Universidade de Aveiro
2008
Departamento de Electrónica, Telecomunicações e informática

Ulisses Miguel Rodrigues França

Metodologias para Identificação de PC Zombies

Universidade de Aveiro
2008
Departamento de Electrónica, Telecomunicações e informática

Ulisses Miguel Rodrigues França

Metodologias para Identificação de PC Zombies

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Mestrado integrado em engenharia de Computadores e Telemática, realizada sob a orientação do Professor Doutor Paulo Salvador e António Nogueira do Departamento de Electrónica, Telecomunicações e informática da Universidade de Aveiro

O júri

Presidente

Doutor João Nuno Pimentel da Silva Matos
Professor Associado da Universidade de Aveiro

Vogais

Doutor Paulo Jorge Salvador Serra Ferreira
Professor Auxiliar da Universidade de Aveiro

Doutor António Manuel Duarte Nogueira
Professor Auxiliar da Universidade de Aveiro

Doutor Joel José Puga Coelho Rodrigues
Professor Auxiliar do Departamento de Informática da Faculdade de Ciências de Engenharia da Universidade da Beira Interior

Agradecimentos

Em primeiro lugar, agradeço aos meus Orientadores, Paulo Salvador e António Nogueira, a oportunidade que me deram para desenvolver esta tese, e pelo apoio prestado durante a execução da mesma.

Aos demais professores da Universidade de Aveiro que contribuíram, directa ou indirectamente, para a minha formação académica.

Aos amigos pelo incentivo e apoio prestado ao longo da realização deste projecto.

Agradeço também aos meus familiares pelo apoio prestado, não só na realização deste trabalho, como no acompanhamento ao longo do curso.

A todos muito Obrigado!

Palavras-chave

Sistema detecção de intrusões, rede neuronal, Botnet, Pc Zombie, cavalo de troia.

Resumo

Este trabalho teve como objectivo o desenvolvimento de um sistema baseado em redes neuronais para a detenção automática de intrusões numa rede local de computadores.

O sistema desenvolvido foi testado num ambiente laboratorial onde foram simuladas intrusões e as acções sobre a rede que as mesmas proporcionaram. Numa primeira fase foram testados vários rootkits para gerar tráfego ilícito embebido em tráfego de aplicações lícitas. Seguidamente utilizando os dados estatísticos recolhidos foi construída uma rede neuronal que depois de treinada mostrou-se capaz de identificar o tráfego que lhe é apresentado à entrada, distinguindo o tráfego que é gerado pelo rootkit como ilícito e o tráfego gerado por uma aplicação de um utilizador autorizado como lícito.

Pretendeu-se com este trabalho iniciar o desenvolvimento de uma arquitectura que dê suporte as técnicas de identificação automática e inteligente de intrusões, usando redes neuronais para a detecção de PC Zombies e BotNets.

keywords

Intrusion detection system, neural network, BotNet, PC Zombie, troyan horse, Traffic Pattern, Traffic signature.

abstract

The purpose of this work was to develop a system based on neural networks for the automatic detention of intrusion on a local network of computers.

The system was tested in a laboratory environment, where was simulated some intrusions and actions on the network that this intrusions provided. Initially where tested out several rootkits to generate illicit traffic (zombie) embedded on licit application traffic. Then, using the statistical data collected was built a neural network, that after trained was able to identify the traffic that was presented at entry, distinguishing the traffic that was generated by the rootkit as illicit and traffic generated by an application of an authorized user as licit.

The main goal of this work is to develop an architecture that supports the techniques of automatic identification and intelligent intrusion detection by using neural networks to detect PC Zombies and BotNets.

Índice

1 INTRODUÇÃO 7

1.1. Objectivo 8

1.2. Motivação 8

1.3. Metodologias do Trabalho 10

1.4. Estrutura da dissertação 11

2. ENQUADRAMENTO DO TRABALHO 13

2.1. Rootkits 13

2.2. BotNets 14

2.3. Malware 17

2.4. Redes Neuronais 18

2.5. Aplicações para detecção e Prevenção de Intrusões 22

3. SISTEMA PROPOSTO 31

3.1. Análise de Requisitos 31

3.2. Arquitectura Utilizada 33

4. TESTE E VALIDAÇÃO 43

4.1. Descrição das Aplicações e do Perfil de Tráfego Gerado 43

4.2. Descrição do Trojan, Funcionalidades e perfil de Tráfego Gerado 48

4.3. Cenários Criados 56

4.4. Resultados 59

5. CONCLUSÃO E TRABALHO FUTURO 69

5.1. Conclusão 69

5.2. Trabalho Futuro 70

6. REFERÊNCIAS BIBLIOGRÁFICAS 73

Índice de Figuras

Figura 1 – Modelo Lógico de uma rede neuronal. 19

Figura 2 – Modelo de um neurónio 20

Figura 3 – Modelo de uma camada com vários neurónios. 20

Figura 4 – Modelo de uma rede neuronal com várias camadas. 21

Figura 5 – Função de transferência hardlim. 22

Figura 6 – Função de transferência purelim. 22

Figura 7 – Função de transferência log-sig. 22

Figura 8 – Intrusion detection agent (IDA). 28

Figura 9 – Arquitectura Rede neuronal proposta. 29

Figura 10 – Sistema proposto para detecção de PC Zombies. 34

Figura 11 – Esquema da rede neuronal utilizada na simulação. 37

Figura 12 – Tráfego gerado quando apenas temos HTTP (download). 43

Figura 13 – Tráfego gerado ao longo do tempo quando apenas temos tráfego FTP (download) 44

Figura 14 – Tráfego gerado ao longo do tempo quando apenas temos tráfego Jogos (download) 44

Figura 15 – Tráfego gerado ao longo do tempo quando apenas temos tráfego Skype (download) 45

Figura 16 – Tráfego gerado ao longo do tempo quando apenas temos tráfego Stream Tv (download) 45

Figura 17– Tráfego gerado ao longo do tempo quando apenas temos tráfego Stream Tv (download) Zoom da figura. 46

Figura 18 – Variação largura de banda ao longo do tempo quando apenas temos tráfego Stream Rádio (download) 46

Figura 19 – Tráfego gerado ao longo do tempo quando temos tráfego Stream Rádio (download) Zoom da figura 16. 47

Figura 20 – Tráfego gerado ao longo do tempo quando apenas temos tráfego RDC (download)..... 47

Figura 21 – Funcionalidade de Port Scan do *subseven*. 48

Figura 22 – Funcionalidade de *Snapshot* em que foi alterado o intervalo entre envio de *snapshots*. 48

Figura 23 – Zoom da figura 21, funcionalidade *subseven* de port scan com *delay* 1 s (canto superior esquerdo), com um *delay* de 2s (canto superior direito), com *delay* de 3 s (canto inferior esquerdo) e com um *delay* de 5 s (canto inferior direito). 49

Figura 24 – Zoom da figura 23, funcionalidade *subseven* de *snapshot* com período 1 S (canto superior esquerdo), com um período de 2 S (canto superior direito), com período de 3 S (canto inferior esquerdo) e com um período de 4 S (canto inferior direito)..... 50

Figura 25– Funcionalidade de *Snapshot* em que foi alterada a qualidade do *snapshot*.. 51

Figura 26 – Zoom da figura acima, funcionalidade *subseven* de *snapshot* alterando a qualidade do *snapshot*, com melhor qualidade (canto superior esquerdo), com metade da qualidade (canto superior direito), com a pior qualidade (canto inferior esquerdo)..... 52

Figura 27– Largura de banda gerada utilizando o *rootkit subseven* com a funcionalidade de transferência de ficheiros. 53

Figura 28 – Zoom da figura acima, funcionalidade *subseven* de transferência de ficheiros, a esquerda temos zoom dos primeiros 1700 s em que foi feito download e a direita temos a parte em que foi feito upload de um ficheiro..... 53

Figura 29 – Utilização da largura de banda quando são utilizadas todas as funcionalidades do *subseven*. 54

Figura 30 – Zoom dos primeiros 500 s em que foram utilizadas todas as funcionalidades do *subseven*, aqui está representada a funcionalidade de *port scan* com um *delay* de 3 S. 55

Figura 31 – Zoom da parte em que foi alterado o intervalo de envio de *snapshots*, a direita com um intervalo de envio de 1 S e a esquerda com um intervalo de envio de 3 S. 55

Figura 32 – Zoom da parte em que foi alterando a qualidade de envio de *snapshots*, a direita *snapshot* com a melhor qualidade e a esquerda com metade da qualidade..... 56

Figura 33 – Zoom da parte em que foi feita a transferência de ficheiros, a esquerda a parte de upload de um ficheiro e a direita a parte de download de um ficheiro..... 56

Figura 34 – Perfil de tráfego do serviço HTTP. 60

Figura 35 – Á esquerda FTP+Streaming Video + port scan e a direita FTP + Streaming Video + snap quality (download). 62

Figura 36 – HTTP + FTP + snap quality (download)..... 65

Figura 37 – HTTP + Streaming Radio + port scan (download). 65

Figura 38 – Skype + HTTP + port scan (download)..... 68

Figura 39 – Skype + Jogos + port scan (download). 68

Índice de Tabelas

Tabela 1 – valores de histórico (Hn) e valores de instantes agregados (Th)..... 36

Tabela 2 – Cenários criados com as aplicações utilizadas e a utilização do *rootkit subseven*. 57

Tabela 3 – Cenários de utilizador criados com as aplicações utilizadas e a utilização do *rootkit subseven*. 58

