflitam caraterísticas recentes de atividade computacional. Por este motivo, o conjunto de dados ADFA-LD é bastante pertinente para treinarmos classificadores a fim de distinguir atividade normal de atividade que potencie ações maliciosas para futuros ataques ao sistema.

O conjunto de dados está dividido em pasta de treino, pasta de validação e pasta de ataque. As pastas de treino e validação contém dados de atividade normal com chamadas de sistema normais do próprio sistema, ou seja, vestígios recolhidos utilizando um programa de auditoria enquanto o sistema estava a ser utilizado para operações normais, tais como, escrita de documentos, navegação na Internet, acesso a ficheiros, etc. A pasta de ataque, contém chamadas de sistema recolhidas durante os ataques contra o *host*.

Para simplificar, cada vestígio consiste em índices das chamada de sistema em vez dos respetivos nomes. O ficheiro *unistd.sh* no sistema operativo Linux pode ser referenciado para mapear os índices numéricos para os nomes das chamadas de sistema correspondentes [40].
A tabela seguinte mostra a descrição e a distribuição geral dos vestígios das chamadas de sistema.

	ADFA-LD		ADFA-WD	
Conjunto de Dados	***Traces***	**Chamadas de Sistema**	***Traces***	**Chamadas de Sistemas**
Dados de Treino	833	308 007	355	13 504 419
Dados de Validação	4732	2 122 085	1827	117 918 735
Dados de Ataque	746	317 388	5542	74 202 804
Total	5951	2 747 550	7724	205 625 958

Tabela 4.1: Descrição Geral dos Dados ADFA-LD

Os serviços Web são ativados no *host* através da instalação do Apache 2.2.17 [41] juntamente com o PHP 5.3.5 [42]. Para permitir o armazenamento de dados e de conteúdo dinâmico, foi instalado o MySQL 14.14 [43]. Os serviços Protocolo de Transferência de Ficheiros (*File Transfer Protocol*) (FTP) e *Shell* Segura (*Secure Shell*) (SSH) foram também iniciados pelas portas predefinidas para aceder remotamente ao *host*. Além disso, a aplicação web Tiki Wiki 8.1 [44] foi instalada e preparada para servir os utilizadores online. Esta versão da aplicação web é escolhida intencionalmente, uma vez que tem uma exploração crítica que permite a execução de código remoto através da injeção de código PHP. Com esta configuração, o servidor Linux é considerado realista possuindo estes serviços ativados com algumas vulnerabilidades existentes. Vários tipos de ataques foram selecionados e emitidos contra o kernel Linux, a fim de representar tentativas reais de atacar um sistema bem protegido.

A tabela seguinte mostra a repartição dos *payloads* e os vetores utilizados para atacar o sistema operativo Ubuntu. As duas primeiras classes de ataque representam tentativas de adivinhar senhas através de força bruta em serviços abertos. FTP e SSH são serviços normalmente disponíveis, e são regularmente atacados se forem expostos a fontes externas. Assim, a sua inclusão recpresenta uma ameaça realista, embora provavelmente originada por um atacante menos qualificado ou porque esgotou todas as outras opções de ataque.

O vetor de ataque seguinte é um executável do lado do cliente, que corresponde ao ataque *Adduser*. O aumento de privilégios num ambiente Linux pode ser extremamente difícil. Uma forma de contornar isto é criar um novo utilizador com privilégios de super utilizador. Isto pode ser feito de muitas maneiras diferentes, mas para este conjunto de dados foi utilizado um ataque do lado do cliente.

O Meterpreter é uma *shell* com funcionalidades adicionais, o que facilita muito o compromisso remoto de um sistema direcionado. Para a execução do ataque Meterpreter foi utilizada a vulnerabilidade *tiwiki*, para carregar uma cópia do *payload* do Java Meterpreter. Depois do executável ser carregado e executado, é iniciada uma ligação TCP do *host* local (acesso remoto) à máquina alvo. Uma vez estabelecida a shell, foram tomadas várias ações no sistema alvo, incluindo a escalada de privilégios locais, tentativas de acesso ao ficheiro de palavra-passe, e a instalação de ferramentas de *backdoor* persistentes.

Do mesmo modo, uma versão executável Linux do *payload* do Meterpreter foi carregada utilizando engenharia social. Embora a funcionalidade e interface do Meterpreter e do Java-

Meterpreter sejam semelhantes, são usadas *syscalls* diferentes e por conseguinte assinaturas diferentes.

O vetor de ataque utilizado para o ataque Webshell, foi a inclusão de um ficheiro php. Através da web, tirando partido de engenharia social, é colocada uma *shell* escondida através de uma interface web [40].

Ataque	***Payload***	**Vetor**	**Contagem**
Hydra-FTP	Força bruta da senha	FTP by Hydra	162
Hydra-SSH	Força bruta da senha	SSH Hydra	176
Adduser	Adicionar novo super utilizador	Executável envenenado do lado do cliente	91
Java-Meterpreter	Meterpreter baseado em Java	Exploração da vulnerabilidade Tikiwiki	124
Meterpreter	Linux Meterpreter payload	Executável envenenado do lado do cliente	75
Webshell	C100 webshell	Vulnerabilidade de inclusão de ficheiro remoto php	118

Tabela 4.2: Classe dos Ataques ADFA-LD/WD

Chamada de Sistema

A unidade de recolha para a coleção deste conjunto de dados é a chamada de sistema, que é o principal elemento a considerar na análise dos ficheiros de registo de um sistema baseado no Unix.
A chamada de sistema é definida como um pedido ao kernel que é gerado por um processo contínuo através de um mecanismo de interrupção. Este pedido é enviado para o kernel porque o processo ativo precisa de aceder a alguns recursos. Por conseguinte, uma chamada de sistema tem um grande impacto no estado do sistema e é exatamente um objeto de análise quando se trata de monitorização do sistema.
Este objeto de análise faz sentido ser investigado em cadeia em vez de forma isolada. Isto porque uma atividade maliciosa contém normalmente uma série de tarefas, cada uma das quais com um pedido de específico para o núcleo. Assim, a fim de identificar um padrão de ataque, uma série de chamadas de sistema sucessivas são recolhidas para serem posteriormente analisadas.
A fim de melhorar a utilidade dos dados, foi utilizada uma janela deslocada cronologicamente, com um determinado *overlap*, para efeitos de extração de caraterísticas.

4.2 Transformações ao Conjunto de Dados

O conjunto de dados ADFA-LD no seu formato original é constituído por milhares de ficheiros, em que cada ficheiro contém centenas de chamadas de sistema em sequência. Estes dados foram compilados em ambiente de sistema operativo Linux e como em qualquer sistema operativo típico, o sistema operativo Linux tem processos em execução durante o tempo de operação. Estes processos utilizam vários recursos do computador através de chamadas ao sistema. O conjunto de dados contém vestígios de chamadas de sistema que são sequências de chamadas de sistema efetuadas por vários processos durante o seu tempo de execução. Uma chamada de sistema é

representada como um número em vez do nome.
A figura seguinte representa parte de uma sub-sequência no ficheiro UTD- 8018 do conjunto de treino.



Figura 4.1: Representação de uma sub-sequência do ADFA-LD original

Com a disposição da informação nos ficheiros do conjunto de dados originais, torna-se complicada a tarefa de detetar e distinguir atividade normal e maliciosa. Assim, foi necessário aplicar transformações aos dados, de modo a facilitar a tarefa para deteção, e de modo a servir de *input* para os algoritmos de classificação. A nova disposição dos dados, tem de fazer sentido para o contexto do estudo em questão, para a própria natureza dos dados e para a sua aplicação real. Além disso, para obter um HIDS preciso, as caraterísticas extraídas devem conter informação suficiente que possa ser útil para diferenciar as chamadas de sistema normais das anormais. Inclusive a técnica de deteção deve ser suficientemente precisa para detetar anomalias. Assim como a precisão do HIDS, o tempo de aprendizagem do perfil normal/anormal, a velocidade de avaliação e o consumo de recursos são todas caraterísticas importantes para a implementação de um bom detetor de intrusões.

Para concluir o impacto que as chamadas de sistema têm na deteção de atividade maliciosa, foi necessário estudar várias abordagens que fizessem sentido no contexto.
Uma das abordagens mais pertinentes é a transformação da sequência com centenas chamadas de sistema, em pequenas sub-sequências, pois verificamos que através de um número reduzido de chamadas, é possível prever um ataque. Outra abordagem pertinente, é a análise de frequências de chamadas de sistema numa determinada janela, pois verificamos que num tamanho curto de sequência de chamadas de sistema, a repetição consistente de certas chamadas, indicava a tentativa de atividade maliciosa. Ambas as técnicas foram aplicadas ao conjunto de dados, variando também tamanhos de janela para comparação de resultados e verificação do impacto da mesma na deteção de intrusões.

4.2.1 Abordagem por Sequência

Em relação ao processo de funcionamento de extração de caraterísticas baseado na técnica de sequências, cada ficheiro de *input*, que contém até centenas de chamadas de sistema, é dividido em sequências curtas com base no tamanho de uma janela, ou seja, é utilizada uma técnica de janela deslizante para extrair sub-sequências.

O algoritmo recebe como *input* cada ficheiro individualmente e produz conjuntos de sequências distintas de tamanho w para esse ficheiro. O parâmetro "w"determina o número de chamadas de sistema numa sequência extraída, ou seja, é o tamanho da sequência.

Neste projeto, os valores de w utilizados foram 2, 3, 4 e 5, no entanto o utilizador pode escolher qual valor de janela quer utilizar. De acordo com um estudo realizado aos dados originais, estes valores foram os tamanhos de janela em que se verificou um melhor aproveitamento da informação do conjunto de dados. Este tema é abordado no capítulo seguinte.

Aplicando a técnica de extração das sequências a todos os ficheiros com atividade normal, é construída uma base de dados de perfil normal que contém sequências distintas para cada registo normal. Esta técnica é aplicada aos dados na totalidade, incluindo dados anormais também.
Através do conhecimento das diferentes sequências com a respetiva classe a que pertencem, é construído um padrão de comportamento, de acordo com os diferentes algoritmos, e posteriormente, a atribuição da classe nos dados de teste, consoante o conhecimento adquirido.

Uma vantagem desta técnica é que preserva a informação posicional das chamadas de sistema, resultando na criação de um perfil preciso e teoricamente diminui a possibilidade de ataques de mímica ao IDS.
No processo de aprendizagem e classificação, foi utilizado um algoritmo de RNN chamado LSTM.

