Universidade do Minho
Escola de Engenharia

Francisco André Guimarães Ribeiro

Detecção de Intrusões em Redes Informáticas, utilizando Redes Neuronais Artificiais

Detecção de Intrusões em Redes Informáticas, utilizando Redes Neuronais Artificiais

Francisco André Guimarães Ribeiro

OUtubro | 2011

Outubro de 2011

Universidade do Minho
Escola de Engenharia

Francisco André Guimarães Ribeiro

Detecção de Intrusões em Redes Informáticas, utilizando Redes Neuronais Artificiais

Tese de Mestrado

Trabalho efectuado sob a orientação do
Professor Doutor Henrique Santos

Julho de 2005

DECLARAÇÃO

Nome
Francisco André Guimarães Ribeiro

Endereço Electrónico
fribeiro@yumeconnections.com

Número do Bilhete de Identidade
13449590

Título da Dissertação
Detecção de Intrusões em Redes Informáticas, utilizando Redes Neuronais Artificiais

Orientador
Henrique Santos

Ano de Conclusão
2011

Designação do Mestrado
Mestrado em Engenharia Informática

Universidade do Minho, Outubro 2011

Assinatura:

Agradecimentos

Faith is taking the first step even when you don't see the whole staircase? Darkness cannot drive out darkness; only light can do that. - Martin Luther King Jr.

Muitas vezes, durante viagens, o que se encontra à nossa frente é uma vasta escuridão, escuridão essa que pode ser iluminada pelos faróis da nossa viatura, ou por elementos externos. Por vezes necessitamos ainda de seguir indicações ou repousar num qualquer lugar.
Até que finalmente chegamos ao destino, e aí agradecemos a todos aqueles que possibilitaram o sucesso da viagem.
Um trabalho de investigação é como uma viagem, por vezes perdemo-nos, por vezes trabalhamos às escuras, por vezes não sabemos por onde seguir. Nestes momentos entram em acção os nossos GPS, os nossos "focos de iluminação", a nossa lua e o nosso sol, que nos guiam e iluminam, indicando o caminho, quando não o conseguimos ver.
Todos estes são elementos essenciais ao sucesso de um projecto, isto porque o sucesso é a conjugação do todo. Ao longo desta dissertação tive várias indicações, vários focos de iluminação, uma lua, um sol e várias estrelas. Mas então o que são todos estes elementos?
Para mim a meu sol é a minha família, composta pela minha mãe, pela minha irmã e pelo meu pai. Pessoas marcantes e que fizeram de mim o ser humano que hoje sou. Pessoas que não só educam, castigam, responsabilizam, mas que também aquecem e reconfortam sempre que há necessidade. Para eles um agradecimento especial por tudo o que me proporcionaram, pedindo-lhes desde já desculpa pela minha forçada ausência.
O meu GPS é sem margem para dúvidas, o meu orientador, o *Prof. Dr. Henrique Santos*, pessoa fantástica que me apoiou em todos os momentos, que me mostrou o caminho a seguir, que me acompanhou e me deu força sempre que me surgiram dúvidas. Sem dúvida muito

deste trabalho é-lhe dedicado, agradecendo-lhe sempre a boa vontade e ajuda disponibilizada.
Os meus focos de iluminação, são os meus amigos, aqueles que nos momentos de maior dúvida me motivaram, aqueles que quando surgiam dúvidas tentavam iluminar o caminho.
Podendo cair no erro de ser injusto, enumero o nome de alguns que foram fundamentais para o meu sucesso: *Tiago Martins, Ricardo Sousa, Jorge Paulo, Rui Moreira e Pedro Sousa.* Este trabalho é também um bocadinho vosso.
Por fim agradecer à minha lua, uma pessoa com enorme significado na minha vida, uma pessoa exceptional que me ajudou nos momentos mais duvidosos, pressionando-me, guiando-me, acreditando em mim. Sendo por isso minha convicção que esta dissertação é muito tua. Obrigado *Carolina.*

De resto, resta-me fazer o agradecimento geral a todos os restantes focos de luz, que apesar de não serem citado, são reconhecidos por mim...

Abstract

The successful intrusion detection on network is a problem without a perfect solution. The general opinion is that this is the most critical problem of modern times because right now we don't know if someone, somewhere is watching our traffic, or most seriously we cannot actually know if what we are receiving is the real information. To solve this we can use cryptography to cypher our data, but the principle remains, shouldn't privacy be protected by default and to every user?

Currently, most used applications to detect intrusions are based on traffic analysis using static patterns using techniques like Data Mining or statistical algorithms, but they usually produce a high ratio of false positives, limiting it's efficiency. Obviously this solution is far from perfect. At this point we need to develop a system that could learn with the traffic, a self sustained system that could analyze traffic and detect strange occurrences.

To solve this problem one idea emerged, the creation of a "Neural Network for Intrusion Detection". This idea is gaining more followers each day as mentioned on ANNCIDS.[9]

This idea is based on brain, with the ability to learn, make mistakes and acquire knowledge. It's a structure more complex than an algorithm based on patterns but is also more reliable and durable. Actually a neural network is a set of simple processors that we call neurons. These neurons have the capability to do operations over inputs to obtain a certain output.

Nowadays the major problem is the creation of false alerts. Each false positive and negative formed using patterns previously created, leads to the necessity of a new reconfiguration made by a system administrator and in consequence lost of performance, time and confidence.

It's more reliable because there's no need to create a dictionary of patterns previously or to have the knowledge of network. Neural systems are more adaptive to the environment around.

Usually a IDs based on NN has the following structure, being that our optimizing focus.

-Data Acquisition;

-Data Formatting;

-Artificial Neural Network;

-Expert System - NN Analysis and Control/Analysis and Decision;

In this work we will be focus in how ANN interact with network and configurations that we can manage/optimize to achieve an improvement.

Keywords: Intrusion, Detection,System, False, Positives, Snort

Resumo

A bem sucedida detecção de intrusões em redes de computador, é um problema para o qual não existe uma solução perfeita. Em geral acredita-se que este é um dos problemas mais críticos da sociedade moderna, uma vez que, actualmente não é possível assegurar que alguém, algures não está a visualizar o tráfego por nós gerado, ou, num caso mais extremo, não podemos assegurar que os dados que recebemos são dados confiáveis.

Para solucionar este problema podemos recorrer à ciência criptográfica, de forma a proteger os nossos dados. No entanto este recurso não é suportado por todos os prestadores de serviços. Sendo que mais uma vez surge o principio que prevê que a nossa privacidade seja assegurada pelos prestadores de serviços.

Actualmente, a maioria das aplicações existentes para detectar intrusões de redes, ou pelo menos as mais utilizadas, recorrem-se de técnicas de *DataMining* ou algoritmos estatísticos, de forma a detectar padrões e encontrar semelhanças com registos prévios de ataques, e monitorização continua por parte de um administrador do sistema. Tendo estas técnicas a desvantagem de gerar um elevado número de falsos positivos, o que limita obviamente a sua eficiência. Por esta razão estas soluções estão longe de ser perfeitas, existindo por isso a necessidade de aprofundar o estudo no campo de forma a desenvolver uma aplicação *autodidácta e autosuficiente* que possa fazer uma análise em tempo real do tráfego, detectando mais eficientemente ocorrências anormais.

Uma ideia que pode permitir uma maior eficiência baseia-se na utilização de *Redes Neuronais Artificiais*, pensamento que tem ganho cada vez mais seguidores com o passar do tempo[16, 13]. O funcionamento desta solução baseia-se no comportamento do cérebro humano, e na forma como este tem capacidade para aprendizagem, errando e adquirindo conhecimento. Este algorimo é mais complexo do que os utilizados até agora, mas é também de maior confiança. Na verdade, uma *RNA* é um conjunto de neurónios, divididos por camadas que têm a capacidade de realizar operações, obtendo determinados resultados.

A presente solução, propõe a reduzção do número de falsos alertas, uma vez que estes são um flagelo que atinge os administradores de sistemas. Cada falso positivo ou negativo gerado exige uma reconfiguração por parte do gestor de forma a que haja a correcção, o que leva a uma perda de *performance*, tempo e confiança na aplicação.

Com a capacidade de aprendizagem e adaptação ao meio, as *RNA* ganham um maior foco no sentido de melhorar estes resultados.

Neste projecto, o foco estará direccionado a uma implementação de *RNA* em sistemas Java, fazendo a integração de dados obtidos de um *IDS* generalizadamente utilizado.

Keywords: Detector, Intrusões, Sistema, Rede, Neuronal, Artificial, Snort

Conteúdo

1 Introdução **1**

1.1 Motivação 1

1.2 Objectivos 3

1.3 Solução Proposta 4

1.4 Estrutura da dissertação 5

2 Ataques e Segurança **7**

2.1 Enquadramento Histórico 7

2.2 O que é a Segurança da Informação 7

2.3 Taxonomia dos Ataques 9

2.3.1 Exemplos práticos de ataques 11

2.4 Anatomia do ataque 12

2.5 Scan, o pré-ataque 13

2.6 Detectores de intrusões de rede 18

2.6.1 Classes de IDS 19

2.6.2 Falsos Positivos 20

2.6.3 Factores que influenciam a eficácia dos *IDS* 21

3 Redes Neuronais Artificiais **23**

3.1 Introdução 23

3.2 Principios Básicos 24

3.3 Propriedades das *RNA* 25

3.4 Estrutura 27

3.4.1 Neurónios 28

3.4.2 Funções de Activação 29

3.4.3 Função *sigmoid* . . . 31
3.4.4 Camadas . . . 32
3.5 Modelos actuais . . . 33
3.5.1 FeedForward . . . 34
3.5.2 Recurrent Network . . . 36
3.6 Algoritmo *Backpropagation* - Feedforward . . . 36
3.6.1 Fases do algoritmo . . . 37
3.7 Considerações a ter com o uso de redes neuronais . . . 39
3.8 Soluções existentes . . . 39

4 Implementação **43**
4.1 Ferramentas utilizadas . . . 43
4.1.1 Snort . . . 43
4.1.2 Utilização do *snort* . . . 45
4.1.3 Nmap . . . 46
4.1.4 Encog Framework . . . 48
4.1.5 IPTables . . . 48
4.2 Metodologia . . . 49
4.2.1 Trabalho Base . . . 50
4.2.2 Trabalho desenvolvido . . . 51
4.2.3 Integração *snort* . . . 62
4.3 Aplicação . . . 66
4.3.1 IDSNN1_ScanDetecion . . . 67
4.3.2 IDSNN1_Snort . . . 70

5 Discussão de Resultados **73**
5.1 DataSet . . . 73
5.1.1 Scans efectuados . . . 74
5.2 IDSNN1_ScanDetection . . . 75
5.3 IDSNN1_Snort . . . 79

6 Conclusões e Trabalho Futuro **83**
6.1 Conclusões . . . 83
6.2 Trabalho Futuro . . . 84

Bibliografia **87**

A Exemplo NMAP **91**

B Blocos de Código **95**

C DataSet **97**

Lista de Figuras

2.1 Três condições para segurança 8
2.2 Man in the Middle 12
2.3 TCP Three Way Handshake 15
2.4 Intrusion detection System Diagram 19

3.1 Camadas Escondidas como *BlackBox* 28
3.2 Estrutura de um neurónio [16] 29
3.3 Função Threshold 30
3.4 Função Piecewise-Linear 31
3.5 Função Sigmoid 32
3.6 Single-Layer *RNA* 34
3.7 Multi-Layer *RNA* 35
3.8 Multi-Layer *RNA* 36
3.9 Recurrent *RNA* 37
3.10 Backpropagation Path [16] 38

4.1 Regras *snort* 45
4.2 IDSNN1 Interface 49
4.3 Modelo Base Dados 54
4.4 Tabelas Esquema *snort* Utilizadas 66
4.5 Menu Detector Scans 68
4.6 Resultados execução Detector Scans 69
4.7 Menu *snort* 70
4.8 Menu *snort* 71

5.1 Gráfico Erro/Iteração IDSNN1_Snort 76

5.2 Gráfico Erro/Iteração IDSNN1_Snort 79

Lista de Tabelas

4.1 Amostra sem ataques 51
4.2 Amostra de Scans 51
4.3 Resultados esperados 53
4.4 Ficheiro Configuração *IDSNN_ScanDetectio* 56
4.5 Opções TCPDump 56
4.6 Rotina Captura ICMP 57
4.7 Rotina Captura Flag 58
4.8 Rede Neuronal Actual 60
4.9 Relação Valor/(Momentum.Learning Rate) 61
4.10 Relação entre resultado de módulos 63
4.11 Ficheiro de Confirguração *IDSNN_Snort* 64
4.12 Query SQL 65
4.13 Rede Neuronal Actual 67
4.14 Exemplo treino rede neuronal 68
4.15 ignorar pacotes com origem no IP declarado 69
4.16 tráfego pelo TCPDump 70

5.1 Resultados IDSNN_Scandetection 76
5.2 Resultados obtidos *IDSNN1_ScanDetection* 78
5.3 Resultados Snort 80
5.4 Resultados Snort 81
5.5 Resultados IDSNN_Snort + IDSNN_ScanDetection 81

A.1 Scan entre endereços de IP 91
A.2 Scan à rede 92

A.3 Scan FIN 93

B.1 Versão Inicial Rede Neuronal Artificial 96

C.1 Dataset Treino Snort 97
C.2 Dataset Treino Snort 98
C.3 Dataset Treino Snort 99
C.4 Dataset Treino Snort 100
C.5 Dataset Treino Snort 101
C.6 Treino rede Neuronal módulo *IDSNN1_Snort* 102
C.7 Treino rede Neuronal módulo *IDSNN1_ScanDetection* 103
C.8 DataSet Treino *IDSNN1_ScanDetection* 104
C.9 DataSet Treino *IDSNN1_ScanDetection* 105
C.10 DataSet Treino *IDSNN1_ScanDetection* 106

Lista de Acrónimos

ARP *Address Resolution Protocol*

DoS *Denial of Service*

H-IDS *Host Based Intrusion Detection System*

IDS *Intrusion Detection System*

IDSNN1 *Intrusion Detection System with artifical Neural Networks One*

IP *Internet Protocol*

MITM *Man In The Middle*

N-IDS *Network Based Intrusion Detection System*

PHP *Hypertext Preprocessor*

RNA *Rede Neuronal Artificial*

CAPÍTULO 1

Introdução

1.1 Motivação

A informática é uma área de investigação multidisciplinar, cuja evolução ao longo da última metade do século XX e primeira década do século XXI tem sido fascinante. Nos últimos 60 anos a informática passou por uma revolução sem precedentes e sem comparação, sendo de tal forma notável que hoje qualquer pessoa dos países desenvolvidos interage diariamente, no mínimo com uma peça informatizada. Seja este um computador, ou uma máquina de *tickets* de fila de espera.

No entanto, esta área é fascinante não só pela sua componente evolutiva, mas também por não excluir nenhum elemento da sociedade. Isto porque existem milhões de autodidactas que conseguem, com poucos recursos, executar tarefas excepcionais, não importando estatuto ou estrato social.

É importante ter em conta, como mencionado acima, que hoje em dia a utilização de equipamentos informatizados é comum. Na verdade, actualmente quase qualquer local público a que nos dirijamos nas cidades, tem um ponto de acesso à internet. Este é um facto extremamente positivo, pois possibilita a conexão, debate, troca de ideias ou até convivência de milhões de pessoas, que tanto podem estar a dois metros como separadas por oceanos. Em boa verdade é possível admitir que nos dias que correm, grande parte da nossa vida, dos nossos dados se encontram, de alguma forma, na rede. Porém, é importante ter consciência que esta banalização tem riscos, por isso é necessário assegurar a autenticidade, confidencialidade, integridade e disponibilidade de acesso aos dados.

A evolução das redes de comunicações permitiu que na actualidade, milhões de computadores estejam conectados e partilhem informação, sendo que hoje é possível afirmar que mesmo

uma pequena ou média Empresa pode deter um servidor central. Servidor onde todos os seus dados podem estar alojados. Num contexto destes, durante toda a actividade da empresa, os terminais comunicam com o servidor central, consumindo e produzindo informação.

Esta centralização tem aspectos positivos e negativos. Se por um lado permite a facilidade de acesso, por outro torna-se um sistema apetecível a ataques por parte de pessoas menos bem-intencionadas. Obviamente nenhuma instituição pretende ter os seus dados digitalizados, se estes estiverem facilmente acessíveis em qualquer lugar por qualquer pessoa e não apenas por quem de direito.

Um exemplo disto é o Sistema Nacional de Saúde. Actualmente planeia-se que os dados clínicos dos pacientes sejam armazenados num *datacenter*. Este sistema tem diversas vantagens, dado permitirem que um habitante de Braga possa ser atendido com a mesma eficácia e rapidez, em qualquer centro hospitalar do país. Porém, em caso de intrusão de um *hacker* no sistema, este poderá fazer alterações aos registos clínicos, podendo vir mesmo a causar a morte de um ou vários pacientes. Esta situação para além de ser extremamente grave para a sociedade, é muito embaraçosa para qualquer administrador de sistemas.

Na verdade e devido à enorme concorrência entre entidades, seja luta por patentes, clientes, ideias, projectos, entre outros, existe uma guerra cibernética constante. Actualmente grupos mal-intencionados organizam-se de forma estruturada, com o objectivo de obter informações de elevada importância, para por exemplo extorquir dinheiro às entidades afectadas, ou vender a entidades interessadas.

Neste contexto surgem então variados sistemas que permitem assegurar a segurança de um sistema informático, ou seja, assegurar a autenticidade, integridade, disponibilidade e confidencialidade deste. De forma a atingir este patamar, existem para utilização diferentes tipos aplicações como Firewall, Antivírus, AntiSpyware, AntiMalware, Sistemas de Detecção de Intrusões em Redes, entre outros, que tratam diferentes tipos de ameaças. Sendo importante ter em consideração que a segurança apenas se atinge com a conjugação de vários tipos distintos, alargando assim o espectro de resposta a diferentes ameaças.

Os *IDS* desempenham um papel importante no sentido de atingir um sistema seguro, pois permitem a detecção em tempo real do acesso de pessoas menos bem-intencionadas em sistemas. Devido à necessidade de melhorar a sua *performance* ser real e imperativa, estes são alvo de constante estudo. Tal como explicado posteriormente, existem diversas classes de *IDS*, focando cada uma delas uma metodologia distinta de detecção.

Actualmente existem variadas soluções criadas a partir de estudos realizados. *Soniya e Wisey*

desenvolveram uma metodologia que permite fazer a detecção de *scans*, utilizando a contagem de *flags* dos pacotes *TCP*.[37] Este método surge devido à importância que o *scan* tem para o sucesso ou não de um ataque a uma rede.[18]
O *scan* é na verdade um dos cinco passos necessários à realização de um ataque[18], pois só através dele é possível ao *hacker* ter noção da tipologia da rede e dos serviços disponiveis. Estes cinco passos são referidos detalhadamente ao longo da dissertação. É então importante ter em consideração, que o desenvolvimento de técnicas de hacking, que permitem subverter os sistemas informáticos, é uma actividade que coloca em risco não só a nossa privacidade, como também a nossa segurança. Elementos essenciais ao nosso quotidiano diário.
Desta forma é possível desenvolver uma nova aplicação que aliada aos *IDS* usualmente utilizados, reduz a emissão de falsos positivos. Estes alertas representam uma quota de todos os alertas gerados[29, 42], sendo prejudiciais ao bom funcionamento do sistema, pois podem influenciar o comportamento, a performance e também implicam a necessidade de existência um administrador que faça a sua validação.[19, 28]

1.2 Objectivos

Dentro do contexto referido na secção anterior, os objectivos delineados para esta dissertação são muito distintos. Sendo que existe o objectivo do topo da pirâmide e por isso apelidada de *objectivo principal* e objectivos mais restritos apelidados de *sub-objectivos*.
O objectivo principal é o desenvolvimento de um protótipo de um sistema que reduz a geração de falsos positivos, sem que haja perda de eficácia na deteção de tráfego malicioso.[29]
De forma a que este seja concretizado, é então necessário seguir uma metodologia de investigação. Para o caso em estudo, a metodologia escolhida é a *Design Science*, dado ser aquela que melhor se aplica ao objectivo apresentado.
Design Science é uma metodologia através da qual existe um estudo e desenvolvimento de novos artefactos, no nosso caso uma solução, e também o estudo dessa mesma através da análise dos resultados obtidos durante e após a sua execução[41]. Desta forma adquirir conhecimento, que permite fazer um retrato mais assertivo do estado da arte. Em boa verdade é aceitável dizer que já existe muita investigação realizada na área, e que esta dissertação vem apenas acrescentar tópicos a visões já existentes, demonstrando na prática como é possível criar cooperação entre diversas metodologias.
A solução apresentada ao longo desta dissertação, demonstra que a cooperação entre me-

todologias distintas, não só é possível, como também permite a melhoria significativa dos resultados.

Para que isto seja possível é então necessário criar um ambiente, onde se pode colocar em prática a investigação realizada. De forma a ser aceitável propor alterações a um modelo computacional, tornando-o mais eficiente, robusto, dinâmico e poderoso, é necessário compreender as metodologias já existentes, validando-as e avaliando as suas diferenças. Para a validação destas metodologias, é necessário criar um ambiente de simulação, capaz de recriar diferentes comportamentos humanos, com distintos objectivos de navegação. Sendo que é também necessário implementar comportamentos anómalos e esperar a sua detecção. Este é então o grande objectivo, a implementação de uma metodologia que permita a redução dos falsos positivos gerados pelos *IDS*, sendo importante ter em consideração, que com esta optimização, o desempenho das máquinas não deve ser muito degradado.

Posto isto e dividindo o objectivo principal em objectivos mais pequenos, esta dissertação propõem-se a apresentar uma metodologia que permite:

- reduzir o número de falsos positivos gerados pelo SNORT;
- utilizar redes neuronais artificiais;
- bloqueio de IP's considerados atacantes;
- detecção em tempo real de tráfego anormal;
- optimização de recursos;

Com isto em mente foi então possível o desenvolvimento de um protótipo que aborda todas estas questões.

1.3 Solução Proposta

A solução aqui proposta pretende apresentar um sistema híbrido, conjugando os benefícios de um detector que especificamente apenas faz análise protocolar[8] e um sistema genérico como é o *snort*, sistemas estes explicados no capítulo três.

Sendo assim, e mais detalhadamente, a solução apresentada é composta por duas redes neuronais, *FeedForward* com algoritmo *Backpropagation*[35, 25, 37, 11, 24, 6, 36, 8], um módulo de análise de tráfego com recurso à aplicação *TCPDump*, a um *parser* para desfragmentação e uma base de dados para guardar a informação e um módulo de integração dos alertas emitidos pelo snort. Esta solução tem o nome *IDSNN1* e o seu algoritmo de funcionamento será

descrito oportunamente nos capítulos seguintes. Apenas foi possível chegar a esta solução graças a uma metodologia de investigação especificada, que se baseou na divisão de tempo por tarefas, sendo que foram dedicadas

- quatro semanas ao estudo de modelos de redes neuronais artificiais aplicados à detecção de intrusão em redes;
- oito semanas ao estudo de optimizações possíveis a sistemas existentes;
- quatro desenvolvimento de uma aplicação seguindo o estudo realizado;
- dezasseis semanas de optimização, correcção e estudo dos resultados obtidos;

Após desenvolvimento, e tal como acima enunciado, o *IDSNN1* foi testados utilizando tráfego de duas origens distintas, facto explicado posteriormente. Os seus resultados foram analisados tendo em conta a prestação do *snort*, verificando se há ou não uma redução na emissão de falsos positivos emitidos.

1.4 Estrutura da dissertação

A presente dissertação encontra-se dividida em sete capítulos principais. O primeiro tem como objectivo fazer uma introdução ao caso de estudo, apresentando uma contextualização e demonstrando as motivações intrínsecas que levaram ao desenvolvimento desta mesma dissertação. Neste capítulo poderemos ainda encontrar enumerados os objectivos principais a que se propõe esta investigação.
Os capítulos posteriores fazem um enquadramento de toda a investigação, fazendo a apresentação de conceitos e linhas de pensamento necessárias para a compreensão dos resultados obtidos. Sendo assim e mais especificamente, no capítulo número dois são abordadas questões de segurança. Da grande panóplia de ataques existentes, existem alguns que se destacam mais do que outros. De forma a ser possível desenvolver uma aplicação que detecte intrusões, é necessário entender como funciona a mente de um *hacker*, bem como as vulnerabilidades a que um sistema está sujeito. Só desta forma poderá ser possível estar atento a factores específicos que permitem detectar e bloquear possíveis intrusões. Este capítulo oferece uma visão geral de como é possível, uma pessoa comum, obter informações confidenciais recorrendo a conteúdos disponibilizados online. Numa primeira fase é feita uma contextualização histórica que nos leva a perceber de que forma têm evoluído as questões de segurança da informação.

Posteriormente é dada uma breve explicação daquilo que no entender do autor, recorrendo sempre à bibliografia, é a segurança da informação. Para finalizar, são enumerados aqueles que, segundo a bibliografia, são alguns dos ataques mais perigosos para os sistemas e sobre os quais os *IDS* se devem focar.

Encontra-se ainda, de forma abrangente, como os sistemas de detecção de intrusões em rede funcionam, havendo a enumeração de alguns deles, explicando ainda como estes interagem com a rede.

No terceiro capítulo é apresentado o *core* da dissertação, isto é, as redes neuronais artificiais. Estas assumem um papel preponderante, uma vez que são elas as responsáveis pela análise e validação de todos os dados recebidos (quer importados do *snort*, quer obtidos directamente). Para o caso de estudo é feita uma apresentação geral dos dois tipos de redes neuronais artificiais, existindo no entanto um foco maior para as redes do tipo *Feedforward*, sendo que a nível dos algoritmos apenas é apresentado o *Backpropagation*. Isto porque foram estes os dois métodos utilizados no decorrer do projecto.

Neste capítulo poderemos encontrar ainda explicação sucinta sobre as funções de activação, tendo sido dada novamente prioridade à utilizada neste projecto, ou seja a função de activação *Sigmoid* e a sua variante *Tranformada Tangente Hiprtbólica*.