Tabela 4 – Resultados obtidos com o serviço HTTP. 60

Tabela 5 - Resultados obtidos com o serviço FTP..... 61

Tabela 6 - Resultados obtidos com o serviço Stream TV. Pela tabela 6..... 61

Tabela 7 - Resultados obtidos com o serviço Stream Rádio On-line. 62

Tabela 8 - Resultados obtidos com o serviço Skype..... 62

Tabela 9 - Resultados obtidos com o serviço JOGOS. 63

Tabela 10 - Resultados obtidos com o serviço RDC. 63

Tabela 11 - Resultados obtidos com o Cenário FTP + Stream Vídeo. 63

Tabela 12 - Resultados obtidos com o Cenário FTP + HTTP. 64

Tabela 13 - Resultados obtidos com o Cenário HTTP + Stream Rádio On-Line. 64

Tabela 14 - Resultados obtidos com o Cenário HTTP + Stream Vídeo. 66

Tabela 15 - Resultados obtidos com o Cenário Stream Rádio + RDC. 66

Tabela 16 - Resultados obtidos com o Cenário HTTP + Skype. 66

Tabela 17 - Resultados obtidos com o Cenário Skype + Jogos..... 67

Tabela 18 - Resultados obtidos com o Cenário Skype + Stream Rádio. 68

Siglas

ARP = Address Resolution protocol.
ASCII = American Standard Code information Interchange.
CBIPS = content based Intrusion Prevention system.
DDoS = Distributed Denial Of Service.
DoS = Denial Of Service.
FTP = File Transfer Protocol.
HIDS = Host Intrusion Detection System.
HIPS = Host based Intrusion Prevention system.
HTML = Hyper Text Markup Language.
HTTP = Hypertext Transfer Protocol.
HTTPS = Hypertext Transfer Protocol over Secure Socket Layer.
ICMP = Internet Control Message Protocol.
IDA = Intrusion Detection Agent.
IDS = Intrusion Detection System.
IP = Internet Protocol.
IPS = Intrusion Prevention system.
IRC = Internet Relay Chat.
ISN = Initial Sequencer Number.
ISP = Internet Service Provider.
Mbps = Megabits por Segundo.
NASL = Nessus Attack Scripting Language.
NIDS = Network Intrusion Detection System.
NIPS = Network Intrusion Prevention System.
POP = Post Office Protocol.
PSK = Pre-Shared Key.
P2P = Peer to Peer.
RDC = Remote Desktop Connection.
RBIPS = Rate based Intrusion Prevention System.
SSH = Secure Shell.
TCP = Transmission Control Protocol.
UDP = User Datagram Protocol.
URL = Uniform Resource Locator.
VoIP = Voice Over IP.
XML = eXtensible Markup Language.

1 Introdução

Actualmente, um dos grandes problemas da Internet é a quantidade de tráfego ilícito que é gerado. Entende-se por tráfego ilícito o tráfego que é gerado com o fim de desabilitar ou destruir sistemas, roubar informações e controlar sistemas. Este toma a designação de *malware*. Inicialmente o *malware* (*spyware*, *ad-ware*, entre outros) era constituído por pequenos programas criados para explorar os sistemas [1]. Com o crescimento da Internet, os sistemas tornaram-se mais robustos, conseguem detectar e eliminar o *malware* gerado, esta evolução também verificou-se nos programas que geravam *malware* (*vírus*, *worms*, *trojans*), que tornaram-se também mais robustos e minuciosos, explorando cada vez mais as falhas nos sistemas para atingir o seu fim. O que inicialmente eram pequenos programas para explorar os sistemas, apenas com o intuito de danificar, actualmente são redes organizadas de computadores (*Botnets*), com computadores a trabalhar para empresas, sem que os donos desses computadores tenham conhecimento (*PC Zombie*) [2]. Torna-se assim importante detectar os computadores que foram comprometidos (*PC zombies*). As técnicas tradicionais para detectar e mapear o tráfego que circula na Internet, e que detectam o *malware,* revelam-se cada vez mais ineficazes, uma vez que o tráfego é mascarado, tornando difícil a tarefa de identificar este tráfego ilícito. Os *trojans* também evoluíram, tornaram-se mais "inteligentes", mascaram também o seu tráfego de modo a confundir este com o tráfego normal gerado pelos utilizadores. Os operadores de rede querem reduzir a quantidade de tráfego ilícito a circular pelas suas redes, mas com as técnicas tradicionais torna-se quase impossível detectar e anular o *malware* gerado. As aproximações tradicionais de mapeamento de tráfego (detecção de serviços) tornaram-se imprecisas, porque muitas aplicações usam números de portos, que não são os seus, usam os mais bem conhecidos, associados a outras aplicações, mudando as assinaturas do tráfego ou usando criptografia. A análise de portos é a técnica mais simples para detectar aplicações e utilizadores na rede, e é baseada no simples conceito que muitas aplicações têm portos por defeito, a análise de protocolos monitoriza o tráfego a passar pela rede e inspecciona os dados dos pacotes de acordo com assinaturas previamente conhecidas. As aplicações IP estão constantemente a evoluir e as assinaturas também, o tráfego pode ser encriptado para dificultar a sua análise, onde esta, é baseada em assinaturas que pode afectar a estabilidade da rede, porque é necessário inspeccionar todo o tráfego da rede, e processar este tráfego de modo a reconhecer assinaturas presentes nos pacotes. Análise

sintáctica e semântica dos fluxos evita algumas das desvantagens da análise de portos, e da análise de protocolos. Esta aproximação pode efectuar reconhecimento de protocolos e extrair os dados de cada protocolo, no entanto esta análise pode ser pesada e não é aconselhado quando é necessária confidencialidade.

O presente trabalho de dissertação tem como objectivo efectuar uma análise das técnicas que existem actualmente para identificar PC Zombies e verificar a viabilidade da utilização de redes neuronais para a detecção de tráfego ilícito, simulando cenários criados com aplicações, que são utilizadas por um utilizador comum, e utilizando os dados capturados para identificação do *malware*.

O *paper* "Detection of Illicit Traffic using Neural Networks" [1], apresentado na conferência internacional (SECRYPT) resulta do trabalho feito ao longo desta dissertação.

1.1. Objectivo

O objectivo desta dissertação é o desenvolvimento de mecanismos de detecção de PC *Zombies,* com base no histórico do perfil de tráfego, dos utilizadores de uma rede. Pretende-se desenvolver mecanismos para detecção de PC zombie. Será usada uma nova abordagem, baseada em redes neuronais para detectar aplicações na Internet, baseado nos agregados do tráfego de rede e estimando o nível de tráfego (isto é, a percentagem de bytes ou pacotes correspondente a uma aplicação do total do tráfego agregado) correspondente a cada aplicação. Com base nas redes neuronais existentes e no *toolkit do matlab,* pretende-se usar perfis de tráfego gerado pelos serviços que estão a ser executados num PC, e detectar tráfego Zombie.

1.2. Motivação

O trabalho desenvolvido nesta dissertação foi motivado pela necessidade de novas técnicas para detecção de intrusões, onde há que ter em conta que, em qualquer parte do mundo, existem *BotNets* (redes organizadas de PC comprometidos) a trabalhar para fins ilícitos, sem que os seus donos tenham conhecimento. A necessidade de novas técnicas mais eficazes e mais rápidas, que as técnicas para a detecção de tráfego ilícito existentes actualmente é crucial. O desenvolvimento de novas técnicas para a detecção de PC zombies, não vem substituir as técnicas existentes actualmente, mas sim, trabalhar em conjunto com estas para detectar PC Zombies.

A maneira mais fácil (tradicional) de detectar os serviços, que estão a ser executados numa rede, é verificar quais os portos que constam nos pacotes que passam pela rede. Aplicações que implementam serviços, como por exemplo FTP, usam os portos 20 e 21, aplicações HTTP usam o porto 80, estes portos são portos normalizados e bem conhecidos. Actualmente para fugir a este mecanismo tradicional de identificação de serviços utilizado por *firewalls*, o tráfego é mascarado de modo a não ser bloqueado pelas *firewalls*, a técnica é utilizar portos bem conhecidos, como por exemplo o porto 80 para outro tipo de serviços que não sejam HTTP (*port forwarding*). O tráfego também pode ser identificado pelos padrões presentes no *payload* dos pacotes, que são assinaturas bem conhecidas, presentes e que identificam os serviços que estão a ser executados, no entanto para ultrapassar a identificação do serviço que está a ser utilizado, usa-se a encriptação dos pacotes, tornando a identificação do tráfego com assinaturas muito difícil. Quando todas estas técnicas de identificação e categorização de tráfego falham, como podemos identificar o tráfego que está a ser gerado numa rede?
As aplicações também podem ser identificadas pela quantidade de tráfego que geram, ao longo do tempo e pelo perfil de tráfego gerado. Por exemplo, a partilha de ficheiros gera grande quantidade de tráfego, em que é usada elevada largura de bandaisto é, caracteriza-se por uma grande variabilidade, variabilidade essa que pode ser atribuída ao grande número de Sessões TCP/IP que são abertas/fechadas. O serviço HTTP é caracterizado pela elevada utilização da largura de banda não periódica, com picos de duração muito curta devido aos *clicks*. Por outro lado, o serviço de jogos é caracterizado por picos com duração periódica, (alta frequência) e uma pequena largura de banda usada. As aplicações com requisitos de tempo real, como por exemplo TV on-line, Rádio on-line, Vídeo-Conversa, videoconferência entre outros, são aplicações caracterizadas pela utilização de uma largura de banda média, com pouca tolerância a perdas e atrasos de pacotes. Nestas aplicações é necessária uma largura de banda constante.
As redes neuronais possuem um enorme potencial no reconhecimento de padrões, visto que podem ser utilizadas para detectar os mais diversos tipos de padrões, por exemplo são utilizadas no reconhecimento de sinais, reconhecimento de caracteres em estatística, para calcular previsões. Também podem ser usadas para detectar aplicações, visto que cada aplicação, gera uma quantidade de tráfego específica ao longo do tempo, a este padrão damos o nome de perfil de tráfego. Por exemplo, a nível de aplicações de *Stream*, o tráfego de vídeo e áudio gerado pelo serviço de mensagens instantâneas do

Messenger, será diferente do tráfego de *Stream* gerado pelo serviço oferecido pelo *Skype*. A nível de aplicações P2P podemos verificar que o tráfego gerado pelo Emul será diferente do tráfego gerado pelo limeware. Outro campo onde as redes neuronais poderão ser utilizadas, é na segurança em redes, onde poderá aparecer como uma técnica auxiliar na detecção e prevenção de infecções de *malware* e na detecção e prevenção de ataques. Por exemplo, se uma rede neuronal for treinada para identificar o perfil de tráfego gerado num ataque DoS (*Denial Of Service*), a rede ao identificar esse perfil de tráfego no inicio de um ataque, poderá bloquear os pacotes que estão a ser usados para atacar a rede, sem necessidade de processar o conteúdo dos pacotes. As redes neuronais também podem ser usadas para alertar um administrador de rede, que um PC na sua rede, está comprometido, e/ou alertar o próprio utilizador, se o administrador da rede mantiver um perfil de tráfego típico de cada utilizador, ao introduzir os dados (número de bytes por exemplo) que saem da interface do utilizador numa rede neuronal previamente treinada, ele poderá identificar quando o perfil de tráfego desse utilizador não corresponde ao perfil guardado, e poderá assim, gerar um alerta. É de notar que esta detecção é feita em tempo real, o que por vezes não pode ser feito por técnicas tradicionais tais como sistemas baseados em assinaturas, onde têm de procurar padrões (assinaturas), dentro dos pacotes e por isso têm que verificar o conteúdo destes. As redes neuronais poderão ser usadas em conjunto com outras técnicas tradicionais como por exemplo identificação dos portos que são utilizados, numa primeira identificação dos serviços, que estão a ser utilizados numa rede. Outra técnica que poderá funcionar em conjunto com a técnica de identificação de tráfego com redes neuronais, seria a identificação de aplicações usando assinaturas.