Para uma melhor compreensão do funcionamento desta abordagem, é ilustrado na figura seguinte uma transformação exemplo das primeiras três extrações de sequências aplicadas ao ficheiro UAD-Adduser-10-2311, que faz parte de dados com atividade maliciosa.
Neste exemplo, é utilizada uma janela deslizante de tamanho 5, e *overlap* de tamanho 1, que se mantém fixo.

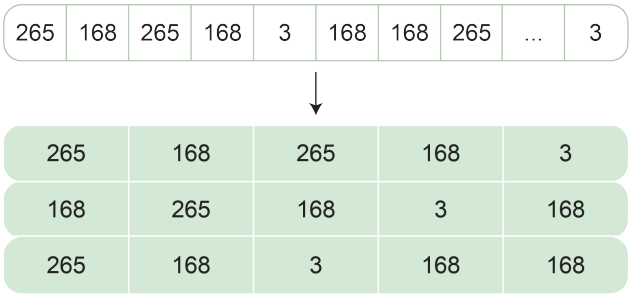


Figura 4.2: Transformação aos Dados Originais por Sequência

4.2.2 Abordagem por Frequência

O conjunto de dados ADFA-LD, no seu formato original, está divido em pastas, cujas possuem dados de atividade normal e maliciosa. Cada pasta contém então centenas de ficheiros, e cada ficheiro contém um enorme sequência de chamadas de sistema.
Na técnica de frequência, é feita uma contagem de chamadas de sistema individuais, num tamanho de janela deslizante definida.

O algoritmo recebe como *input* cada ficheiro individualmente e produz novas linhas de entrada distintas, em que cada instância corresponde à contagem de chamadas de sistema que estão presentes nas sub-sequências de tamanho w. O parâmetro "w"determina o número de chamadas de sistema da sequência extraída para análise das frequências.
Assim como no estudo das sequências, os valores de w utilizados foram 2, 3, 4 e 5, para comparação de resultados tanto variando a técnica aplicada utilizando o mesmo tamanho de janela, como comparação de resultados entre janelas diferentes, para averiguar o impacto que o tamanho das sub-sequências de chamadas ao sistema tem no estudo de deteção de intrusões.

Aplicando a técnica de contagem das chamadas de sistema a todos os ficheiros com atividade tanto normal como anormal, é construída uma base de dados de perfil normal e anormal que contém tabelas de frequências distintas para cada natureza dos registos normal e registos de ataque.
Depois da transformação dos dados para este formato, os algoritmos de classificação aplicados, aprenderam o padrão dos dados normais e anómalos consoante a classe que está atribuída para na fase seguinte classificarem dados novos e serem detetados ataques.

Uma vantagem desta técnica é o facto de uma forma geral o tempo execução na aprendizagem, classificação e teste ser relativamente rápido, dependendo do algoritmo utilizado.

No processo de aprendizagem e classificação, foram utilizados vários algoritmos e diferentes tamanhos de janela para comparação de resultados. Os algoritmos utilizados nesta abordagem foram: KNN, *Naïve Bayes*, RF e SVM.

Para uma melhor compreensão do funcionamento desta abordagem, é ilustrado na figura seguinte uma transformação exemplo das primeiras três extrações de sub-sequências para contagem da frequência. Estes dados são também retirados do ficheiro UAD-Adduser-10-2311, que faz parte de dados com atividade de ataque.
Neste exemplo, é utilizada uma janela deslizante de tamanho 5, e *overlap* de tamanho 1, que se mantém fixo.

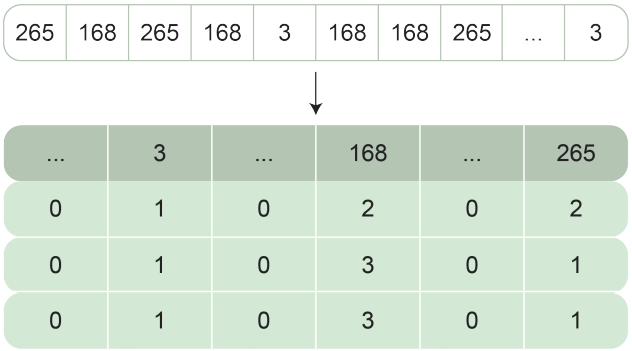


Figura 4.3: Transformação aos Dados Originais por Frequência

4.3 Modelo Geral

Em cada uma das abordagens descritas nas secções anteriores, o processo de funcionamento, desde a transformação dos dados até à fase de teste e atribuição da classe pertencente, é semelhante. No entanto, a transformação dos dados originais, para extração de caraterísticas é diferente, variado pela própria abordagem, como foi analisado nas secções anteriores.

O modelo geral é constituído por três elementos principais: extração de caraterísticas, treino dos dados e classificação, e por fim, geração do modelo de deteção. Depois de obtermos o modelo de deteção, são introduzidos dados novos, onde será efetuado o teste e validação do modelo.
A extração de caraterísticas é um componente crucial para o HIDS, pois extrai caraterísticas informativas e representativas dos dados originais. Se as caraterísticas extraídas forem irrelevantes, então os algoritmos de classificação não dariam bons resultados, e o HIDS não executaria a sua função.
Na fase de treino, o algoritmo de classificação toma como *input* dados de atividade normal e dados de atividade maliciosa, para aprender padrões de acordo com o funcionamento próprio do algoritmo em questão e posteriormente classificar dados novos na fase de teste. O *output* desta fase é o modelo de deteção. Este modelo, na fase seguinte, fase de avaliação, toma como input dados novos, com informação normal e maliciosa. Com base no modelo de deteção, na fase de avaliação, é decidido se os dados de *input* correspondem a ações caraterísticas de um utilizador normal, ou se correspondem a ações caraterísticas de ataque. Caso se verifique, que é uma ação maliciosa, será emitido um alarme. É também nesta fase, que é verificada a eficácia e performance dos modelos de classificação, comparando a percentagem de casos em que o algoritmo acerta na atribuição da classe 0, ou classe 1. Sendo a classe 0 referente a atividade normal e a classe 1 a atividade anormal.
A figura seguinte mostra o funcionamento do componente HIDS de deteção.

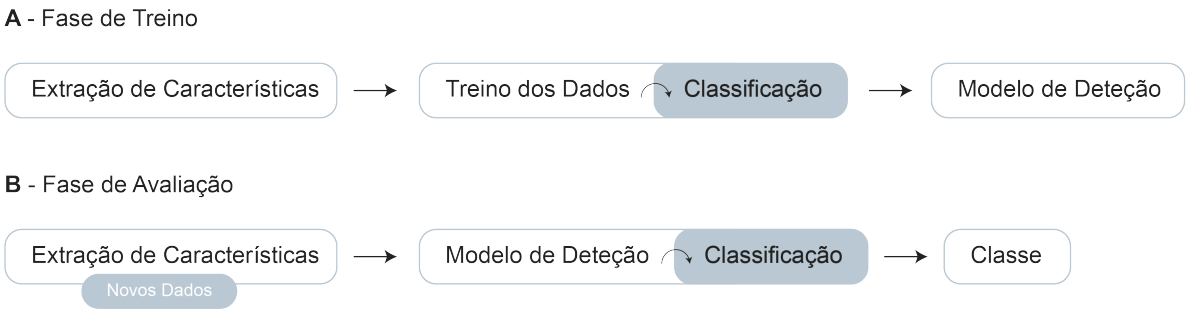


Figura 4.4: Modelo Geral do HIDS Proposto

Capítulo 5

Implementação dos Modelos e Resultados

Neste Capítulo são apresentados os resultados para as diferentes abordagens descritas nas secções anteriores (secções 4.2.1 e 4.2.2).
Para cada uma das abordagens, sequência e frequência, foram gerados resultados para vários tamanhos de janela, usando algoritmos diferentes, para comparação de resultados e discussão das possíveis causas da discrepância entres eles.

De modo a gerar valores mais realistas, foram realizados 4 testes para cada modelo de deteção, em cada uma das técnicas. Para o primeiro teste é utilizado 30% dos dados para treino, incidindo nos dados da pasta de treino (ficheiro UTD-0418 até UTD-0833) com atividade normal e todos os ficheiros de dados das pastas 1 até à 5, representativas de cada ataque. O segundo teste utiliza a pasta de validação para dados de atividade normal (ficheiro UVD-0001 até UVD-0390), e todos os ficheiros que constam da pasta 6 até à 10, de cada ataque diferente. No terceiro teste, para dados de atividade normal é utilizado o ficheiro UVD-0391 até ao UVD-0753, e todas os ficheiros das pastas 3 até à 7 de cada tipo de ataque. Por último, no último teste, é utilizada outra fração dos dados reservados para treino.
Todos os testes realizados aos modelos, utilizam o modelo de deteção guardado gerado durante o treino. De modo a obter resultados realistas, os dados utilizados em cada um dos testes são diferentes e novos para o classificador. Para cada uma das abordagens diferentes foram realizados o mesmo número de testes, utilizando exatamente os mesmos dados. Assim, podemos comparar e discutir os resultados, alterando apenas entre as duas técnicas e os algoritmos utilizados em cada uma delas, para a escolha dos melhores modelos.

Uma caraterística importante nos resultados apresentados nas secções seguintes, é a escolha da janela. Na fase de transformação dos dados, foram utilizadas janelas de 2 a 7 e 10, para ambas as técnicas. O objetivo é comparar os resultados obtidos de ambas as abordagens utilizando a mesma janela, e verificar o impacto do número de chamadas de sistema para a deteção de intrusões.

A partir dos dados originais, foi realizada uma distribuição, com vários valores de janela, para verificar que valores faziam mais sentido ser utilizados. Verificou-se através da apresentação dos gráficos, uma distribuição exponencial, como esperado. Ou seja, à medida que o valor da janela aumentava, a ocorrência de repetição de padrões diminuía. Verificou-se também que a partir do valor 10, deixavam de existir ocorrências de repetições de padrões. Como é lógico, com um valor de janela baixo, por exemplo, um valor igual a 2, o número de ocorrências de padrões de chamadas de sistema é bastante superior a um valor de janela igual a 5. No entanto, o objetivo é utilizar valores cujos resultados sejam representativos de padrões, seja padrões de ataque, seja padrões de atividade normal. Então foram utilizados vários valores de dimensão de janela considerados pertinentes na transformação dos dados para comparação de resultados.