O capítulo quatro demonstra de forma explicativa toda a componente prática levada a cabo, fazendo inicialmente uma apresentação das ferramentas utilizadas, incluindo ainda as tecnologias e *frameworks* de desenvolvimento. Nele poderemos ainda encontrar questões como desafios, especificações, obstáculos e apresentações esquemáticas e gráficas de como a aplicação desenvolvida deve trabalhar.

No capítulo cinco apresenta a análise dos resultados. São apresentadas as expectativas e o que efectivamente resultou de toda a investigação/implementação.

Por fim, o capítulo seis traz consigo as conclusões deste projecto, e aquilo que poderá vir a ser feito num futuro próximo. Nele é possível encontrar de forma sucinta as descobertas relevantes e ainda o que pode vir a ser feito de forma a melhorar a metodologia existente.

CAPÍTULO 2

Ataques e Segurança

2.1 Enquadramento Histórico

A necessidade da segurança informática está intimamente ligada à necessidade de segurança de informação. Esta existe desde que existem conflitos e a necessidade de manter em segredo informações de grande importância. No entanto a grande evolução surgiu entre os anos 1939 e 1945 durante a Segunda Guerra Mundial, com o desenvolvimento de código e máquinas de cifragem/decifragem de mensagens. Nessa altura a segurança era muito linear, baseando-se apenas em políticas de confidencialidade e cifragens de mensagens com algoritmos hoje em dia bastante simplistas.[20, 10]

Posteriormente, as necessidades foram evoluindo o que levou à criação de departamentos que apenas se dedicam a manter os dados seguros. Instituições usualmente ligadas aos departamentos militares dos diversos países.

2.2 O que é a Segurança da Informação

Genericamente é possível afirmar que a segurança informática é a protecção de forma a preservar a *integridade, confidencialidade, disponibilidade e autenticidade* dos dados electrónicos de indivíduos e instituições.[20, 10]

A segurança de redes, figura 2.1 é apenas uma das três componentes que permitem atingir a segurança da informação, porém deve ser-lhe dada especial importância devido ao facto, já referido, da banalização da Internet e consequente difusão de informação através desta.

A figura 2.1 apresenta de forma esquemática as três principais classes que permitem obter

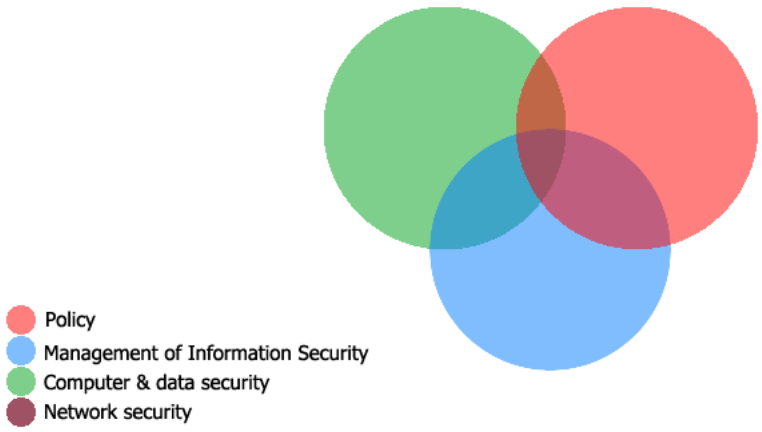

Figura 2.1: Três condições para segurança

segurança dos nossos dados, sendo que é necessário conjugar aplicações de *software*, com componentes de *hardware* e comportamentos humanos, de forma a assegurar esta:[20, 10]

1. Segurança da rede - esta é a componente a ser tratada neste trabalho, sendo o objectivo melhorar os resultados obtidos pelos *IDS*, através da redução dos valores falsos positivos.

2. Gestão da informação de segurança - a forma como o administrador de sistema armazena, transporta ou divulga as metodologias de segurança, são um factor de extrema importância. A confidencialidade destes dados é crucial para manter um sistema mais seguro;

3. Segurança dos dados e dos equipamentos - a cifragem ou não de dados, aliada à protecção dos sistemas, conseguida através de *firewalls, anti-spywares, anti-malwares, anti-vírus* será a terceira e última componente que permite atingir um sistema seguro, fiável e confiável.

A conjugação das três consições permite ao administrador ter políticas de segurança importantes para que seja mantida a confidencialidade, integridade e disponibilidade não só da instituição, mas também dos serviços prestados.

2.3 Taxonomia dos Ataques

Tipicamente pode-se dividir os ataques na rede em duas classes distintas, sendo estas as classes de *Ataques Activos* e *Ataques Passivos*. Estas duas, diferenciam-se na forma como operam e por vezes nos objectivos.[26]

Os ataques passivos, genericamente podem-se descrever como sendo aqueles em que o *hacker* apenas pretende *escutar* a rede, não havendo qualquer tipo de alteração do tráfego visionado. Este tipo de ataques normalmente é utilizado para que seja possível a obtenção de dados confidenciais deixando o mínimo de rasto possível.

Os ataques activos são aqueles no qual o *hacker* faz alteração ao tráfego da rede. Tipicamente estes têm objectivos mais agressivos e imediatos como o isolamento de uma instituição ou o roubo de informação, causando prejuízos muitas vezes inestimáveis. Um exemplo de ataques activos foi o que no ano de 2011 se passou com a empresa *SONY*, mais propriamente com a *playstation network*. Deste ataque resultou a captura de milhões de dados confidenciais e posterior indisponibilidade do serviço.

Na actualidade existem sete ataques principais [26] que preocupam todos os adminstradores de segurança. Isto porque são não só de relativa fácil implementação como também os que causam mais prejuízo às entidades responsáveis.[26]

- **Denial of Service**: Os ataques por *DoS*, como mencionado anteriormente, têm o objectivo de isolar o alvo de comunicação com o exterior. Na actualidade, os requisitos necessários para poder efectuar este ataque com sucesso e facilmente são:
  - o alvo ter recursos limitados ao nível de conexões;
  - o hacker ter recursos suficientes para exceder o limite do alvo;

  Hoje em dia, e mais uma vez devido à evolução das ligações de internet e redes sociais, não é difícil reunir estas condições, isto não só porque a grande maioria das aplicações de internet têm um limite máximo de conexões não muito elevado, mas também porque é com facilidade que se consegue colocar um grande volume de máquinas, por vontade própria dos seus utilizadores ou não, a gerar tráfego de forma a ocupar um recurso. Esta subclasse é apelidada de *DDoS* e é a forma encontrada para com rapidez ser possível fazer com que uma máquina destino fique em baixo. Um exemplo de como este ataque é levado muito a sério, recentemente a agência de *rating Moody's* bloqueou o acesso de todos os pedidos oriundos de Portugal, após no facebook ter sido criado um movimento sincronizado para fazer pedidos ao seu local na internet. Era expectável,

através do número de utilizadores que confirmaram aderir ao processo, que os servidores não fossem capaz de lidar com a carga e ficassem indisponíveis.[26, 18]

- **War Dialing** esta técnica teve o seu pico máximo durante as décadas de 1980 e 1990. Nessa altura eram utilizados *modems* directamente ligados à internet, existindo um número de telefone associado. Actualmente existem outras soluções de ligação, o que diminuiu a utilização deste método.

  Este ataque é efectuado com o objectivo de identificar o número de telefone directamente associado ao *modem*. Após ter esta associação, o *hacker* poderá inserir estes números de telefone, e dependendo dos resultados, existem diversas possibilidades, desde acesso a linha telefónica o que permitirá fazer chamadas a cobrar nesse número, escutas de chamadas, recepção/envio de fax, entre outros;[26]

- **Spanning-Tree-Attacks** método através do qual se torna possível fazer um ataque tendo por base a análise dos protocolos de rede e as suas diversas camadas. Com este ataque é possível obter acesso a redes internas;[26]

- **Penetration-Testing** são todos os métodos que permitem ultrapassar barreiras de forma a entrar em máquinas individuais. Usualmente este é combinado com outros de forma a ser possível inicialmente entrar numa rede privada e então depois fazer o ataque à máquina individual. Através deste é também possível verificar qual a capacidade máxima de resistencia da rede.
  [26, 2]

- **Man-in-the-Middle** método através do qual o *hacker* fica a funcionar como central de tráfego, colocando-se no meio de duas máquinas e capturando todo o tráfego entre elas as duas;[26, 2]

- **Protocol Tunneling** metodologia que permite fazer a ligação segura entre duas máquinas. Este método permite que haja fugas de informação de instituições, acesso a serviços bloqueados, entre outros.

  Este método consiste na criação de uma ligação VPN segura entre duas máquinas, como por exemplo um servidor caseiro, através disto, o *hacker* integra a sua máquina na rede

caseira, podendo desta forma ter acesso a conteúdos da sua rede, ou acesso a serviços inicialmente bloqueados.

A particularidade deste sistema reside no facto dos dados poderem ser cifrados, passando todo o tráfego pela rede de casa do utilizador e aí decifrados. Desta forma os serviços de monitorização da empresa têm dificuldades em fazer o bloqueio de informação;[26]

- **Password Replay** método através do qual um *hacker* pode obter acesso às credenciais de login em diversos serviços. Este método pode ser aplicado através da criação de falsas páginas web, *scripts* que pedem *login*, ou software malicioso que faz o registo destes dados, dando posterior acesso ao *hacker* aos dados obtidos.[26]

2.3.1 Exemplos práticos de ataques

Hoje em dia é possível, e com relativa facilidade, executar ataques não só a redes, como também a máquinas. Na internet facilmente se encontram explicações e ferramentas que permitem de forma fácil e até incógnita levar a cabo ataques potencialmente perigosos. Exemplo disso é a suite de *hacking BackTrack*, que conjugando com o manual *Gray Hat Hacking - Third Edition* permite a um autodidacta aprender técnicas poderosas.

Um exemplo prático é a utilização do plugin *firesheep* para o browser *firefox* que permite ao utilizador, numa rede aberta, obter cookies que dão acesso aos mais variados sites, desde o *GMAIL*, até ao *Facebook*. No entanto isto só acontece porque os utilizadores comuns não se previnem.

Um outro exemplo prático, mas com maior complexidade é fazer um ataque por *Man in the Middle*. Este ataque aproveita-se de uma funcionalidade/vulnerabilidade a partir do qual é possível fazer uma actualização das entradas da *Cache ARP*, obrigando a que todo o tráfego destinado a uma máquina seja redireccionado para uma outra distinta. Desta forma a máquina *C* da figura 2.2 fica a funcionar como sistema central para onde todo o tráfego é redireccionado. Após esta acção o *hacker* poderá passar para um modo mais activo procedendo mesmo a alteração de informação ou bloqueio de acessos à rede. A grande desvantagem deste ataque para o *hacker* é a necessidade de acesso físico à rede.

Uma outra forma de fazer *hacking* consiste na procura de falhas na componente administrativa. Ao longo da investigação para este projecto, foi possível encontrar, *online*, graves falhas de segurança causadas pelos administradores de sistema. Através da utilização do motor de

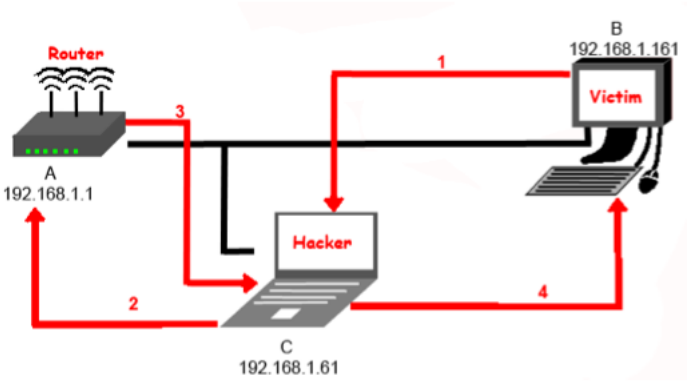

Figura 2.2: Man in the Middle

busca *Google* é possível obter dados, como por exemplo ficheiros de backup php, que possibilitam a obtenção de dados confidenciais, como credenciais de bases de dados, sites, entre outros[23], o que pode levar ao controlo por parte de pessoas não autorizadas de sistemas.

2.4 Anatomia do ataque

Um ataque a um sistema informático é uma operação complexa que deve ser cuidadosamente planeada. Esta é uma necessidade, uma vez que só assim será possível ao *hacker* realizar as operações que pretende, no menor espaço de tempo possível. Um bom planeamento e execução, para além de aumentar a probabilidade de sucesso do ataque, reduz a possibilidade de detecção e consequentemente a possibilidade de bloqueio por parte do sistema vítima.
Sendo assim, e tal como referido no capítulo um, um ataque pode ser dividido em cinco fases [18] *Recon, Scan, Exploring, Exploiting e Expunging.*

- **Recon** não deixa qualquer traço na rede por não gerar tráfego anómalo. Durante esta fase, motores de busca como o *Google, Yahoo! ou Bing* entre outros, são utilizados de forma a verificar dados sobre os sistemas vítima. Na actualidade, os motores de busca funcionam como agregadores de informação, sendo por isso muito poderosos. Executando os comandos apropriados é possível obter todas as informações sensíveis;[18]

- **Scan** é a primeira fase do ataque que deixa registos na rede neste ponto o *hacker* deixou de fazer uma observação comum, para iniciar um processo de pesquisa de vulnerabilidades. Neste ponto se o sistema de detecção for eficiente, será difícil para o *hacker*

obter informações correctas e importantes sobre os serviços e máquinas disponíveis;[18]

- **Explore** neste ponto, o *hacker* começa a recolher informação mais relevante e a criar um *profile* de ataque para que o passo seguinte, *Exploiting*, seja bem-sucedido;[18]

- **Exploit** com isto, o *hacker* aproveita-se de vulnerabilidades detectadas nos pontos anteriores, de forma a obter acessos a dados não autorizados, podendo mesmo vir a tomar controlo das máquinas vítima. Neste ponto o ataque já ocorre "dentro de portas". Durante o *Exploiting* o agente malicioso pode efectuar operações tais como instalar serviços, adicionar excepções à *firewall*, até aceder/alterar/remover ficheiros, remover/adicionar/alterar utilizadores, entre outros.[18]

- **Expung** este ponto é crítico para manter o anonimato do *hacker* este é o último processo que este executa. Aqui o agente malicioso procede à remoção de todas as evidências que possam apontar para si no caso de análise. Sendo bem-sucedido neste ponto, no sistema podem ter sido criados serviços que podem permitir a execução de diversas tarefas, sendo que não existindo evidências de comprometimento de informação, estes são tipicamente ignorados pelos sistemas automáticos de protecção.[18]

2.5 Scan, o pré-ataque

Para que um *hacker* seja bem-sucedido, de entre vários factores, é necessário que ele esteja bem preparado. Sabendo quais são as possíveis vulnerabilidades e desta forma como poderá ele proceder para ultrapassar os obstáculos que lhe vão surgindo.[18, 2, 38]
Existem duas formas de este poder obter estas informações. Subvertendo os administradores do sistema, dando-lhe informações importantes, ou através de *scans* efectuados à rede ou máquina de forma a obter directamente a maior quantidade de informações possível. A grande diferença entre estas metodologias reside em dois factores. No primeiro ponto é necessário que o *hacker* saiba quem é o administrador de sistema, sendo que por este método não pode ser detectado por nenhum *IDS*. Por sua vez o segundo método pode ser efectuado remotamente e pode ser detectado pelo *ID* se este estiver correctamente configurado.
Os *scans* são então uma forma de poder obter informação útil sobre as variadas máquinas,

podendo até mesmo dizer quais as vulnerabilidades e como se podem explorar. Existem duas ferramentas comummente utilizadas para este efeito, o *nmap* e o *nessus* [31]. Estas duas são extremamente poderosas, funcionais e de fácil utilização, sendo que o *nessus* tem um papel mais profundo, dado verificar quais as vulnerabilidades dos sistemas e não apenas apresentando os serviços e portas disponíveis.

As aplicações de *scan* são muito úteis, dado permitirem aos administradores de sistema saber o que existe a trabalhar naquilo que estão a gerir, verificando se existem ou não serviços não autorizados ou duvidosos. Estas aplicações permitem um grande leque de configurações, fazendo com que elas possam trabalhar em modo praticamente furtivo, passando despercebidas aos *IDS*.

Um *portscan* baseia toda a sua funcionalidade na forma como os protocolos trabalham, retirando então algumas conclusões tendo em conta a resposta obtida. Para o caso de estudo apenas teremos em consideração a forma como o protocolo TCP funciona, sendo assim é importante considerar:

- existem cerca de 65500 portas disponíveis numa máquina e às quais pode ser associado um serviço;[7]
- se a porta estiver aberta ou um serviço funcional, a máquina irá responder dizendo que se encontra disponível para comunicar;[7]
- se a porta estiver fechada é dado um reply de que esta não está a aceitar conexões;[7]
- se a máquina estiver configurada para não divulgar os serviços disponíveis, esta não responderá.[7]

Este é uma parte do processo *three way handshake*. Por definição este processo tem dois objectivos máximos, garantir que as duas máquinas estão prontas para iniciar comunicação e assegurar que os números de sequência estão acordados mutuamente, sendo este um ponto muito importante pois só assim é possível garantir a fiabilidade e a não perda de pacotes durante a transmissão.[7]

Estes números são então aleatórios e diferentes entre todas as transmissões durante um largo período de tempo, sendo que este processo também é bastante rápido dado que é realizado apenas em três etapas. Isto apenas é possível porque ambas as tramas contêm um número de sequência e um *ACK*. Tendo isto em conta e considerando ainda que na figura 2.3 a máquina do lado esquerdo é o Bob e a do lado direito é a Alice o algoritmo explica-se da seguinte forma:

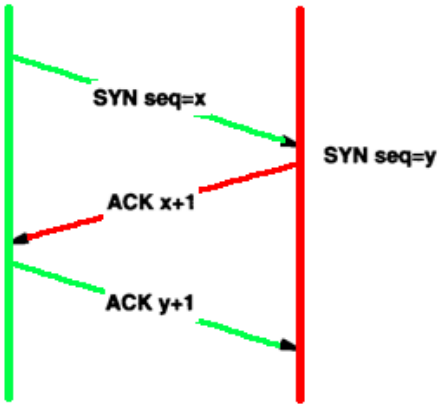

Figura 2.3: TCP Three Way Handshake

1. o Bob envia para a Alice um *SYN* com o seu número de sequência *y*;
2. a Alice guarda esse valor e responde ao Bob com o seu número de sequência*x* e com o *ACK y+1*;
3. por fim Bob recebe a trama e responde para a Alice com o *ACK x+1* terminando desta forma o processo.

Existem diversas variantes de sistemas de *scan*, no caso mais básico, também conhecido por *SYN scan*, o processo de *three way handshake* raramente é terminado, isto é, após o *nmap* emitir um *SYN* a uma determinada porta [code:vermelho] caso a porta esteja disponível emite um *SYN-ACK* [code:verde-claro], então o *nmap* responde com um pacote *RST*, terminando de imediato a ligação [code:azul]:[3, 30, 5, 5]

```
nmap 192.168.0.1
(...)
20:37:11.372374 IP blacktigermacbook.home.52279 > zonhub.home.http: Flags [S], seq 167611
20:37:11.373500 IP zonhub.home.http > blacktigermacbook.home.52279: Flags [S.], seq 14176
20:37:11.373627 IP blacktigermacbook.home.52279 > zonhub.home.http: Flags [R.], seq 1, ac
(...)
```

Este é o método utilizado pelo *nmap* em modo pré-definido, sem nenhuma opção activa. É também o mais comum por parte de quem está a aprender a utilizar esta ferramenta. Este método é muito pouco aconselhável uma vez que itera todas as portas, o que provoca a geração de elevadas quantidades de tráfego fora do normal, sendo por isso facilmente detectável pelos

IDS.

Os tipos de *scan* mais comuns são aqueles resumidamente descritos de seguida:

- **TCP connect *scan*** este método é o mais básico e mais facilmente detectável, já que completa o *Three Way Handshake*, terminando a ligação logo de seguida. Esta característica torna este *scan* facilmente detéctavel, uma vez que não há transmissão de dados entre as máquinas após o estabelecimento de comunicação;[3, 30, 5]

- **TCP SYN *scan*** neste caso o *Three Way Handshake* não é completo.[3, 30, 5]

  - se a porta estiver disponivel, a máquina vítima irá responder com um *SYN-ACK*;[3, 30, 5]
  - se a porta não estejá disponível, é emitido um *RST ACK* ou um *ICMP unreachable*.[3, 30, 5]

  No primeiro caso, a máquina que providenciou o *scan*, após receber a trama *SYN-ACK* irá responder com um *RST ACK* terminando de imediato com a ligação. Esta técnica tem a vantagem de passar despercebida à maioria dos *IDS*, isto porque, por defeito, estes não se encontram configurados para tomar atenção a este detalhe. No entanto e como poderá conferir no capítulo 4 este é um dos aspectos que o *IDSNN1* toma em consideração. Em contrapartida este tipo de *scan* pode causar indisponibilidade do serviço por *DoS* devido ao elevado número de conexões não terminadas.[3, 30, 5]

- **TCP FIN *scan*** para proceder a este *scan* o atacante envia um pacote *FIN* e então espera que seja recebido um pacote *RST*. Este pacote apenas é recebido se as portas estiverem fechadas;[3, 30, 5]

- **TCP Xmas Tree *scan*** com base no *RFC 793*, a máquina responderá com um pacote *RST* se a porta estiver fechada após a recepção de um pacote *FIN*, *URG* e *PUSH,1*;[3, 30, 5]

- **TCP Null *scan*** novamente com base no *RFC 793*, a máquina destino responderá com um pacote *RST* se a porta estiver fechada, após receber um pacote enviado pelo atacante, que não contém nenhuma *flag* activa.[3, 30, 5]

- **TCP ACK *scan*** se a porta estiver a ser filtrada por uma *firewall*, não haverá resposta, ou então será emitido um *ICMP-unreachable*, no caso contrário será emitido um pacote *RST*. Este tipo de *scan* é útil na medida em que permite saber o que está ou não está a ser controlado e deste modo planear com maior destreza o ataque a ter em conta. É no entanto também uma metodologia para detectar possíveis *scans*, uma vez que se houver um grande volume de pacotes *TCP-ACK* a ser emitidos e um igualmente grande número de pacotes *RST* ou *ICMP* como resposta significa que a probabilidade de vir a ocorrer um ataque é elevada;[3, 30, 5]

- **TCP Windows *scan*** tal como a técnica anterior, o objectivo desta é detectar a monitorização por parte das *firewall* em alguns sistemas. Esta técnica aproveita-se de uma falha reportada em sistemas *AIX e FreeBSD* no tamanho da janela *TCP*;[3, 30, 5]

- **TCP RPC *scan*** técnica exclusiva para sistemas *Unix*. É utilizada para detectar e identificar portas *RPC* assim como as aplicações associadas e as suas versões;[3, 30, 5]

- **UDP *scan*** Através deste método é enviado um pacote *UDP* para a máquina destino.[3, 30, 5]

  Dado o protocolo *UDP* ser um protocolo não orientado à conexão, a fiabilidade desta técnica é duvidosa, sendo o processo também demorado (em comparação com técnicas que utilizam *TCP*). O sucesso desta operação depende entre outros de:

  1. monitorização da máquina destino por uma *firewall*;
  2. carga da rede;
  3. número de saltos da rede.

  Sendo assim, se a máquina responder com um pacote *ICMP unreachable* significa que esta se encontra fechada. Se não houver resposta, isto pode significar que a porta está aberta, ou que o pacote pode ter-se perdido. Uma nota importante a ter em conta é que a taxa de sucesso desta técnica através da internet é muito diminuta.

Exemplos de *scan*s com Nmap

O resultado A.2 é referente a um *scan* cujo objectivo era detectar *hosts* activos, e respectivos serviços. Este *scan* enquadra-se no tipo um, enumerado anteriormente, uma vez que existe a pesquisa em toda a gama de *IPs* e um elevado número de portas.

O caso acima demonstra que existem três serviços a correr na máquina 192.168.0.4 desconhecidos, o que pode representar uma ameaça para a segurança da máquina e da própria rede.
O resultado A.1 representa os dados originados por uma pesquisa de *hosts*, cujo único objectivo é a pesquisa de máquinas activas. Este é um *scan* mais cuidadoso, mas que ainda assim pode ser detectado.[30, 5]
Por fim o teste A.3 é o resultado de um *scan* furtivo com a utilização apenas de *FIN*. Este é o teste mais bem-sucedido no que toca a escapar a detecções, uma vez que os *IDS* apenas fazem parelha *ACK/FIN*.
Outra nota importante prende-se com o facto de este teste ter retornado quais são as portas/serviços a ser filtrados por uma *firewall*.