1.3. Metodologias do Trabalho

A primeira tarefa a realizar será procurar um *rootkit* que gere o tráfego com as características necessárias (tráfego ilícito), para que este tráfego possa ser identificado pela rede neuronal.

O passo seguinte será escolher uma rede neuronal que melhor se adapte as necessidades.Para que o tráfego ilícito seja identificado, é necessário que a rede neuronal identifique este tráfego correctamente, e o mais rapidamente possível. Depois de escolher a rede neuronal a usar, é necessário modelar os parâmetros dessa rede neuronal, para obter os melhores resultados possíveis na identificação de cada perfil de

tráfego. Depois de escolhida a rede neuronal e dos seus parâmetros, será feito o treino da rede neuronal, e finalmente a simulação dos diversos perfis de tráfego na rede neuronal. Serão criados vários perfis de tráfego, seleccionando os serviços que os utilizadores usam mais regularmente, como por exemplo, HTTP, FTP, Skype, Streaming TV, Streaming Rádio, Jogos e RDC.

A etapa seguinte será usar os dados capturados, para treinar e posteriormente simular na rede neuronal, sendo obtida à saída da rede neuronal uma probabilidade de acerto, que representa o número de vezes que a rede neuronal consegue identificar correctamente o tráfego que é apresentado à entrada. De modo a obter uma aproximação de um cenário real, serão criados Cenários com perfis de utilizador, um exemplo de um cenário de utilizador poderá ser Tráfego HTTP e FTP. Depois de obter os dados dos perfis de utilizador, estes dados serão introduzidos na rede neuronal.

O passo seguinte será simular os diversos tipos de tráfego mencionados acima, mas agora gerando também tráfego ilícito com o *rootkit. Serão* utilizadas algumas das funcionalidades do *rootkit* escolhido. E finalmente a rede neuronal será testada com os diversos dados recolhidos (número de bytes e número de pacotes, em download e upload).

1.4. Estrutura da dissertação

A presente dissertação está dividida em 5 capítulos. O primeiro capítulo é dedicado a apresentação do problema existente actualmente, onde é apresentada uma pequena introdução do problema. Neste capítulo, é também apresentado o objectivo e a motivação que levaram ao desenvolvimento desta dissertação.

No segundo capítulo é feita uma descrição dos *rootkits* e das *botnets*, quais os elementos que existem numa *botnet*, as funcionalidades que uma *botnet* oferece e o seu modo de operação. Seguidamente são apresentados vários tipos de *malware* e o seu modo de funcionamento. Neste capitulo é também dada um pequena introdução as redes neuronais, quais os elementos constituintes das redes neuronais, seu modo de funcionamento e quais as suas aplicações. Por fim são apresentados os sistemas e tecnologias existentes para detecção e prevenção de intrusões, são apresentadas as potencialidades das redes neuronais e sua utilização. È apresentada uma pequena introdução e descrição dos sistemas, à medida que vão sendo apresentados.

No terceiro capítulo é apresentada a arquitectura utilizada, a configuração dos parâmetros utilizados na rede neuronal e uma descrição do modo de funcionamento da rede neuronal.

No quarto capítulo são descritas as aplicações e o perfil de tráfego gerado, também é feita uma descrição do trojan utilizado (*rootkit*), suas funcionalidades, finalmente é feita uma descrição dos cenários de utilizador criados. Por fim, é apresentado o resultado do trabalho desenvolvido, ou seja, os resultados obtidos à saída da rede neuronal. São apresentados todos os testes realizados e uma explicação para os resultados obtidos.

No quinto e último capítulo serão apresentadas as conclusões bem como as principais dificuldades encontradas durante a realização das experiências. Serão enumeradas quais as vantagens do modelo apresentado e apresentadas algumas ideias de trabalho futuro, bem como alguns aspectos que poderiam ainda ser melhorados na arquitectura actual.

2. Enquadramento do Trabalho

Na secção 1 são dados alguns exemplos de *rootkists* existentes que podem gerar tráfego zombie. Neste capítulo é descrito o modo de operação das *BotNets*, quais os elementos que intervêm na organização e gestão, destas e quais as principais dificuldades encontradas na identificação de *BotNets*. Também é dada a definição de *malware*, quais os tipos de *malware* existentes, e modo de operação típico destes. Na secção 4 é dada um pequena introdução às redes neuronais, como são constituídas, como é que funciona uma rede neuronal, e finalmente são dados alguns exemplos de quais as potenciais utilizações das redes neuronais. Finalmente na secção 5 são dados alguns exemplos funcionais de sistema de detecção e prevenção de intrusões, alguns destes baseados em redes neuronais.

2.1. Rootkits

Um *rootkit* é um conjunto de ferramentas que são usadas para esconder processos a correr; ficheiros ou dados do sistema e do sistema operativo; atacar e infectar outros computadores; roubar informações e interagir em tempo real com o sistema comprometido. Um *rootkit* pode tomar o controlo total de um sistema dando acesso privilegiado a utilizadores não autorizados. Com um *rootkit* podemos aceder a um sistema, onde podemos, executar tarefas; alterar as permissões de acesso; esconder ficheiros; efectuar ataques DoS (*Denial Of Service*) e tomar conta das ligações de rede. Seguidamente temos 3 exemplos de *rootkits* bem conhecidos:

- Subseven – programa *backdoor* que dá acesso não autorizado a outros sistemas Windows 9x, através de uma ligação de rede. Este programa possui 3 ficheiros que são: o programa cliente (subseven.exe), que é a aplicação que vai permitir ao intruso, aceder remotamente ao PC onde está a correr o programa server.exe. O programa servidor (server.exe) permite aceder remotamente ao PC infectado. O terceiro ficheiro é o ficheiro de edição do servidor (editserver.exe) que permite ao intruso configurar o servidor para receber uma mensagem por correio electrónico, a notificar que o PC *zombie* esta activo, também permite que seja estabelecido um canal seguro entre o PC atacado e atacante, proteger o server.exe com *password* para não ser modificado, entre outras funcionalidades.

- Back oriffice – o programa servidor BO2K tem o tamanho de 100K, enquanto o programa cliente tem o tamanho de 500 K. Uma vez instalado no PC da vítima, o servidor dá acesso total ao atacante. Possui diversos *plugins*: BOPeep, controlo remoto completo snap in, *Encryption* encripta os dados enviados entre o BO2K GUI e o servidor. BOSOCK32, mais robusto, usa ICMP em vez de TCP ou UDP. O STCPIO, providencia controlo de fluxo encriptado entre o GUI e o servidor, tornando o tráfego mais difícil de detectar na rede.
- Netbus – Existem dois componentes neste *rootkit*, um servidor e um cliente. O servidor é instalado no computador que vai ser remotamente controlado. O servidor fica a escuta num porto à espera que a aplicação cliente, contacte o servidor. O programa cliente possui uma interface gráfica que permite ao utilizador efectuar um grande número de operações no computador remoto. Algumas das potencialidades oferecidas por este *rootkit* são *capture screen*, *keystroke logging*; executar aplicações; download de ficheiros e reiniciar o sistema operativo.

2.2. BotNets

O alargamento das ligações de *Broadband* levou ao aparecimento de novas ameaças contra os ISP e aos clientes, que os ISP servem. Uma *BotNet* é uma rede de computadores comprometidos, sob o controlo de um único *botmaster*, que possui a capacidade de lançar ataques de *Denial Of Service* (DoS); enviar uma grande quantidade de mensagens de e-mail (SPAM) não solicitadas; infectar outros sistemas vulneráveis com *spyware* de violação de privacidade, entre outras formas de *malware*. *BotNet's* são difíceis de detectar e ainda mais difíceis de parar devido as suas características dinâmicas e adaptativas que lhes permitem ultrapassar os meios de detecção tradicionais. Ao falhar as tecnologias baseadas em portos e em assinaturas, os ISP são forçados a adoptar novas formas de anular esta nova ameaça.

Uma *BotNet* [2] começa quando um indivíduo, que é conhecido como o *botmaster*, faz o download de um programa de *bot*. O *botmaster* pode não actuar sozinho, investigações criminais começaram por ligar *BotNet´s* a redes organizadas de crime [2]. Programas como *AgoBot*, *SGBot* e *IRCBot* estão disponíveis livremente na Internet como "*exploit code*", tornando a criação de *BotNet* simples, onde geralmente, são exploradas as

vulnerabilidades do SO Windows. Estas vulnerabilidades são atractivas, não só devido ao elevado número de vulnerabilidades de segurança, mas também devido ao SO Windows ser o mais popular, o mais usado entre os utilizadores e empresas. Depois de seleccionar o *bot* e a falha a explorar, o *botmaster* tem de usar um ou mais planos de controlo. A técnica mais comum é usar servidores de IRC públicos, para controlar a *BotNet*, apesar de existir outras técnicas disponíveis. O *botmaster* precisa de ter um plano de controlo, para poder executar comandos e receber *feedback* da *BotNet*, usando canais de IRC torna-se simples coordenar actividades através de milhares de máquinas distribuídas, Planos de controlo são frequentemente movidos para evitar a detecção [3]. O *botmaster* tem de agora construir o exército *zombie* que ligará à *BotNet*. Usando a vulnerabilidade escolhida o *botmaster* toma controlo de um grande número de sistemas. Uma vez que uma máquina esteja comprometida, imediatamente começa a escutar no plano de controlo para receber instruções. Durante a fase inicial da *BotNet,* as máquinas são apenas usadas para automaticamente procurarem outras máquinas para infectar.

Um PC *zombie* pode fazer o *scan* de milhares de IP´s por minuto, assim *BotNet´s* podem crescer rapidamente no número de PC *zombies*. Cada sistema comprometido comunica pelo plano de controlo para aguardar instruções futuras, neste ponto um *botmaster* tem um único ponto de controlo sobre um exército de PC ligados por uma ligação *broadband*.