Para comparação dos resultados dos 4 testes efetuados a cada modelo, são apresentados gráficos da Curva Caraterística de Operação do Receptor (*Receiver Operating Characteristic Curve*) (ROC) com o valor de Área sob a Curva (*Area Under the Curve*) (AUC) de cada um dos testes. Estas ferramentas são poderosas para medir e comparar o desempenho de modelos de classificação binários, em *machine learning*.
A curva ROC é um gráfico robusto, que permite estudar a relação da taxa de verdadeiros positivos e da taxa de falsos positivos, ou seja, número de vezes que o classificador acertou a previsão em relação ao número de vezes que o classificador errou a previsão. Quanto mais a curva ROC se aproxima do canto superior esquerdo do gráfico melhor é o classificador.
A AUC resume a curva ROC num único valor, calculando a "área sob a curva", sendo uma medida de área que facilita a comparação entre curvas ROC, pois em alguns casos é difícil comparar visualmente os modelos, pois as curvas podem se cruzar em diferentes pontos [45]. Esta medida representa o grau de separabilidade, quanto maior o seu valor, melhor o modelo está a efetuar a sua função de prever 0s como 0s e 1s como 1s. Isto é, quanto maior o valor AUC, melhor o modelo está em distinguir entre atividade maliciosa e atividade normal.

Nas tabelas das secções seguintes são apresentados os resultados referentes à *precision* e à *recall* de cada teste para cada tamanho de janela diferente.
No contexto de *machine learning*, *precision* e *recall* são métricas de desempenho para algoritmos de classificação.
Considerando uma tarefa de classificação com duas classes, precisão é quantas vezes ocorre uma previsão correta de uma determinada classe em relação ás falsas previsões dessa classe. *Recall* é a percentagem dos dados pertencentes a uma determinada classe que o modelo prevê corretamente como pertencendo a essa classe.
Ou seja, *precision* é definida como a quantidade de verdadeiros positivos dividida pela soma de verdadeiros positivos e falsos positivos. A *recall* é definida como a quantidade de verdadeiros positivos dividida pela soma dos verdadeiros positivos e dos falsos negativos.

Neste caso, é muito importante que o modelo detete de facto a atividade maliciosa, e não deixe passar por atividade normal. Pois se o modelo classificasse muitas ocorrências que são ataque como atividade normal, teríamos um modelo inútil. Isto traduz-se na importância do valor da métrica *recall* ser elevada, pois significa um número baixo de falsos negativos. Por outro

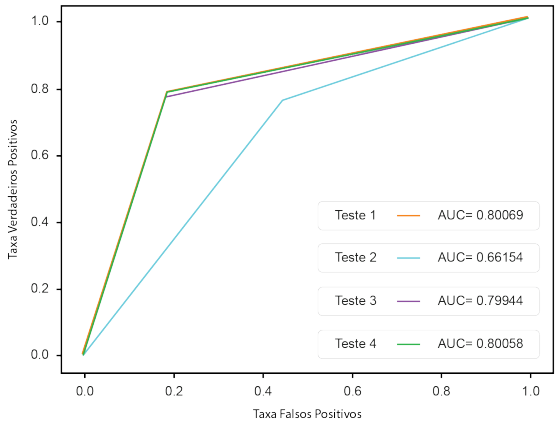
lado, a métrica *precision* também é extremamente importante, pois dita os falsos positivos. Não é do nosso interesse construir um sistema de deteção de intrusões que possua uma taxa baixa de *precision*, pois traduz-se em muitos falsos positivos, ou seja, casos em que o classificador assume aquela atividade como maliciosa, e na verdade é atividade normal.

Nas duas secções seguintes são apresentados os resultados dos testes executados, utilizando o algoritmo LSTM para a abordagem de sequência, e quatro algoritmos clássicos para abordagem de frequência. De seguida, é feita a escolha de qual modelo faz mais sentido para o caso de estudo de deteções de intrusões, de acordo com os resultados obtidos.

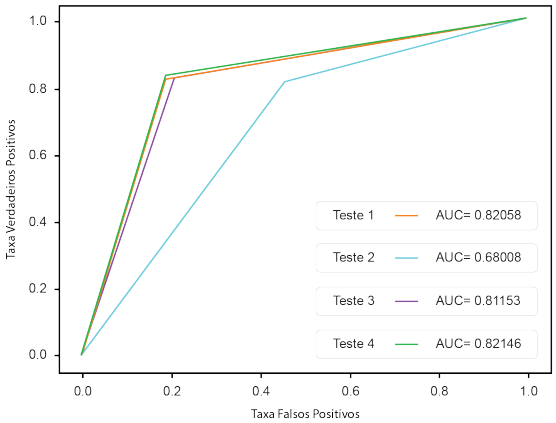
5.1 Sequências

Na fase de geração de resultados para a técnica das sequências, como foi dito anteriormente, foram realizados testes para valores de janela entre 2 a 7 e 10.
As Figuras seguintes (Figuras 5.1, 5.2, 5.3 e 5.4) dizem respeito aos testes efetuados, após o treino dos dados. Cada gráfico resume a performance dos testes para cada valor de janela utilizando o algoritmo de classificação LSTM, com uma camada de *embedding* de dimensão igual a 32, uma camada densa com a função de ativação *sigmoid* e o treino utiliza 30 *epochs*.

Podemos verificar de forma geral que os testes que obtiveram melhor performance foram o primeiro e o último. Isto deve-se ao facto, de durante a fase de teste dos testes 1 e 4 terem sido usados dados diferentes, no entanto fazem parte da mesma pasta dos dados que foram usados durante a fase de treino, podendo haver semelhança no padrão desses dados.

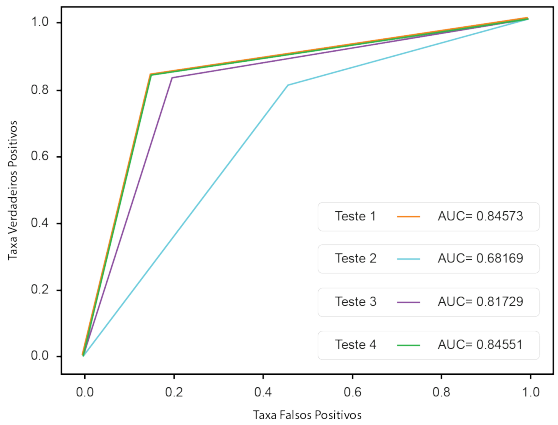


(a) Curva ROC para Janela de 2 (W=2)

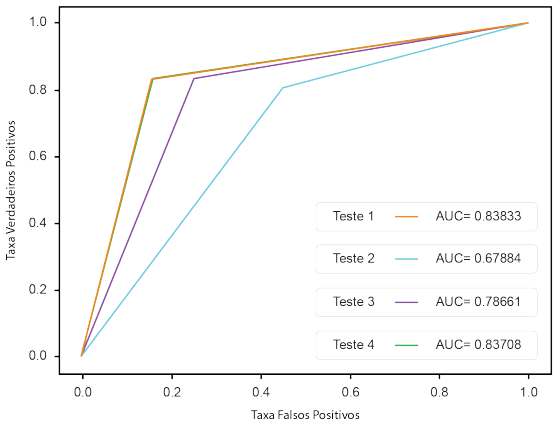


(b) Curva ROC para Janela de 3 (W=3)

Figura 5.1: Curvas ROC para Janela de 2 e 3

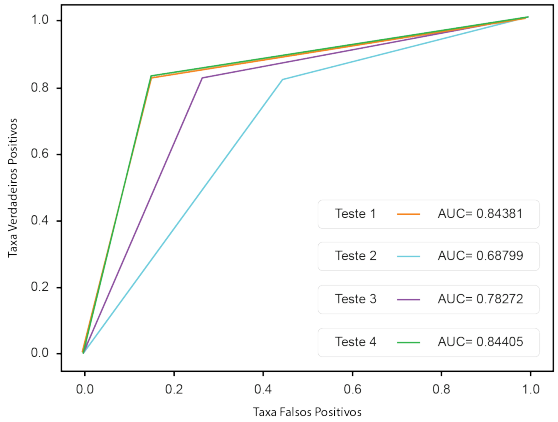


(a) Curva ROC para Janela de 4 (W=4)

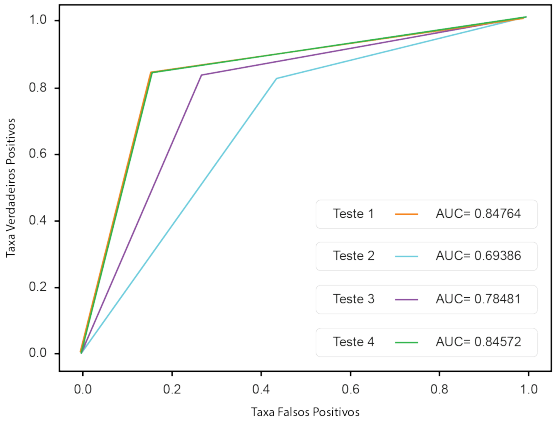


(b) Curva ROC para Janela de 5 (W=5)

Figura 5.2: Curvas ROC para Janela de 4 e 5



(a) Curva ROC para Janela de 6 (W=6)



(b) Curva ROC para Janela de 7 (W=7)

Figura 5.3: Curvas ROC para Janela de 6 e 7

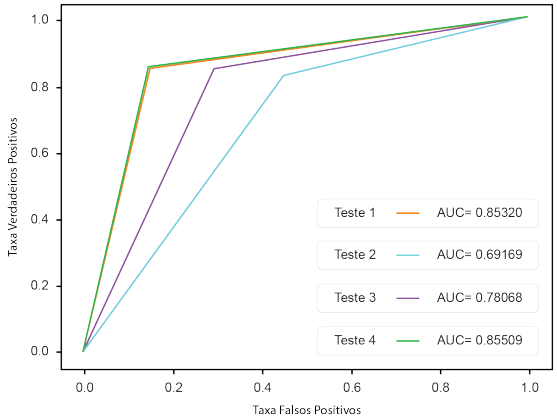


Figura 5.4: Curva ROC para Janela de 10 (W=10)

Na Tabela 5.1 são apresentados os valores das métricas *precision* e *recall* correspondente a cada teste, na respetiva janela e na Tabela 5.2 as médias dos resultados de teste e tempos de execução da fase de teste e da fase de treino (T.M.E).
Podemos verificar que à medida que a janela aumenta, o valor da *recall* aumenta, havendo pequenas alterações no valor da *precision*. Ou seja, à medida que o tamanho da janela de chamadas de sistema aumenta, o número de falsos negativos diminui, pois o valor da *recall* aumenta.
Estes resultados devem-se ao facto dos padrões da atividade maliciosa e atividade normal, ficarem mais claros para o classificador à medida que a janela aumenta. Com janelas maiores, acaba por haver menos "ruído"nos dados, ou seja, as pequenas sequências que tanto podem corresponder a padrões de ataque como a atividade maliciosa são eliminadas, acabando por se tornar mais clara a diferença entre as sequências de ataque e as sequências de atividade normal.
Em relação aos tempos de execução da fase de treino, tendem a diminuir à medida que a janela aumenta, o que é expectável, pois o número de instâncias do conjunto de dados é menor em janelas maiores. Os tempos de execução da fase de teste tende a oscilar, não se verificando nenhum padrão.
Concluindo, verifica-se que à medida que a janela de chamadas de sistema aumenta, a performance do modelo de sequências também aumenta. Com uma janela igual a 10 conseguimos um menor número de falsos negativos e um tempo de execução de treino menor também, em relação a uma janela igual a 2, e assim sucessivamente.