2.6 Detectores de intrusões de rede

Os detectores de intrusões são aplicações de *software* que fazem a monitorização do tráfego analisando-o, de forma a detectar qualquer situação anómala que possa colocar em risco a confiabilidade, integridade ou disponibilidade de um sistema informático.
Este conceito foi inicialmente introduzido por Anderson em 1980, tendo sido profundamente estudado ao longo dos últimos 30 anos.
Os *IDS* baseiam-se no facto de um *hacker* ter que deixar sempre um "rasto"por onde passa, dependendo sempre do tipo de acção que pretendem tomar no futuro. Se por ventura pretendem fazer nova conexão ao sistema, são obrigados a alterar alguma configuração deste, de forma a garantir sempre o acesso. Por norma, este rasto é detectável em tempo real, porém e como mencionado anteriormente, é possível fazer uma previsão, detectando o apelidado *pré-ataque* ou *scan*.[28, 19]
É importante perceber ainda que estes dispositivos podem encontrar-se instalados em máquinas em modo *furtivo* de forma a não ter um endereço de IP, evitando assim a sua detecção e possível aproveitamento por parte de um *hacker*. Uma outra vantagem de não ter endereço de IP atribuido, passa por não haver geração de tráfego para aquele *host* e, deste modo, menos carga na rede.
Estes sistemas podem dividir-se em duas classes distintas, os *host-based* e *network-based*. Os sistemas *host-based* encontram-se instalados e configurados numa máquina local e o seu objectivo é tratar o tráfego que chega ao computador onde se encontra configurado.
Os sistemas *network-based* são aplicações instaladas numa máquina central por onde passa

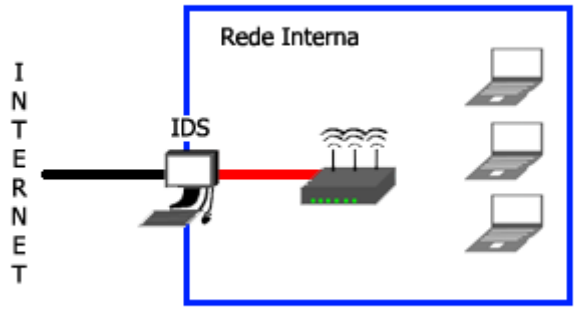

Figura 2.4: Intrusion detection System Diagram

todo o tráfego gerado na rede.[6] Estas aplicações têm o objectivo de monitorizar apenas o tráfego, não verificando se há alteração de ficheiros em máquinas individuais. Tal como os *IDS host-based*, a grande maioria destes fazem uma detecção por comparação de padrões entre o tráfego recebido/enviado e os dados existentes em registo.[28, 19]

2.6.1 Classes de IDS

Apesar da diversidade de *IDS*, é possível fazer o seu agrupamento em classes gerais, aqui apresentam-se quatro delas, *Anomaly IDS, Signature IDS, Behavior-Based IDS, Hybrid IDS*:[19, 1, 28, 12]

- *Anomaly IDS*: Através da análise dos cabeçalhos dos pacotes interceptados é possível fazer numa primeira instância, a verificação da validade dos pacotes. Isto acontece devido ao facto de um grande número de ataques fazerem alterações activas nos pacotes. Se estas alterações diferirem mais do que o valor de *threshold*, existe a geração de um alerta. Dada a generalização da utilização das redes, constantemente existem pequenas alterações e consequentemente existe a geração de um grande volume de falsos positivos.[19, 1, 28]

- *Signature IDS*: Esta é a técnica mais comum, utilizando um *parser* e uma base de dados é possível verificar se as assinaturas correspondem ou não a um ataque.

- *Behavior-Based IDS*: Esta técnica assume que é possível fazer a detecção de anomalias,

através da detecção de desvios no comportamento dos utilizadores ou do sistema, isto é, geração de tráfego não esperado. Este processo é feito através da comparação dos dados recolhidos com os esperados, caso exista um desvio, é gerado um alerta.
A maior desvantagem deste sistema é a geração elevada de falsos positivos.[19, 1, 28]

- *Hybrid IDS*: Os sistemas híbridos, são aqueles que fazem a conjugação de duas ou mais classes de *IDS*.[19, 1, 28]

2.6.2 Falsos Positivos

Falsos positivos são todos os alertas gerados por um *IDS*, que não têm nenhuma relação com questões de segurança. Estes alertas são gerados a partir de tráfego completamente legítimo que é mal interpretado pelo *IDS*. Na verdade este tipo de resultados tem, normalmente, um valor bastante elevado, quando os *IDS* não sao configurados correctamente. Este é um factor extremamente negativo, tendo em conta que, usualmente, os alertas devem ser inspeccionados por um administrador de sistema.[28, 19]
O desafio proposto a um *IDS* é ter rácio elevado entre os alertas legítimos e os falsos positivos. Este é um desafio muito complicado uma vez que por vezes, para ser possível a detecção de uma intrusão, será necessário fazer uma análise de todo o contexto de tráfego, objectivo que os *IDS* actuais não conseguem cumprir. Isto porque por vezes o tráfego, em particular, não contém nenhuma evidência do ataque, sendo necessária uma vista superior de tudo o que se passa.
Outro desafio que os *IDS* enfrentam é a necessidade de ser facilmente configuráveis a diferentes ambientes, isto porque num ambiente profissional de redes certas acções não devem ser consideradas maliciosas enquanto. Num ambiente distinto estas já o podem ser. [28, 19] Por outro lado, e tendo em conta que alguns *IDS* têm por base o *Pattern Matching*, será importante perceber que a dificuldade na criação destas regras é elevada[28, 19], sendo necessário criar um equílibrio, de forma a que não passem em claro as intrusões, mas que também não seja considerado malicioso, tráfego comum.

2.6.3 Factores que influenciam a eficácia dos *IDS*

Desde o trabalho publicado por Anderson em 1980, até aos dias de hoje, os *IDS* sofreram uma grande evolução, tendo sempre como objectivo o aumento da sua eficácia.
No entanto, e no seguimento da secção anterior, é importante ter em conta que há vários factores que influenciam a eficácia destes sistemas. Numa primeira instância deve-se considerar o ambiente no qual este se encontra integrado, isto porque este deve ser pensado cuidadosamente, não só de forma a garantir que a máquina *IDS* é uma máquina central, como o próprio sistema se encontra configurado de forma consistente. O próprio sistema deve ser alimentado com informações, que devem ser obtidas pelo administrador do sistema (utilizando por exemplo *scanners*, *exploit systems*) de forma a ser configurado apenas para aquele meio em concreto. Desta forma é possível fazer a relação entre vulnerabilidades e alertas de forma a reduzir os falsos positivos.[28, 19, 2, 38, 18]
O administrador do sistema é também uma peça fundamental. É necessário que este seja um indivíduo entendido na área da segurança e seja multidisciplinar podendo avaliar de forma geral e num curto espaço temporal os eventos que vão ocorrendo, adaptando não só o *IDS* de forma a torná-lo mais eficiente, como também eliminar falhas de segurança que apenas permitirão a geração de mais tráfego malicioso.
Por fim, numa terceira instância, a robustez do próprio sistema. A robustez destes sistemas foi evoluindo ao longo dos últimos 30 anos, passando de aplicações de *software* que só faziam comparação de assinaturas, para aplicações com aprendizagem, capacidade de contextualização e rapidez na detecção. Isto aconteceu graças à evolução da própria engenharia informática, que permitiu o melhoramento de performance, a inclusão de concorrência e até a simulação do comportamento cerebral, utilizando *RNA* de forma a optimizar este processo.
É, ainda importante saber qual é o tipo de *IDS* que melhor se adapta à rede onde vai ser implementado. No caso de estudo apenas se abordam os *N-IDS* sendo que mesmo nesta classe é necessário ter em consideração se se pretende um sistema que análise aspectos específicos ou um que tome em consideração um contexto mais alargado. Para referência ficam quatro soluções que propõe quatro tipos distintos, três são específicas (baixo número de falsos positivos) e uma outra é generalizada (elevado número de falsos positivos):

- análise de baixo nível, de forma a detectar *scans*, tentativas de *DoS*;[2, 38, 18]
- análise ao nível da aplicação, de forma a detectar *VNC-Injection*, *SQL-Injection*, *PHP-Oriented Attacks*;[2, 38, 18]

- detecção de *worms, vírus*;[2, 38, 18]
- análise genérica de tráfego (todo o que chega e sai da rede) como o *Snort*;[2, 38, 18]

CAPÍTULO 3

Redes Neuronais Artificiais

3.1 Introdução

As redes neuronais surgiram nas décadas de 40/50 através dos estudos realizados por *Warren McCulloch* e *Warren Pitts*. Foi então em 1943 que estes dois estudiosos apresentaram o primeiro modelo de neurónios artificiais, naquele que foi o primeiro artigo a ser publicado sobre o tema, e cujo título é *"A Logical Calculus of Ideas Immanent in Nervous Activity"*.[32] Obviamente e dada a novidade do tópico de estudo, este artigo acabou por se tornar fonte de inspiração para a grande maioria dos trabalhos posteriormente realizados.

É possível então depreender que este tema é de elevada importância, dado que a compreensão do funcionamento do cérebro é a única forma existente, que nos permitirá usufruir mais das suas reais capacidades. Na realidade, ao longo dos últimos 50 anos, 10% dos prémios Nobel foram atribuídos a estudos directamente relacionados com o funcionamento dos neurónios, sinapses e outras estruturas das *RNA*.[32]

Na actualidade, as *RNA* desempenham um papel preponderante para o processamento de dados, sendo que a sua grande vantagem e diferença relativa à programação usual, reside no facto de permitir paralelismo, aprendizagem, fiabilidade e consequentemente generalização. Esta generalização advém do facto de ser possível obter aproximações com grande grau de confiabilidade, apesar de os dados processados poderem ser distintos dos dados utilizados para treino. Por esta razão, hoje em dia estas estruturas são muito utilizados para realizar tarefas computacionalmente pesadas, como por exemplo reconhecimento de padrões, aproximação de funções, entre outros.[32]

Outra grande vantagem é a possibilidade de mais facilmente conviver com a falha, onde a perda de performance é diminuta. Isto é, no cérebro, anualmente, "morrem"centenas de

neurónios, no entanto o ser humano não nota essa perda, pois graças ao paralelismo, somos capazes de ainda assim continuar a desempenhar as mesmas tarefas. O mesmo acontece com as *RNA*. Se por alguma razão houver alguma falha, o sistema continuará a funcionar, podendo no entanto haver alteração do resultado final.[32]

Tipicamente uma *RNA* é a agregação de vários neurónios dispostos em camadas, sendo que, por norma neurónios de uma mesma camada têm o mesmo comportamento. Esta característica é de significante importância, dado que desta forma é possível fazer uma leitura directa sobre aquilo que cada neurónio está a fazer. [32]

Na actualidade as *RNA*, começam a ser amplamente aceitáveis como uma forma fidedigna de programação, simplificando muitas vezes não só a programação como também a algoritmia, podendo ser aplicada a uma grande panóplia de casos distintos.[7]

3.2 Principios Básicos

Como enunciado na secção anterior, *Warren McCulloch e Warren Pitts* foram os primeiros estudiosos a propor o uso das *RNA*, tendo enumerado então quatro princípios básicos:[32]

1. um neurónio ou está em actividade ou está em repouso, não havendo meio termo;
2. o único atraso de processamento apenas pode estar associado às sinapses;
3. se uma sinapse tem uma actividade inibidora, o neurónio a si associado não poderá ser excitado;
4. a estrutura de ligações entre neurónios (sinapses) não poderá ser alterada ao longo do tempo;

Com estas restrições estabelecidas é então possível retirar uma série de conclusões:

1. um neurónio é um elemento binário;
2. em caso de falha de uma sinapse/neurónio, existe uma produção de resultado que pode não ser fidedigno;
3. a rede não pode evoluir com o passar do tempo;

Estas conclusões são de grande importância pois demonstram limitações associadas, na época, a este tipo de estrutura.[7]

3.3 Propriedades das *RNA*

As *RNA* tem então diversas propriedades e capacidades, resumidas seguidamente:[16, 13]

- **uma *RNA* pode ser linear ou não linear** uma rede neuronal feita através da ligação de neurónios cuja função seja não linear, é considerada também uma rede neuronal.

- **input-output mapping** uma propriedade importante reside no paradigma de *aprendizagem supervisionada*. Esta propriedade dita que os *pesos* atribuídos a cada uma das sinapses seja alterado através da análise de uma amostra exemplo. Cada amostra consiste no conjunto constituído por dados de entrada e resultado esperado. Com isto, os valores dos *pesos* das sinapses são alterados de forma a reduzir o erro entre a previsão e o resultado esperado. O treino da rede é repetido quantas vezes for necessário de forma a que o erro final seja um valor aceitável ou até que este valor não sofra alterações significativas.

- **adaptatividade** uma *RNA* tem a funcionalidade de se adaptar ao longo do tempo, fazendo alterações aos pesos atribuídos às sinapses. Em particular uma rede neuronal que seja treinada para responder de uma determinada forma, pode facilmente ser retreinada de forma a responder positivamente a alterações que surjam. Por esta razão se diz que uma rede neuronal tem sempre em consideração o ambiente envolvente. Isto acontece, uma vez que em alguns casos os ambientes podem não ser *estacionários*. Esta condição permite assegurar então que:

  > "Quanto mais adaptável for o sistema, assegurando sempre que este se mantém estável, mais robusto será e maior será a sua performance."

  Porém, deve-se ter a noção que esta condição nem sempre é aplicavél. Se por exemplo, as adaptações foram em grande número num curto espaço de tempo, o sistema não terá a capacidade de interpretar as perturbações, o que levará uma rápida degradação deste. Por esta razão, devemos considerar a seguinte condição para que:

  > *um sistema adaptativo apenas pode ser considerado fiável e robusto se o seu rácio entre actualizações e tempo for o suficiente para que este possa interpretar correctamente perturbações que ocorram durante a alteração*

Na prática o que isto significa é que se durante uma análise de dados, houver alteração dos pesos das sinapses, os resultados estarão adulterados não sendo por isso fidedigno. Por esta razão deve haver tempo suficiente para que após a alteração possam ser gerados novos resultados fidedignos.[14, 16, 13]

- **Tolerância à falha** uma rede neuronal implementada ao nível do hardware, tem que ter a habilidade de ser tolerante a falhas, caso contrário existe a certeza que a sua performance se irá degradar substancialmente ao longo do tempo. A título de exemplo, se um neurónio ou uma sinapse falharem por alguma razão, os resultados dos dados a ser processados teoricamente serão afectados. No entanto neste caso devemos relembrar o ponto anterior. Adaptando o sistema neste caso, verificar-se-á que este responderá de forma fiável sendo o seu resultado de elevada confiança. Apesar deste processo poder levar algum tempo a acontecer, isto é preferível à falha crítica que possivelmente inviabilizará o sistema;[13]

- **Nível de Confiança** quando se tratam de redes neuronais, cujo objectivo é a detecção de padrões, uma informação extremamente útil é o valor de confiança. Esta característica das redes neuronais apenas tem eficácia quando são utilizadas funções contínuas, isto porque o resultado varia continuamente entre dois valores. No caso de duas detecções ambiguas, este valor permite que seja escolhido o padrão com maior nível de confiança;[16, 13]

- **Paralelismo em larga escala** a possibilidade de implementar uma rede neuronal de grandes dimensões, permite que o nível de paralelismo seja extremamente elevado. Se a isto se adicionar o facto de o paralelismo poder não ser apenas por software, mas também por hardware, temos uma das propriedades mais importantes das redes neuronais. Esta propriedade permite que ao longo do tempo o processamento informático se aproxime do processamento cerebral;[16, 13]

- **Uniformidade no design e estrutura** as redes neuronais são estruturas que cujo design segue um padrão predefinido. Esta propriedade facilita o trabalho dos programadores, uma vez que é possível fazer uma generalização básica. Esta propriedade é demonstrada de variadas formas, três delas encontram-se enunciadas:[16, 13, 32]

1. todas as redes neuronais são constituídas por camadas de neurónios, contendo sempre no mínimo duas (entrada e saída);
2. esta propriedade permite a troca de teorias e a optimização dos algoritmos de aprendizagem, já que o leque de aplicações é muito elevado;
3. possibilidade de implementar uma rede neuronal através da conjugação de diversos módulos previamente programados.

- **Semelhança ao comportamento cerebral** tendo a sua génese sido motivada para simular o comportamento humano, a comparação entre estas e o cérebro está intimamente ligada. Este é um factor muito positivo já que torna possível a evolução destas estruturas ao longo do tempo e à medida que o conhecimento sobre o cérebro vai aumentando. Na realidade, os neurobiólogos estudam não só o comportamento do cérebro, mas também o comportamento destas estruturas, de forma perceberem empiricamente como reage ou não o cérebro a determinados estímulos. No entanto não são apenas os neurobiólogos que beneficiam desta relação, os engenheiro também, uma vez que desta surgem soluções para resolver problemas relacionados com o *design* e as técnicas utilizadas.[32]

3.4 Estrutura

As *RNA* podem ser constituídas por três camadas, uma de entrada (*input Layer*), uma de processamento (*middle layer*) e por fim a de saída (*output layer*). É importante ter em conta que a camada de processamento pode ser constituídas por várias camadas de neurónios, ou por nenhuma, havendo assim uma ligação entre a camada de entrada e a camada de saída.
As redes neuronais artificiais, funcionam muitas vezes como uma *black box*, para o programadores, dado ele poder não saber qual a sua organização, quantas camadas tem, ou até quais as funções de activação.[16, 13] A figura 3.1 representa exactamente este conceito de abstracção, demonstrando que há uma entrada de valores, e uma saída de valores. Não sabendo porém que processos são realizados internamente.
Geralmente as *RNA* têm o objectivo de gerar m resultados a partir de n valores de entrada. Isto significa que a rede se comporta como uma máquina de "conversão"ou "mapeamento"capaz de modelar a função

$$F : \mathbb{R}^n \mapsto \mathbb{R}^m, \tag{3.1}$$

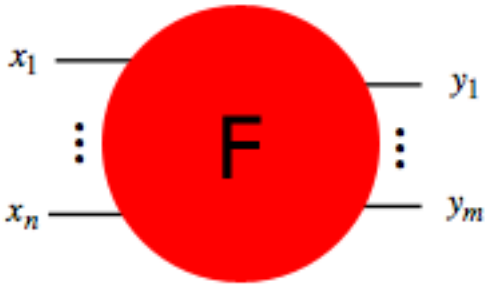

Figura 3.1: Camadas Escondidas como *black box*

3.4.1 Neurónios

Um neurónio é uma unidade de processamento fundamental para as redes neuronais, já que são estas as unidades básicas que constituem as *RNA*. Um neurónio, ou unidade de processamento tem três propriedades básicas:[16, 13, 32]

1. é constituída por um conjunto de sinapses que fazem a ligação entre os diversos neurónios. Cada uma delas tem um *peso* atribuído, que é um factor para regulação dos valores a ser processados.[16, 13, 32] De forma sucinta, dadas

$$y_a, \tag{3.2}$$

e

$$w_{na}, \tag{3.3}$$

em que y é um sinal de entrada na sinapse a, n é o neurónio ligado à sinapse e w é o *peso* da sinapse o resultado será obtido através da multiplicação dos dois factores

$$rt = y_a * w_{na}, \tag{3.4}$$

2. um somatório que permite fazer a operação sobre os dados de entrada e os pesos das respectivas sinapses, tendo estas operações o mesmo número de *combinadores lineares*

3. uma função de activação que permite limitar a amplitude do resultado do neurónio, permitindo assim que este tenha um topo máximo e mínimo uniformizando os resultados e possibilitando, deste modo, a sua comparação.[16, 13, 32] Para o nosso caso de estudo, a função de activação é contínua não linear variando o seu resultado entre zero (0) e um (1).

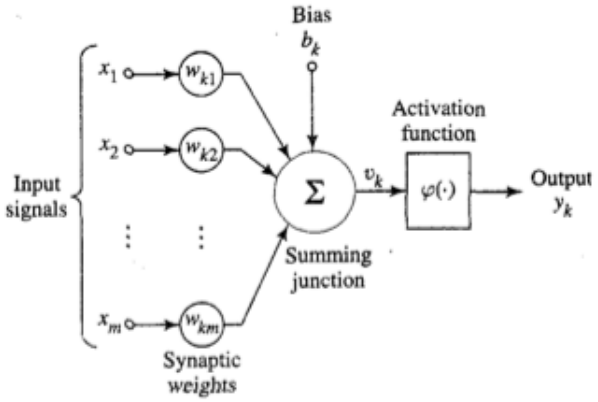

Figura 3.2: Estrutura de um neurónio [16]

A estrutura de um neurónio pode então ser representada pela figura 3.2. Aqui encontra-se ainda representado um *bias*. Este elemento tem o objectivo de aumentar ou diminuir a entrada da rede da função de activação, dependendo se o valor é respectivamente, positivo ou negativo.[16, 13, 32]

De forma matemática descrevemos o neurónio k através do seguinte conjunto de equações:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j, \tag{3.5}$$

e

$$y = \varphi(u_k + b_k), \tag{3.6}$$

onde as seguintes variáveis representam:

- $x_1, ..., x_m$: dados de entrada;
- $w_{k1}, ..., wkj$: os pesos das sinapses do neurónio k;
- u_k: o combinador linear do sinal de entrada;
- b_k: o *bias*;
- y_k: resultado do neurónio;
- $\varphi()$: a função de activação.

3.4.2 Funções de Activação

De forma a simplificar a representação, declara-se que v_k representa a seguinte combinação:[16, 13]

$$v_k = u_k + b_k, \tag{3.7}$$

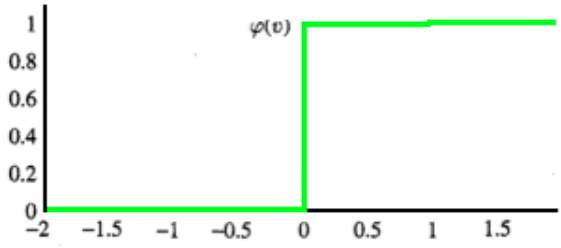

Figura 3.3: Função *Threshold*

Tendo em consideração a função 3.7, podemos então explanar de forma simplificada as funções de activação do tipo $\varphi(v)$.
Neste subcapítulo são abordados três tipos básicos de funções de activação:

Treshold

Esta é a função que descreve o principio básico número 1 do subcapítulo 3.2. Sendo que as equações:

$$\varphi(v) = \begin{cases} 1 & sev \geq 0 \\ 0 & sev < 0 \end{cases} \tag{3.8}$$

e:

$$y_k = \begin{cases} 1 & sev_k \geq 0 \\ 0 & sev_k < 0 \end{cases} \tag{3.9}$$

onde v_k representa:

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{3.10}$$

e que permitem que o resultado final seja de apenas zero ou um. Análogamente ao descrito anteriormente, estes resultados representam activo ou inactivo.[16, 13] Esta encontra-se representada grafiicamente pela figura 3.3 e analiticamente pelas equações 3.8 e 3.9.

Piecewise-Linear

$$\varphi_k = \begin{cases} 1, & v \geq \frac{+1}{2} \\ v, & \frac{+1}{2} > v > \frac{-1}{2} \\ 0, & v \leq \frac{-1}{2} \end{cases} \tag{3.11}$$

Graficamente esta função encontra-se representada na figura 3.4, analiticamente pela equação 3.11 onde o factor de amplificação dentro da região linear da operação, assunme-se como sendo

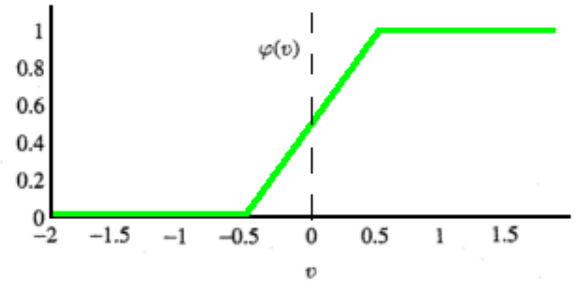

Figura 3.4: Função *Piecewise-Linear*

a unidade. Esta forma de função de activação pode ser vista como uma aproximação a um amplificador não linear. Os dois casos seguidamente descritos podem ser tidos como formas específicas de funções lineares:*piecewise*[16, 13]

- Combinador linear: poderá aparecer se a região linear do operando não entrar em saturação;
- esta função reduz o *threshold* se o factor de amplificação da região for infinitamente grande.

3.4.3 Função *sigmoid*

Esta função é aquela que é usualmente utilizada como função de activação utilizada na construção de redes neuronais, tendo sido também a escolhida como função de activação para este projecto.

A sua definição diz que é uma função com o objectivo de aumentar, exibindo um grande equilíbrio entre o comportamento linear e não linear.[13] A função 3.12 é apenas um exemplo representativo de uma função *sigmoid*, sendo a sua representação gráfica apresentada na figura 3.5.

$$\varphi_v = \frac{1}{1 + exp(-av)} \tag{3.12}$$

onde *"a"* é um parâmetro de inclinação, cuja variação resulta em diferentes inclinações da função tal como ilustrado pela figura 3.5. Na verdade a inclinação na origem, ponto (0,0), tem a inclinação de $\frac{a}{4}$, sendo que no limite, à medida que o parâmetro de inclinação se aproxima do infinito, a função *sigmóide* converte-se numa função *threshold*.