Para além de providenciar um meio conveniente para enviar instruções para o exército, o plano de controlo também permite rapidamente alterar ou apagar código para explorar novas vulnerabilidades, capacidades que são cruciais para qualquer *BotNet* permanecer activa. Existem um grande número de razões para um *botmaster* precisar de actualizar a *BotNet*, pode ser necessário modificar o código *bot* para evitar detecção por dispositivos que aplicam assinaturas a pacotes e fluxos, ou talvez o *botmaster* precise de adicionar funcionalidades ao exército (novos comandos, algoritmos de *scan* optimizados, novos vectores de ataque). Como mencionado anteriormente planos de controlo IRC é um dos meios mais usados pelos *botmaster's* para controlar a *BotNet*, IRC é um plano de controlo muito atractivo visto que existem vários servidores IRC espalhados pela Internet e além do mais, quaisquer comandos que entrem no canal é feito o *broadcast* para todos os *zombies* que se tenham juntado à *BotNet*. Uma vez estabelecido o exército, o *botmaster* pode começar a efectuar ataques, por exemplo, pode ser utilizado para mandar a um *website* a baixo, ou para instalar *spyware*, *spam* ou *trojans* nas máquinas comprometidas, noutro exemplo, o *botmaster* seleccionou comprometer outras

máquinas usando um ataque *buffer overflow* num servidor IIS da Microsoft conhecido como ASN1HTTP. O ataque começa com os *zombies* a receber instruções para executar um *scan* avançado por servidores com a falha ASN1HTTP. Depois de infectar uma maquina, uma *password* é transmitida aos *zombies* para evitar que outras pessoas tomem conta do canal e consequentemente da *BotNet*. Os *zombies* começam por efectuar *scan's* e explorar qualquer sistema vulnerável que encontre. Cada vez que um sistema é comprometido, o endereço IP do sistema comprometido é transmitido pelo canal de controlo IRC de volta ao *botmaster*. O objectivo de um *botmaster* não é conhecido, mas com o aumento dos esquemas de *phishing*[3] é normal que o *botmaster* queira manter controlo sobre servidores bem ligados de modo a efectuar o *scan* a *websites*. È também comum *botmaster's* compilarem listas de sistemas comprometidos, de maneira a venderem ou alugarem parte da rede (para uso em ataques de *Denial Of Service*, redes de *spam*, ou para outras actividades maliciosas). Se uma *BotNet* particular tomar conta de uns 100 sistemas residenciais num servidor POP de um ISP, o poder combinado dos PC pode ser usado em conjunto para lançar um ataque de *Denial Of Service* a um servidor POP externo, causando falha do serviço a vários clientes que utilizam esse servidor. Os clientes frequentemente viram-se para o ISP para suporte técnico, mesmo que estejam a participar no ataque sem saberem. Ataques e um grande volume de *spam* originado de um ISP particular, pode levar a que esse ISP seja percebido como a origem de tráfego malicioso, o que pode levar a que o ISP fique com o seu espaço de endereços bloqueado por outros ISP directamente ou via *blacklists* disponíveis na Internet. Localizar *o botmaster* é um grande desafio. De maneira a escapar à detecção, *zombies* quase sempre ligam-se a um canal de controlo que não é do *botmaster*. Frequentemente estes são servidores públicos de IRC ou servidores que foram desviados para uso da *BotNet*. Adicionado a complexidade da *BotNet* está o facto de o *botmaster,* tipicamente fazer *proxy* da sessão de controlo por um grande número de máquinas comprometidas, que estão distribuídas por um grande número de redes e ISP.
As ligações *proxy's* são mudadas com relativa frequência, para quem tente fazer o *trace* das ligações não consiga e o plano de controlo seja mudado com um único comando emitido. Como consequência destas técnicas de evasão, determinar o *botmaster,* requer uma imensa quantidade de cooperação e coordenação eficiente, através dos ISP e empresas. Detectar e inspeccionar *BotNets* não é um assunto trivial, visto que as *BotNet's* são uma ameaça dinâmica e em constante evolução.

As capacidades de actualizações rápidas permitem ao *botmaster* rapidamente modificar o "*exploit code*" do *zombie*, canal de controlo e comprometer outros sistemas através do canal, sendo os sistemas baseados em assinaturas estáticas ineficazes. Bloqueio de portos apenas previne o uso de simples portos. Assim, por exemplo, bloqueando o porto 4000 vai fazer com que a B*otNet* escolha outro porto a utilizar, além do mais, o uso de portos não *standard* vai fazer com que se use outros portos, o porto HTTP (TCP - 80) ou 443 se for HTTPS e estes portos não podem ser bloqueados. *BotNet's* embora bastantes simples em desenho são ferramentas de ataque eficazes. Elas podem dar grande quantidade de largura de banda a um indivíduo, providenciar cobertura para o *botmaster*, e são facilmente capazes de escapar a métodos baseados em assinaturas estáticos e métodos baseados em portos. Técnicas inteligentes que usam a análise do comportamento oferecem o único meio efectivo para detectar e defender-se da proliferação das *BotNet's*. *Bot's* tal como *worms* comportam-se de uma maneira previsível, com tarefas, como por exemplo, efectuar o *scan* a novos computadores, transmitindo código para as máquinas a infectar e envolvendo-se em ataques. Detectando estas actividades e aplicando uma politica de heurísticas é possível identificar *bot's* e implementar políticas para parar um futuro alargamento da infecção. Desta maneira ISP podem eliminar grandes quantidades de *spam* e parar ataques DoS, enquanto protegem os clientes de invasão de privacidade e consumo dos recursos do sistema atribuído a um *bot*.

2.3. Malware

Tem a designação de *Malware, o software* que é desenhado para danificar sistemas sem conhecimento dos seus donos. Estão incluídos nesta definição *vírus*, *worms*, *backdoors*, *trojans*, *spyware* e *adware* malicioso. Os tipos de *malware* mais conhecidos são os *vírus* e os *worms*, conhecidos pelo modo como se espalham, assim a designação de vírus aplica-se a um programa que está infectado, enquanto um *worm* transmite-se pela rede, infectando outros computadores [1]. Para um programa malicioso atingir o seu objectivo, tem de o fazer sem ser desligado, ou apagado pelo sistema infectado. Por vezes os vírus são disfarçados em algo "desejável e apetecível" (engenharia social), levando os utilizadores a instalar programas, sem saberem o que esses programas realmente fazem, é esta a técnica do cavalo de Tróia (*trojan horse*). Por vezes, na tentativa de apagar um ficheiro infectado pode levar o *worm* a infectar outros ficheiros

no sistema, tornando a sua eliminação ainda mais difícil. Assim, um cavalo de Tróia é basicamente, um programa que convida o utilizador a instalar e executar, mas que concede permissões a terceiros. A execução do código pode tomar efeito logo de imediato, ou pode esperar que o seu criador o contacte através de outro computador. Pode também criar, apagar e alterar ficheiros. No fundo, toma conta do sistema e das permissões para aceder a este, poderá também espalhar e instalar outro tipo de *malware [5]*. A maneira mais comum do *spyware* se espalhar é por cavalos de Tróia, aliciando o utilizador a fazer o download de um programa da Web. Quando o programa é instalado, o *spyware* também é instalado.

Programas como *rootkits* permitem a quem os controla, obter acesso a todos os recursos do sistema onde foi instalado, modificando o sistema operativo para que o *malware* seja escondido do utilizador. *Rootkits* podem esconder um processo malicioso para que não seja mostrado na tabela de processos, e podem esconder ficheiros para que não sejam lidos [5]. *Backdoor* é o termo normalmente utilizado para programas que evitam procedimentos de autenticação para aceder a um sistema, alguns fabricantes instalam *backdoors* para poderem aceder ao sistema para suporte técnico. *Crackers,* são geralmente *backdoors* usados para oferecer acesso remoto seguro a computadores, enquanto se mantêm escondidos de uma inspecção.

Para instalar *crackers* e *backdoors* geralmente usam-se *worms* ou *trojans*. Inicialmente o *malware* era criado como forma de vandalismo, actualmente é a forma de *malware* que mais custos acarretam em termos de tempo e dinheiro, é conhecida como *spyware*. *Spyware, são* programas comercialmente produzidos para obter informação sobre os utilizadores, mostrando *pop-ups*, alterando os *web-browsers* para que o seu criador obtenha lucro. *Spywar*e é muitas vezes instalado por trojans.

Também é possível ao criador do *malware* roubar da pessoa que possui o computador infectado, alguns programas de *malware* instalam *key loggers* que copiam as *password's*, números de cartão de crédito e outras informações importantes para o seu criador, que depois as usa para roubar a pessoa que possui o computador infectado.

2.4. Redes Neuronais

O principal objectivo deste estudo é detectar tráfego ilícito gerado por PC *zombies*, usando para o efeito redes neuronais.

As redes neuronais são compostas por elementos que operam em paralelo, estes elementos foram inspirados pelos sistemas nervosos neuronais, que tal como na natureza, a função da rede é determinada pelas ligações entre os elementos. Uma rede neuronal pode ser treinada para efectuar uma função particular, ajustando os valores (pesos) das ligações entre os elementos para, reconhecer determinado padrão. Redes neuronais são ajustadas ou treinadas para uma entrada específica, dê origem a uma saída específica. A rede é ajustada baseada na comparação entre a saída e o objectivo (*target*), até que a saída coincida com o objectivo. Tipicamente muitas entradas/objectivos são precisos para treinar a rede.

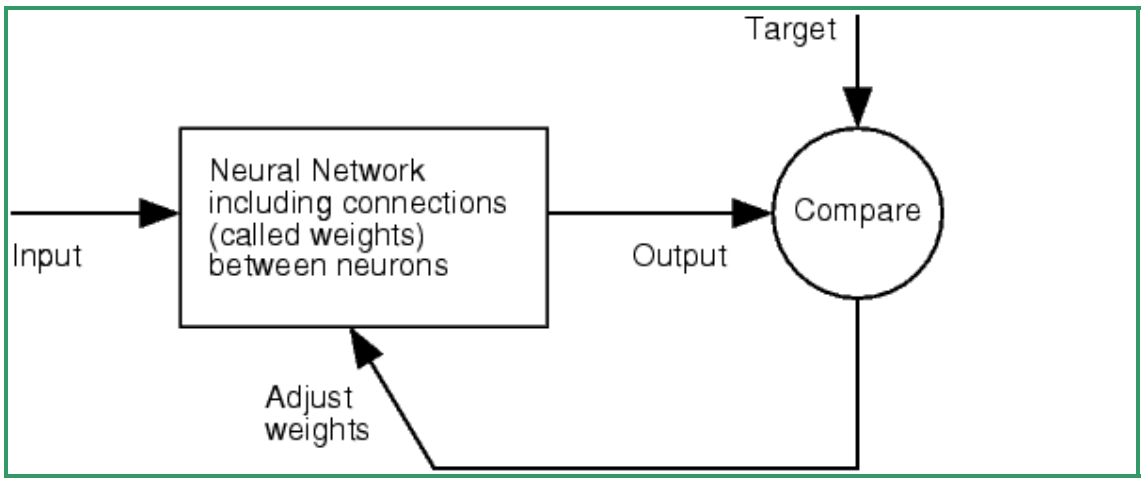

Figura 1 – Modelo Lógico de uma rede neuronal, figura retirada do matlab [16].