	W=2		**W=3**		**W=4**		**W=5**		**W=6**		**W=7**		**W=10**	
	Precision	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***
1º Teste	**0.82**	**0.79**	0.85	0.83	0.86	0.84	0.85	0.83	0.85	0.84	0.86	0.85	**0.86**	**0.86**
2º Teste	**0.54**	**0.77**	0.55	0.81	0.55	0.82	0.55	0.81	0.56	0.82	0.56	0.83	**0.55**	**0.83**
3º Teste	**0.84**	**0.78**	0.83	0.83	0.84	0.83	0.80	0.83	0.79	0.83	0.79	0.84	**0.78**	**0.85**
4º Teste	**0.82**	**0.79**	0.85	0.83	0.86	0.84	0.85	0.83	0.85	0.84	0.86	0.85	**0.86**	**0.86**

Tabela 5.1: *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo LSTM

	W=2		W=3		W=4		W=5		W=6		W=7		W=10	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
Média	**0.76**	**0.78**	0.77	0.82	0.78	0.83	0.76	0.83	0.76	0.83	0.77	0.84	**0.76**	**0.85**
T.M.E. Fase de Treino (seg)	43076.399		43704.398		60245.339		46579.637		27743.851		28876.2510		28525.587	
T.M.E. Fase de Teste (seg)	5.839		3.826		4.087		5.214		3.659		3.791		4.144	

Tabela 5.2: Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo LSTM

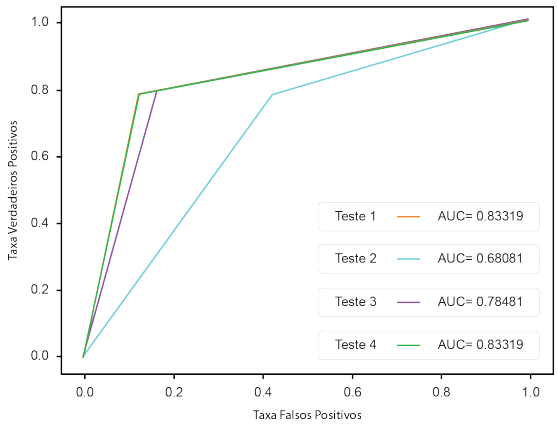
5.2 Frequências

Na fase de geração de resultados para a técnica das frequências, como foi dito anteriormente, foram realizados testes para valores de janela entre 2 a 7 e 10, assim como na abordagem anterior. As Figuras seguintes dizem respeito aos testes efetuados, após o treino dos dados. Cada gráfico resume a performance dos testes para cada valor de janela utilizando os algoritmos de classificação KNN, *Naïve Bayes*, RF e SVM.

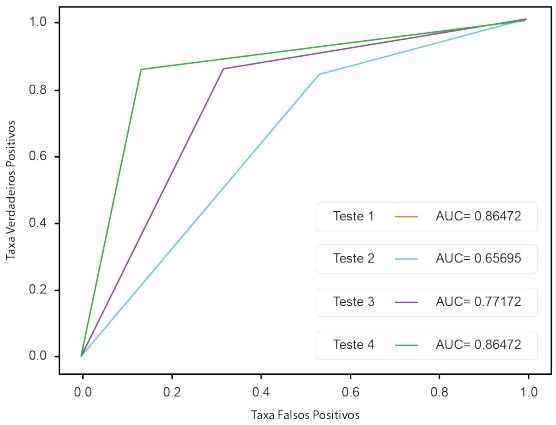
Podemos verificar de forma geral que os testes que obtiveram melhor performance foram o primeiro e o último. Os dados utilizados para teste foram os mesmos dados utilizados para teste da abordagem de sequência. Portanto era expectável que os resultados se assemelhassem.
Durante a fase de teste dos testes 1 e 4 foram usados dados diferentes, no entanto fazem parte da mesma pasta dos dados que foram usados durante a fase de treino, podendo haver semelhança no padrão desses dados. Por isso, os resultados são consideravelmente melhores, visto que o treino foi executado com dados com padrões parecidos e o classificador obteve melhor classificação de acerto.
Em relação aos resultados apresentados nas Tabelas 5.4, 5.6, 5.8 e 5.10 verificamos que à medida que a janela de chamadas de sistema aumenta, a performance do modelo de frequências usando os diferentes algoritmos também aumenta. Com uma janela igual a 10 conseguimos um menor número de falsos negativos e falsos positivos, em relação a uma janela igual a 2.
Os tempos de execução diferem bastante consoante o algoritmo. O KNN é o algoritmo mais rápido na fase de treino, e o algoritmo *Naïve Bayes* é o mais rapido na fase de teste. No entanto a performance de classificação do algoritmo *Naïve Bayes* é muito baixa, sendo o RF o algoritmo seguinte mais rápido da fase de teste, com a melhor taxa de *precision* e *recall*. O SVM é o terceiro algoritmo com melhor taxa de *precision* e *recall*, a seguir ao RF e ao KNN, e apresenta tempos de execução de treino e teste elevados. O algoritmo *Naïve Bayes* embora possua tempos de execução bons, apresenta taxas de acerto muito baixas, sendo por isso pouco pertinente neste estudo.

KNN

Os gráficos das Figuras 5.5, 5.6, 5.7 e 5.8 dizem respeito à curva de ROC dos 4 testes executados do algoritmo KNN, para cada valor de janela.
A Tabela 5.3 resume em valores numéricos a *precision* e a *recall* de cada teste para cada janela e a Tabela 5.4 apresenta os valores da média dos testes dessas métricas, assim como os tempos médios de execução (T.M.E) de treino e teste.

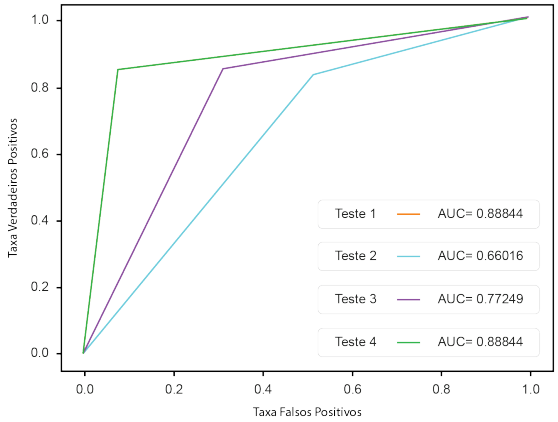


(a) Curva ROC para Janela de 2 (W=2) - KNN

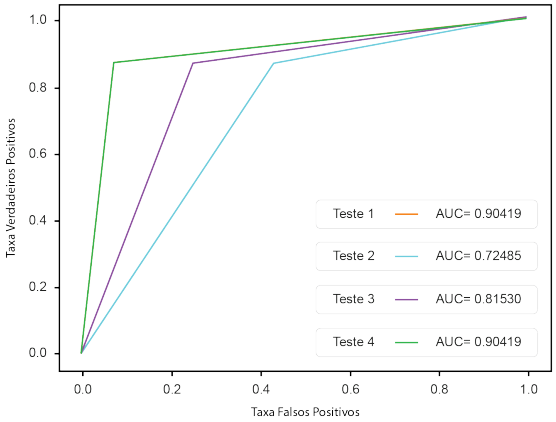


(b) Curva ROC para Janela de 3 (W=3) - KNN

Figura 5.5: Curvas ROC para Janela de 2 e 3 - KNN

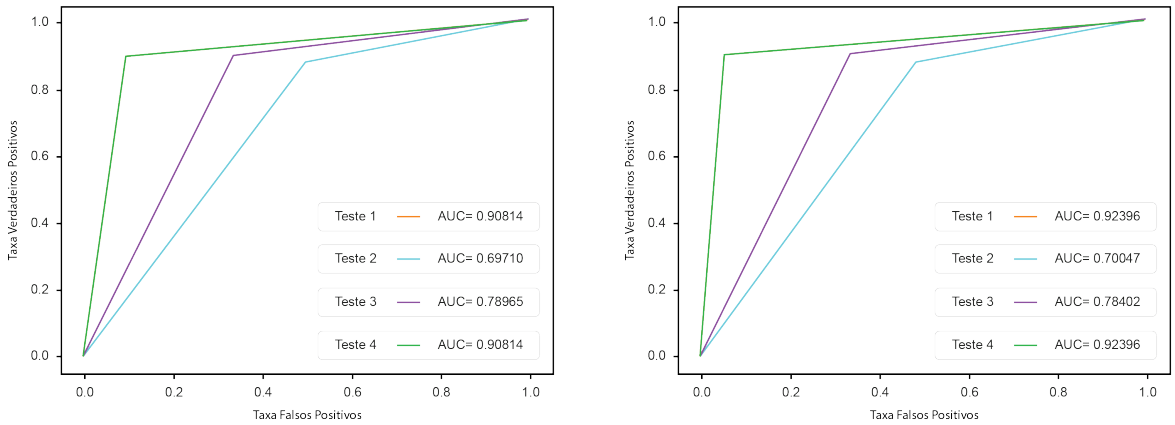


(a) Curva ROC para Janela de 4 (W=4) - KNN



(b) Curva ROC para Janela de 5 (W=5) - KNN

Figura 5.6: Curvas ROC para Janela de 4 e 5 - KNN



(a) Curva ROC para Janela de 6 (W=6) - KNN (b) Curva ROC para Janela de 7 (W=7) - KNN