Em particular esta função, distingue-se da função *threshold* por ser uma função continua no intervalo [0,1], ao contrário da função *threshold* cujos valores apenas podem ser 0 ou 1. E por a função *sigmóide* ser uma função diferenciável, ao contrário da *threshold*.[16, 13, 32]

Com esta transformação é então possível ter uma função de activação do tipo *sigmóide* assumindo valores negativos, o que traz consigo benefícios de análise de resultados.[16, 13]

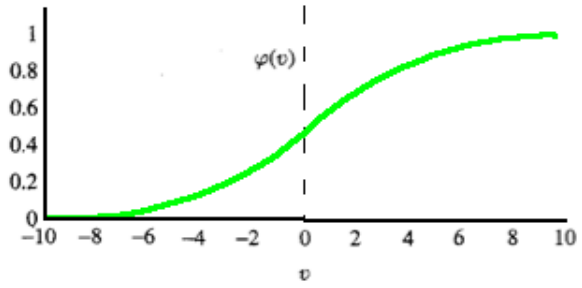

Figura 3.5: Função Sigmoid

Por vezes, é desejável que as funções de activação tenham um resultado variável entre -1 e +1, sendo que neste caso a função assume uma forma assimétrica tendo o seu ponto de origem na coordenada (0,0). Tendo isto em consideração, pode-se fazer a transposição da função *threshold* 3.8 para a função 3.13, referindo-nos a esta como função *signum*

$$\varphi_v = \begin{cases} 1 & sev > 0 \\ 0 & sev = 0 \\ -1 & sev < 0 \end{cases} \tag{3.13}$$

Da mesma forma pode-se fazer a transposição da função *sigmóide*, utilizando a função *tangente hiperbólica*, definida por 3.14.[16, 13]

$$\varphi_v = tahn(v) \tag{3.14}$$

3.4.4 Camadas

Input Layer

A camada de entrada, é como o próprio nome indica, a camada que recebe os dados a ser processados. Como todas as outras, partilha da propriedade de todos os seus neurónios terem características semelhantes. Esta camada pode ter *0 ou n* neurónios.[16, 13]

Camada Escondida

Como previamente declarado, as redes neuronais contêm uma camada de entrada e uma camada de saída, sendo que por vezes podem também conter uma camada intermédia.
Usualmente esta camada pode ser apelidada de *camada de processamento*, sendo que este

nome pode ser falacioso. Isto porque não é apenas esta camada que faz o processamento de dados, a camada de saída também processa os valores recebidos de forma a dar um resultado final. Numa rede neuronal a única camada que é "cega"é a camada de entrada, camada esta, cujos neurónios não interferem no resultado final.[16, 13]
No entanto, o nome mais usualmente utilizado é de *hidden layer*, em português, camada *escondida*, isto porque é a camada que não tem qualquer tipo de interacção com o ambiente em seu redor, mas sim e apenas com outras camadas de neurónios da mesma rede.
Importante reparar ainda, que esta camada não necessita ter o mesmo número de neurónios por camada que a camada de entrada, sendo no entanto isso o mais usual.[13]

Camada de saída

A camada de saída é o último passo de processamento a que os dados são sujeitos.
Esta camada, tal como a anterior, não necessita ter o mesmo número de neurónios que as anteriores, sendo que muitas vezes apenas se espera que gere um único valor. Dependendo das funções utilizadas este pode ser, por exemplo, entre 0 e 1.[16, 13] Sendo que neste caso as funções seriam do tipo contínuas, aplicando-se por isso o **Teorema da aproximação universal para as *RNA*** que declara:

> *"Qualquer função contínua que mapeia intervalos de números reais para um resultado de números reais, pode ser aproximada por uma rede neuronal multi-camada com apenas uma camada escondida."*[32]

3.5 Modelos actuais

A metodologia, ou algoritmo, utilizado para fazer o treino da *RNA*, está intimamente ligado à forma como a rede se encontra organizada e estruturada. Por esta razão é necessário ter um cuidado especial na forma como se cria a rede neuronal. Para o caso de estudo serão apresentados as duas classes genéricas de organização, mas apenas um algoritmo de treino, o ***BackPropagation***. As *RNA* podem, então, ser agrupadas em duas classes distintas:[16, 13]

1. **FeedForward** Os resultados gerados por uma camada de neurónios, apenas alimenta a camada seguinte. Os resultados obtidos não vão influenciar a série seguinte de resultados;

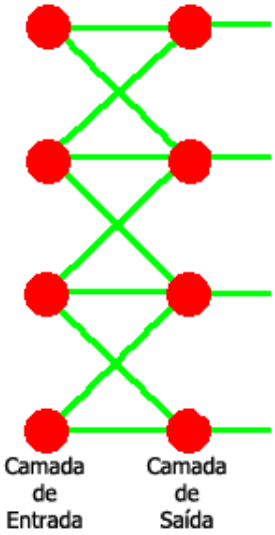

Figura 3.6: Single-Layer *RNA*

2. **Recurrent** Os resultados gerados por uma camada de neurónios, alimenta não só a camada seguinte, como também as anteriores, influenciando deste modo os resultados de todas as outras.

3.5.1 FeedForward

Unicamada *FeedForward*

É uma rede neuronal onde existem apenas duas camadas, uma de entrada e outra de saída. Como enunciado anteriormente, os dados apenas alimentam as camadas de neurónios seguintes, nunca havendo regressão. A figura 3.6 representa este tipo de estruturas.

Multi-Camada *FeedForward*

Este tipo de redes contém a já acima mencionada camadas *escondida* ou do *meio*. A função dos neurónios escondidos é fazer o processamento dos *input* para que estes sejam mais correctos. Com a adição de n camadas escondidas, a rede ganha a capacidade de obter dados estatísticos com mais precisão. De forma genérica pode-se afirmar que a rede ganha uma perspectiva mais geral. No entanto, é importante ter em conta que deve haver um equilíbrio entre o número de camadas escondidas e o tamanho do *input*. Sendo que quanto maior for este, mais camadas deverá ter. Não se deve no entanto descurar a parte do desempenho, já que este será tanto pior, quanto maior for a rede. Esta rede encontra-se representada esquematicamente pela figura 3.7.[16, 13] Os nodos de entrada alimentam com dados, os vectores

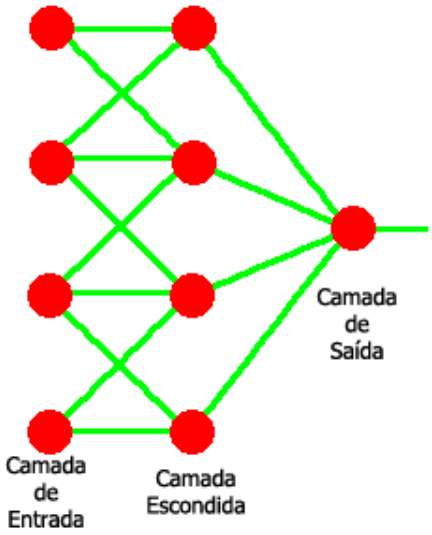

Figura 3.7: Multi-Layer *RNA*

de activação. Estes vectores constituem o sinal de entrada aplicado aos neurónios da segunda camada, também conhecida por primeira camada escondida. Consequentemente, o sinal de saída desta camada é utilizado como entrada da terceira camada, e assim sucessivamente ao longo de toda a rede neuronal. Importante ter em consideração que em cada iteração cada neurónio apenas receberá os sinais da camada imediatamente anterior.[16, 13]

O conjunto de dados saídos dos neurónios será a resposta que este dá tendo em consideração *os dados de entrada e peso*, tal como enunciado anteriormente. Sendo por isso a resposta da rede ao padrão recebido pelos nodos desta. A figura 3.7 representa então o esquema arquitectacional de uma rede *Feedforward*, sendo que neste caso apenas contém uma camada escondida. Esta rede, tipicamente, é descrita como sendo uma 10-4-2, referindo-se respectivamente aos dez nodos de entrada, quatro nodos escondidos e dois de saída. Analogamente é possível referir-se a uma rede com y nodos de entrada, $u_1^{+\infty}$ nodos escondidos e x nodos de saída como: y-$u_{11}^{+\infty}$-...-$u_{1m}^{+\infty}$-x.

Esta rede é também uma rede completamente ligada, do inglês *fully connected*, uma vez que todos os neurónios de uma camada x_y estão ligados a todos os neurónio da camada x_{y+1}. No entanto, esta não é regra, podendo por isso existir redes onde os neurónios não se encontram ligados entre si. Nesse caso a rede designa-se por parcialmente ligada, do inglês *partially connected*.[16, 13]

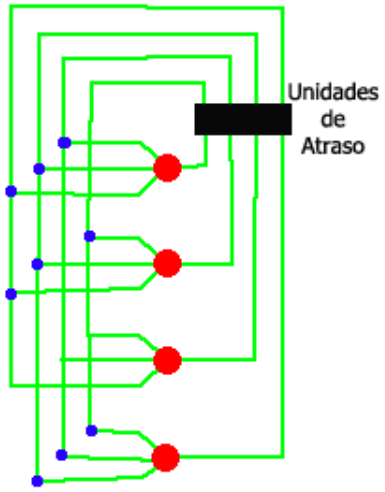

Figura 3.8: Multi-Layer *RNA*

3.5.2 Recurrent Network

Uma rede neuronal artificial recorrente distingue-se de uma rede *feedforward* no sentido em que os dados de saída de um neurónio alimentam pelo menos uma vez os neurónios dessa mesma camada (todos, alguns ou apenas um).[16, 13]

A título de exemplo apresenta-se a figura 3.8, neste caso representativa de uma rede recorrente uni-camada. Neste caso pode-se aferir que o resultado de um neurónio alimenta todos os restantes neurónios da mesma camada, não havendo no entanto alimentação do próprio. À semelhança da primeira rede neuronal apresentada na secção anterior, esta apenas contém uma única camada, sendo que este tipo de redes pode ter camadas escondidas à semelhança das anteriormente apresentadas. A figura 3.9 apresenta um exemplo de uma rede recorrente multi-camada.[16, 13] O impacto na performance de processamento ou treino causado pela utilização desta arquitectura é muito elevado, uma das razões é a existência de um novo elemento chamado *Unit-Delay Operator* representado por z^{-1}. Assumindo que a rede contém unidades não lineares, é aceitável considerar que a utilização das unidades de atraso resulta num comportamento dinâmico não linear.

3.6 Algoritmo *Backpropagation* - Feedforward

Este método foi inicialmente introduzido por *Bryson* em 1969, tendo sido ignorado devido ao elevado número de cálculos necessários. Apenas durante a década de 80 voltou a ser reutilizado, tendo-se então tornado genericamente aceite como um algoritmo viável para o

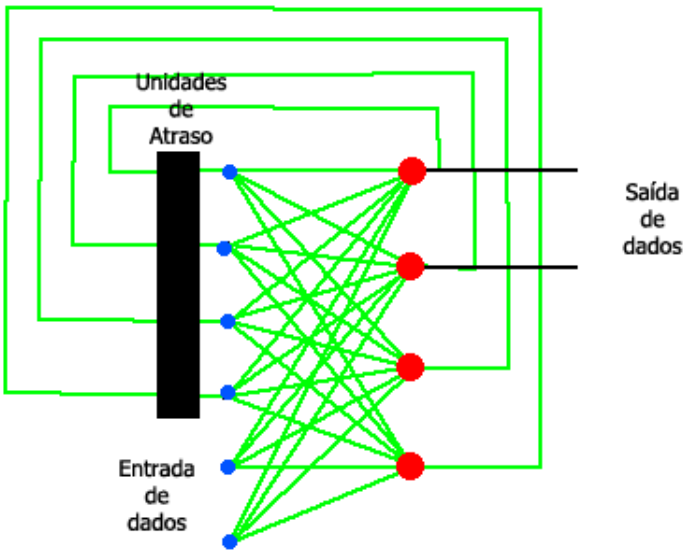

Figura 3.9: Recurrent *RNA*

treino de redes neuronais.

A ideia do processo é encontrar a função erro minímo $e(w)$ em relação aos pesos das conexões.[16, 13]

3.6.1 Fases do algoritmo

Para o caso de estudo, apenas será abordado o algoritmo *backropagation* aplicado às redes do tipo *feedforward* com camadas escondidas.

Após a entrada dos valores para processamento numa camada, essa gera um resultado, que será por sua vez passado à camada seguinte. Este processo acontece até chegar à última camada, onde é obtido o resultado final e o erro. No final deste processo, os pesos das sinapses são alterados, de forma a reduzir o valor de erro.[16, 13] Mais especificamente o algoritmo processa-se da seguinte forma:[16, 13]

- Processamento *Feedforward*: explicado na secção anterior;
- *Backpropagation* até à camada de saída. Neste caso surgem o primeiro *set* de derivadas parciais. O caminho encontra-se representado pela figura 3.10. A partir daqui pode-se verificar que o erro δ_j^2 é calculado tendo por base diversos termos, resultando na equação 3.15

$$\delta_j^2 = o_j^2(1 - o_j^2)(o_j^2 - t_j), \tag{3.15}$$

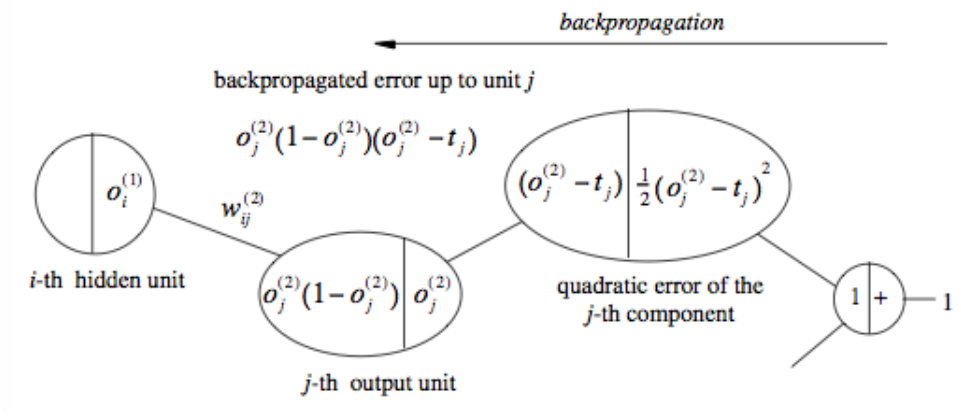

Figura 3.10: Backpropagation Path [16]

A derivada parcial que se pretende está representada na equação 3.16, sendo que se considera o peso $w_{ij}^{(2)}$ um valor variável e o seu *input* $o_i^{(1)}$ uma constante. Daqui é possível concluir que, geralmente, o valor peso será uma relação entre a entrada $o_i^{(1)}$ e a saída $\delta_j^{(2)}$.

$$\frac{\partial E}{\partial w_{ij}^2} = [o_j^2(1-o_j^2)(o_j^2-t_j)]o_i^= \delta_i^2 o_i^{(1)} \tag{3.16}$$

- *Backpropagation* até à camada escondida após o passo seguinte é necessário calcular a derivada 3.16.

  Cada unidade j existente na camada escondida está ligada a uma unidade q da camada de saída com um peso de $w_{jq}^{(2)}$, no qual $q = 1, ..., m$.

  O erro propagado até à unidade j da camada escondida deve ser calculado tendo em consideração todos os caminhos possíves, como demonstra a equaçõe 3.17

$$\delta_j^{(1)} = o_j^{(1)}(1-o_j^{(1)}) \sum_{q=1}^{m} w_{jq}^{(2)} \delta_q^{(2)} \tag{3.17}$$

  Sendo então o peso obtido através da derivação 3.18

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_j \tag{3.18}$$

  O erro pode ser calculado desta mesma maneira, 3.17, para várias camadas escondidas e não apenas uma. Sendo que a derivada parcial mantém a mesma forma analítica.

- Actualização dos pesos: Após calcular todas as derivadas parciais, os pesos da rede são actualizados. A variável γ define a longevidade da correcção. De forma analítica, as equações 3.19 e 3.20 demonstram como os pesos são obtidos.

$$\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)} \delta_j^{(2)}, para i = 1, .., k+1; j = 1, ..., m \tag{3.19}$$

$$\Delta w_{ij}^{(1)} = -\gamma o_i \delta_j^{(1)}, parai = 1, .., n+1; j = 1, ..., k \tag{3.20}$$

por convenção estipula-se que $o_{n+1} = o_{k+1}^{(1)} = 1$.

A correcção aos pesos apenas podem ser realizadas após o erro ser calculado para todas as sinapses da rede. Caso contrário as correcções podem tornar-se incoerentes e estas não corresponderão à direcção negativa do gradiente.

3.7 Considerações a ter com o uso de redes neuronais

1. para problemas de menor carga computacional, o valor de aprendizagem de 0.7 é aceitável, no entanto por norma o recomendado não deverá ser distante dos 0.2.

2. de forma a que o método *momentum* seja acelerado, o valor de α deverá ser de 0.9.[16, 13]

3. o tipo de função de activação a ser escolhido, deverá depender do tipo de problema em estudo. Para redes com camadas escondidas, as funções *sigmóide* são preferidas, uma vez que têm a vantagem de ser deriváveis em qualquer ponto do seu domínio (requisito para utilização do algoritmo *backpropagation*).[16, 13] Sendo que a maior influência a nível de performance será na simetria em relação à origem.[16, 13]

4. um pequeno valor na aprendizagem leva a uma lenta convergência do algoritmo. Enquanto um elevado valor pode levar a que não seja encontrada solução. Podendo mesmo haver o salto sobre a solução.[16, 13]

5. uma rede deve ser capaz de ser adaptável e ser generalizavel.[16, 13]

3.8 Soluções existentes

Existem diversas soluções que permitem fazer a detecção de intrusões em redes utilizando redes neuronais.

Moradi e Zulkernine propõem uma solução offline, que utiliza uma rede neuronal artificial multi-camada, onde o objectivo é não só fazer a detecção de um ataque mas também perceber qual é o tipo do ataque, permitindo desta forma o aconselhamento de acções a ser tomadas.

Para isso desenvolveram uma solução em *MatLab* com recurso a rede neuronal de duas camadas escondidas com trinta e cinco neurónios de entrada. O resultado é formado por três valores, sendo a sua combinação binária relativa a cada tipo de ataque existente.

A rede neuronal artificial utilizada é *feedforward* sendo o algoritmo de treino utilizado o *bakpropagation*.[25]

Seliya e Khoshgoftaar apresentam uma outra solução que utiliza *RNA* no sentido de permitir não só a detecção de instruções mas também uma aprendizagem activa do sistema.

Após diversos testes realizados pelos mesmos, estabeleceram que uma *RNA* com duas camadas escondidas, trinta neurónios de entrada e um de saída que fornece os resultados óptimos. Nas duas camadas escondidas é utilizada a função activação *tan-sigmóide*, enquanto na camada de saída a função é *log-sigmóide*. A implementação foi desenvolvida em *Matlab*, tendo o algoritmo de treino utilizados sido baseado na *framework Bayesian*.

Os resultados obtidos demonstraram a possibilidade de uma aprendizagem activa por parte da rede neuronal, permitindo assim uma melhor precisão dos resultados.[35]

Lee, Roedel e Silenok desenvolveram um estudo sobre a detecção e caracterização de ataques por *port-scan*. *Snort* e *UCSD ActiveWeb* foram duas aplicações utilizadas, sendo que a primeira tinha o objectivo de fazer uma filtragem dos dados, enquanto a segunda faz a análise dos dados filtrados. Desta forma reduz-se o volume de dados para aqueles que apenas interessam.

Com esta abordagem, é possível obter dados estatísticos relativos a dois tipos de *scans Vertical e Horizontal*. São ainda obtidos resultados relativos a portas que sofreram *scan*, padrões, ou distribuição geográfica.[21]

Em 2009, Sheikhan e Sha'bani, desenvolveram um novo sistema baseado em redes neuronais artificiais, optimizadas para rápido processamento, tendo em conta algoritmos de optimização de pesos das camadas escondidas. Este trabalho aborda a detecção de ataques abusivos da rede, através da implementação de dois mecanismos. Sendo assim, na primeira fase é utilizado um algoritmo de optimização de pesos da rede neuronal. No segundo ponto permitiu a redução do tamanho da entrada da *RNA*, o que aliado ao facto de melhorar o desempenho de treino, permite a obtenção de um *IDS* efectivo quer na taxa de detecção, quer na quantidade de falsos positivos.[24]

Uma metodologia diferente, baseia-se na detecção de *scans* tendo por base a contagem de ligações *TCP* bem-sucedidas e interrompidas. Esta metodologia, abordada por Soniya e Wisey, utiliza uma rede neuronal com uma camada escondida. A camada de entrada contém

dez neurónios, a única camada escondida cinco e por fim existe um neurónio de saída. Como algoritmo de treino é utilizado *backpropagation*.

Este sistema foi desenvolvido para interagir apenas com um computador, em modo *offline*, não possibilitando por isso detecção em tempo real. Este método aliado a características dos métodos apresentados anteriormente, esteve como base para o desenvolvimento do *IDSNN1*, ou seja, da solução apresentada nesta dissertação.[37]

CAPÍTULO 4

Implementação

4.1 Ferramentas utilizadas

4.1.1 Snort

Snort é um *IDS*. Na verdade é considerado o melhor *IDS OpenSource* existente actualmente. Tendo mesmo sido premiado em 2003 é um exemplo de como é possível a comunidade unir-se e trabalhar activamente em busca de um bem comum.
O criador do *snort*, *Martu Roesch* visualizou o *snort* como sendo um detector leve e prático. Sendo possível correr o *snort* sem especificar nenhum conjunto de regras, podendo desta forma visualizar todo o tráfego que passa na rede (se o *snort* estiver numa máquina central), ou no segmento, se estiver num *host* particular.
Com o contributo de toda a comunidade o *snort* tornou-se uma aplicação muito completa, tendo capacidade para:

- análise em tempo real de tráfego IP;
- registo de pacotes;
- regras de detecção;
- linguagem intuitiva e fléxivel;
- emissão de alertas;
- verificação de padrões;
- detecção de scans;

- detecção de scans furtivos;
- detecção de ataques *CGI*;

O *snort* é então uma aplicação baseada no *sniffer libpcap* que pode ser utilizado como um detector de intrusões de rede por ser leve e prático. O motor de detecção é programado utilizando uma linguagem proprietária no qual se especificam regras. Estas regras são apenas a descrição por pacote dos testes e acções que o *snort* deverá realizar. Dada a sua arquitectura, facilmente se criam novas regras para detectar novos ataques ou novos *exploits* que possam surgir com o tempo.[28][1]
A arquitectura do *snort* é focada maioritariamente em três aspectos chave: *Performance, Simplicidade e Flexibilidade*. E pode ser dividido em três subsistemas: *Descodificador de pacotes, motor de detecção e regists/alerta*, sendo que estes três subsistemas correr sobre a biblioteca *lipcap*.

Descodificador

O Descodificador está organizado sobre as camadas dos protocolos. Cada rotina deste impõe uma ordem sobre o tráfego. Neste subsistema a velocidade de processamento é uma das prioridades, pelo que a maioria das funções por ele desempenhadas baseiam-se apenas na criação de apontadores de pacotes a ser analisados pelo motor de detecção.[28][1]

Motor de Detecção

As regras do *snort* encontram-se guardadas numa matriz *nxm*, sendo a primeira linha chamada de *Chain Header* e as restantes de *Chain Option*, como apresentado na figura 4.1. A pesquisa das regras é feita recursivamente para cada um dos pacotes, de forma a verificar se há um encontro, em primeira instância com a *Chain Header*, e caso esta condição se verifique, numa segunda instância com as condições a si ligadas *Chain Option*. Se existe uma regra específica para um determinado pacote, o *snort* desencadeia todas as acções descritas nesse pacote, sejam elas de alerta, ou de ignorar o pacote. [28][1] foi

Registo e alerta

O processo de registo pode ocorrer de diversas formas, uma delas é o registo optimizado para leitura humana, facilmente compreensível pelo ser humano, ou o formato binário tal como extraído pelo *tcpdump*. Sendo que este último é um recurso de muita mais rápida utilização pela máquina e deve ser o escolhido, no caso de um ambiente onde a eficiência seja

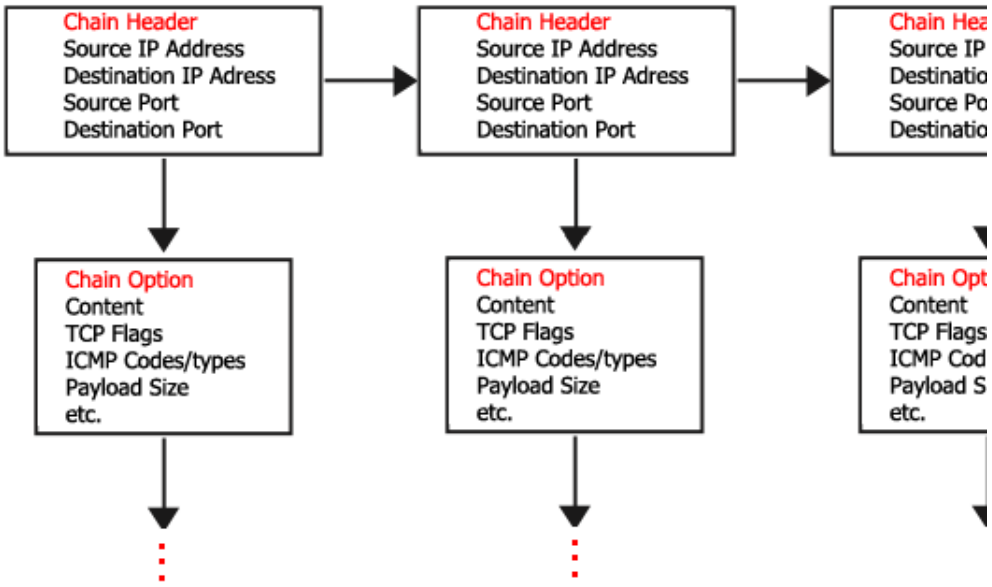

Figura 4.1: Regras *snort*

exigida Em casos extremos é possível desactivar o registo ficando apenas o sistema de alertas activo. Desta forma a velocidade de processamento é esmagadoramente maior.

Por sua vez os alertas podem ser enviados para o sistema de registos do sistema, enviados para um ficheiro de alertas ou então a criação de uma mensagem *pop-up*. Estes eventos são enviados como mensagens seguras e autorizadas que podem ser facilmente monitorizadas por diversas ferramentas.

Ainda dentro da categoria dos alertas, é possível fazer um alerta detalhado onde se pode encontrar informações como o cabeçalho do pacote, e mensagem de alerta, ou então um alerta simples onde apenas aparece uma mensagem de alerta. Este último módulo é também mais rápido, mas caso o administrador pretenda verificar a veracidade do alerta terá que fazer pesquisa do pacote no sistema de registos.[28][1]

4.1.2 Utilização do *snort*

Para o caso de estudo, a versão do *snort* utilizada foi a *2.9.1*. Para este projecto as regras utilizadas pelo *snort* foram as originais cujo *download* foi efectuado a partir dos repositórios

do *snort*, não havendo por isso qualquer regra específica gerada durante o projecto.
O registo de todos os alertas foi efectuado numa base de dados sql, a correr na mesma máquina onde se encontra instalado o *snort*.[28][1]

4.1.3 Nmap

O *nmap* é uma aplicação que permite ao seu utilizador fazer *scan* de outras máquinas localizadas na rede. Tem a sua origem em sistemas linux, onde se encontra muito enraizada, mas com o passar dos anos tornou-se compatível quer com sistemas Windows quer com o sistema Mac OS X. Sendo que o objectivo final dos programadores é tornar esta ferramenta totalmente compatível com todos os sistemas operativos.
Esta aplicação é normalmente utilizada por administradores de sistemas para fazer *scan* a grandes redes, com o intuito de procurar *hosts*, serviços e sistemas operativos.
É uma aplicação muito completa com uma característica muito especial, é capaz de criar pacotes IP do zero e enviá-los utilizando metodologias únicas que lhe permitem obter resultados fiáveis.[30][3]
O *nmap* traz consigo uma duas de funcionalidades que cobrem tudo o que um *scanner* deve fazer:

- capacidade de fazer *scans* básicos;
- capacidade de fazer *scans* avançados;

Entre várias opções, o *nmap* , permite ao administrador verificar se a rede está em conformidade com o planeado. A mesma metodologia utilizada pelo administrador de sistema para verificar isto, é também utilizada pelo *hacker* para verificar vulnerabilidades, serviços, *hosts* numa rede. Esta é a beleza do *nmap*, sendo também a sua grande falha e risco.
O objectivo disto é estimar o nível de segurança dos componentes de uma rede, de forma a ter noção do panorama geral.
Com o *Nmap* é possível fazer uma série de testes que se enquadram nas duas funcionalidades acima enunciadas. É possível por exemplo:

- testar portas abertas nas interfaces das *firewall*;
- efectuar scans numa determinada gama/intervalo de IPs, ou até mesmo IPs especificos de forma a verificar a existência de serviços de rede a correr sem autorizados;

- verificar se as versões de *webservice* estão actualizadas na zona em questão;
- verificar que sistemas estão a partilhar ficheiros através de portas especificas para o efeito;
- verificar a existência de serviços de *FTP*, impressoras ou sistemas não autorizados;

Outra vantagem da utilização do *nmap* é a sua versatilidade, dando ao utilizador uma ferramenta poderosa. Esta versatilidade verifica-se não só pela enorma panóplia de configurações possíveis, mas também pela forma como este pode apresentar os resultados ao utilizador. Na actualidade existem *GUI* que permitem ter uma visualização mais *user-friendly*, no entanto, por definição o *nmap* permite guardar os resultados em quatro mais n formatos distintos. Quatro deles predefinidos pela aplicação (apresentados de seguida), sendo os seus ficheiros facilmente manipulados recorrendo a um simples editor de texto, e n definidos pelo utilizador.