As redes neuronais têm sido treinadas para executar funções complexas em vários campos, incluindo reconhecimento de padrões, identificação, classificação do discurso e sistemas de controlo. As redes neuronais podem ser classificadas como estáticas ou dinâmicas. Redes Estáticas (*feedforward*), não têm elementos de feedback e não contém atrasos. A saída é calculada a partir da entrada através de ligações *feedforward*, isto é, as ligações não formam um ciclo directo. Em redes neuronais dinâmicas, a saída depende não só das entradas actuais, mas também das entradas ou saídas da rede anteriores. Redes dinâmicas podem ser divididas em dois grupos, as que tem ligações *feedforward* e as que tem ligações *feedback* ou ligações recorrentes. Um neurónio com um vector de n elementos p1, p2, p3, ... pn é multiplicado pelos pesos W1,1, W1,2, ..., W1,R, os valores dos pesos são adicionados a junção. A sua soma é simplesmente Wp. A soma é o produto de uma (única coluna) da matriz W e do vector p.

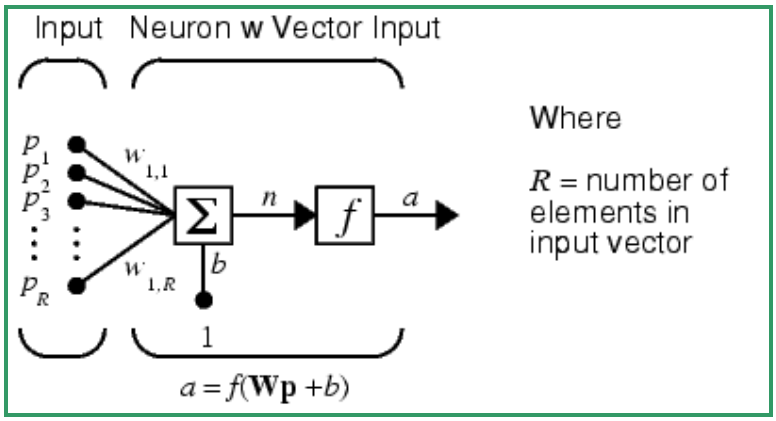

Figura 2 – Modelo de um neurónio, figura retirada do matlab [16].

O neurónio tem uma polarização b, que é somada aos pesos das entradas para formar a entrada de rede n. A soma n, é o argumento de entrada da função de transferência f.

A = f (wp+b)

n = w1,1p1 + w1,2p2 + ... + w1,RpR + b

Uma camada de rede com R entradas e S neurónios.

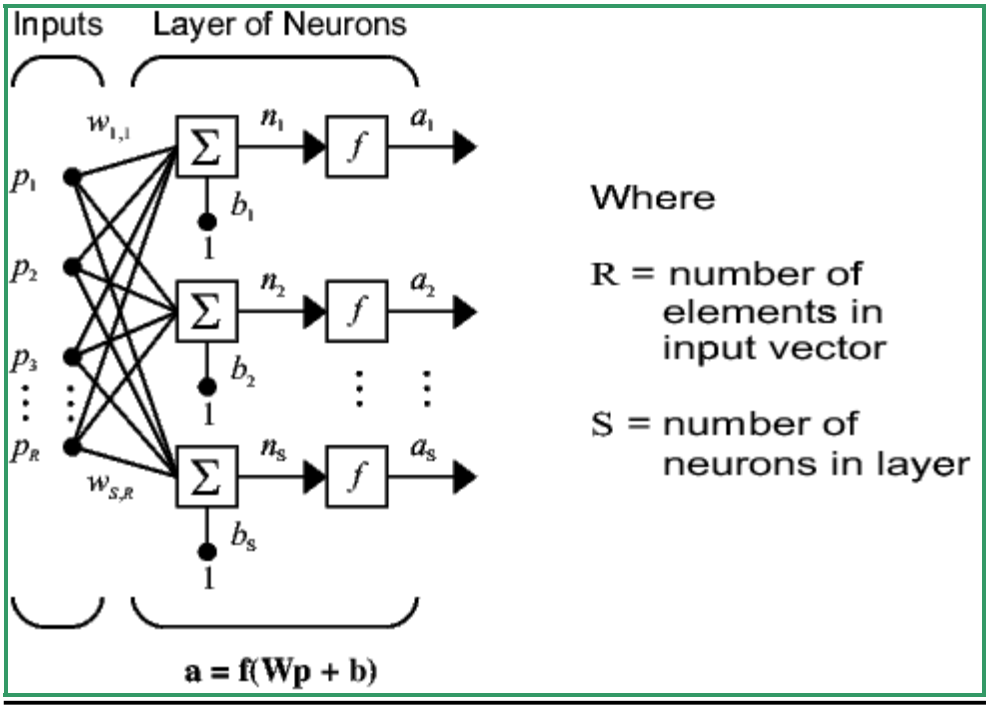

Figura 3 – Modelo de uma camada com vários neurónios, figura retirada do matlab [16].

Na rede cada elemento do vector de entrada p é ligado a cada neurónio através da matriz de pesos W. O neurónio i reúne a soma dos pesos das entradas e polarização para formar a sua própria saída escalar n (i). As varias entradas n(i) formam um elemento S que é o vector de entrada n. Finalmente a camada de saída dos neurónios formam o vector a. Uma camada pode ter o número de entradas diferente do número de neurónios.

Cada camada tem uma matriz de pesos W, um vector de polarização b, e um vector de saída a.

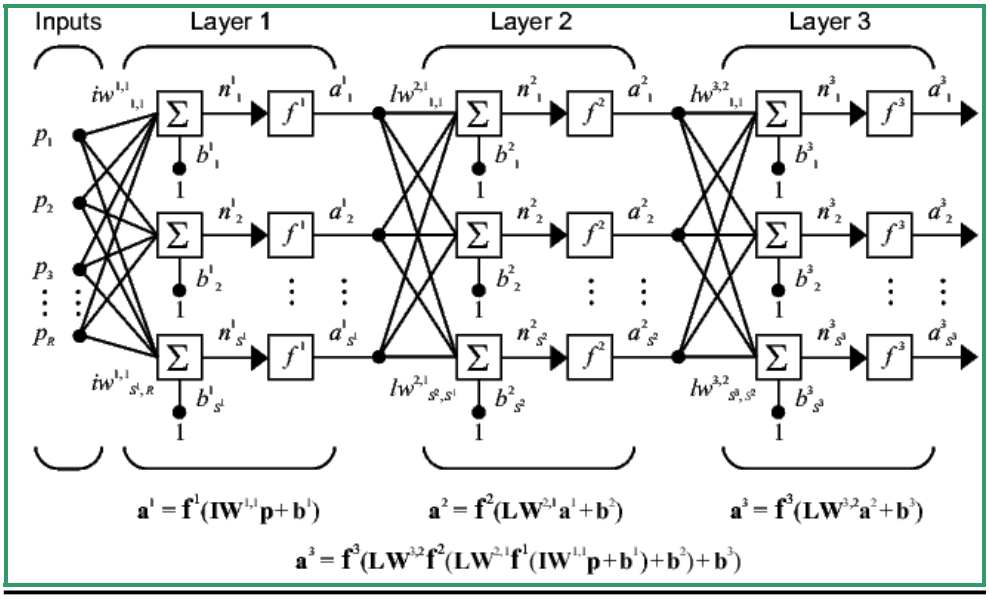

Figura 4 – Modelo de uma rede neuronal com várias camadas, figura retirada do matlab [16].

A rede neuronal apresentada na figura 4 tem R1 entradas, S1 neurónios na camada 1, S2 entradas na camada 2 que corresponde ao número de elementos do vector a. Assim, a saída de uma camada é a entrada da próxima, a saída da camada 1 é a entrada da camada 2 e a saída da camada 2 é a entrada da camada 3. Uma camada que produz uma saída é chamada camada de saída (*output layer*), na figura representada por *layer* 3, a camada do meio é designada camada escondida (*hidden layer*) e a camada inicial é a camada de entrada (*input layer*). Para treinar e simular a nossa rede neuronal, foi utilizado o *toolkit* de redes neuronais do *matlab*. Uma função de transferência calcula a saída da camada a partir da entrada da rede, o *matlab* possui as seguintes funções de transferência:

- Hard-limit – limita a saída do neurónio a zero (0), se os argumentos de entrada da rede forem menores que zero, ou 1 se os argumentos de entrada da rede forem maiores que 1. Usada em perceptrons.

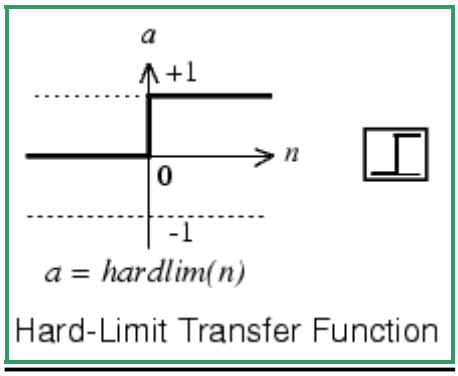

Figura 5 – Função de transferência hardlim.

- Purelin – Usada em filtros.

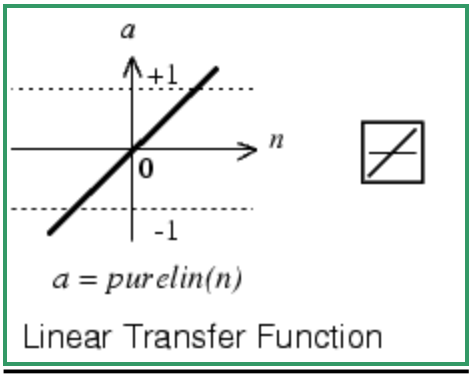

Figura 6 – Função de transferência purelim.