Figura 5.7: Curvas ROC para Janela de 6 e 7 - KNN

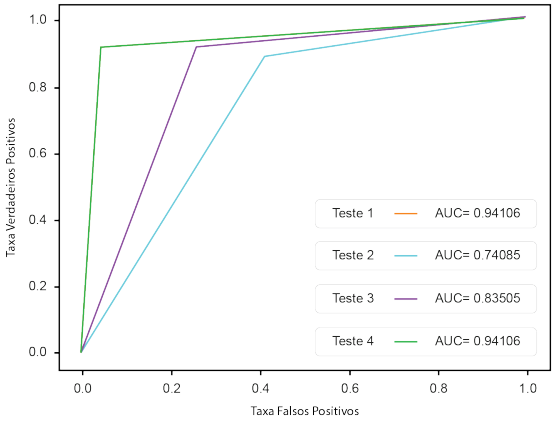


Figura 5.8: Curva ROC para Janela de 10 (W=10) - KNN

	W=2		W=3		W=4		W=5		W=6		W=7		W=10	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
1º Teste	**0.87**	**0.79**	0.87	0.86	0.92	0.86	0.93	0.88	0.92	0.91	0.94	0.91	**0.96**	**0.92**
2º Teste	**0.56**	**0.78**	0.52	0.84	0.52	0.84	0.59	0.88	0.55	0.88	0.55	0.88	**0.57**	**0.89**
3º Teste	**0.86**	**0.79**	0.77	0.86	0.77	0.86	0.82	0.88	0.77	0.91	0.77	0.91	**0.82**	**0.92**
4º Teste	**0.87**	**0.79**	0.87	0.86	0.92	0.86	0.93	0.88	0.92	0.91	0.94	0.91	**0.96**	**0.92**

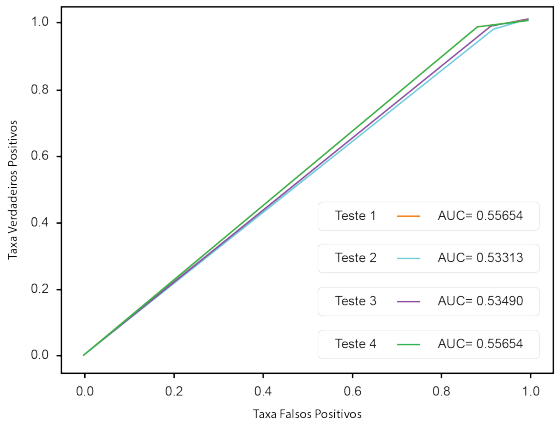
Tabela 5.3: *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo KNN

	W=2		W=3		W=4		W=5		W=6		W=7		W=10	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
Média	**0.79**	**0.79**	0.75	0.85	0.78	0.85	0.81	0.88	0.79	0.90	0.80	0.90	**0.83**	**0.91**
T.M.E. Fase de Treino (seg)	0.397		0.319		0.354		0.380		0.374		0.300		0.518	
T.M.E. Fase de Teste (seg)	256.226		254.421		254.339		255.826		305.479		304.905		306.997	

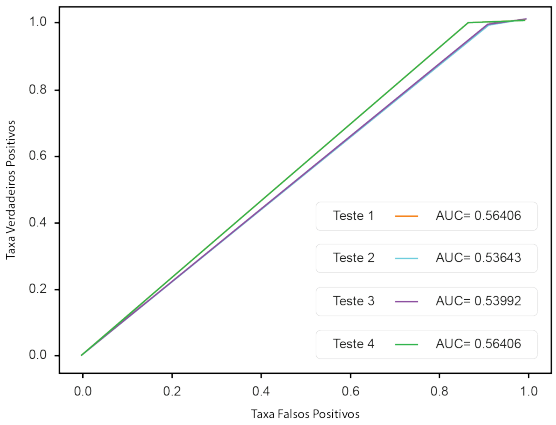
Tabela 5.4: Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo KNN

Naïve Bayes

Os gráficos das Figuras 5.9, 5.10, 5.11 e 5.12 dizem respeito à curva de ROC dos 4 testes executados do algoritmo *Naïve Bayes*, para cada valor de janela.
A Tabela 5.5 resume em valores numéricos a *precision* e a *recall* de cada teste para cada janela e a Tabela 5.6 apresenta os valores da média dos testes dessas métricas, assim como os tempos médios de execução (T.M.E) de treino e teste.
Podemos verificar que os valores de AUC se aproximam de 0.50, apresentando assim uma curva no gráfico que se posiciona muito distanciada do canto superior esquerdo. Isto é indicador de um classificador que se aproxima de classificar à sorte, sendo por isso, um mau algoritmo de classificação neste contexto, independentemente da janela de chamadas de sistema utilizada.

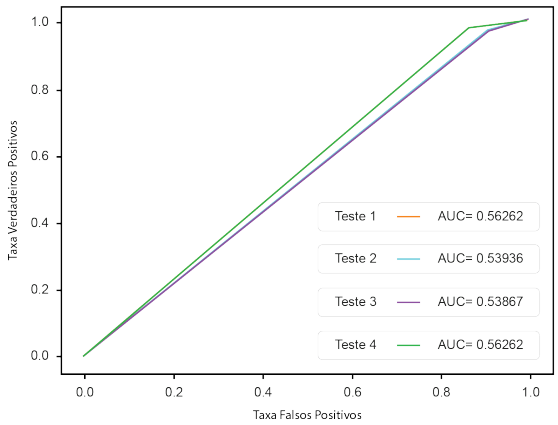


(a) Curva ROC para Janela de 2 (W=2) - *Naïve Bayes*

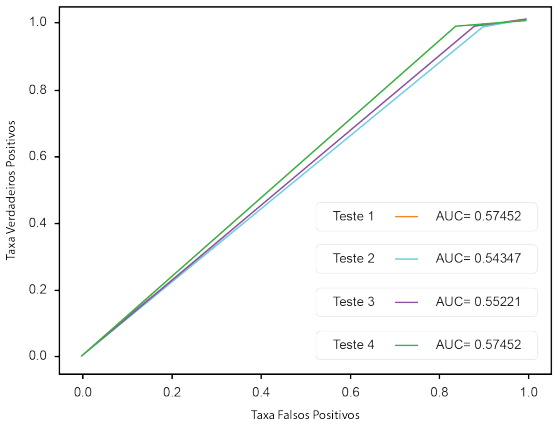


(b) Curva ROC para Janela de 3 (W=3) - *Naïve Bayes*

Figura 5.9: Curvas ROC para Janela de 2 e 3 -*Naïve Bayes*

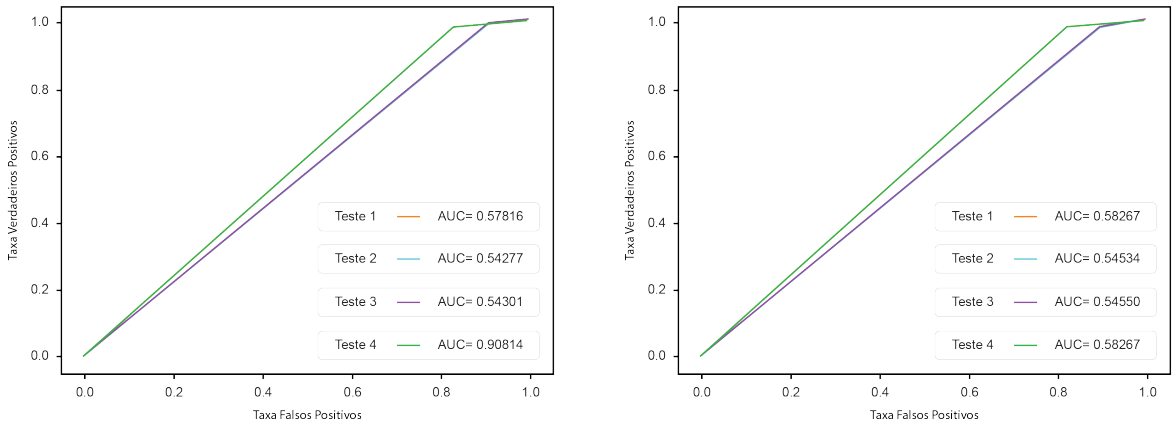


(a) Curva ROC para Janela de 4 (W=4) - *Naïve Bayes*



(b) Curva ROC para Janela de 5 (W=5) - *Naïve Bayes*

Figura 5.10: Curvas ROC para Janela de 4 e 5 - *Naïve Bayes*



(a) Curva ROC para Janela de 6 (W=6) - *Naïve Bayes* (b) Curva ROC para Janela de 7 (W=7) - *Naïve Bayes*

Figura 5.11: Curvas ROC para Janela de 6 e 7 - *Naïve Bayes*

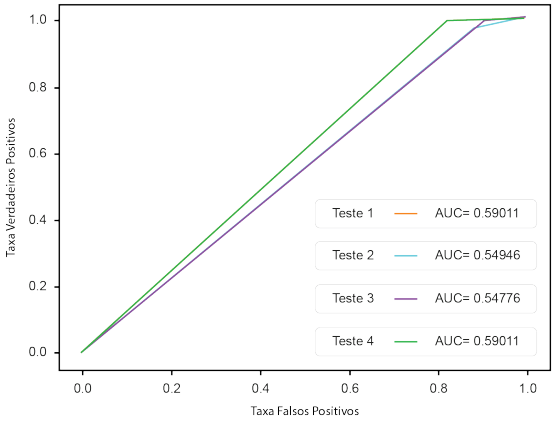


Figura 5.12: Curva ROC para Janela de 10 (W=10) -*Naïve Bayes*

	W=2		**W=3**		**W=4**		**W=5**		**W=6**		**W=7**		**W=10**	
	Precision	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***
1º Teste	0.55	0.99	0.55	0.99	0.55	0.99	**0.56**	**0.99**	0.56	0.99	0.56	0.99	0.56	0.99
2º Teste	0.42	0.98	0.42	0.98	0.42	0.98	**0.43**	**0.99**	0.42	0.98	0.42	0.98	0.40	0.97
3º Teste	0.58	0.99	0.57	0.99	0.57	0.99	**0.58**	**0.99**	0.57	0.99	0.58	0.99	0.58	0.99
4º Teste	0.55	0.99	0.55	0.99	0.55	0.99	**0.56**	**0.99**	0.56	0.99	0.56	0.99	0.56	0.99

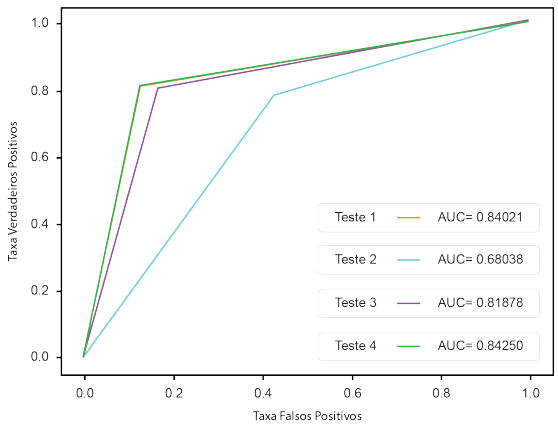
Tabela 5.5: *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo *Naïve Bayes*