- *interactivos* resultados apresentados no momento, enquanto o utilizador efectua o scan;
- *XML* com este formato é possível fazer a manipulação dos dados através de diversas aplicações, podendo haver a criação de um relatório em *html* por exemplo;
- *grepable* um formato orientado a ferramentas de linhas de comando, que processam linha-a-linha como por exemplo a ferramenta *grep*;
- *normal* formato tal como apresentado na linha de comandos, sendo que neste caso esta informação é automáticamente salva para um ficheiro;
- *script kiddie* formato adicionado *à posteriori* cujo objectivo é substituir as letras pelo digito idêntico. A utilidade deste último é duvidosa, existindo apenas para descontrair o utilizador enquanto trabalha; [30][3]

Utilização do *nmap*

O *nmap* foi utilizado com o intuito de fazer testes em tempo real para detecção de scans a ocorrer na rede. Na generalidade estes testes focaram-se em dois aspectos distintos:

- *scan genérico* não havendo preocupação em delimitar as portas a escutar, nem a gama de endereços a verificar. Este é o modo predefinido de acção do *nmap* e que mais facilmente é detectado pelos *IDS*;
- *scan furtivo* há preocupação na forma como o scan é efectuado. Utilizam-se técnicas que permitem verificar apenas alguns serviços, verificar se o *host* está activo, entre outros.

Com estes scans pretendeu-se gerar tráfego malicioso que o *IDSNN1* pudesse capturar de forma a verificar se é possível ou não detectar scans através da contagem de ligações partidas, e se sim verificar se era possível fazer a confirmação de alertas gerados pelo *snort*, de forma a verificar a existência ou não de scans/ataques.

4.1.4 Encog Framework

FeedForward Neural Network with Backpropagation Algorithm

Encog é uma *Framework* de inteligência artificial em *Java* e *.Net*. Apesar desta suportar diversas áreas da inteligência artificial o seu foco principal são as redes neuronais.
A primeira versão foi lançada em Julho de 2008, tendo evoluído desde aí de forma consistente e prática. Encontra-se licenciada por *LGPL* e todo o seu código encontra-se alojado nos repositórios disponibilizados pelo *Google Code project*.
Esta ferramenta integra funcionalidades que permitem a criação de diversos tipos de redes neuronais. Sendo que para o caso de estudo, e como referido anteriormente, apenas foram utilizadas duas redes do tipo *feedforward*[17]

4.1.5 IPTables

O *IPTables* é uma aplicação com base em *linux* que permite ao administrador de sistema configurar as tabelas integradas no *kernel da firewall do linux*, utilizadas pelos diferentes protocolos *IPv4, IPv6, ARP, ebtables*. O *IPTables* é uma aplicação que necessita de previlégios de super-utilizador e encontra-se integrada no IDSNN1, permitindo que a máquina bloqueie ligações de cariz suspeito.
Esta aplicação permite que o administrador defina as tabelas nas quais se encontram as regras que permitem o tratamento de pacotes. Estas regras permitem que um pacote seja reencaminhado para um determinado serviço, seja ignorado, ou então siga o seu percurso normal. Cada regra da corrente contém a especificação que coincide com o pacote.
Funcionalmente, cada pacote que chega à máquina é processado pela *firewall*, que faz *match* entre as regras e o pacote, percorrendo para cada um dos pacotes todas as regras internas.
O *IPTables* permite, por si só, fazer a detecção em tempo real de scans, no entanto torna-se computacionalmente pesado, levando a lentidão no processamento dos pacotes e consequente perda de performance.

!h

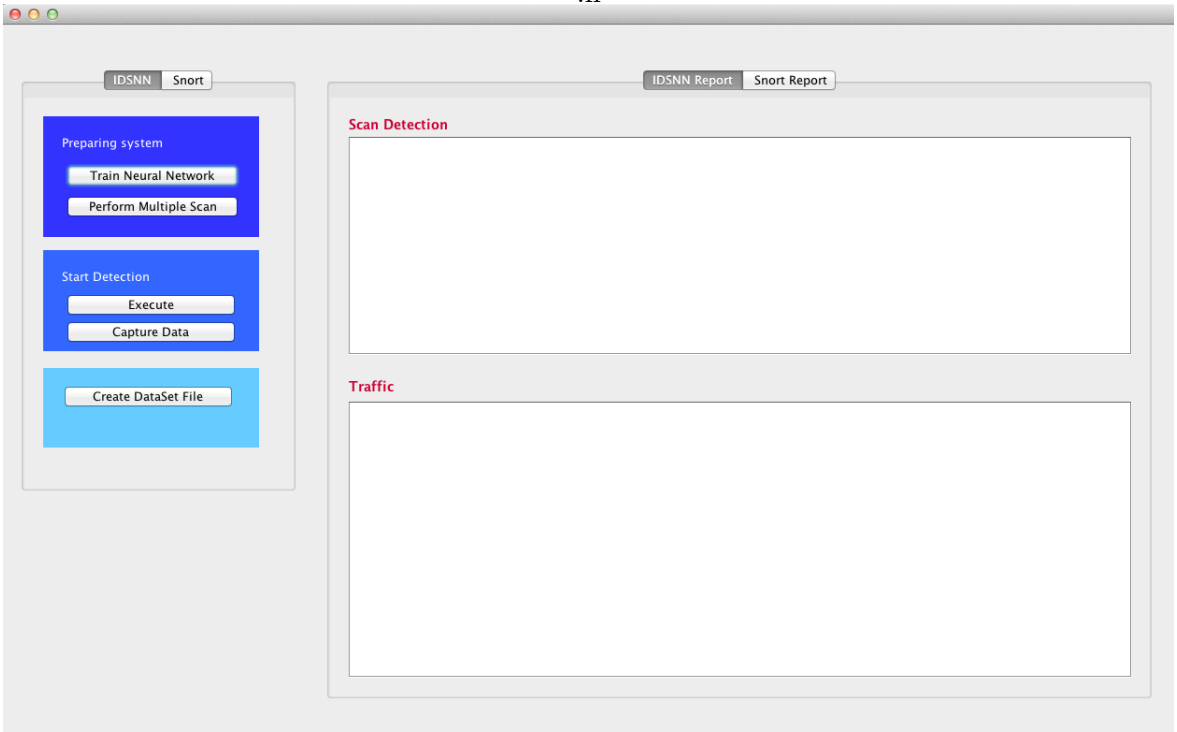

Figura 4.2: IDSNN1 Interface

4.2 Metodologia

O protótipo aqui implementado tem o intuito de ser instalado num servidor central, tal como qualquer *IDS*. Este encontra-se desenvolvido em java e permite a captura, interpretação e processamento dos dados. Uma vez que é desenvolvido em java, com recurso a um motor base de dados, em mysql, é possível fazer a sua portabilidade para diversos sistemas operativos. Tendo sido testado em sistemas Windows 7, Windows 8 Developers Edition, Linux Ubuntu 11.04, Linux BackTrack 4 r1, Mac OS X Snow Leopard e Mac OS X Lion. Para captura de pacotes o *IDSNN1* suporta tanto o *TCPDump* como o *WinDump*.
O *IDSNN1* encontra-se desenvolvido em três camadas, *Dados, Negócio, Apresentação*.

- a camada de dados é constituída pelas classes que fazem a leitura e escrita, tanto em ficheiro como na base de dados.
- a camada de negócio é a camada que trata todo o processamento dos dados, e que prepara os resultados para apresentação na camada de Apresentação;
- a camada de apresentação contém todos os controlos que o utilizador pode utilizar no sentido de realizar tarefas. A representação desta camada pode ser vista através da figura 4.2.

O *IDSNN1* encontra-se dividido em dois módulos de que se destinam a acções diferentes, apelidados *IDSNN1_ScanDetection* e *IDSNN1_SnortDetection*, sendo que podem actuar de forma independente ou em cooperação.

4.2.1 Trabalho Base

Trabalho de investigação já realizado, comprova que é possível fazer a detecção de scans a um *host* especifco fazendo a contabilização de pacotes [37]. Para isso, e segundo os autores, é utilizada uma rede neuronal com dez nodos de entrada, sendo constituidos por sete dados obtidos directamente e três resultados de operações:

- C1:Pacotes *SYN* Entrada;
- C2:Pacotes *SYN- ACK* Saída;
- C3:Pacotes *RST* Saída;
- C4:Pacotes *SYN* Saída;
- C5:Pacotes *SYN-ACK* Entrada;
- C6:Pacotes *FIN* Saída;
- C7:Pacotes *FIN* Entrada;
- P1 = *SYN* Entrada - *SYN-ACK* Saída;
- P2 = *RST* Saída / (Total de Pacotes - (Valores de entrada e Saída de pacotes *ACK*));
- P3 = Máx(*FIN* Entrada,*FIN* Saída);

Os pacotes neste caso são capturados utilizando a aplicação *WinDump*.
Em teoria, no caso da ocorrência de um *scan*, o valor da função P1 será muito elevado.

O que os autores perceberam foi que para uma comunicação normal deveria existir uma parelha entre os dados, tal como demonstra a tabela 4.1. Neste caso os valores de *SYN* e *SYN-ACK* são muito próximos um do outro, podendo ou não ser o mesmo. O mesmo deverá acontecer com os valores de *FIN*, isto é, tanto o de entrada como o de saída devem ter valores muito idênticos[40]. Comprovando que existiram sempre conexões totais, ou que as perdidas são diminutas.
Por sua vez, na eventualidade de um *scan*, estes valores não ocorrerão desta maneira. Neste

Tabela 4.1: Amostra sem ataques

Serial Num.	C1	C1	C3	C4	C5	C6	C7
1	25	25	1	0	0	23	23
2	53	50	7	0	0	46	46
3	33	33	3	0	0	29	29
4	48	45	8	0	0	40	41
5	0	0	6	71	71	65	67
6	4	4	3	15	15	16	18
7	0	0	2	7	7	5	5
8	0	0	11	145	145	132	134

caso, e ao contrário dos anteriores, iremos verificar o aumento do número de pacotes *RST* de saída e a diminuição do número de pacotes de saída *SYN-ACK*. A tabela 4.2 apresenta esta situação. [37]

Tabela 4.2: Amostra de Scans

Serial Num.	C1	C1	C3	C4	C5	C6	C7
1	9	0	9	9	9	8	8
2	8	2	9	11	11	11	12
3	11	1	17	16	16	9	12
4	25	5	20	500	500	505	508

4.2.2 Trabalho desenvolvido

O trabalho desenvolvido para este módulo partiu do estudo do comportamento do protocolo TCP em condições normais, e do estudo do comportamento deste quando se efectuam *scans*, seguindo as linhas orientadoras do trabalho apresentado na subsecção anterior, havendo no entanto alterações quer na arquitectura, quer na própria implementação.

O protótipo aqui apresentado, desenvolvido em *Java/MySql*, permite fazer dois testes aos dados recolhidos.

De forma geral, esta aplicação captura pacotes, e faz *parse* deles, verificando o que cada um representa. Esta interpretação leva a que novos valores sejam adicionados à base de dados,

sendo então os pacotes apagados. Ao mesmo tempo que a captura e inserção de dados é feita, os módulos de análise obtêm os valores, utilizando-os para processamento pelas *RNA*.[8, 33] Estas tarefas são executadas concorrentemente.

Para referência, ao longo desta dissertação às *flags* é acrescentada letra "a"ou "b", representando respectivamente a vítima e o atacante.

Os dados utilizados para fazer os testes são dependentes da forma como um *scanner* interage com a rede. Dados como o número de portas é importante uma vez que durante um *scan* normal, a diferença entre o número de portas escutadas por um elemento tenderá a ser mais elevada do que o número de portas origem. Por sua vez, o número de pacotes *SYN/SYNACK/RST* também devem ser mais elevados, uma vez que são testadas diversas portas. Isto levará à geração de pacotes *RST* como resposta ou ao pacote *SYN* ou ao pacote *SYNACK*. Sendo assim, e caso haja um elevado número de pacotes *RST*, deverá haver um baixo número de pacotes *FIN*. Por fim, e tendo em conta o modo de funcionamento do *nmap*, é feita também a contabilização do valor da *window* associada, isto porque, por diversas vezes, em *scans* furtivos, o valor desta variavél é 0. [30, 21, 5] De seguida encontram-se declarados os dados contabilizados em cada comunicação:[7, 30]

- Número de portas destino (PD)
- *SYN*;
- *SYN ACK*;
- *RST*;
- *FIN*;
- *Window*;
- *ICMP*.

Seguindo as linhas orientadoras do trabalho base[37], são também criadas operações que permitem dar mais informação sobre o conjunto de interacções *ip origem* e *ipdestino*. Estas operações encontram-se discriminadas de seguida, assim como a sua explicação e os resultados esperados representados na tabela 4.3.

1. *ABS(PDa -PDb)*;
2. *SYNa- (RSTa+RSTb)*;

3. *RSTa/SYNa*;
4. *FINa/SYNa*;
5. *Max(ICMP)*;
6. *Windowa*;

Tabela 4.3: Resultados esperados

Serial Num.	Scan	Normal
0	»0	≈0
1	≈ 0	»0
2	>0.5	<0.5
3	x<0.5 || x>1.0	0.5< x <1.0
4	»0	> 0
5	»0	> 0

A tabela 4.3 representa quais são os resultados esperados da execução das fórmulas quer em caso de *scan* quer não, sendo que:

1. tendo em conta o comportamento do *nmap*, o número de portas que o computador atacante irá pesquisar, será muito maior do que o número de portas para o qual o computador vitíma deverá responder;[30]
2. tendo em conta o comportamento do *nmap* este valor será elevado, uma vez que o *nmap* responderá com *RST* à máquina destino;[30]
3. este valor deverá ser muito elevado em caso de *scan*;
4. o valor será diminuto uma vez que muito poucas conexões são terminadas com sucesso;
5. o atacante poderá utilizar *pings* para verificar actividade de um *host*;
6. o valor poderá ser muito elevado, uma vez o *nmap*, por vezes, utiliza como técnica de *scan* furtivo, a colocação deste valor a zero.

Modelo de dados

A figura 4.3 representa o modelo de base de dados utilizados pelo IDSNN1. Esta é constituída por quatro tabelas, sendo *Source* a tabela central, que permite criar as relações entre todas as outras. Esta é constitúida por três entradas:

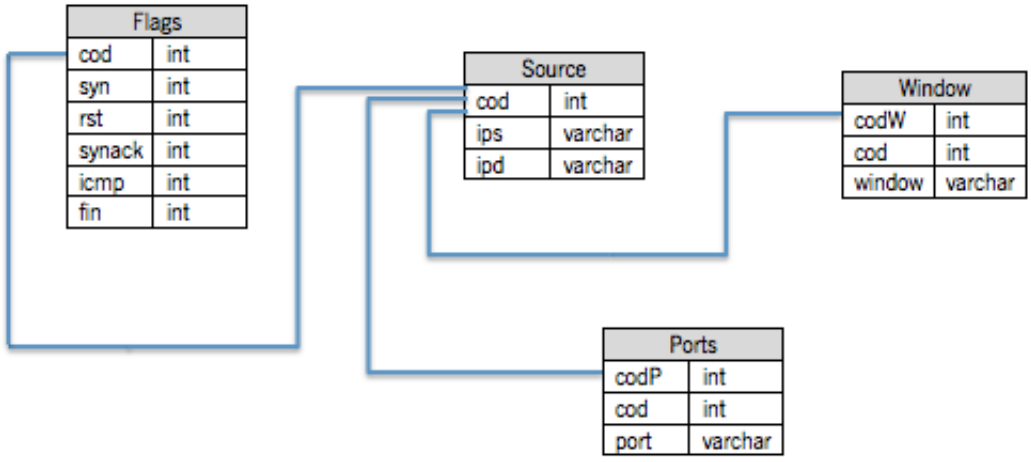

Figura 4.3: Modelo Base Dados

- cod: chave primária da tabela. Este registo por ser único permite fazer a ligação com todas as outras tabelas do sistema;
- ips: endereço IP origem da trama capturada;
- ipd: endereço destino da trama capturada;

Nestas tabelas apenas é guardado um par (ips,ipd), isto é, não há repetição de pares ao longo da tabela, esta característica permite não só a redução de entradas, mas também uma maior rapidez de acesso aos dados.
A tabela *Ports* é também constituída por três entradas:

- codP: chave primária da tabela.
- cod: chave estrangeira, permite criar a relação com a tabela *source*;
- port: porta destino da trama capturada;

Nesta tabela, apenas é guardado um par (cod,port), não havendo, tal como no caso anterior, repetição dos pares ao longo da tabela. Desta forma para se obter o número de portas destino escutadas, apenas é necessário fazer um $Selectcount(port)fromscanDetection.ports\ where\ cod\ like\ XXX$ onde "XXX"é o código pretendido.
A tabela window tem o mesmo comportamento da tabela *ports*, sendo que neste caso a sua utilização é distinta. Ela é constituída por:

- codW: chave primária da tabela.

- cod: chave estrangeira, permite criar a relação com a tabela *source*;
- window: janela destino da trama capturada;

Por fim a tabela *Flags* é constituída por:

- *cod*: chave primária da tabela e também chave estrangeira.
- *SYN*:contador desta *flag* na globalidade das tramas capturadas;
- *RST*:contador desta *flag* na globalidade das tramas capturadas;
- *SYN ACK*:contador desta *flag* na globalidade das tramas capturadas;
- *ICMP*:contador desta *flag* na globalidade das tramas capturadas;
- *FIN*:contador desta *flag* na globalidade das tramas capturadas;

Na tabela *Flags* são guardados os valores que contabilizam os dados as *flags* activas nas tramas capturadas. Existe apenas um registo para cada *cod*, sendo que ao longo do tempo, os valores aqui registados vão sendo incrementados.

Ficheiro Configuração

Para que seja possível utilizar o *IDSNN1*, é necessário ter um ficheiro de configuração na mesma directoria do executável. Este ficheiro de configuração dá ao utilizador a possibilidade de alterar as definições relativas a *Data-Set, Entrada de Dados, Nome de Utilizado/Palavra Chave MySql, Endereço/Porta MySql*, contendo a seguinte estrutura:

1. directoria onde se encontra o *trainset*;
2. comando para entrada de dados;
3. *username* da base de dados;
4. palavra-chave da base de dados, caso não exista coloca-se *"nnn"*;
5. directoria de comandos para efectuar *scans*;
6. endereço IP e porta da máquina onde se encontra o motor base de dados;
7. endereços a ser ignorados (um por linha).

Os dois pontos mais importantes a referir sobre o ficheiro de configuração são: a possibilidade de passar um ficheiro como entrada de dados, em vez de utilizar a aplicação *TCPDump*. Esta funcionalidade existe porque a aplicação executa um comando de *Shell* e depois captura o retorno desse comando. Por esta razão, se se tiver um ficheiro com a mesma estrutura do *outuput* de um ficheiro *TCPDump* e não se tiver esta aplicação instalada, pode colocar $catdirectoriaficheiro$ no ficheiro de configuração e este será o *input* da aplicação. O outro ponto importante é o facto da aplicação suportar uma base de dados que não se encontre a correr na mesma máquina do *IDSNN1*. Esta funcionalidade é importante pois diminui a carga computacional associada ao processamento dos dados.

```
    /Users/francisco/Documents/Universidade/Mestrado/2Ano/TeseMestrado/Dataset/trainSet.ts
tcpdump -i en1 tcp[13] \& 4!=0 || tcp [13] \& 1!=0 || tcp[13] \& 2!=0 || tcp[13] =18 || icmp[icmptype]==icmp-echo
root
nnn
/Users/francisco/Documents/Universidade/Mestrado/2Ano/TeseMestrado/Dataset/nmapList4.nm
localhost:3306
```

Configuração 4.4: Ficheiro Configuração *IDSNN_ScanDetectio*

Captura de dados

O *IDSNN* faz também o trabalho de *sniffer*, capturando todos os pacotes que recebe. De forma a optimizar este processo, convencionou-se que apenas seria necessário capturar os pacotes que contêm as *flags* pretendidas activas, por essa razão, quando se utiliza o *TCPDump*, escolhem-se as *flags* através das opções de configuração, tal como demonstra o código 4.5 Desta forma optimizam-se recursos, evitando maior processamento de dados.
O incremento das variáveis na base de dados é feito utilizando dois métodos, *routineICMP* e *routineFlags*. Cujos algoritmos se apresentam de seguida:

Sendo assim, o método *routineICMP* recebe uma *String*, contendo os dados passados à aplicação (capturados ou lidos por ficheiro). Esta rotina apenas recebe dados relativos a tramas *ICMP*.
O código 4.7 apresenta o algoritmo de como são extraídos as *flags* das tramas e como são incrementados os respectivos valores na base de dados. Neste ponto é também adicionado o

```
tcp[13] & 4!=0 || tcp [13] & 1!=0 || tcp[13] & 2!=0 || tcp[13] =18 || icmp[icmptype]==icmp-echo
```

Configuração 4.5: Opções TCPDump

```
private void routineICMP(String)
    {
        Partir string "aux" em blocos usando o char " "
        Inicializar "cod" e "codF" a "-1"
        Verificar se existe a parelha "bloco 2(ips)" "bloco 4(ipd)" na
                    base de dados e retornar "cod"

        Se "cod" existe
        {
            Buscar "codF"  à tabela flags
            Se "codF" existe
                Fazer update da row na tabela
            Se "codF" não existe
                Adicionar nova row à tabela
        }
        Se "cod" não existe{
            Adicionar par ("bloco 2(ips)","bloco 4(ipd)") à tabela
                    source e pedir "cod"
            Adicionar nova row à tabela flags
        }
    }
```

Algoritmo 4.6: Rotina Captura ICMP

valor da *window* à tabela respectiva.

Obtenção dos dados

Ao longo do tempo a base de dados é alimentada com novos dados. Tal como referido anteriormente, estes dados são então processados pela *RNA*[25, 24, 35, 29] sendo o processo descrito no subcapítulo seguinte, no entanto há uma fase intermédia que é a fase de *obtenção e preparação* dos dados. Esta é uma fase importante dadas as restrições e cuidados a ter.
Estando o sistema preparado para trabalhar com redes de internet de grandes dimensões, os dados o volume de dados será extremamente elevado, pelo que é necessário que haja aleatoriedade, para que um *hacker* não possa prever como poderá ser detectado. Por esta razão, a obtenção dos dados é feita de forma aleatória e não sequencial. Deve-se ter em consideração que a performance não deve ser afectada pela forma como são obtidos os dados. Posto isto, a obtenção dos dados encontra-se em execução dentro de um ciclo "infinito", seguindo os seguintes passos:

1. criar lista de códigos já consultados;
2. obter *cod* máximo e mínimo da tabela *source*, desta forma sabe-se qual é o limite inferior e superior para a aleatoriedade;
3. calcular *cod* aleatório tendo em conta os limites previamente estabelecidos;
4. verificar se *cod* já foi consultado, e se sim, calcular novo *cod*;

```
private void routineFlags(String)
    {
        Obter valor de "Window"
        Inicialização de "cod"s a -1 e "flags" a 0
        Se fo trama "SYN-ACK"
            incrementa "flag _fsp"
        } Caso Contrário
            Se for trama "SYN"
            {
                incrementa "flag _fs"
            }Caso Contrário
               Se for trama "RST"
                {
                    incrementa "flag _fr"
                }Caso Contrário
                   Se for uma trama "FIN"
                    {
                        incrementa "flag ff"
                    }
        Se houver incremento de alguma "flag"
        {
           Obtenção de "_cod" utilizando par (IPs,IPo);
           Se "_cod" existe
            {
                Obter "_codP" a partir do par(_cod, _port);
                Se "_codP" não existe
                    Adição de ("cod",porta) à bd e obtenção do respectivo "_codP"
                Se "_codF" existe:
                    Incrementar "flag"  à bd;
                Caso contrário
                    Adicção da "flag"  à bd;
            }
            Caso Contrário{
                Adição de (IPs, IPo) e obtenção de respectivo "cod";
       Adição de ("cod",porta) à bd;
                Adição da "flag" à bd;
            }
            Adição do valor de "Window" à bd;
        }
    }
```

Algoritmo 4.7: Rotina Captura Flag

5. obter dados relativos ao *cod* calculado;

6. processar cálculos e enviar para a *RNA*;

7. adicionar *cod* à lista de códigos consultados

8. reiniciar no ponto 2;

Sendo que, quando a lista de códigos consultados se encontra igual aos registos da base de dados, esta é limpa e o processo contínua, verificando todos os códigos novamente;

Rede Neuronal

O *ISDNN1_ScanDetection* teve a sua génese em fevereiro de 2011, tendo vindo a evoluir desde então. Existem duas grandes diferenças desde a sua versão inicial, até à versão actual, no que a *RNA* diz respeito.
Tendo em conta o estudo, no qual se comprova que a diferença de resultado entre uma *RNA* de três camadas e de quatro[39], optou-se pela utilização de três camadas escondidas.