- Log-sigmoid – recebe uma entrada que pode ter qualquer valor entre +/- infinito e distribui a saída entre zero e um. Muito usada em redes de *backpropagation*.

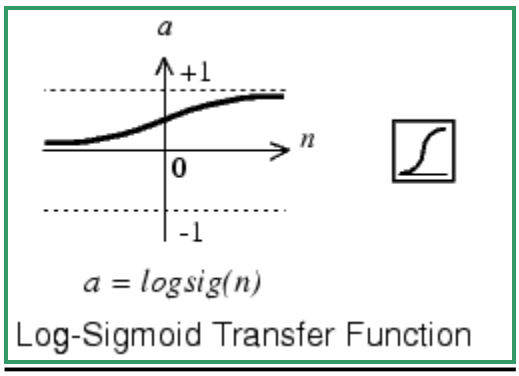

Figura 7 – Função de transferência log-sig.

2.5. Aplicações para detecção e Prevenção de Intrusões

Antes de iniciar o desenvolvimento de uma aplicação destinada a identificar, convém verificar alguns exemplos funcionais já existentes. Neste momento existem alguns sistemas que identificam intrusões e outros que previnem Intrusões. Também existem

algumas arquitecturas que identificam Intrusões baseadas em redes neuronais ainda em fase de desenvolvimento. *Intrusion detection system* (IDS) são programas que detectam computadores comprometidos e que estão a ser controlados através da Internet. Um IDS é usado para detectar diversos tipos de tráfego malicioso, que não pode ser detectado por um firewall convencional. Esta detecção inclui ataques de rede contra serviços vulneráveis, ataques direccionados a aplicações, logins não autorizados, acesso a ficheiros e distribuição de *malware*.

Um IDS é composto por vários componentes:

- Sensores: que geram eventos de segurança.
- Uma consola: para monitorizar eventos, alertas e controlar os sensores.
- Um motor central: que grava os eventos gerados pelos sensores numa base de dados e usa as regras do sistema para gerar alertas de segurança dos eventos recebidos.

Tipos de IDS:

- *Network intrusion detection system* é uma plataforma de rede independente que identifica intrusões examinando o tráfego de rede, monitorizando múltiplos computadores. O sistema ganha acesso ao tráfego ligando-se a um *Hub* ou *Switch* configurado para *port mirroring*, ou *network tap*. Um exemplo de um NIDS é o *Snort*.

- *Protocol-Based intrusion detection system* é composto por um sistema ou agente, que tipicamente fica a frente de um servidor, monitorizando e analisando os protocolos de comunicação, entre um dispositivo ligado e o servidor.

- *Application protocol-based intrusion detection system* é um sistema ou agente que geralmente fica a frente de um grupo de servidores monitorizando e analisando a comunicação de protocolos específicos da aplicação.

- *Host Based intrusion detection system* é composto de um agente num PC, que detecta intrusões analisando as *system calls* do sistema, *logs* de aplicação, modificações no sistema de ficheiros (binários, ficheiros de passwords) entre outras actividades.

- Um *Hybrid intrusion detection system* combina uma ou mais das aproximações definidas. Por exemplo o Snort pode funcionar com NIDS e HIDS, outro exemplo é o Prelude.

Os IDS podem ser classificados como passivos ou reactivos. Num IDS passivo, o sistema detecta uma possível falha de segurança, faz o *log* do evento e gera um alerta para a consola. Num IDS reactivo o sistema responde com a uma actividade suspeita desligando a ligação. Um IDS analisa as ligações, analisa o tráfego à procura de intrusões e impedindo-as de acontecerem. Um firewall limita o acesso entre redes, para prevenir a intrusão de fora de uma rede para dentro, mas não previne um ataque de dentro para fora da rede. Os IDS também podem ser categorizados em duas subcategorias:

- *Signature-Based*.
- *Anomaly detection system*.

NIDS – *Network* IDS são IDS que capturam pacotes que passam na rede e comparam-nos com assinaturas guardadas numa base de dados. HIDS – HOST IDS são instalados como agentes num PC. Assinatura (*signature*) – É um padrão que obtemos ao analisar um pacote. A assinatura é usada para detectar um ou mais tipos de ataques. A assinatura pode estar presente em diferentes partes do pacote dependendo da natureza do ataque (IP, TCP, UDP…). *Anomaly detection systems* são sistemas usados para detectar intrusões monitorizando o sistema e classificando o tráfego como normal ou anómalo. A classificação é baseada em heurísticas ou regras, identificando melhor que padrões ou assinaturas que estão presentes, e detectando qualquer abuso que se classifique fora do normal funcionamento do sistema, isto é oposto dos sistemas baseados em assinaturas que só podem detectar ataques em que a assinatura foi previamente criada. De maneira a detectar o que é um ataque, o sistema tem que ser ensinado a reconhecer o funcionamento normal do sistema, isto pode ser conseguido de diferentes maneiras, uma possibilidade será o uso de inteligência artificial. Outro método é definir qual o uso normal do sistema usando um modelo matemático e assinalar qualquer desvio deste modelo como sendo um ataque, isto é conhecido como *strict anomaly detection*.

IPS é o nome do software de sistemas de prevenção (*Intrusion prevension system*), é um software que faz controlo dos computadores de modo a proteger da exploração das falhas ao nível do software que os computadores possuem. IPS é considerado por alguns uma extensão da tecnologia IDS, mas é na realidade outra forma de controlo de acesso, como por exemplo um *firewall*. Existem vários tipos de IPS que são os seguintes:

- *Host Based* – HIPS é onde o sistema de prevenção existe, num endereço IP específico.

- *Network* – É onde o software/hardware IPS ou qualquer acção tomam lugar para evitar a intrusão num PC especifico a partir de outro PC. NIPS são construídos de propósito para analisar, detectar e alertar para possíveis intrusões, NIPS analisam o tráfego com base na política de segurança.
- *Content based* – CBIPS inspecciona o tráfego de rede para sequências únicas, chamadas assinaturas, para evitar ataques conhecidos, como por exemplo infecção de *worms*.
- *Protocol analysis* – Este IPS pode descodificar aplicações de rede como por exemplo HTTP, FTP. Uma vez que os protocolos estejam todos descodificados o IPS pode procurar por comportamentos anómalos e exploração de falhas no protocolo.
- *Rate based* – RBIPS foi inicialmente desenvolvido para evitar ataques Denial Of Service (DoS) e Distributed Denial Of Service (DDoS). Trabalham monitorizando e aprendendo comportamentos de rede normais. Através da analise de tráfego em tempo real, e comparando com as estatísticas armazenadas, pode identificar comportamento anormal de diversos tipos de tráfego como por exemplo pacotes TCP, UDP ou pacotes ARP, ligações por segundo, pacotes por segundo. Os ataques são detectados quando os limites são excedidos. Os limites são dinamicamente ajustados baseados nos diversos parâmetros (estatísticas).

O Snort [12] é um IDS, e é também um sistema de prevenção capaz de efectuar o *log* de pacotes e análise do tráfego em tempo real nas redes IP. O Snort utiliza uma linguagem baseada em regras, que combina os benefícios da inspecção de pacotes, baseado em assinaturas, protocolos e anomalias. Este software efectua análise de protocolos, procura de conteúdos e é muito usado para bloquear ataques e/ou passivamente detectar uma grande variedade de ataques ou sondas a um sistema. Alguns ataques conhecidos são detectados pelo snort, como por exemplo: *Port scans*, ataques de *buffer overflow*, ataques em aplicações Web e detecção de Sistemas Operativos. O snort é principalmente conhecido por parar os ataques a medida que vão acontecendo, isto é, funciona como um *Intrusion detection system*. Uma característica relevante é a capacidade de gerar alertas em tempo real, que incorpora mecanismos de alerta para o *syslog*, para arquivo, para socket UNIX ou, com auxílio do SAMBA (smbclient), para o WinPopup messages em máquinas Windows clientes. O Snort pode ser usado como um

sniffer de pacote directo, da mesma forma que o *tcpdump*. Também, pode ser usado como um sistema de detecção de intrusões de rede completamente integrado. O registo dos pacotes pode ocorrer no formato binário do *tcpdump* ou no formato Snort ASCII descodificado, em directórios de registo nomeados com base no endereço IP do *host* externo. Snort também é baseado no pacote de bibliotecas *libpcap* para capturar tráfego de rede de baixo nível. É uma ferramenta de domínio público, robusta e portável. Pode ser encontrada na sua *web-page* [12], onde são disponibilizados o código fonte para algumas plataformas, e os próprios binários. É possível encontrar também, neste mesmo endereço, arquivos de regras para identificar os mais actuais *exploits*. Por ser uma ferramenta leve, a utilização do Snort é indicada para monitorar redes TCP/IP pequenas, onde pode detectar uma grande variedade do tráfego suspeito, assim como ataques externos e então, fornece argumentos para as decisões dos administradores. Os chamados IDSs peso leve, são pequenos, poderosos e suficientemente flexíveis, tornaram-se elementos permanentes da infra-estrutura de segurança da rede. Ao contrário do *tcpdump*, a principal característica do Snort é inspecção de conteúdo do pacote. Para detectar actividade hostil, o Snort descodifica a camada de aplicação do pacote. Então, através de regras pode-se agrupar o tráfego de dados específicos, como por exemplo, assinaturas digitais contidas nessa camada.

O Ossec [13] é um *Host-Based Intrusion Detection System open source* que realiza operações de Analise de *Logs*, Integridade de Sistemas, Detecção de *Rootkits*, gera alertas e respostas activa (regras no firewall). Tem uma plataforma centralizada, que permite monitorizar e gerir múltiplos sistemas. Possui um motor de análise de *logs* muito potente, sendo possível analisar *logs* de vários sistemas em diferentes formatos. O Ossec também se baseia em regras, regras estas descritas na linguagem XML, também possui alertas de tempo real. Possui um agente integrado para detecção de *Rootkits*, assinaturas geridas centralmente, uma base de dados de ficheiros na qual podem ser procurados ficheiros de *rootkits* usando *stats*. Possui também uma base de dados de *trojans*, com binários de ficheiros usados por *trojans*. Permite analisar anomalias verificando problemas de permissões, ficheiros escondidos, consegue verificar processos escondidos, portos activos, entre outros. O Ossec pode ser instalado em três modos: servidor, agente e local. Possui dois tipos de ligações: *syslog* e *secure*, para

máquinas não monitorizadas, as ligações são feitas via *syslog*, para máquinas em que existe monitorização é feito com comunicação segura (PSK).