	W=2		**W=3**		**W=4**		**W=5**		**W=6**		**W=7**		**W=10**	
	Precision	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***
Média	0.53	0.99	0.53	0.99	0.53	0.99	0.53	0.99	0.53	0.99	0.53	0.99	0.53	0.99
T.M.E. Fase de Treino (seg)	0.815		0.725		0.621		0.752		0.835		0.669		0.698	
T.M.E. Fase de Teste (seg)	0.243		0.199		0.202		0.249		0.233		0.191		0.217	

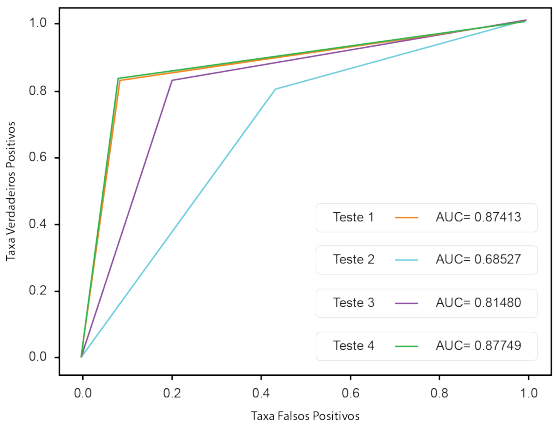
Tabela 5.6: Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo *Naïve Bayes*

RF

Os gráficos das Figuras 5.13, 5.14, 5.15 e 5.16 dizem respeito à curva de ROC dos 4 testes executados do algoritmo RF, para cada valor de janela.
A Tabela 5.7 resume em valores numéricos a *precision* e a *recall* de cada teste para cada janela e a Tabela 5.8 apresenta os valores da média dos testes dessas métricas, assim como os tempos médios de execução (T.M.E) de treino e teste.
Como podemos verificar, a curva de ROC gerada a partir dos resultados do modelo RF se apresenta mais próxima do canto superior esquerdo em relação aos restantes modelos, com valores de AUC mais próximos de 1.

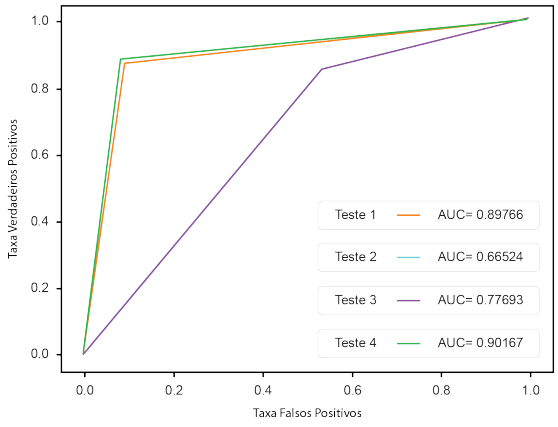


(a) Curva ROC para Janela de 2 (W=2) - RF

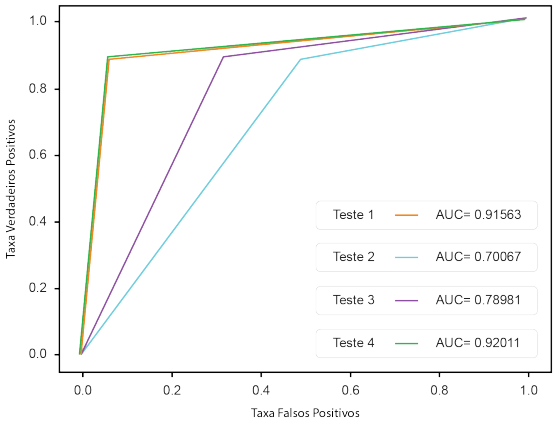


(b) Curva ROC para Janela de 3 (W=3) - RF

Figura 5.13: Curvas ROC para Janela de 2 e 3 -RF

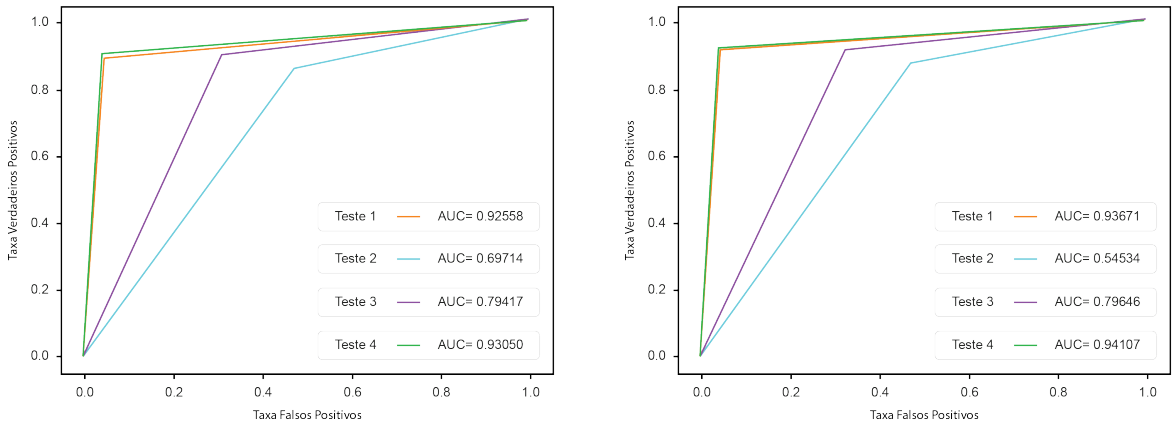


(a) Curva ROC para Janela de 4 (W=4) - RF



(b) Curva ROC para Janela de 5 (W=5) - RF

Figura 5.14: Curvas ROC para Janela de 4 e 5 - RF



(a) Curva ROC para Janela de 6 (W=6) - RF

(b) Curva ROC para Janela de 7 (W=7) - RF

Figura 5.15: Curvas ROC para Janela de 6 e 7 - RF

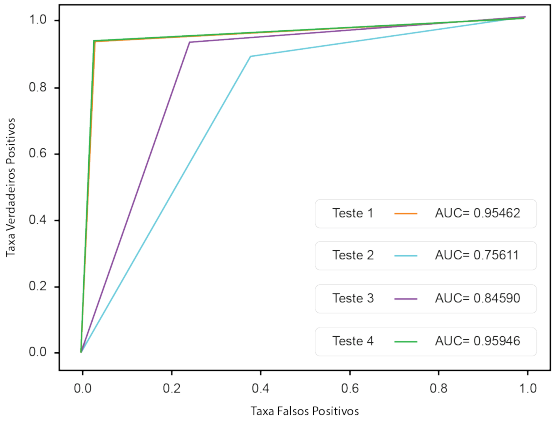


Figura 5.16: Curva ROC para Janela de 10 (W=10) - RF

	W=2		**W=3**		**W=4**		**W=5**		**W=6**		**W=7**		**W=10**	
	Precision	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***
1º Teste	0.87	0.81	0.91	0.83	0.92	0.88	0.94	0.89	0.95	0.90	0.95	0.92	**0.97**	**0.94**
2º Teste	0.56	0.78	0.55	0.80	0.52	0.85	0.56	0.89	0.55	0.87	0.56	0.89	**0.58**	**0.89**
3º Teste	0.86	0.80	0.84	0.83	0.77	0.88	0.78	0.90	0.78	0.90	0.78	0.92	**0.82**	**0.94**
4º Teste	0.87	0.81	0.92	0.84	0.92	0.89	0.94	0.90	0.96	0.91	0.96	0.92	**0.97**	**0.95**

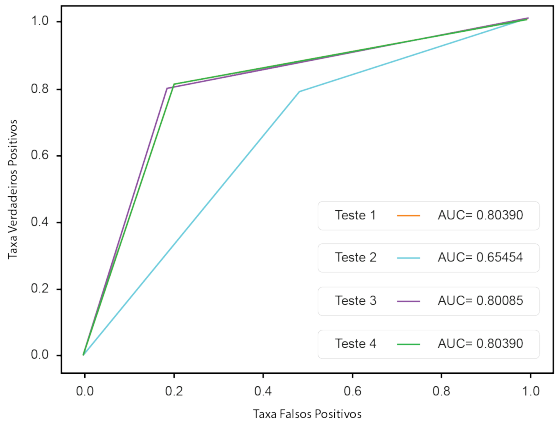
Tabela 5.7: *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo RF

	W=2		**W=3**		**W=4**		**W=5**		**W=6**		**W=7**		**W=10**	
	Precision	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***	***Precision***	***Recall***
Média	0.79	0.80	0.81	0.83	0.78	0.88	0.81	0.90	0.81	0.90	0.81	0.91	**0.83**	**0.93**
T.M.E. Fase de Treino (seg)	48.790		23.993		23.018		28.862		41.798		40.581		48.176	
T.M.E. Fase de Teste (seg)	1.052		1.785		1.291		1.471		1.323		1.315		1.353	

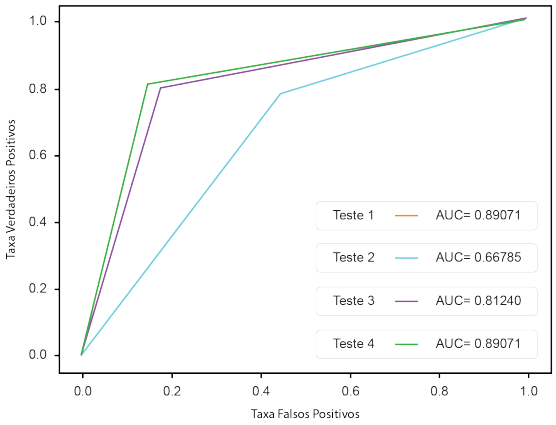
Tabela 5.8: Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo RF

SVM

Os gráficos das Figuras 5.17, 5.18, 5.19 e 5.20 dizem respeito à curva de ROC dos 4 testes executados do algoritmo SVM, para cada valor de janela.
A Tabela 5.9 resume em valores numéricos a *precision* e a *recall* de cada teste para cada janela e a Tabela 5.10 apresenta os valores da média dos testes dessas métricas, assim como os tempos médios de execução (T.M.E) de treino e teste.