1. a primeira versão corre sobre o *encog 2.3*, enquanto a segunda corre sobre a versão 3 da *framework*.

2. na primeira versão, os métodos que permitem a construção, invocação e treino de uma rede pertencem a uma aplicação já existente. Não utilizando por isso os construtores da *framework*. Por sua vez, a segunda versão é totalmente construida com base na *framework encog 3.0*, não fazendo uso de mais nenhuma biblioteca.

Pela análise do código 4.8, é possível aferir que existem três métodos relevantes para as redes neuronais. Na verdade graças à biblioteca utilizada a implementação de redes neuronais está muito facilitada e directa.
Podemos então concluir que a rede é constituida por cinco camadas. Uma de entrada, uma de saída e as restantes são camadas escondidas. A razão pela qual se escolheram três camadas escondidas, tem a ver com o tamanho do *dataset* de treino. Três será o tamanho ideal tendo em consideração o número de variáveis, o tipo de variáveis e o objectivo a que se propõem esta solução.[17]
Em termos práticos existem dois momentos distintos para utilização desta classe, sendo eles ***Treino da RNA e Detecção usando a RNA***.
O treino da rede é efectuado, obrigatoriamente antes de se poder utilizar a *RNA* e é feito utilizando um *dataset*, previamente guardado em ficheiro num formato específico. Uma vez que se está a utilizar o algoritmo *Backpropagation* é importante ter em atenção dois factores, o *Learning Rate e Momentum*, pois são tão importantes quanto o *Data-Set*, estando as relações entre os dois valores representados na tabela 4.9.[16]
Pela análise da mesma, verifica-se que com certeza, apenas é possível atingir um ponto de equilíbrio se ambos os valores *Momentum, Learning Rate*[16, 13] estiverem em equilíbrio A

```
public RedeConta(JTextArea _reportTextBox) {
                network = new BasicNetwork();
                network.addLayer(new BasicLayer(null, true, 13));
                network.addLayer(new BasicLayer(new ActivationTahn(), true, 13));
                network.addLayer(new BasicLayer(new ActivationTahn(), true, 13));
                network.addLayer(new BasicLayer(new ActivationTahn(), true, 13));
                network.addLayer(new BasicLayer(new ActivationTahn(), false, 1));
                network.getStructure().finalizeStructure();
                network.reset();
                new ConsistentRandomizer(-1,1,500).randomize(network);
                System.out.println(network.dumpWeights());
                this._reportTextBox =_reportTextBox;
}

public void trainNN(double[][] inputTest, double[][] idealTest){
                // train the neural network
                trainingSet = new BasicMLDataSet(inputTest,idealTest);
                train = new Backpropagation(network, trainingSet, 0.1, 0.7);
                int epoch=0;
                do
                 {
                    train.iteration();
                    if(epoch%1==0)
                       System.out.println("Epoch #" + epoch + " Error:" + train.getError());
                    epoch++;
                }while((train.getError() > 0.01));
}


public int testNN(double[] inputComp, String ips, String ipd){
                int attack =0;
                MLData mldt = new BasicMLData(inputComp);
                final MLData output = network.compute(mldt);
                if(output.getData(0)>0.5){
                _reportTextBox.append("IP Source: "+ips +" - IP Destination: "+ipd+"\n");
                _reportTextBox.append(inputComp[0] + "," + inputComp[1] + "," + inputComp[2] + ","
                + inputComp[3] + "," + inputComp[4] + "," + inputComp[5] + "," + inputComp[6] + ","
                + inputComp[7]
                + ", actual=" + output.getData(0)+"\n");
                attack=1;
                }
        return attack;
}
```

Code 4.8: Rede Neuronal Actual

única forma de determinar estes valores com precisão para cada caso específico é por tentativa erro. Por esta razão foi também desenvolvido um método que faz o cálculo de todas as situações possíveis, armazenando aquele que for o melhor par resultado.

O treino apenas é terminado quando for atingido um erro abaixo do 1%. Como referido anteriormente, treino da *RNA* é possível através da introdução de um *dataset*. Esse *dataset* contém valores de entrada na rede e o resultado estimado. Sendo por isso um treino supervisionado e do qual se espera um erro inferior a 1%.

O *Dataset* encontra-se guardado em ficheiro, contendo os valores separados por *;*. Este ficheiro é carregado, quando o botão *Train Neural Network* é pressionado, alimentando de

Code 4.9: Relação Valor/(Momentum.Learning Rate)

X	(Mo)Elevado	(Mo)Correcto	(Mo)Baixo
(LR)Elevado	Não atinge equilibrio	Pode ou não atingir Equilibrio	Salta o ponto de equilibrio
(LR)Correcto	Pode ou não atingir Equilibrio	Atingido o ponto de equilibrio	Pode ou não atingir Equilibrio
(LR)Baixo	Muito tempo para equilibrio	Pode ou não atingir Equilibrio	Salta o ponto de equilibrio

seguida o método *trainnn* acima apresentado.

Um nota importante a ter em consideração prende-se com o facto de ao longo da utilização do IDSNN1, a rede sofrer evolução. Isto é, ao longo da sua utilização, conjunto de valores cujo resultado seja superior a 95% são considerados *scans*, valores cujo resultado seja inferior a 5% são considerados como *não scan*. Em ambos os casos, os resultados são adicionados ao *dataset* de treino. Este novo conjunto de dados alimentará a rede ao longo da sua utilização, em momentos de treino, de forma a optimizar os dados adicionados e a evoluir com o tempo. Daqui pode-se concluir que de x em x tempo a aplicação faz uma pausa no processamento e treina a rede novamente. Findo este processo, a rede volta a processar os dados capturados. É importante salientar que este x deverá ser um valor aleatório, relativamente grande de forma a impedir que a rede esteja sempre em treino e de forma a permitir que sejam capturados dados suficientes para aperfeiçoar a rede. Sendo ele aleatório, impossibilita a previsão da ocorrência deste treino por parte de um elemento mal-intencionado que pretenda subverter o sistema.

Como entrada da rede neuronal, existem treze neurónios. Estes neurónios recebem valores *double* com a seguinte sequência:

- $\#0 ABS(PDa - PDb)$;
- $\#1\ SYNa$;
- $\#2\ SYNb$;
- $\#3\ SYN\ ACKa$;
- $\#4\ SYN\ ACKb$;
- $\#\ 5\ RSTa$;

- # 6 $FINa$;
- # 7 $FINb$;
- # 8 $SYNa - (RSTa + RSTb)$;
- # 9 $RSTa/SYNa$;
- # 10 $RSTb/SYNa$;
- # 11 $Max(ICMP)$;
- # 12 $RSTb$;

Sendo que o conjunto [1..7] é constituído por elementos retirados da base de dados, enquanto os restantes são calculados com base nos valores da base de dados.
O resultado do processamento dos dados de entrada é uma *string* que contém o endereço de IP origem e endereço IP destino, e o valor probabilidade entre zero e um.
Caso o valor seja superior a 0.5, uma mensagem é enviada para a *interface* da aplicação, alertando para um possível ataque.

4.2.3 Integração *snort*

Existe também a integração dos dados existentes no *snort* com um módulo desenvolvido para o efeito, desta feita tenta-se diminuir o número de falsos positivos.
Este módulo é em tudo idêntico ao módulo de *scan detection* ao nível do desenvolvimento, complementando-se mutuamente, dando origem a um sistema híbrido, que chega ao baixo nível de processamento.
Genericamente o modo de funcionamento deste módulo é:

- Obter dados do *snort* de forma aleatória (tal como no módulo anterior). A forma como o módulo *IDSNN1_snort* obtém os dados é semelhante à do módulo IDSNN1_ScanDetection, explicado anteriormente;
- Preparar os dados para a rede neuronal;
- Submeter os dados a apreciação da rede neuronal;
- Se o resultado for abaixo de 0.5, passa para o ponto um, caso contrário continua;

Code 4.10: Relação entre resultado de módulos

	snort (+)	*snort* (+) && IDSNN1_Snort (+)
IDSNN1_Snort (-)	Falso Positivo	X
IDSNN1_ScanDetection(-)	X	Falso Positivo
IDSNN1_ScanDetection(+)	X	Positivo

- Sendo o valor acima dos 0.5, este módulo activa o módulo *scandetection*, utilizando como valores chave o par (IPs,IPo). Tendo em consideração que ambos os sistemas *snort e IDSNN1* se encontram a correr na mesma máquina, a base de dados do *IDSNN1* encontra-se actualizada;

- Processamento dos dados do par (IPs,IPo) por parte do módulo *scan detection*;

- Reinicio do processo;

Através desta breve explicação, verifica-se que para o mesmo pacote, existem três verificações, a primeira, é efectuada pelo *snort*, o segundo é pelo módulo *IDSNN1_Snort* e por fim a confirmação é dada pelo módulo *IDSNN1_ScanDetection*. Sendo os seus resultados práticos explicitados na tabela 4.10 Através da análise da mesma, pode-se concluir que caso os dois módulos do *IDSNN1* tenham resultado positivo em ambas as *RNA*, que se está perante um ataque e/ou um scan à rede, pelo que as comunicações devem ser cortadas de imediato, através da geração de regras automáticas no *IPTables*. Caso o módulo de *IDSNN1_ScanDetection* tenha o resultado abaixo dos 0.5, mas por sua vez, o módulo *IDSNN1_Snort* for positivo, não será tirada nenhuma conclusão ficando o resultado para análise pelo administrador. Se por último o módulo *IDSNN1_Snort* tiver um resultado abaixo dos 0.5, o alerta é considerado um falso positivo e nenhuma acção tomada.
Tendo isto em conta, torna-se importante perceber como o módulo *IDSNN1_Snort* funciona e quais as variáveis que tem em consideração.
Este módulo foi todo criado com base no módulo *IDSNN1_ScanDetection*, pelo que apenas houve algumas adaptações de forma a que este pudesse interagir tanto com o *snort* com o *IDSNN1_ScanDetection*. Este módulo, tal como o anterior, tem um ficheiro de configuração com a estrutura apresentada e exemplificada pelo bloco 4.11:

1. endereço ip e porta da máquina onde se encontra o motor base de dados;

2. nome de utilizador da base de dados;
3. palavra-chave da base de dados, caso não exista coloca-se *"nnn"*;
4. directoria onde se encontra o *trainset*;

Exemplo:

```
localhost:3306
root
nnn
/Users/francisco/Documents/Universidade/Mestrado/2Ano/TeseMestrado/Dataset/trainSetSnort.ts
```

Configuração 4.11: Ficheiro de Confirguração *IDSNN_Snort*

Modelo de dados

O *Snort* possibilita que o registo dos alertas seja efectuado numa base de dados *MySql*. Esta foi a configuração utilizada na máquina de testes, sendo que foi assim projectado para facilitar a integração com *IDSNN1*. Sendo assim obtêm-se dez valores referentes aos alertas registados no *Snort*.
Esses valores apresentam-se de seguida:

1. Tabela *Signature*
   - sig_id;
   - sig_priority;
2. Tabela *IPHdr*
   - ip_len;
   - ip_src;
   - ip_dst;
3. Tabela *TCPHdr*
   - tcp_sport;
   - tcp_dport;
   - tcp_flags;

- tcp_ win;
- tcp_rev;

4. Tabela *Data*

- Payload;

Todos estes dados são utilizados pelo *IDSNN1*, sendo que como entrada na rede neuronal, apenas *sig_id, sig_priority, ip_len, tcp_sport, tcp_dport, tcp_flags, tcp_win, tcp_rev.* Os restantes dois *ip_src, ip_dst* são utilizados para referência e para ser possível fazer a ligação com o módulo *IDSNN1_ScanDetection* e *IPTables.*
Através da análise do modelo de base de dados do *Snort*, figura 4.4 verifica-se que existe uma relação entre as tabelas. Esta relação permite a união das mesmas de modo a obter os dados de forma eficiente. Utilizando uma *query mysql* apresentada pelo código **??**, onde code é a listagem de todos os pares já verificados e que por isso não devem ser verificados novamente até a lista estar completa.

```
"select "
+" sig.sig _priority as sig _priority,"
+" sig.sig _rev as sig _rev,"
+" iphdr.ip _src  as ip _src,"
+" iphdr.ip _dst  as ip _dst,"
+" iphdr.ip _len as ip _len,"
+" iphdr.ip _id as ip _id,"
+" tcphdr.tcp _sport as tcp _sport,"
+" tcphdr.tcp _dport as tcp _dport,"
+" tcphdr.tcp _flags as tcp _flags,"
+" tcphdr.tcp _win as tcp _win,"
+ "data.data _payload as payload"
 +" from ((snort.tcphdr as tcphdr join "
     +" \textit{snort}.iphdr as iphdr on (iphdr.cid like tcphdr.cid)) join"
         +" \textit{snort}.event as event on (tcphdr.cid like event.cid) left   join \textit{snort}.signature as sig on"
           +" (sig.sig _sid like event.signature) join \textit{snort}.data as data on (iphdr.sid like data.sid)) "+ code
+ " where iphdr.sid like "+sid+" and iphdr.cid like "+cid+" "
+ " limit 1;"
```

Query 4.12: Query SQL

Rede Neuronal

O código agora apresentado é em tudo semelhante ao código do módulo *IDSNN1_ScanDetection*, variando apenas no número de nodos de entrada da rede. Mais uma vez, o treino tem que ser feito, obrigatoriamente, antes da utilização da *RNA* sendo feito utilizando um *dataset.*
Este *Dataset* é previamente guardado em ficheiro num formato proprietário do *IDSNN1.* O *Dataset* é o conjunto de dados que irá alimentar o treino da *RNA*. Novamente o algoritmo de

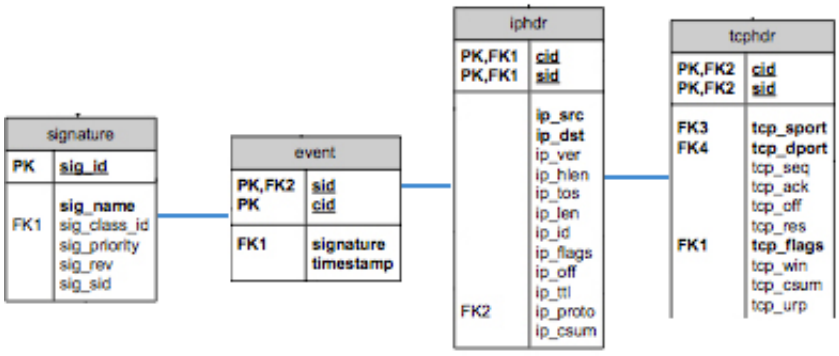

Figura 4.4: Tabelas Esquema *snort* Utilizadas

treino utilizado é *Backpropagation* sendo importante ter em atenção dois factores, já referidos anteriormente, o *Learning Rate e Momentum*. Para este caso especifico o treino da rede termina apenas quando o seu erro for abaixo do 1%. À semelhança do que acontece com o módulo *IDSNN_ScanDetection*, a *RNA* sofre evolução ao longo do tempo.

A aprendizagem dinâmica da rede é feita tendo em conta os dados já processados e salvos para o ficheiro *dataset*.[11]

Relembrar apenas que a rede faz uma pausa de x em x tempo para se re-treinar com novos valores, e que esses valores são todos os resultados acima de 95% e abaixo de 5%, representando respectivamente "*scan*"e "não *scan*".

Mais uma vez é importante ter em consideração as preocupações previamente descritas.

Para este módulo a rede neuronal apenas contém oito neurónios no máximo e um no mínimo. Sendo que, pela análise do código acima apresentado, esta rede é constituida por seis camadas, uma de entrada com oito neurónios, quatro camadas escondidas, também com oito neurónios e por fim uma camada de saída com apenas um neurónio.

4.3 Aplicação

Como enunciado no capítulo anterior, o *IDSNN1* é constítuido por dois módulos que interagem entre si, mas que podem trabalhar de forma independente.

Nesta secção são apresentadas imagens da aplicação.

Quando a aplicação é iniciada, é exibido no ecrã o *layout* do módulo *IDSNN1_ScanDetection*, sendo que, e tal como demonstra a figura 4.2 este é constítuido por dois paineis do tipo *tab* que permitem a mudança entre os módulos.

```
public RedeConta(JTextArea _reportTextBox) {
                network = new BasicNetwork();
                network.addLayer(new BasicLayer(null, true, 8));
                network.addLayer(new BasicLayer(new ActivationSigmoid(), true, 8));
                network.addLayer(new BasicLayer(new ActivationSigmoid(), true, 8));
                network.addLayer(new BasicLayer(new ActivationSigmoid(), true, 8));
                network.addLayer(new BasicLayer(new ActivationSigmoid(), false, 1));
                network.getStructure().finalizeStructure();
                network.reset();
                new ConsistentRandomizer(-1,1,500).randomize(network);
                System.out.println(network.dumpWeights());
                this._reportTextBox =_reportTextBox;
}

public void trainNN(double[][] inputTest, double[][] idealTest){
                double lr=0.5;
                double m=0.005;
                // train the neural network
                trainingSet = new BasicMLDataSet(inputTest,idealTest);
                train = new Backpropagation(network, trainingSet, 0.05, 0.87);
                int epoch=0;
                do
                 {
                    train.iteration();
                    if(epoch%1==0)
                       System.out.println("Epoch #" + epoch + " Error:" + train.getError());
                    epoch++;
                }while((train.getError() > 0.01));
}


public int testNN(double[] inputComp, String ips, String ipd){
                int attack =0;
                MLData mldt = new BasicMLData(inputComp);
                final MLData output = network.compute(mldt);
                if(output.getData(0)>0.5){
                _reportTextBox.append("IP Source: "+ips +" - IP Destination: "+ipd+"\n");
                _reportTextBox.append(inputComp[0] + "," + inputComp[1] + "," + inputComp[2] + ","
                + inputComp[3] + "," + inputComp[4] + "," + inputComp[5] + "," + inputComp[6] + ","
                + inputComp[7]
                + ", actual=" + output.getData(0)+"\n");
                attack=1;
                }
        return attack;
```

Code 4.13: Rede Neuronal Actual

4.3.1 IDSNN1_ScanDetecion

No caso de se optar por executar este módulo, o utilizador tem à sua disposição um menu, figura 4.5, que lhe permite:

- *Train Neural Network* método que permite realizar o treino da *RNA*, utilizando o *DataSet* estipulado previamente. O utilizador tem oportunidade de ver como é a evolução do treino da rede, através da linha de comandos (resultado 4.14), uma vez que esses

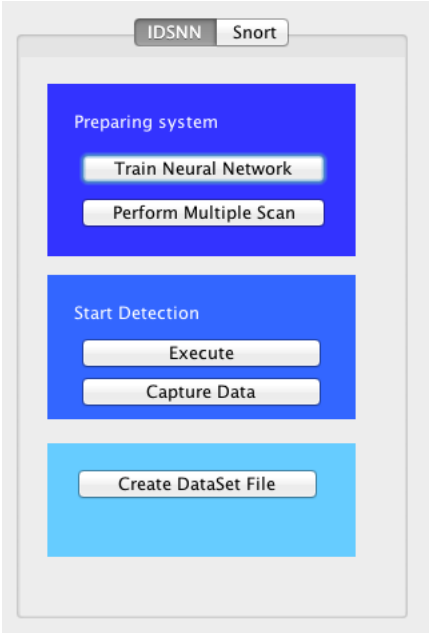

Figura 4.5: Menu Detector Scans

resultados são lá impressos.

```
Epoch \#0 Error:0.31733093191993694
Epoch \#1 Error:0.3870967712956013
Epoch \#2 Error:0.30042720284017904
```

Output 4.14: Exemplo treino rede neuronal

- *Perform Multiple Scans* método que permite a realização de multiplos scans, neste caso, é lido um ficheiro nos quais são colocados comandos de execução de uma aplicação de *scan*. Para execução de cada um destes comandos será criada um *thread* de forma a haver concorrência tornando o processo mais rápido;

- *Execute* método que executa as rotinas de detecção. Esta detecção pode ser *live*, neste caso o módulo *Capture Data* deverá ser activo utilizando uma aplicação como o *TCP-Dump*, ou pode ser pós captura, sendo que neste caso, os dados já estão guardados em base de dados, ou em ficheiro. A execução desta rotina inicia a emissão do relatório que é apresentado na primeira *TextBox* da figura 4.6;

- *Capture Data* método que permite fazer a captura dos pacotes, fazendo o seu registo na base de dados interna. Este método pode ser utilizado para fazer a captura *live* de pacotes, ou para ler resultados através de ficheiro;

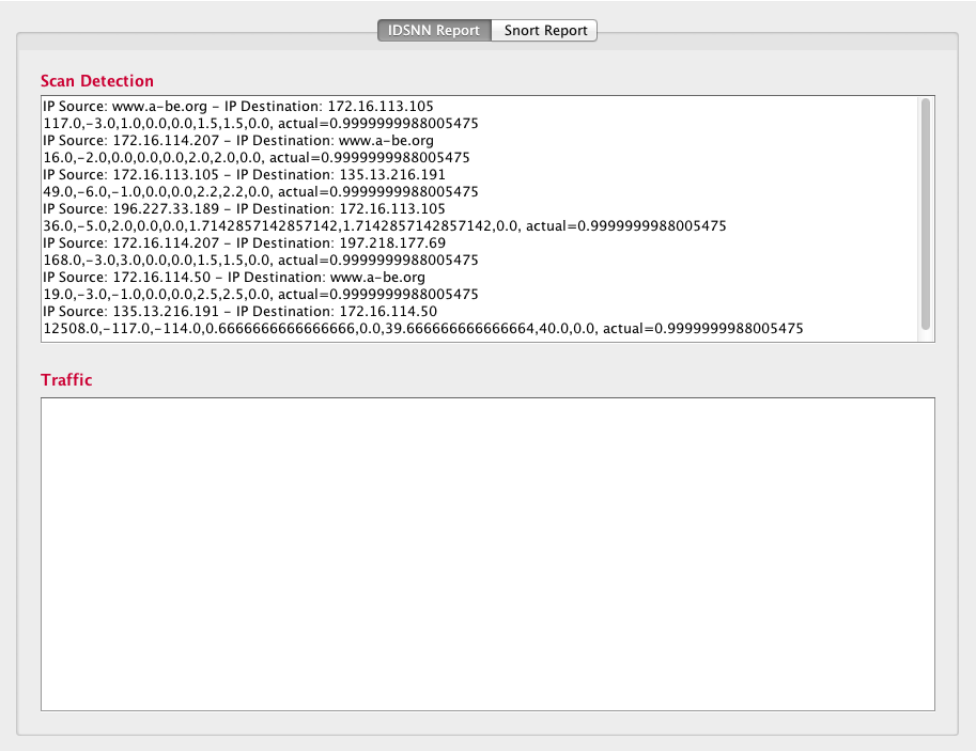

Figura 4.6: Resultados execução Detector Scans

- *Create DataSet File* método que permite a criacção de um ficheiro *dataset*, sendo que neste caso o utilizador terá que escolher quais os pacotes a ser considerados maliciosos, e os que não o são. Este ficheiro faz a exportação de resultados das diversas tabelas para um ficheiro com a estrutura de entrada da *RNA*.

A execução das rotinas *Capture Data e Execute* produzem a impressão de resultados no ecrã de resultados. Este é constituído por duas *TextBox*, sendo que a primeira apresenta apenas os resultados relativos a comunicações maliciosas, fazendo a impressão do endereço de IP origem e destino, as variáveis utilizadas e o resultado obtido pela *RNA*. Relembrando no entanto que uma comunicação apenas é considerada maliciosa se o resultado da *RNA* for superior a 0.5. No caso de ser considerado uma comunicação maliciosa, e caso a aplicação esteja a ser executada numa máquina com base *Unix*, pode ser criada uma regra no *iptables*, de forma a bloquear as comunicações com aquela origem, tal como demonstra o código **??** A segunda

```
iptables --insert input 0 --source IP_a_bloquear --jump DROP
```

Code 4.15: ignorar pacotes com origem no IP declarado

TextBox apresenta o tráfego que está a ser capturado. Aqui aparece exactamente o que é lido

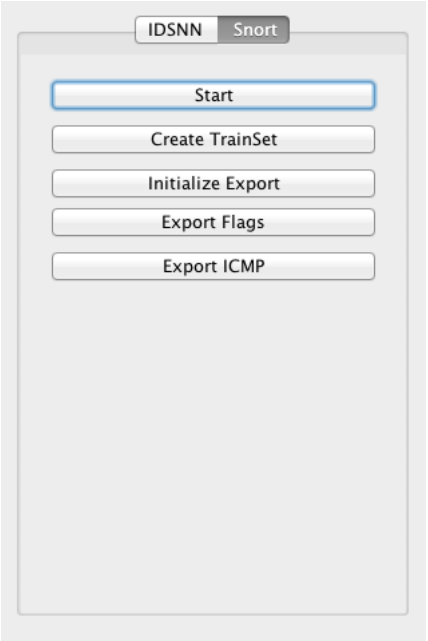

Figura 4.7: Menu *snort*

pelo *IDSNN1*, seja através de ficheiro ou através de captura *live* utilizando uma aplicação como o *TCPDump* (resultado 4.16).

```
16:02:36.644117 IP blacktigermacbook.home.63385 > mad01s03-in-f23.1e100.net.https: Flags [S], seq 100013671, win(...)
16:02:36.688136 IP mad01s03-in-f23.1e100.net.https > blacktigermacbook.home.63385: Flags [S.], seq 3300758141, ack 100013672, win (...)
16:02:37.229055 IP 68.232.35.119.http > blacktigermacbook.home.63378: Flags [F.], seq 1345, ack 985, win 125, length 0
16:02:37.229226 IP blacktigermacbook.home.63378 > 68.232.35.119.http: Flags [F.], seq 985, ack 1346, win 65535, length 0
16:02:37.245841 IP 68.232.35.119.http > blacktigermacbook.home.63378: Flags [F.], seq 1345, ack 985, win 125, length 0
16:02:37.245905 IP blacktigermacbook.home.63378 > 68.232.35.119.http: Flags [F.], seq 985, ack 1346, win 65535, length 0
16:02:37.278081 IP blacktigermacbook.home.63378 > 68.232.35.119.http: Flags [R], seq 327700846, win 0, length 0
```

Captura 4.16: tráfego pelo TCPDump

4.3.2 IDSNN1_Snort

No caso do módulo *snort*, o utilizador é confrontado com uma apresentação idêntica à do módulo descrito na secção anterior. As duas grandes diferenças encontram-se nas opções de execução e na secção de relatório. Neste módulo apenas se conta com uma *TextBox*. Tal como apresentado na figura 4.7, existem cinco opções para execução desta aplicação.