Cfengine [14] é um software de detecção de anomalias desenvolvido na universidade de Oslo, o objectivo deste projecto é desenvolver um sistema imune, baseado na detecção de anomalias e no comportamento do sistema, este projecto esta no seu estado inicial. Cfengine é uma ferramenta para criar e manter sistemas. Esta ferramenta consiste em vários componentes:

- Cfagent – agente de configuração autónoma.
- Cfservd – um servidor de ficheiros e serviço de activação remota.
- Cfexecd - serviço de *scheduling* e de relatório.
- Cfenvd – Serviço de detecção de anomalias.
- Cfrun – ferramenta para activar cfagent remotamente.
- Cfshow - ferramenta de análise de conteúdos de base de dados.
- Cfkey – ferramenta de geração de chaves.

O Cfengine incorpora uma linguagem declarativa, de mais alto nível que o perl ou shell. Uma única declaração pode resultar em centenas de operações a serem executadas. O principal objectivo do cfengine é criar um sistema central de configuração, que irá definir como cada computador na rede deve ser configurado.

Nessus [15] é um programa de verificação de falhas/vulnerabilidades de segurança. É composto por um cliente e servidor, sendo que o *scan* propriamente dito feito pelo servidor. O nessusd (servidor Nessus) faz um *port scan* ao computador alvo, depois disso, vários scripts (escritos em NASL, *Nessus Attack Scripting Language*) ligam-se a cada porta aberta para verificar problemas de segurança. Uma das principais características do Nessus é a sua tecnologia Cliente/Servidor, onde os servidores podem ser alocados em pontos estratégicos da rede, permitindo testes de vários pontos diferentes. Um cliente central ou múltiplos clientes podem controlar todos os servidores.

O nmap [16] é uma ferramenta *Open Source* que detecta serviços e sistemas operativos. Detecta portos TCP e/ou UDP através de ligação normal, meia ligação, ligação camuflada e UDP ICMP *port unreacheable*. Os sistemas operativos são identificados através dos graus de liberdade dados aos protocolos, cada sistema operativo usa-os de forma diferente, por exemplo o valor do ISN (pode ser constante, incrementos fixos, incrementos temporais), opções TCP (teste de opções suportadas, teste de opções retornadas, teste da ordem das opções retornadas), FIN *probe* (envio de FIN para um porto aberto, o sistema poderá não responder ou fazer RESET). Os serviços usam portos

fora do comum que não são registados oficialmente (para testes ou despiste de ataques automatizados), registados em nome de outros serviços (telnet sobre *sshd* para passar firewall) e sobrepostos (multiplexagem *SSH*). A identificação requer que o nmap dialogue com o máquina para recolher características dos serviços a correr. A nível da utilização de inteligência artificial para detecção de intrusões existem algumas arquitecturas que estão em fase de desenvolvimento, mas ainda não existe uma ferramenta que tenha emergido.

O instituto de tecnologia de *New Jersey* da Universidade Heights (*Newark*), está a desenvolver uma nova arquitectura de detecção de intrusões baseada em redes neuronais. Esta nova arquitectura tem a designação "Hierarchical Network Intrusion Detection system" [17]. O sistema proposto é um sistema hierárquico distribuído, que consiste em várias camadas, cada camada contém um agente detector de intrusões (IDA). IDAs são componentes que monitorizam a actividade de um *host,* ou de uma rede. Diferentes camadas correspondem a diferentes alcances da rede, que são protegidos por agentes associados a essas camadas.

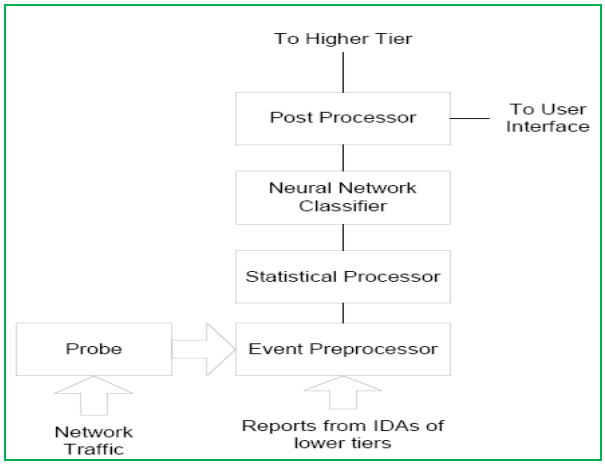

Figura 8 – Intrusion detection agent (IDA), figura retirada do paper que apresenta o sistema [17].

Na figura 8 está representado um agente da arquitectura proposta. È constituído por uma *probe* – recolhe o tráfego de uma rede ou *host*, abstrai esse tráfego para um conjunto de variáveis estatísticas, *Event Processor* – recebe eventos da *probe* e de camadas mais baixas, *Statistical processor* – mantém um modelo de referência com as actividades da rede mais comuns, *Neural Network Classifier* – analisa o vector estatístico gerado pelo *statistical processor* e por um *post processor* que gera relatórios para os agentes das camadas mais altas.

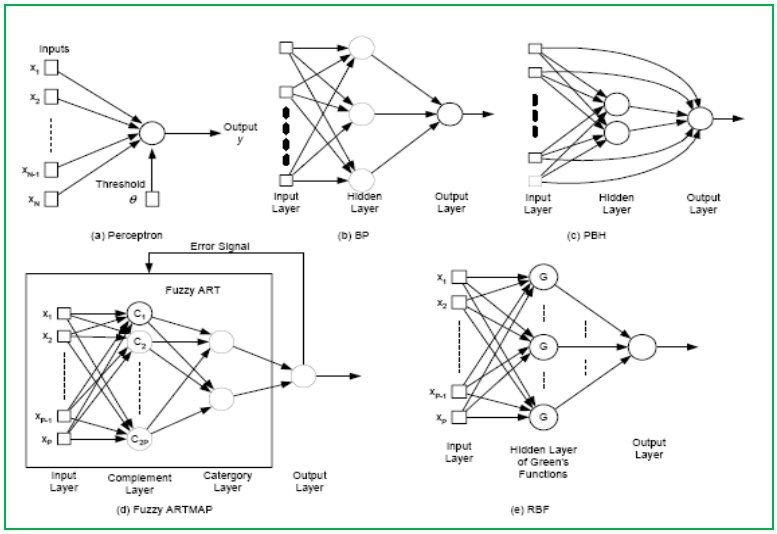

Figura 9 – Arquitectura Rede neuronal proposta, figura retirada do paper que apresenta o sistema [17].

Esta arquitectura ainda está em fase de desenvolvimento. Os resultados apresentados por esta equipa de desenvolvimento são bons, com erros inferiores a 50% na identificação de Intrusões. Esta arquitectura pode detectar ataques de *flooding* UDP. Existe uma proposta de um IDS que foi apresentada por Mehdi Moradi e Mohammad Zulkernine [18]. Esta arquitectura diferencia os ataques em 4 categorias: Ataques de *denial of service*, Ataques em que o utilizador ganha permissões de *Root*, ataques com acesso remoto, e ataques em que o sistema é analisado para descobrir possíveis falhas no sistema (*probe*). No *paper* apresentado foram usados dois ataques como conjunto de dados de entrada, *SYN Flood (Neptune)* e *Satan.* Estes dois ataques foram seleccionados de diferentes categorias (*denial of service e probing*), para verificar a facilidade do sistema em identificar ataques de diferentes categorias. A rede neuronal proposta para a classificação de tráfego utiliza uma camada de saída com 3 neurónios (estados de saída), o vector [1 0 0] é um estado normal, para o ataque Neptuno o estado é [0 1 0] e para o ataque *satan* [0 0 1]. A rede neuronal apresentada possui 3 camadas, mais uma camada a entrada que não é contada porque actua como um buffer, logo não é feito nenhum processamento nesta camada. Para a simulação este grupo de investigação utilizou a *toolbox* de redes neuronais do *matlab*. Nesta proposta à saída da rede neuronal existem três tipos de saídas, falso positivo, falso negativo e saídas irrelevantes. As saídas irrelevantes são as saídas que não representam nenhuma das classes de saída do conjunto. Uma análise feita por este grupo demonstra que, uma rede neuronal com 3 camadas consegue uma classificação de 90,9%, e mais de metade dos resultados incorrectos são da categoria de resultados irrelevantes. Este projecto ainda esta numa

fase inicial e o seu objectivo era verificar a viabilidade de construir um classificador para um IDS baseado em redes neuronais.

3. Sistema Proposto

Neste capítulo é apresentada a arquitectura utilizada para identificação de PC zombies. São apresentados também as configurações da rede neuronal e as razões que levaram a essa configuração.

3.1. Análise de Requisitos

Nesta secção serão apresentados os requisitos necessários que o sistema deverá apresentar de modo a efectuar as necessidades do sistema.

Para gerar o tráfego ilícito (malware) será necessário o uso de um rootkit para gerar este tráfego.

De entre os rootkits apresentados (subseven, netbus e backorifice) o escolhido foi o subseven, porque foi o rootkit que apresentou as melhores funcionalidades. Assim, é necessário que seja gerado uma quantidade de tráfego para que a rede neuronal consiga identificar este tráfego ilícito, depois de testar os vários rootkits apresentados, verificou-se que o subseven é o rootkit que deverá ser utilizado porque é o rootkit que apresenta maior número de funcionalidades a usar na geração de tráfego ilícito.

Das funcionalidades oferecidas pelo subseven serão utilizadas 3 funcionalidades, port scan, transferência de ficheiros e captura de snapshots do desktop comprometido.