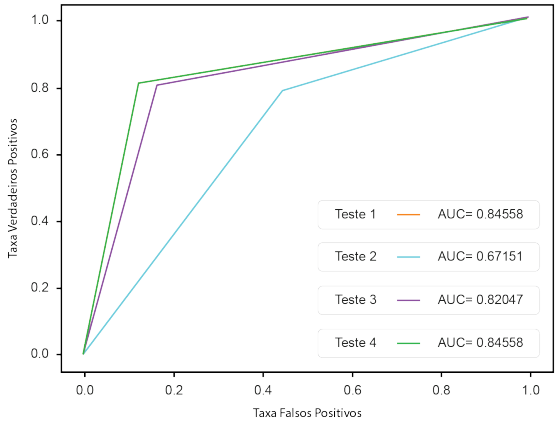


(a) Curva ROC para Janela de 2 (W=2) - SVM

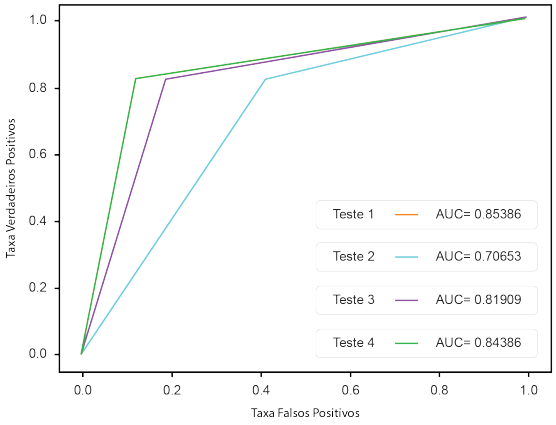


(b) Curva ROC para Janela de 3 (W=3) - SVM

Figura 5.17: Curvas ROC para Janela de 2 e 3 -SVM

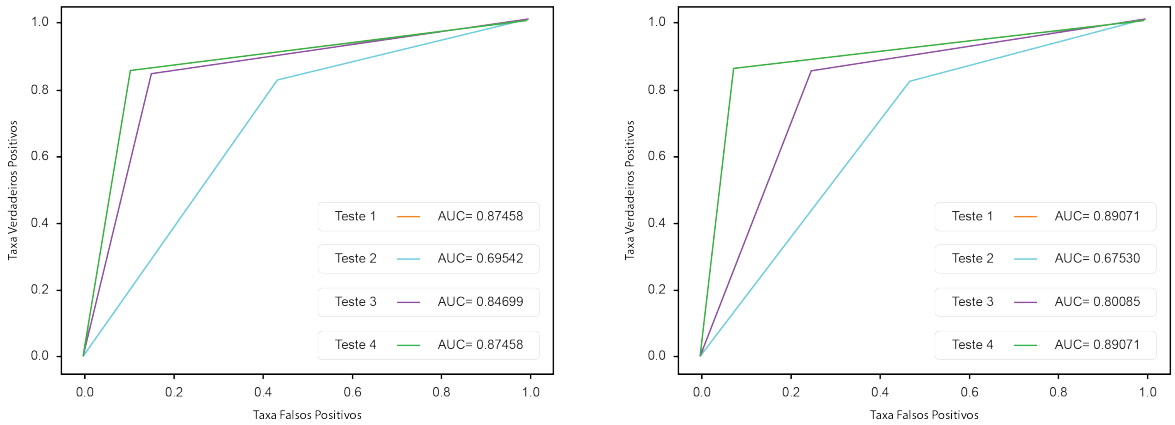


(a) Curva ROC para Janela de 4 (W=4) - SVM



(b) Curva ROC para Janela de 5 (W=5) - SVM

Figura 5.18: Curvas ROC para Janela de 4 e 5 - SVM



(a) Curva ROC para Janela de 6 (W=6) - SVM (b) Curva ROC para Janela de 7 (W=7) - SVM

Figura 5.19: Curvas ROC para Janela de 6 e 7 - SVM

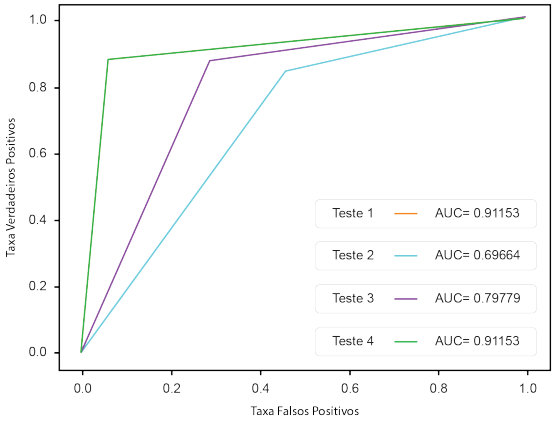


Figura 5.20: Curva ROC para Janela de 10 (W=10) - SVM

	W=2		W=3		W=4		W=5		W=6		W=7		W=10	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
1º Teste	0.81	0.81	0.86	0.81	0.88	0.81	0.88	0.83	0.90	0.95	0.92	0.86	**0.94**	**0.88**
2º Teste	0.53	0.79	0.54	0.78	0.54	0.78	0.58	0.83	0.56	0.82	0.54	0.82	**0.53**	**0.85**
3º Teste	0.85	0.80	0.85	0.80	0.86	0.81	0.85	0.82	0.87	0.85	0.81	0.85	**0.79**	**0.88**
4º Teste	0.81	0.81	0.86	0.81	0.88	0.81	0.88	0.83	0.90	0.85	**0.92**	**0.86**	**0.94**	**0.88**

Tabela 5.9: *Precision* e *Recall* correspondente a cada Teste na respetiva Janela do Algoritmo SVM

	W=2		W=3		W=4		W=5		W=6		W=7		W=10	
	Precision	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*	*Precision*	*Recall*
Média	0.75	0.80	0.78	0.80	0.79	0.80	0.80	0.83	0.81	0.87	0.80	0.85	**0.80**	**0.87**
T.M.E. Fase de Treino (seg)	6751.407		24167.779		5187.357		8261.263		6185.110		3732.737		6690.118	
T.M.E. Fase de Teste (seg)	2209.342		1943.912		1763.986		1633.335		1601.635		1414.745		1145.741	

Tabela 5.10: Média da *Precision* e *Recall* correspondente a cada Janela e respetivos Tempos de Execução das Fases de Treino e Teste do Algoritmo SVM

5.3 Comparação dos Resultados

Nesta secção é feita a seleção dos modelos mais pertinentes e com melhor performance dos modelos testados. É importante ressalvar que no caso de deteção de intrusões, a taxa de falsos negativos e o tempo de execução gasto para a deteção é particularmente importante, pois é do nosso interesse obter um detetor eficaz de ataques sem deixar passar atividade maliciosa como sendo atividade normal. O tempo de deteção também tem bastante importância, pois o objetivo é sempre detetar ataques na fase mais preliminar possível, diminuindo assim os danos e possibilitar maior espaço de manobra para os especialistas de segurança na resolução e prevenção do ataque. Na Tabela 5.11 é apresentado o tempo de pré-processamento dos dados da abordagem por sequência e da abordagem por frequência. Podemos verificar, segundo os resultados apresentados na secção anterior, que apesar dos resultados de deteção da abordagem por frequência serem melhores comparativamente aos resultados de deteção da abordagem por sequência, em relação ao tempo de pré-processamento dos dados, a abordagem por sequência apresenta melhores tempos de execução, e portanto, menor tempo gasto na preparação dos dados para o detetor.

	Sequência	**Frequência**
Tempo de Pré-Processamento (seg)	3077.004	16979.814

Tabela 5.11: Tempo de Execução do Pré-Processamento de ambas as Abordagens (Dados de Validação W=7)

Após a escolha dos algoritmos mais pertinentes para o contexto do estudo do impacto das chamadas de sistema, da escolha da janela e da abordagem utilizada, os resultados da performance desses diferentes modelos foram analisados nas duas secções anteriores.
Podemos verificar que com o aumento do tamanho da janela até 10, a performance do detetor também aumenta. Isto verifica-se devido a facto de, com uma janela igual a 2, os dados pré-processados contém mais ruído, ou seja, existem registos nestes dados que podem não ser descritivos de um ataque ou não ataque. À medida que a janela aumenta, o registo da atividade das sequências de chamadas ao sistema, quer na abordagem de análise de sequências, quer na abordagem de frequências, fica mais claro. Isto é, apesar de haver menos frequência de padrões, os padrões existentes são mais fidedignos de atividade concreta. De acordo com a análise prévia ainda partindo dos dados originais, verificou-se que com janelas superiores a 10, a frequência de padrões decai consideravelmente, perdendo-se registos de sequências importantes para o reconhecimento da deteção.
Em contrapartida, apesar de obtermos melhores resultados com janelas maiores do que 2 até 10, quanto maior é a janela mais informação e mais tempo é necessário para reconhecer um ataque. No contexto de deteção de atividade maliciosa, esta questão é fulcral, pois quanto mais tempo passa desde a tentativa de ataque (neste caso, a sequência e frequência de certas chamadas de sistema específicas já pode ser suficiente e indicar atividade maliciosa) até ao ataque propriamente dito, ou seja, ação final, mais danos podem ser efetuados ao sistema, mais prejuízo vai ocorrer e mais difícil será a sua prevenção e resolução. Portanto, a escolha e o balanço entre performance

de deteção e tempo gasto para a deteção deve ser bem considerada de acordo com vários aspetos. Uma sugestão para um sistema completo, com alta taxa de deteção, mas também rápido na prevenção de ataques, seria um detetor utilizando um número mais baixo de chamadas de sistema, ou seja, que utilizasse uma janela menor, que quando detetasse alguma atividade maliciosa gerasse um alarme para alerta de possível ataque, assim podiam ser tomadas medidas de prevenção e alerta mais cedo. No entanto o gerador de alertas para janelas de chamadas de sistema maiores, estar ativo simultaneamente ou não, conforme fizesse sentido para o contexto em questão. Um sistema com estas duas abordagens a funcionar em simultâneo, poderia fazer sentido para aproveitar os pontos fortes de cada uma delas.

Os modelos que obtiveram taxas mais baixas de falsos positivos e falsos negativos foram o LSTM para a abordagem de sequências, e RF para a abordagem de frequências.
Na Figura 5.21 estão representadas as curvas de ROC de cada teste para cada uma das abordagens. Na Figura 5.22 estão representadas as curvas de ROC médias também de cada uma das abordagens.
Podemos verificar que a abordagem das frequências obteve melhor performance de deteção e tempo de execução de treino e teste, no entanto a abordagem de sequências obteve melhor tempo no pré-processamento dos dados.
Em relação à performance melhor do modelo de deteção, foi utilizado o algoritmo RF. Este algoritmo faz parte dos métodos *ensemble*, que é a combinação de diferentes modelos para se obter um único resultado. No caso do RF são criadas várias árvores de decisão. Esta caraterística torna este algoritmo mais robustos e complexo, levando a um maior custo computacional que costuma ser acompanhando de melhores resultados. Neste caso específico, como a natureza dos dados de *input* não é muito complexa, os custos computacionais não foram elevados em relação os outros modelos testados, e por isso foi escolhido o melhor modelo para a deteção.