- *Start* executa a rotina de confirmação dos alertas do *Snort*. Neste caso cada pacote é analisado pela *RNA*. Para este caso especifico, ao executar esta rotina, a rede neuronal é treinada, automáticamente, tendo em conta o *TrainSet* designado;

Figura 4.8: Menu *snort*

- *Create TrainSet* método semelhante ao módulo analisado anteriormente. Neste caso os dados existente na base de dados do *snort* são preparados e guardados no formato de *TrainSet*;
- *Initialize Export* prepara dados básicos do *snort* para exportação, este método poderia estar inserido nos dois seguintes, mas desta forma optimiza-se recursos, pois permite paralelismo de execução;
- *Export Flags* método que permite fazer a passagem dos dados relativos às *flags* do *snort* para a base de dados do *IDSNN1*;
- *Export ICMP* método que permite fazer a exportação dos dados relativos a pacotes *ICMP* do *Snort* para o *IDSNN1*;

Será importante no entanto ter em conta que estas três últimas opções apenas são úteis se o *snort* fizer o registo de todos os pacotes que por ele passam. Neste caso torna-se desnecessário executar o *TCPDump* do lado do *IDSNN1_ScanDetection*, uma vez que os resultados do *snort* podem ser aproveitados.
Por fim é importante realçar que a figura **??**, apresenta todos os alertas emitidos pelo *snort* e que foram considerados como tráfego malicioso pelo *IDSNN1_Snort*. Estes dados são então passados para o módulo *IDSNN1_ScanDetection* que os irá analisar.

CAPÍTULO 5

Discussão de Resultados

A discussão de resultados do *IDSNN1* é feita analisando os seus módulos de forma independente, isto porque ambos têm uma *RNA* dedicada, com funções de activação e parâmetros distintos.

5.1 DataSet

Para ser possível fazer a utilização deste sistema, foi necessário a criar dois *dataset* de teste distintos, um para treino da *RNA* e outro para verificação da viabilidade do sistema. Os *dataset* de treino e teste foram gerado a partir de duas fontes distintas.
Para obtenção de pacotes que não contêm nenhum tipo de *scan*, utilizou-se o *DARPA Intrusion Detection Evaluation Dataset* de 1999, fornecido pelo *MIT* e comummente aceite como *dataset* de teste. Para obtenção de pacotes referentes a *scans*, foi gerado internamente tráfego utilizando *nmap*, capturando-o com o *tcpdump*. De forma a concretizar esta tarefa, foram gerados quinze scans distintos, a seis máquinas distintas. Será importante considerar que foram utilizados quinze endereços de IP diferente de forma a simular as quinze máquinas, através da utilização das abaixo discriminadas:

- Acer Aspire One ZG5, Ubuntu 11.04, 2Gb Ram;
- MacBook Pro late 2009, Mac os X lion, 5 Gb Ram;
- Asus EEEPC 1201NL, Windows XP SP3, 2 Gb Ram;
- Desktop, Intel P4 2.4gh, Windows XP SP1, 768 Mg Ram;
- Acer Aspire 5601, Windows 8Developers Edition, 2 Gb Ram;

- MacBook Pro late 2009, BackTrack 4 r1, 2Gb Ram, Pen Wireless;

No caso da utilização do Macbook, a mesma máquina foi utilizada com configurações diferentes de memória RAM e placa *wireless*. Nas restantes máquinas, foi-se alternando entre conectividade por cabo de rede e por *wireless*. Será importante anotar ainda que foram criados e encerrados serviços distintos ao longo dos testes.
Alguns serviços activos:

- *Xampp/Lampp* portas 80, 8080, 443;
- *Vuze/utorrent/transmission* portas 52531, 56342;
- *RDP* porta 3389;
- *IOW'10 Stocks* porta 60000;
- *IOW'10 Order* porta 60001;
- *IOW'10 Report* porta 60002;
- *MySQL* porta 3306;
- *kerberos-sec* porta 88;
- *Mobile Mouse* porta 51101;

O *Dataset* com *scans* foi gerado em dois dias distintos, sendo que no segundo dia, foram gerados *scans* em dois momentos do dia diferentes, um da parte da tarde e outro durante a madrugada.
Durante os testes nenhuma das máquinas teve connectividade à internet, estando ligadas entre si através de um router *Draytek Vigor 2820* com firewall inactiva.

5.1.1 Scans efectuados

O *DataSet* de teste e treino foi criado através da execução dos seguintes *scans*[30]

1. nmap IP_Destino;
2. nmap -O IP_Destino;
3. nmap -sL IP_Destino;

4. nmap -sP IP_Destino;
5. nmap -sT IP_Destino;
6. nmap -sF IP_Destino;
7. nmap -sN IP_Destino;
8. nmap -sX IP_Destino;
9. nmap -sI IP_Destino;
10. nmap -sP IP_Destino;
11. nmap -sS IP_Destino;
12. nmap -sW IP_Destino;
13. nmap -sR IP_Destino;
14. nmap -sS -T4 -vv -r -sV -O -n -F -oA IP_Destino;
15. nmap -PN -sT -A -p T:80,443,8080,8888,8088 -oA webapps -T28 IP_Destino;
16. nmap IP_Destino[ip1...ipn];

5.2 IDSNN1_ScanDetection

Como enunciado anteriormente, este módulo utiliza uma rede neuronal artificial com treze neurónios em cada uma das camadas, exceptuando a última, onde existe um. Neste caso particular, é utilizada uma transformação da função de activação *Sigmoid* através de uma *Tangente Hiperbólica*, e desta forma os seus resultados variam numa amplitude maior, entre -1 e 1, representando tráfego malicioso e não malicioso respectivamente. O algoritmo de treino foi *BackPropagation* sendo o *linear rate* de 0.01 e o *momentum* de 0.03.
Os resultados da execução deste módulo encontram-se apresentados nas tabelas 5.2 e 5.1, representando a primeira um extracto dos pacotes analisados, e a segunda um resumo da prestação geral.
Pela análise da tabela 5.2 é possível verificar que as linhas **1,2,3,4** são representativos da emissão de um alerta, enquanto as linhas **5,6,7,8,9,10,11,12,13,14** representam tráfego legítimo.

Através desta análise é possível então aferir de que forma os *scans* interferem com o comportamento normal da rede. A primeira diferença que se verifica entre um *scan* e tráfego legítimo, é a geração de um elevado número de *RST's*, isto é, quebra de comunicações a meio, situação recorrente aquando de um *scan*. Por sua vez, a diferença entre as portas escutadas e as portas que responderam positivamente, isto é, sem um pacote *RST*, poderá ser muito elevada existindo ainda, a geração de poucos pacotes *SynAck* por parte da máquina vítima.

Por sua vez, o tráfego legítimo cria um baixo número pacotes *RST*, gerando em contrapartida um número de pacotes *FIN* muito idêntico ao número de ligação inicializadas/estabelecidas. Sendo assim, é possível verificar que em situações normais, o resultado das colunas **11,12** será zero, ou muito próximo disso, enquanto que em casos de *scan* estes valores devem ser superiores.

O caso particular da linha *1* representa a geração de pacotes *FIN*, para que o atacante possa verificar quais os serviços disponíveis e também quais as portas sob vigilância de uma firewall.

Captura 5.1: Resultados IDSNN_Scandetection

Tipo de resultado	Legítimo	Scan	Falsos Positivos	Eficácia
Teórico	5485	15	X	100%
Prático	5436	15	49	99,10%

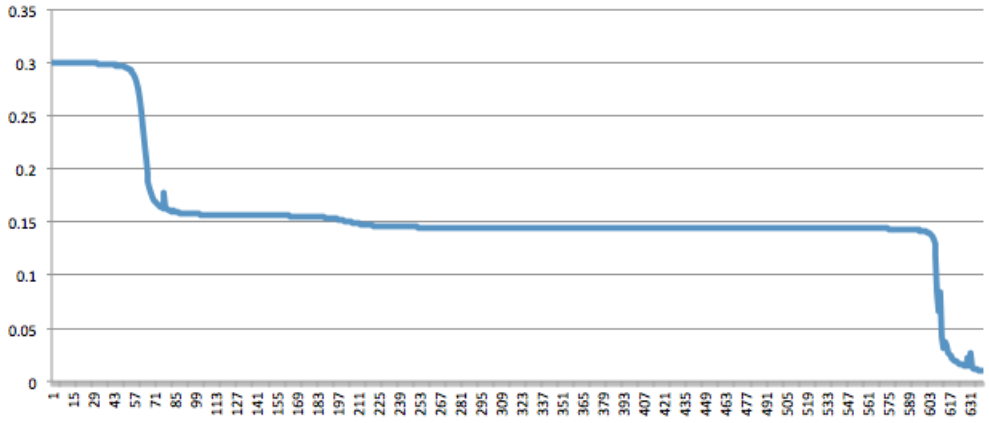

Figura 5.1: Gráfico Erro/Iteração IDSNN1_ScanDetection

Utilizando o *TrainSet* representado pelas tabelas C.8, C.9, C.10 que se encontram no apêndice foi possível atingir um *erro* de *0.0099* ao final de 640 iterações, tal como demonstrado pela figura 5.1 e pela tabela C.7.

Como acima enunciado, a tabela 5.1 apresenta o resumo dos resultados. Para o caso de estudo apresentado, foram analisados dados relativos a 5500 ligações *TCP/IP*, 5485 retirados do *Dataset do MIT* e 15 gerados internamente. Sendo que para as condições apresentadas, o *IDSNN1_ScanDetection* teve uma taxa de sucesso de 99,10%, tendo gerado 49 falsos positivos, detectando os 15 *scans* efectuados e as restantes 5436 ligações como de tráfego legítimo.

Captura 5.2: Resultados obtidos *IDSNN1_ScanDetection*

Nº	Abs(PDa-PDb)	Syna	Synb	SynAcka	SynAckb	RSTa	FINa	FINb	Syna-(RSTa+RSTb)	RSTa/Syna	RSTb/Syna	Max(ICMP)	RSTb		Resultado
1	981.0	27881.0	0.0	0.0	189.0	189.0	12953.0	0.0	27692.0	0.00677	0.0	0.0	189.0	-1	0.4698
2	991.0	13797.0	0.0	0.0	35.0	35.0	0.0	0.0	13699.0	0.0025	0.0045	0.0	35.0	-1	0.4698
3	267.0	13104.0	0.0	0.0	112.0	119.0	0.0	0.0	2408.0	0.0090	0.8071	0.0	119.0	-1	0.9172
4	989.0	13923.0	0.0	0.0	329.0	329.0	0.0	0.0	13552.0	0.02362	0.0030	35.0	42.0	-1	0.46980
5	104.0	105.0	0.0	105.0	0.0	0.0	105.0	105.0	105.0	0.0	0.0	0.0	0.0	-1	-0.99803
6	28.0	29.0	0.0	29.0	0.0	0.0	29.0	29.0	29.0	0.0	0.0	0.0	0.0	-1	-0.9980
7	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	-1	-0.9866
8	59.0	60.0	0.0	60.0	0.0	0.0	60.0	60.0	60.0	0.0	0.0	0.0	0.0	-1	-0.9980
9	189.0	190.0	0.0	190.0	0.0	0.0	190.0	190.0	190.0	0.0	0.0	0.0	0.0	-1	-0.998
10	109.0	110.0	0.0	110.0	0.0	0.0	110.0	110.0	110.0	0.0	0.0	0.0	0.0	-1	-0.9980
11	3.0	3.0	6.0	3.0	6.0	0.0	9.0	9.0	3.0	0.0	0.0	0.0	0.0	-1	-0.9955
12	27.0	28.0	0.0	28.0	0.0	0.0	28.0	28.0	28.0	0.0	0.0	0.0	0.0	-1	-0.9980
13	28.0	29.0	0.0	29.0	0.0	0.0	29.0	29.0	29.0	0.0	0.0	0.0	0.0	-1	-0.9980
14	75.0	76.0	0.0	76.0	0.0	0.0	76.0	76.0	76.0	0.0	0.0	0.0	0.0	-1	-0.9980

5.3 IDSNN1_Snort

O *IDSNN1_Snort* é composto por uma rede neuronal artificial com oito neurónios na camada de entrada e nas camadas escondidas, e um neurónio na camada de saída. A função de activação utilizada neste caso é do tipo *Sigmoid* normal, sendo os seus resultados com valores distribuídos entre 0 e 1.

Ao contrário do módulo apresentado anteriormente, neste é considerado *scan* todos os *inputs* cujo resultado de processamento seja superior a 0.5 e tráfego legitimo todos os que tenham resultado inferior a esse mesmo valor.

Tendo em consideração o *TrainSet* representado pela tabela do apêndice C.1, obteve-se um erro ao final de 28 iterações de *0.0044%*, tendo sido testados diversos valores para as variáveis *momentum e learning rate* até se atingir este ponto de equilibrio. Os valores ficaram então estabelecidos em 0.7 e 0.1 para *momentum e learning rate* respectivamente. Os resultados

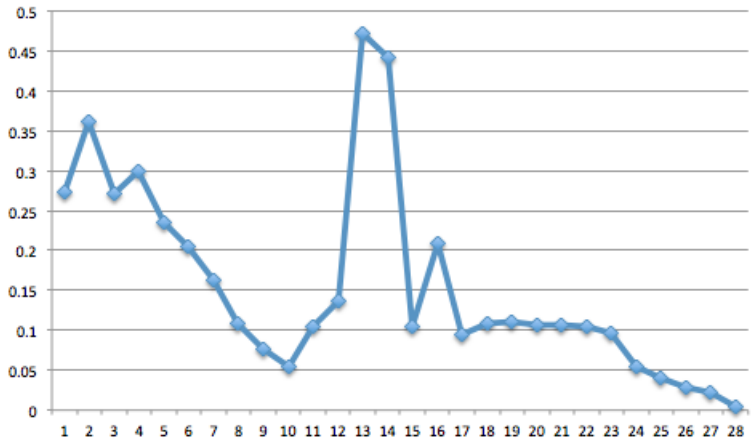

Figura 5.2: Gráfico Erro/Iteração IDSNN1_Snort

deste processo encontram-se representados pela tabela C.6 que deu origem ao gráfico representado na figura 5.2, onde o eixo dos x representa o número de iterações e o eixo dos y o erro. Pela análise da mesma, é possível verificar a influência da variável *momentum* na aprendizagem da *RNA*, verificando-se que a utilização de um valor elevado, para esta, leva a que possam existir saltos sobre a solução, elevando o erro, tal como demonstrado na iteração 13.

A tabela 5.3, apresenta os resultados obtidos pelo *IDSNN1_Snort* para um determinado conjunto de dados aleatóriamente retirados da base de dados.

A base de dados contém resultados com e sem *scans* efectuados, pelo que alguns registos são então falsos positivos.

Captura 5.3: Resultados Snort

Col	Payload	sig_priority	ip_len	tcp_sport	tcp_dport	tcp_flags	tcp_win	sig_rev	Resultado
1	73624	3	1500	80	52716	16	73	0	0.010045443764750936
2	50726	-11	40	61135	7676	1	2048	0	0.9636579780558172
3	50726	-11	40	61135	1094	1	2048	0	0.9636579780558172
4	50726	-11	40	61135	28201	1	3072	0	0.9636579780558172
5	1000	3	1400	80	60836	16	10622	0	0.009816676418109859
30	1000	3	143	3306	1027	24	274	0	0.9636579780558172
6	73624	3	52	53214	80	16	33304	0	0.01595376360450151
7	50726	-11	40	61134	9502	1	4096	0	0.9636579780558172
8	50726	-11	40	61135	9100	1	3072	0	0.9636579780558172
9	73624	3	1500	80	53213	16	215	0	0.010045443764750936
10	50726	-11	40	61134	1100	1	3072	0	0.9636579780558172
11	73624	3	1500	80	53214	16	979	0	0.010045443764750936
12	73624	3	1420	80	52439	24	8276	0	0.010045443764750936
13	73624	3	52	80	52318	16	125	0	0.010045443764750936
14	50726	-11	40	61134	3828	1	3072	0	0.9636579780558172
15	50726	-11	40	61134	63331	1	2048	0	0.9605188671952722
16	50726	-11	40	61135	2608	1	1024	0	0.9636579780558172
17	73624	3	52	52192	80	16	33304	0	0.015953763604501518
18	73624	3	1500	80	52365	16	43	0	0.010045443764750936
19	50726	-11	40	61135	783	1	1024	0	0.9636579780558172
20	73624	3	1500	80	53213	16	215	0	0.010045443764750936
21	50726	-11	40	61135	51493	1	1024	0	0.9641550860047846
22	50726	-11	40	61135	1033	1	1024	0	0.9636579780558172
23	73624	3	52	53214	80	16	33304	0	0.015953763604501518
24	73624	3	260	3306	1027	24	274	0	0.010045443764750936
25	73624	3	1500	80	53056	16	86	0	0.010045443764750936
26	50726	-11	40	61135	2638	1	4096	0	0.9636579780558172
27	73624	3	1500	80	52660	16	14	0	0.010045443764750936
28	50726	-11	40	61135	2522	1	3072	0	0.9636579780558172
29	50726	-11	40	61135	687	1	2048	0	0.9636579780558172

Para o caso apresentado, 30 registos, as linhas **1 ,5 ,6 , 9, 11, 12, 13, 17, 18, 20, 23, 24, 25, 27 e 30** representam falsos positivos gerados pelo *snort*, enquanto que as linhas **2, 3, 4, 7, 8, 10, 14, 15, 16, 19, 21, 22, 26, 28 e 29** representam *scans* efectuados à rede.

Para esta amostra, a taxa de detecção é elevada, sendo que em trinta registos analizados houve a geração de um falso positivo, ou seja uma taxa de sucesso de 97,77%. Continuando a análise, é perceptivel que a alteração de valores provoca a imediata alteração do resultado final gerado pela *RNA*. No caso apresentado, verifica-se que as linhas 5 e 30 apesar de terem o mesmo valor de *payload*, têm resultados muito diferentes, isto causado pela variação das restantes variáveis. Se se tomar em consideração, por sua vez, as linhas 24 e 25, verifica-se que apesar de o valor de *payload* ser o mesmo, o resultado final do processamento das duas amostras não é diferente, tal como no caso anterior, mas sim o mesmo. Estes dois exemplos demonstram que o resultado final da *RNA* é dependente da globalidade das variáveis.

Para evitar que o seja gerado novamente um falso positivo ao receber uma amostra do género da linha 30, o administrador de sistema deverá adicionar manualmente esta ao *TrainSet*, sendo que lhe deverá acrescentar ";0"no final.

A tabela 5.4 gerou 121 falsos positivos que foram enviados para o módulo ScanDetection.

Captura 5.4: Resultados Snort

Número de alertas	Alerta Snort	Legítimo	Scan	Eficácia
10260	Falsos Positivos	10139	121	98.82%
331	Scan	0	331	100.00%

Este gerou um falso positivo a partir dos dados recebidos. Simplificando, com a conjugação da aplicação *IDSNN1* e *Snort*, houve uma redução de falsos positivos de 99.99% na emissão de falsos positivos ao nível do *scanning*. A tabela 5.4 demonstra o nível de eficácia do módulo

Captura 5.5: Resultados IDSNN_Snort + IDSNN_ScanDetection

Número de alertas	Legítimo	Scan	Eficácia
121	120	1	99,17%

IDSNN1_Snort na detecção de scans de rede. No caso apresentado, dos alertas gerados, 331 eram efectivamente *scans* enquanto que 10260 são falsos positivos.

Pela análise da tabela, verifica-se que dos 10260 (falsos positivos do *Snort*), o *IDSNN1_Snort* considerou, erradamente, que 121 eram *scans*, enquanto ilibou os restantes 101143 pacotes, tendo por isso uma eficácia de 98,81%.

Por sua vez, no processamento dos 331 alertas legítimos, o *IDSNN1_Snort* teve uma eficácia de 100% validando todos os registos como *scans*.

Este dado é importante, tendo em conta que a única consideração que o administrador da rede terá que fazer, será a de criar e validar um *TrainningSet*.

A tabela 5.5 apresenta o resultado da conjugação dos dois módulos integrantes do *IDSNN1*. Pela sua análise é possível verificar que dos 121 resultados considerados *scans* pelo *IDSNN1_Snort*, 120 foram considerados com tráfego legitimo e 1 como *scan*, gerando por isso 1 falso positivo. Sendo assim é aceitavel afirmar que o *IDSNN1* produz uma redução substancial ao nível dos falsos positivos, tendo por isso uma eficácia de *99,17%*.

CAPÍTULO 6

Conclusões e Trabalho Futuro

6.1 Conclusões

Os objectivos a que se propôs esta dissertação foram alcançados. O protótipo criado é uma aplicação funcional *cross-plataform* que pode facilmente integrar dados recolhidos pelo *IDS Snort*.

Ao longo deste projecto houve diversos problemas, quer na implementação das estruturas essenciais, quer na conclusão do próprio protótipo. Isto porque ao longo dos meses, as próprias *frameworks* foram alteradas. Exemplo disso foi a necessidade de alterar as redes neuronais por evolução do *encog* da versão 2 para a versão 3, na qual foram dispensados alguns métodos anteriormente implementados. Esta situação levou a que fosse necessário perceber em que influenciavam as alterações, e de que forma a *performance* poderia ser afectada positiva ou negativamente.

É importante realçar que para os testes efectuados, a aplicação apresentou uma melhoria acentuada nos resultados, sendo que no entanto se deve ter em consideração que o ambiente de teste não foi o melhor. Isto deveu-se a limitações de equipamento e tempo, causado por não cumprimento do primeiro prazo estabelecido para o protótipo. Porém é importante realçar que a aplicação passou com sucesso no processamento de informação fornecida por um *DataSet* internacionalmente aceite para teste de *IDS* e no qual o *Snort* falhou.

Sendo assim foi então possível definir uma metodologia híbrida que trata a problemática da geração de falsos positivos por *IDS*, sendo que fica provada a possibilidade de fazer a detecção comportamentos anómalos por sistemas específicos que analisam desvios comportamentais, não estando por isso necessitados de uma monitorização constante por parte do Administrador de Sistema. Isto apenas é possível graças à implementação de estruturas de análise de

comportamento com capacidade de aprendizagem.
Tal como demonstrado neste protótipo a utilização de redes neuronais artificiais é uma solução plausível para esta questão, uma vez que preenche requisitos como paralelismo, rapidez, eficácia, aprendizagem.
Será importante concluir ainda que a possibilidade de adição automática por parte da aplicação de dados ao *trainset* deve ser ponderada e seguida com atenção. Isto porque é importante verificar que não há a adição de falsos positivos e ainda impedir o aumento desmesurado desta estrutura. Tal ocorrência levará a tempos maiores de treino, o que colocará o detector mais tempo em espera.
Por fim os resultados apresentados são satisfatórios uma vez que cumprem o objectivo primário deste protótipo, ou seja, a redução efectiva da emissão de falsos positivos por parte de um *IDS* relativamente a scans.
O conjunto *IDSNN1* obteve então uma taxa de eficácia de 99,17% reduzindo o número de alertas gerados pelo *snort*, quando este utiliza as regras genéricas. Desta forma houve uma redução de 10260 alertas para 1, demonstrando a possibilidade de detecção fazendo contagem de pacotes e utilizando redes neuronais artificiais.

6.2 Trabalho Futuro

Haverá alguns aspectos a melhorar no prótotipo desenvolvido.

- a metodologia utilizada para fazer o *parsing* de dados recebidos, deverá ser substituido por expressões regulares, mais facilmente adaptaveis;
- no caso da execução do método *Executar Scan*, as *threads* inicializadas devem ser todas terminadas, sendo que em caso de bloqueio, estas devem ser terminadas.
- as consultas à base de dados deve ser optimizada, de forma a que os valores random obtidos sejam gerados internamente, não sobrecarregando o motor de base de dados;
- optimização do *layout* de apresentação;
- possibilidade de exportação de resultados;
- integração com outros *IDS*;

Uma solução diferente é baseada em *Support Vector Machines*, genericamente, este método recebe dados de entrada e classifica estes dados tendo em consideração duas classes de entrada. Desta forma os *SVM*, são classificadores bilineares não probabilísticos Esta solução foi comparada com as *RNA*, num estudo realizado por Mukkamala, Janoski e Sung, que utilizando apenas 13 dos 41 um dados disponibilizados pelo *Darpa Dataset* permitiram ter elevada precisão. Apesar de tanto as *RNA* como as *SVM* terem tido resultados pouco acima dos 99% de correcção, as *SVM* tiveram um resultado um pouco melhor, deixando em aberto esta área de investigação.[27]

Bibliografia

[1] chapter 3 - Dissecting Snort, pages 43–68.

[2] James C. Foster Aaron W. Bayles. *Penetration Tester's.* Number 1-59749-021-0. Syngress, 2006.

[3] Shon Harris Allen Harper. *Gray Hat Hacking Third Edition*, volume 978-0-07-174256-6. McGrawHill, 2011.

[4] S. Kalaiarasi Anbananthen, G. Sainarayanan, Ali Chekima, and Jason Teo. Data mining using artificial neural network tree. In *Computers, Communications, Signal Processing with Special Track on Biomedical Engineering, 2005. CCSP 2005. 1st International Conference on*, pages 160 –164, nov. 2005.

[5] David Barroso Berrueta. A pratical approach for defeating nmap os-fingerprinting, March 2003.