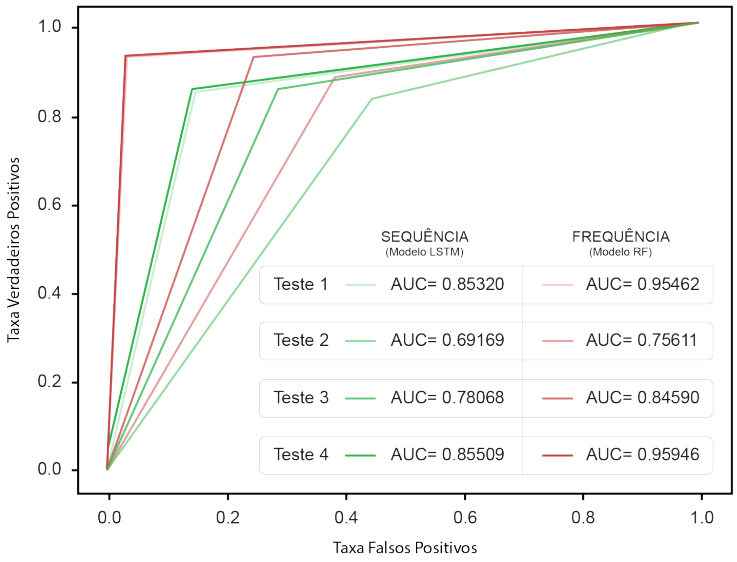


Figura 5.21: Curva de ROC dos Testes dos Algoritmos com Melhores Resultados

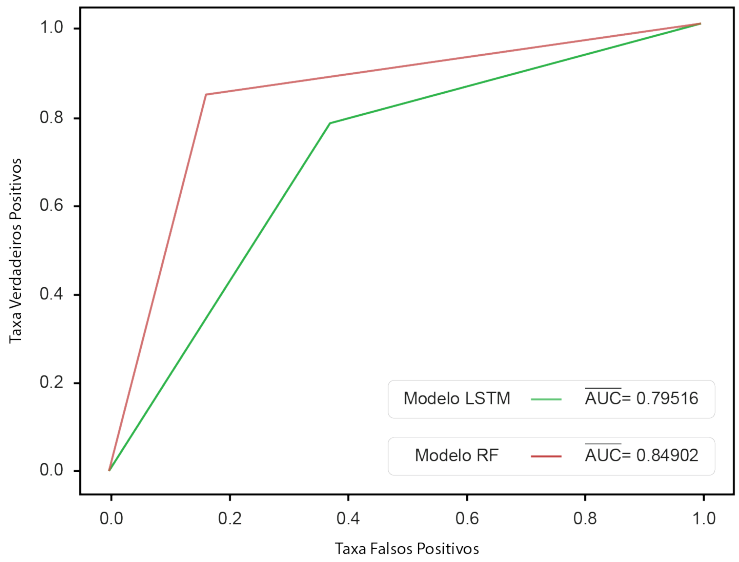


Figura 5.22: Curva de ROC Média dos Testes dos Algoritmos com Melhores Resultados

Capítulo 6

Conclusão

No desenvolvimento deste projeto foi proposto e desenvolvido um modelo adicionável ao JBriareos, um componente de deteção de atividades maliciosas utilizando chamadas de sistema através de algoritmos de *machine learning*, aumentando assim a sua gama de ataques detetáveis.
Foram utilizados diferentes algoritmos, diferentes tamanhos de sequências de chamadas de sistema, diferentes abordagens de deteção e também diferentes técnicas de pré-processamento de dados, para comparação de resultados e estudo do impacto que estas alterações têm no detetor.
Concluindo, o tamanho da janela de chamadas de sistema que têm de ser analisada para de facto concluir se estamos perante atividade potencialmente ameaçadora ou não aproxima-se de 10. A partir de janelas inferiores verifica-se uma taxa alta de deteção de atividade normal e anormal, no entanto os resultados não são tão favoráveis devido ao possível ruído que os dados possuem. Conclui-se também que uma abordagem com dados menos complexos e utilizando um modelo mais simples, como o *Random Forest*, obtém-se melhores resultados no que diz respeito a taxas de falsos positivos e falsos negativos.
Em relação à abordagem de sequências que utiliza um modelo de *deep learning*, os resultados são favoráveis mas menos bons comparando com o modelo anterior, devido à elevada complexidade do próprio modelo e ao tempo de execução gasto no treino e no teste dos dados.
Para a utilização e aproveitamento na prática das vantagens das técnicas estudadas, seria a combinação de ambas as abordagens, utilizando diferentes tamanhos de janela de acordo com o objetivo que queremos que o sistema desempenhe. Se for prioritário uma mais deteção rápida, mesmo que possua taxas menores de desempenho, pode-se optar por tamanhos de janela mais baixos.
Em contrapartida, se for importante uma deteção próxima dos 100%, e o tempo para a prevenção não for tão prioritário, pode-se optar pela utilização de janelas superiores, equilibrando o tempo de deteção com a taxa de deteção precisa.

6.1 Trabalho Futuro

Para trabalho futuro, uma sugestão seria a integração do modelo de *machine learning* proposto no sistema JBriareos, para atingir um maior número de ataques que o sistema deteta, e assim melhor aproveitamento da utilidade do sistema.
Uma outra sugestão, treinar para além de dados de chamadas de sistema, dados com outra natureza e outros atributos para aumentar ainda mais a gama de ataques detetáveis, tendo sempre em conta os tempos de execução.

Bibliografia

[1] J. P. Anderson. Computer security threat monitoring and surveillance. 1980.

[2] Shijoe Jose et al. A survey on anomaly based host intrusion detection system. 2018.

[3] André Zúquete. Segurança em redes informáticas. 2018.

[4] Peter Vamplew Joarde Kamruzzaman Ansam Khraisat, Iqbal Gondal. Survey of intrusion detection systems: techniques, datasets and challenges. 2019.

[5] Bo Lang Hongyu Liu. Machine learning and deep learning methods for intrusion detection systems. 2019.

[6] Aaron Adler Rachel Greenstadt Michael Mayhew, Michael Atighetchi. Use of machine learning in big data analytics for insider threat detection. 2015.

[7] Fadl Mutaher Ba-Alwi Ammar Thabit Zahary Suad Mohammed Othman, Nabeel T.Alsohybe. Survey on intrusion detection system types. 2018.

[8] Marteau. Sequence covering for efficient host-based intrusion detection. 2018.

[9] S. Biswas Subba, B. and S. Karmakar. Host based intrusion detection system using frequency analysis of n-gram terms. 2017.

[10] D. Seethalakshmi and G.M. Nasira. Detecting and preventing intrusion in multi-tier web applications using double guard. 2016.

[11] https://www.snort.org/faq/what-is-snort.

[12] https://www.snort.org.

[13] http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node3.html.

[14] https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/023/original/ids2ips.txt.

[15] https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/013/original/Snort_IPS_using_DAQ_AFPacket.pdf.

[16] Home. https://suricata.io, Aug 2022.

[17] Jon Chase Sep 21. Open source ids tools: Comparing suricata, snort, bro (zeek), linux. https://cybersecurity.att.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview.

[18] All features. https://suricata.io/features/all-features/, Mar 2021.

[19] James Stanger. Suricata admin magazine. https://www.admin-magazine.com/Archive/2015/27/Detecting-intruders-with-Suricata.

[20] Open source hids - fim, rootkit detection, malware detection. https://www.ossec.net/about/.

[21] Ossec project team. ossec's documentation. website. , february 2017. https://ossec.github.io/docs/.

[22] Tripwire inc. open source tripwire. website. august 2017. https://github.com/Tripwire/tripwire-open-source.

[23] Zhenqi Wang1 and Dankai Zhang. Hids and nids hybrid intrusion detection system model design. 2012.

[24] T. M. Mitchell. Machine learning. mcgraw-hill science/engineering/math. 1997.

[25] Aaron Adler Rachel Greenstadt Michael Mayhew, Michael Atighetchi. Use of machine learning in big data analytics for insider threat detection.

[26] N. Zhu Teng Zhang Teng, S Wu. Svm-dt-based adaptive and collaborative intrusion detection.

[27] Wenjuan Li Weizhi Meng1 and Lam-For Kwokz. Design of intelligent knn-based alarm filter using knowledge-based alert verification in intrusion detectiont. 2015.

[28] Sumeet Dua and Xian Du. Data mining and machine learning in cibersecurity.

[29] Geon Heo Yuji Roh and Steven Euijong Whang. A survey on data collection for machine learning: a big data - ai integration perspective. 2018.

[30] Sofiane MAZA and Mohamed TOUAHRIA. Feature selection algorithms in intrusion detection system: A survey. 20179.

[31] Xue .B. Article swarm optimisation for feature selection in classification. 2014.

[32] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[33] Steven Huang. https://www.kaggle.com/datasets/galaxyh/kdd-cup-1999-data, Dec 2018.

[34] Asra Kalim Syed Md. Minhaz Hossain Sheikh Ikhlaq Sohrab Hossain Hamed Alqahtani, Iqbal H. Sarker. Cyber intrusion detection using machine learning classification techniques. 2020.

[35] Andrew Hamilton Christos Tachtatzis Robert Atkinson Elike Hodo, Xavier Bellekens. Shallow and deep networks intrusion detection system: A taxonomy and survey. 2017.

[36] C.Deisy Tran DucChung M.K.A.Ahamed Khan T.Saranyaa, S.Sridevi. Performance analysis of machine learning algorithms in intrusion detection system: A review. 2020.

[37] Jacek Rumiński Anna Drewek-Ossowicka, Mariusz Pietrołaj. A survey of neural networks usage for intrusion detection systems. 2020.

[38] Ruhul Sarker Nam Nhat Tran and Jiankun Hu. An approach for host-based intrusion detection system design using convolutional neural network.

[39] 1999 darpa intrusion detection evaluation dataset.

[40] Peter Vamplew Joarder Kamruzzaman Ansam Khraisat, Iqbal Gondal. Survey of intrusion detection systems: techniques, datasets and challenges.

[41] The apache software foundation. https://www.apache.org/. [Online; accessed 19-Oct-2022].

[42] Hypertext preprocessor. https://www.php.net/, note = "[Online; accessed 19-Oct-2022]".

[43] Mysql.

[44] Tiki8.x- documentation for tiki wiki cms groupware.

[45] Wlademir Ribeiro Prates. Curva roc e auc em machine learning. https://cienciaenegocios.com/curva-roc-e-auc-em-machine-learning/, Oct 2021.