[6] Alan Bivens, Rasheda Smith, Chandrika Palagiri, Boleslaw Szymanski, and Mark Embrechts. Network-based intrusion detection using neural networks, 2002.

[7] Douglas E. Comer. *InternetWorking with TCP/IP Principles, Protocols, and Architectures.* Number 0-13-18380-6. Prentice Hall, 2000.

[8] H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 240 –250, may 1992.

[9] Hervé DEBAR, Monique BECKER, and Didier SIBONI. A neural network component for an intrusion detection system. *IEEE*, 1992.

[10] Gurpreet Dhillon. *Principles of Information Systems Security: text and cases.* Number 978-0471450566. John Wiley & Sons, 2007.

[11] Haddadi F., Khanchi S., Shetabi M., and V. Derhami. Intrusion Detection and Attack Classification using Feed-Forward Neural Network. page 4, 2010.

[12] Anup K. Ghosh and Aaron Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, pages 12–12, Berkeley, CA, USA, 1999. USENIX Association.

[13] Daniel Graupe. *Principles of artificial neural networks*, volume 6. World Scientific, 2007.

[14] Stephen Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. 1988.

[15] M. Gudadhe, P. Prasad, and K. Wankhade. A new data mining based network intrusion detection model. In *Computer and Communication Technology (ICCCT), 2010 International Conference on*, pages 731 –735, sept. 2010.

[16] Simon Haykin. *Neural Networks - A Comprehensive Foundation*. Number 81-7808-300-0. Prentice Hall, 1999.

[17] Jeff Heaton. *Programming Neural Networks With Encog 2 in Java*. Number 1-60439-011-5. Heaton Research, Inc, March 2010.

[18] Kent Nabors Jayson E. Street. *Dissecting The Hack - The Forbiden Network*. Number 978-1-59749-478-6. Elsevier, 2010.

[19] Christopher Gerg Kerry J. Cox. *Snort and IDS Tools*. Number 0-596-00661-6. O'Reilly, August 2004.

[20] Timothy P. Layton. *Information Security: Design, Implementation, Measurement, and Compliance*. Number 978-0-8493-7087-8. Auerbach, 2007.

[21] Cynthia Bailey Lee, Chris Roedel, and Elena Silenok. Detection and characterization of port scan attacks, 2003.

[22] Richard Lippmann, Joshua Haines, David Fried, Jonathan Korba, and Kumar Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In Hervé Debar, Ludovic Mé, and S. Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 162–182. Springer Berlin / Heidelberg, 2000. 10.1007/3-540-39945-3_11.

[23] Johnny Long. *Google Hackinh - For Penetration Testers - Volume2*. Number 978-1-59749-176-1. Syngress, 2008.

[24] Amir Ali Sha'bani Mansour Sheikhan. Fast neural intrusion detection system based on hidden weight optimization algorithm and feature selection. *World Applied Sciences Journal*, 2009.

[25] Mohammad ZULKERNINE Mehdi MORADI. A neural network based system for intrusion detection and classification of attacks. 2004.

[26] Stacy Prowell Mike Borkin, Reb Kraus. *Seven Deadliest Network Attacks*. Number 978-1-59749-549-3. Syngress, 2010.

[27] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1702 –1707, 2002.

[28] Stephen Northcutt. *Snort 2.1 Intrusion Detection*. Number 1-931836-04-3. Syngress, 2004.

[29] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *In Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID*, pages 102–124. Springer, 2004.

[30] Becky Pinkard. *Nmap In the enterprise your guide to network scanning*. Number 978-1-59749-241-6. Syngress, 2008.

[31] Jan-Oliver Wagner Renaud Deraison. *Nessus User's Manual*. Teenable, January 2001.

[32] R. Rojas. *Neural Networks*. Springer-Verlag,, 1996.

[33] Jake Ryan, Meng jang Lin, and Risto Miikkulainen. Intrusion detection with neural networks. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 943–949. MIT Press, 1998.

[34] Jake RYAN, Meng-Jang LIN, and Risto MIIKKULAINEN. Intrusion detection with neural networks. *AAAI Technical Reports*, 1997.

[35] N. Seliya and T.M. Khoshgoftaar. Active learning with neural networks for intrusion detection. In *Information Reuse and Integration (IRI), 2010 IEEE International Conference on*, pages 49 –54, aug. 2010.

[36] J. Shun and H.A. Malki. Network intrusion detection system using neural networks. In *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, volume 5, pages 242 –246, oct. 2008.

[37] M Wiscy Soniya B. Detection of tcp syn scanning using packet counts and neural network. *International COnference on Signal Image Technology*, 2008.

[38] George Kurtz Stuart McClure, Joel Scambray. *Hacking Exposed 6 - Network Security Secrets and Solutions*. Number 978-0-07-161385-0. McGrawHill, 2009.

[39] S. Tamura and M. Tateishi. Capabilities of a four-layered feedforward neural network: four layers versus three. *Neural Networks, IEEE Transactions on*, 8(2):251 –255, mar 1997.

[40] K.M.C. Tan and B.S. Collie. Detection and classification of tcp/ip network services. *Computer Security Applications Conference, Annual*, 0:99, 1997.

[41] Vijay Vaishnavi. Design science research in information systems.

[42] Fu Xiao and Xie Li. Using outlier detection to reduce false positives in intrusion detection. *Network and Parallel Computing Workshops, IFIP International Conference on*, 0:26–33, 2008.

APÊNDICE A

Exemplo NMAP

```
blacktigermacbook:~ francisco$ nmap -n -sP 192.168.0.1-10
Starting Nmap 5.21 ( http://nmap.org ) at 2011-10-10 17:00 WEST
Nmap scan report for zonhub.home (192.168.0.1)
Host is up (0.0043s latency).
Nmap scan report for BlackTigerMacBook.home (192.168.0.2)
Host is up (0.00081s latency).
Nmap scan report for rabidkittenw8.home (192.168.0.4)
Host is up (0.0032s latency).
Nmap done: 10 IP addresses (3 hosts up) scanned in 1.23 seconds
```

Code A.1: Scan entre endereços de IP

```
blacktigermacbook:~ francisco$ nmap 192.168.0.1/24
Starting Nmap 5.21 ( http://nmap.org ) at 2011-10-10 16:39 WEST
Strange error from connect (65):No route to host
Nmap scan report for zonhub.home (192.168.0.1)
Host is up (0.012s latency).
Not shown: 992 closed ports
PORT     STATE SERVICE
80/tcp   open  http
139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
631/tcp  open  ipp
5000/tcp open  upnp
8080/tcp open  http-proxy
8443/tcp open  https-alt
Nmap scan report for BlackTigerMacBook.home (192.168.0.2)
Host is up (0.00021s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
88/tcp   open  kerberos-sec
548/tcp  open  afp
3306/tcp open  mysql
Nmap scan report for rabidkittenw8.home (192.168.0.4)
Host is up (0.0022s latency).
Not shown: 989 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
443/tcp   open  https
445/tcp   open  microsoft-ds
554/tcp   open  rtsp
1723/tcp  open  pptp
2869/tcp  open  unknown
3389/tcp  open  ms-term-serv
5357/tcp  open  unknown
10243/tcp open  unknown
Nmap done: 256 IP addresses (3 hosts up) scanned in 12.08 seconds
```

Code A.2: Scan à rede

```
blacktigermacbook:~ francisco$ sudo nmap -sF 192.168.0.1/24
Password:
Starting Nmap 5.21 ( http://nmap.org ) at 2011-10-10 17:04 WEST
Warning: Unable to open interface vmnet1 -- skipping it.
Warning: Unable to open interface vmnet8 -- skipping it.
Nmap scan report for zonhub.home (192.168.0.1)
Host is up (0.013s latency).
Not shown: 992 closed ports
PORT     STATE         SERVICE
80/tcp   open|filtered http
139/tcp  open|filtered netbios-ssn
443/tcp  open|filtered https
445/tcp  open|filtered microsoft-ds
631/tcp  open|filtered ipp
5000/tcp open|filtered upnp
8080/tcp open|filtered http-proxy
8443/tcp open|filtered https-alt
MAC Address: 00:05:CA:E9:8F:45 (Hitron Technology)
Nmap scan report for BlackTigerMacBook.home (192.168.0.2)
Host is up (0.000087s latency).
Not shown: 997 closed ports
PORT     STATE         SERVICE
88/tcp   open|filtered kerberos-sec
548/tcp  open|filtered afp
3306/tcp open|filtered mysql
Nmap scan report for rabidkittenw8.home (192.168.0.4)
Host is up (0.0011s latency).
All 1000 scanned ports on rabidkittenw8.home (192.168.0.4) are open|filtered
MAC Address: 00:16:36:5E:2A:B3 (Quanta Computer)
Nmap done: 256 IP addresses (3 hosts up) scanned in 39.30 seconds
```

Code A.3: Scan FIN

APÊNDICE B

Blocos de Código

```
public RedeConta(JTextArea _reportTextBox) {
        network = new FeedforwardNetwork();
        network.addLayer(new FeedforwardLayer(10));
        network.addLayer(new FeedforwardLayer(10));
        network.addLayer(new FeedforwardLayer(1));
        network.reset();
        this._reportTextBox =_reportTextBox;
}
public void trainNN(double[][] inputTest, double[][] idealTest){
        // train the neural network
        train = new Backpropagation(network, inputTest, idealTest, 0.01, 0.3);
        int epoch=0;
        do  {
          train.iteration();
          if(epoch%100==0)
                System.out.println("Epoch #" + epoch + " Error:" + train.getError());
                epoch++;
          } while((train.getError() > 0.01));
}
public int testNN(double[] inputComp, String ips, String ipd){
        int attack =0;
        double actual[] = network.computeOutputs(inputComp);
        if(actual[0]>0.5){
                _reportTextBox.append("IP Source: "+ips +" - IP Destination: "+ipd+"\n");
                 _reportTextBox.append(inputComp[0] + "," + inputComp[1] + "," + inputComp[2]
                + ","+ inputComp[3] + "," + inputComp[4] + "," + inputComp[5] + "," +
                inputComp[6] + "," + inputComp[7] + ", actual=" + actual[0]+"\n");
                attack=1;}
        return attack;
}
```

Code B.1: Versão Inicial Rede Neuronal Artificial

APÊNDICE C

DataSet

Code C.1: Dataset Treino Snort

sig_id	sig_priority	ip_len	tcp_sport	tcp_dport	tcp_flags	tcp_win	sig_rev	ataque
73624.0	3.0	1500.0	80.0	53506.0	16.0	315.0	0.0	0
73624.0	3.0	52.0	53507.0	80.0	16.0	31579.0	0.0	0
73624.0	3.0	1500.0	80.0	53470.0	16.0	540.0	0.0	0
73624.0	3.0	601.0	53465.0	80.0	24.0	33304.0	0.0	0
73624.0	3.0	148.0	53478.0	80.0	24.0	33304.0	1.0	0
73624.0	3.0	1500.0	80.0	53214.0	16.0	979.0	0.0	0
73624.0	3.0	52.0	53214.0	80.0	16.0	33304.0	0.0	0
73624.0	3.0	52.0	80.0	53433.0	16.0	68.0	0.0	0
73624.0	3.0	1272.0	80.0	53214.0	24.0	979.0	0.0	0
73624.0	3.0	52.0	53214.0	80.0	16.0	33304.0	0.0	0
73624.0	3.0	1500.0	80.0	53214.0	16.0	979.0	0.0	0
73624.0	3.0	52.0	53214.0	80.0	16.0	33304.0	0.0	0

Code C.2: Dataset Treino Snort

sig_id	sig_priority	ip_len	tcp_sport	tcp_dport	tcp_flags	tcp_win	sig_rev	ataque
73624.0	3.0	1500.0	80.0	53056.0	16.0	86.0	0.0	0
73624.0	3.0	1500.0	80.0	53044.0	16.0	644.0	0.0	0
73624.0	3.0	1500.0	1935.0	52998.0	16.0	114.0	0.0	0
73624.0	3.0	198.0	443.0	52958.0	24.0	964.0	0.0	0
73624.0	3.0	52.0	52968.0	80.0	16.0	33304.0	0.0	0
73624.0	3.0	350.0	80.0	52948.0	24.0	20.0	0.0	0
73624.0	3.0	1500.0	80.0	52949.0	16.0	91.0	0.0	0
73624.0	3.0	52.0	52919.0	80.0	17.0	33304.0	0.0	0
73624.0	3.0	1500.0	80.0	52842.0	16.0	52.0	0.0	0
73624.0	3.0	1500.0	80.0	52430.0	16.0	362.0	0.0	0
73624.0	3.0	52.0	80.0	52693.0	17.0	55.0	0.0	0
73624.0	3.0	1433.0	80.0	52791.0	25.0	7826.0	0.0	0
73624.0	3.0	1500.0	80.0	52192.0	16.0	58.0	0.0	0
73624.0	3.0	40.0	52777.0	80.0	16.0	65272.0	0.0	0
73624.0	3.0	925.0	52752.0	80.0	24.0	32850.0	1.0	0
73624.0	3.0	1500.0	80.0	52716.0	16.0	73.0	0.0	0
73624.0	3.0	1500.0	80.0	52660.0	16.0	14.0	0.0	0
73624.0	3.0	1500.0	80.0	52707.0	16.0	104.0	0.0	0
73624.0	3.0	1500.0	443.0	52661.0	16.0	144.0	0.0	0
73624.0	3.0	1420.0	80.0	52655.0	16.0	63615.0	0.0	0
73624.0	3.0	1500.0	80.0	52557.0	16.0	91.0	0.0	0
73624.0	3.0	52.0	1935.0	52510.0	17.0	114.0	0.0	0
73624.0	3.0	652.0	80.0	52492.0	24.0	1013.0	0.0	0
73624.0	3.0	1500.0	80.0	52430.0	16.0	362.0	0.0	0
50726.0	-11.0	40.0	61135.0	55600.0	1.0	3072.0	0.0	1

Code C.3: Dataset Treino Snort

sig_id	sig_priority	ip_len	tcp_sport	tcp_dport	tcp_flags	tcp_win	sig_rev	ataque
50726.0	-11.0	40.0	61135.0	5510.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	31038.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61134.0	1600.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61134.0	1183.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	9290.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	987.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	1024.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	2251.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61135.0	5678.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	6059.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	8500.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	8290.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	1947.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	1086.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	9290.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	987.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61134.0	1024.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	2251.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61134.0	5678.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	6059.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	8500.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61134.0	8290.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61134.0	1947.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61134.0	1086.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	1035.0	1.0	4096.0	0.0	1

Code C.4: Dataset Treino Snort

sig_id	sig_priority	ip_len	tcp_sport	tcp_dport	tcp_flags	tcp_win	sig_rev	ataque
50726.0	-11.0	40.0	61135.0	7911.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	1050.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	5718.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	9595.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61135.0	16080.0	1.0	1024.0	0.0	1
50726.0	-11.0	40.0	61135.0	722.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	1718.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	8994.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	7627.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61134.0	1035.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	7911.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	1050.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	5718.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	9595.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61134.0	16080.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	722.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61134.0	1718.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	8994.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61134.0	7627.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	34572.0	1.0	2048.0	0.0	1
50726.0	-11.0	40.0	61135.0	2910.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	2190.0	1.0	4096.0	0.0	1
50726.0	-11.0	40.0	61135.0	2135.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	1839.0	1.0	3072.0	0.0	1
50726.0	-11.0	40.0	61135.0	7000.0	1.0	3072.0	0.0	1

Code C.5: Dataset Treino Snort

sig_id	sig_priority	ip_len	tcp_sport	tcp_dport	tcp_flags	tcp_win	sig_rev	ataque
73624.0	3.0	52.0	80.0	53312.0	16.0	68.0	0.0	0
73624.0	3.0	1500.0	80.0	53262.0	16.0	175.0	0.0	0
73624.0	3.0	1500.0	80.0	53241.0	16.0	315.0	0.0	0
73624.0	3.0	1500.0	80.0	53228.0	16.0	248.0	0.0	0
73624.0	3.0	1500.0	80.0	53214.0	16.0	979.0	0.0	0
73624.0	3.0	52.0	53214.0	80.0	16.0	33304.0	0.0	0
73624.0	3.0	1500.0	80.0	53213.0	16.0	215.0	0.0	0
73624.0	3.0	52.0	53189.0	80.0	16.0	33304.0	0.0	0
73624.0	3.0	1500.0	80.0	53187.0	16.0	449.0	0.0	0
73624.0	3.0	1500.0	80.0	53137.0	16.0	67.0	0.0	0
73624.0	3.0	1500.0	80.0	53140.0	16.0	216.0	0.0	0
73624.0	3.0	40.0	53121.0	80.0	16.0	32850.0	0.0	0
73624.0	3.0	64.0	53081.0	80.0	2.0	65535.0	0.0	0

Code C.6: Treino rede Neuronal módulo *IDSNN1_Snort*

Iteração	Erro
0	0.273639493888362
1	0.36211208971361136
2	0.2712079705901923
3	0.2990502849062694
4	0.2350527071315675
5	0.20443637391357697
6	0.16264102622476515
7	0.1089470549545033
8	0.07639448386217679
9	0.05521193195561933
10	0.10415919474961984
11	0.13640476331267745
12	0.47272295488789257
13	0.4414883758717167
14	0.10436016544897665
15	0.2081850157600968
16	0.09405971048605602
17	0.10882894910993564
18	0.10973260601393037
19	0.10682983082852472
20	0.10679793120485179
21	0.10375156842872865
22	0.09748611478515415
23	0.05487154456618648
24	0.03997572748832674
25	0.029381163756775373
26	0.02178625095031585
27	0.004428046867020521

Code C.7: Treino rede Neuronal módulo *IDSNN1_ScanDetection*

Iteração	Erro
630	0.02169369950553468
631	0.01743901188567081
632	0.02693355155746952
633	0.012494351251615122
634	0.011855000721357364
635	0.011471856013159122
636	0.011125795603633698
637	0.010799734622308379
638	0.010492290152646222
639	0.010205446305442942
640	0.009945782348629828

Code C.8: DataSet Treino *IDSNN1_ScanDetection*

PDa-PDb	Syna	Synb	SynAcka	SynAckb	RSTa	FINa	FINb	Syna-(RSTa+RSTb)	RSTa/Syna	RSTb/Syna	Max(ICMP)	RSTb	
4	2	7	2	7	0	9	9	2	0	0	0	0	-1
4	0	5	0	5	0	5	5	0	0	0	0	0	-1
0	6	5	6	5	0	11	11	6	0	0	0	0	-1
3	6	3	6	3	0	9	9	6	0	0	0	0	-1
13	8	8	8	8	0	15	16	8	0	0	0	0	-1
5	4	8	4	8	0	12	12	4	0	0	0	0	-1
1	7	7	7	7	0	14	14	7	0	0	10	0	-1
2	5	7	5	7	0	12	12	5	0	0	0	0	-1
9	1	11	1	11	0	12	12	1	0	0	7	0	-1
0	4	3	4	3	0	7	7	4	0	0	0	0	-1
3	5	8	5	8	0	13	13	5	0	0	0	0	-1
4	5	9	5	9	0	14	14	5	0	0	0	0	-1
1	0	3	0	3	0	3	3	0	0	0	0	0	-1
0	3	3	3	3	0	6	6	3	0	0	0	0	-1
6	3	9	3	9	0	12	12	3	0	0	0	0	-1
4	3	7	3	7	0	10	10	3	0	0	0	0	-1
1	6	8	6	8	0	14	14	6	0	0	1	0	-1
20	25	7	25	7	0	32	32	25	0	0	0	0	-1
0	4	3	4	3	0	7	7	4	0	0	4	0	-1
1	3	3	3	3	0	3	5	1	0	0.67	0	2	-1
3	2	4	2	4	0	6	6	2	0	0	0	0	-1
6	0	7	0	7	0	7	7	0	0	0	0	0	-1
1	2	4	2	4	0	6	6	2	0	0	0	0	-1
1	3	4	3	4	0	7	7	3	0	0	0	0	-1
76	18	79	18	79	0	79	96	18	0	0	0	0	-1
0	5	4	5	4	0	9	9	5	0	0	0	0	-1
5	6	12	6	12	0	18	18	6	0	0	0	0	-1
2	0	3	0	3	0	3	3	0	0	0	0	0	-1
5	3	8	3	8	0	11	11	3	0	0	0	0	-1
3	4	8	4	8	0	12	12	4	0	0	0	0	-1

Code C.9: DataSet Treino *IDSNN1_ScanDetection*

PDa-PDb	Syna	Synb	SynAcka	SynAckb	RSTa	FINa	FINb	Syna-(RSTa+RSTb)	RSTa/Syna	RSTb/Syna	Max(ICMP)	RSTb	
4	3	8	3	8	0	11	11	3	0	0	0	0	-1
2	2	3	2	3	0	5	5	2	0	0	0	0	-1
1	2	3	2	3	0	5	5	2	0	0	0	0	-1
3	2	5	2	5	0	7	7	2	0	0	0	0	-1
0	0	1	1	0	0	0	0	0	0	0	0	1	-1
2	0	2	2	0	0	0	0	0	0	0	0	1	-1
12	0	4	4	0	0	0	0	0	0	0	0	1	-1
20	0	5	5	0	0	0	0	0	0	0	0	1	-1
31	1	0	0	1	31	0	0	-30	31	0	0	2	-1
43	1	0	0	1	43	0	0	-42	43	0	0	2	-1
0	0	1	1	0	0	0	0	0	0	0	0	1	-1
2	0	2	2	0	0	0	0	0	0	0	0	1	-1
12	0	4	4	0	0	0	0	0	0	0	0	1	-1
20	0	5	5	0	0	0	0	0	0	0	0	1	-1
31	1	0	0	1	31	0	0	-30	31	0	0	2	-1
43	1	0	0	1	43	0	0	-42	43	0	0	2	-1
7.0	0.0	8.0	8.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	0.0	0.0	-1
2.0	0.0	3.0	3.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	0.0	0.0	-1
1.0	0.0	2.0	2.0	0.0	0.0	0.0	0.0	-4.0	-1.0	-1.0	0.0	0.0	-1
12.0	0.0	13.0	13.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	0.0	0.0	-1
3.0	0.0	4.0	4.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	0.0	0.0	-1
997.0	1001.0	0.0	0.0	17.0	17.0	0.0	0.0	0.0	0.016983016983016984	0.983016983016983	1.0	984.0	1
7.0	0.0	8.0	8.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	0.0	0.0	-1
98.0	0.0	99.0	0.0	99.0	0.0	99.0	99.0	0.0	-1.0	-1.0	0.0	0.0	-1
11.0	0.0	12.0	0.0	12.0	0.0	12.0	12.0	0.0	-1.0	-1.0	0.0	0.0	-1
120.0	0.0	120.0	0.0	120.0	0.0	120.0	120.0	0.0	-1.0	-1.0	0.0	0.0	-1
20.0	0.0	21.0	0.0	21.0	0.0	21.0	21.0	0.0	-1.0	-1.0	0.0	0.0	-1
2.0	0.0	3.0	0.0	3.0	0.0	3.0	3.0	0.0	-1.0	-1.0	0.0	0.0	-1
29.0	0.0	30.0	0.0	30.0	0.0	30.0	30.0	0.0	-1.0	-1.0	0.0	0.0	-1
12.0	8.0	7.0	8.0	7.0	0.0	14.0	15.0	8.0	0.0	0.0	0.0	0.0	-1
0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	-1

Code C.10: DataSet Treino *IDSNN1_ScanDetection*

PDa-PDb	Syna	Synb	SynAcka	SynAckb	RSTa	FINa	FINb	Syna-(RSTa+RSTb)	RSTa/Syna	RSTb/Syna	Max(ICMP)	RSTb	
2.0	2.0	3.0	2.0	3.0	0.0	5.0	5.0	2.0	0.0	0.0	0.0	0.0	-1
3.0	6.0	5.0	6.0	5.0	0.0	9.0	9.0	6.0	0.0	0.0	0.0	0.0	-1
2.0	2.0	3.0	2.0	3.0	0.0	5.0	5.0	2.0	0.0	0.0	0.0	0.0	-1
0.0	4.0	5.0	4.0	5.0	0.0	9.0	9.0	4.0	0.0	0.0	0.0	0.0	-1
1.0	2.0	3.0	2.0	3.0	0.0	5.0	5.0	2.0	0.0	0.0	0.0	0.0	-1
3.0	5.0	1.0	5.0	1.0	0.0	6.0	6.0	5.0	0.0	0.0	0.0	0.0	-1
14.0	15.0	3.0	15.0	3.0	0.0	18.0	18.0	15.0	0.0	0.0	0.0	0.0	-1
17.0	21.0	4.0	21.0	4.0	0.0	25.0	25.0	21.0	0.0	0.0	0.0	0.0	-1
20.0	25.0	7.0	25.0	7.0	0.0	32.0	32.0	25.0	0.0	0.0	0.0	0.0	-1
25.0	22.0	6.0	22.0	6.0	0.0	28.0	28.0	22.0	0.0	0.0	0.0	0.0	-1
1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	-1
1.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0	0.0	0.0	0.0	0.0	-1
1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	-1
4.0	2.0	7.0	2.0	7.0	0.0	9.0	9.0	2.0	0.0	0.0	0.0	0.0	-1
2.0	1.0	4.0	1.0	4.0	0.0	5.0	5.0	1.0	0.0	0.0	0.0	0.0	-1
4.0	1.0	5.0	1.0	5.0	0.0	6.0	6.0	1.0	0.0	0.0	0.0	0.0	-1
14.0	0.0	929.0	11.0	0.0	879.0	0.0	0.0	-890.0	0.0	0.0	0.0	879.0	1
14.0	0.0	929.0	11.0	0.0	879.0	0.0	0.0	-890.0	0.0	0.0	0.0	879.0	1
989.0	3965.0	0.0	0.0	87.0	87.0	0.0	0.0	3869.0	0.02194199243379571	0.002269861286254729	8.0	87.0	1
199.0	373.0	0.0	0.0	1.0	1.0	0.0	0.0	372.0	0.002680965147453083	0.0	0.0	1.0	1
991.0	25623.0	0.0	0.0	65.0	65.0	0.0	0.0	25441.0	0.0025367833587011668	0.0045662100456621	0.0	117.0	1