No nosso caso utilizamos apenas três das funcionalidades oferecidas por este *rootkit*, para ser possível observar o tráfego gerado por este *rootki*t. Foi utilizada a transferência de ficheiros, o serviço em que o *rootkit* envia *snapshots* periódicos do *desktop* comprometido, permitindo visualizar todos os movimentos do utilizador e por último utilizamos o serviço de *port scan*, que pode ser efectuado remotamente no computador comprometido. Foram definidas 5 tipos de simulações que vão ser feitas para todos os cenários definidos, que são as seguintes:

1. *Port Scan* – È Utilizada a funcionalidade de *port scan* do *rootkit subseven*. Cada simulação demorou 3600 segundos. Assim, foi feito um *port scan* a rede do PC comprometido ao intervalo de portos 10-65535. Dos 0-900 s, com um *delay* de 1 s, 900-1800 s com um *delay* de 2 s, 1800-2700s com um *delay* de 3 s e dos 2700-3600 s com um *delay* de 5 s.
2. *Snapshot Interval* – Nesta simulação foi utilizada a funcionalidade de captura de *snapshots* da máquina comprometida. Nesta experiência o que se pretende simular é o efeito do intervalo de envio de *snapshots* em diferentes intervalos de

tempo. Assim, 0-900 s o intervalo de envio de *snapshots* é de 1 s, 900-1800 s o intervalo é de 2 s, 1800-2700 s o intervalo é de 3 s e de 2700-3600 s o intervalo é de 4 s.

3. *Snapshot quality* – Utilizando a funcionalidade de *snapshot* foi simulado o envio periódico de *snapshots*, aqui foi mantido o período de envio e foi alterada a qualidade dos *snapshots* enviados. Dos 0-600 s, foram enviados *snapshots* com a melhor qualidade, 600-1200 s foram enviados *snapshots* com metade da qualidade, 1200-1800 s com a pior qualidade, 1800-2400 s com a melhor qualidade, 2400-3000 s com metade da qualidade, 3000-3600 s com a pior qualidade.
4. *File transfer* – Foi utilizada a funcionalidade de transferência de ficheiros para download e upload de um ficheiro para a máquina comprometida. Assim, 0-1800 s foi feito upload de ficheiros da máquina comprometida, 1800-3600 s download de ficheiros para a máquina comprometida.
5. *Subseven all* – Nesta simulação foram utilizadas todas as funcionalidades do *rootkit* sequencialmente. Dos 0-900 s foi utilizada a funcionalidade de *port scan* com um *delay* de 2 s, 900-1380 s *snapshot* na melhor qualidade com intervalo de 1 s, 1380 -1800 s *snapshot* na melhor qualidade com intervalo de 3 s, 1800-2280 s *snapshot* na melhor qualidade com intervalo 1 s, 2280-2700 s *snapshot* na qualidade media, 2700-3180 s download de ficheiros para à máquina comprometida e dos 3180-3600 s upload de ficheiros da máquina comprometida.

Para identificação do tráfego ilícito serão utilizadas redes neuronais.

Como dimensionar e parametrizar o sistema de detecção?

1. Número de entradas de rede neuronal = número de entradas do problema.
2. Número de saídas do neurónio de saída = número de saídas do problema.
3. A escolha da Função de transferência da Camada de saída (*layer*) é determinada pelas especificações do problema.

Assim, numa rede neuronal à entrada são apresentados os dados com o padrão que queremos identificar e à saída obtemos a identificação desse padrão.

A tecnologia utilizada neste sistema será o toolkit de redes neuronais do matlab, visto que apresenta uma vasta gama de redes neuronais para teste e parametrização de modo a optimizar os resultados obtidos.

As experiências foram feitas usando a rede *wireless* da universidade de Aveiro, utilizando uma placa com a capacidade *downstream/upstream* de 54 Mbps. Para estas

experiências foi escolhido um período de medição do tráfego de 1 hora (3600 segundos). Foi capturado todo o tráfego gerado utilizando o *wireshark* (sniffer) e o *Trafana* que é um programa escrito em C que utiliza *winpcap* para capturar pacotes. Este programa (*Trafana*) gera um output de 4 colunas em que a primeira coluna corresponde número de pacotes que saíram da interface, a segunda ao número de bytes que saíram pela interface, a terceira ao número de pacotes que entraram pela interface e a quarta coluna corresponde ao número de bytes que entraram na interface.

Para não comprometer nenhum PC foi utilizado o VMWare *workstation*. Foram criadas e configuradas duas máquinas virtuais, uma que servia de PC *Zombie*, isto é, simulava o PC comprometido e outra que simulava o atacante. Ambas as máquinas virtuais tiveram acesso a internet.

3.2. Arquitectura Utilizada

A utilização das redes neuronais para identificação de PC Zombies, através do tráfego gerado por estes, não deverá ser utilizada isoladamente, mas em conjunto com outras técnicas de detecção de intrusões já existentes. Assim, arquitectura final de detecção de intrusões, terá ao seu dispor várias técnicas de detecção de intrusões em que a detecção de PC zombie utilizando redes neuronais, faz parte desse leque de opções. O estudo das outras técnicas de detecção de intrusões sai fora do âmbito desta dissertação, sendo aqui apenas abordada a perspectiva de detecção de intrusões utilizando redes neuronais.

Na rede neuronal desta arquitectura à entrada é apresentado o perfil de tráfego, que são os dados capturados durante a simulação do cenário criado e à saída obtemos a identificação do tipo de tráfego, que neste caso apenas identifica se o tráfego apresentado a entrada é lícito (sem tráfego zombie), ou ilícito (com tráfego zombie). Na figura 10 está esquematizado sistema utilizado para a detecção de tráfego ilícito.

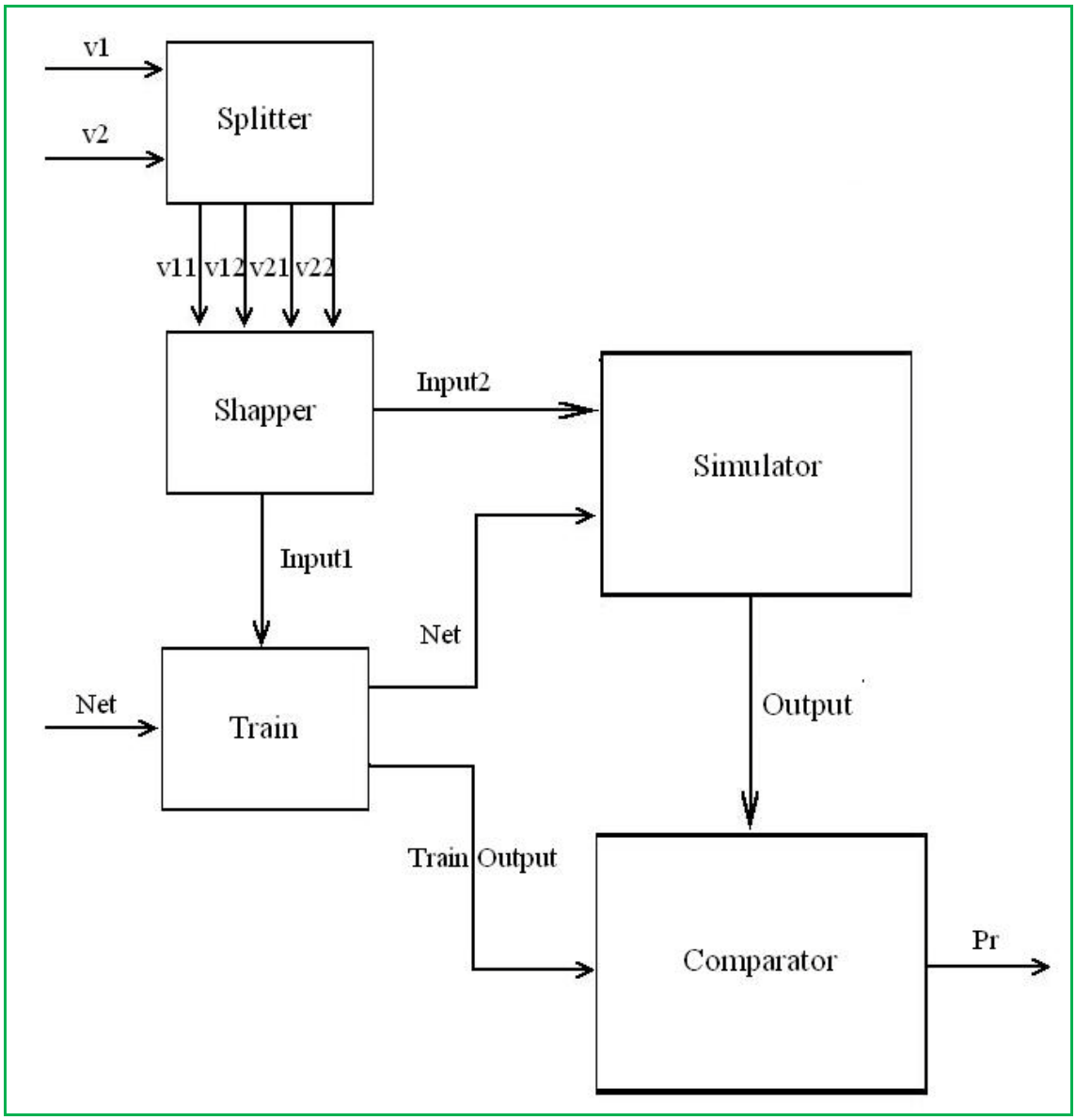

Figura 10 – Sistema proposto para detecção de PC Zombies.

O primeiro módulo que estes vectores encontram tem a designação de *Splitter,* o que este módulo faz, é dividir os dados de cada vector em dois vectores, por exemplo o vector v1 dá origem aos vectores v11 e v12, e o vector v2 dá origem aos vectores v21 e v22. È necessário dividir os vectores de entrada em dois conjuntos porque metade dos dados serão usados para treino da rede neuronal e a outra metade será usada para simulação da rede neuronal, estes quatro vectores dão entrada no próximo módulo. Cada amostra corresponde aos dados (número de bytes e número de pacotes) capturados durante o intervalo de 1 segundo.

O módulo seguinte tem a designação de *shapper,* a função deste módulo é agregar amostras, isto é instantes de tempo, para que a rede neuronal consiga detectar o padrão existente. Por exemplo se tivermos 10 amostras estas amostras poderão ser agregadas

duas a duas, ou quatro a quatro, este módulo têm essa função. Foram feitas várias simulações, em que foi alterado o número de instantes de tempo agregados (Hn), no final foram escolhidos os valores de Hn com que obteve-se melhores resultados à saída da rede neuronal. È no módulo *Shapper* que é adicionado o histórico. O histórico neste contexto consiste em utilizar instantes anteriores à entrada da rede neuronal. Assim, supondo que existe um vector X à entrada da rede neuronal, então de modo a introduzir 1 instante de atraso, é criado um novo vector, X1 em que cada elemento X1(i), deste vector X1 é igual ao elemento X(i-1) do vector X. Logo, obtemos uma matriz M=[X,X1], em que os vectores (X e X1) serão as linhas da matriz. Se quisermos três instantes de tempo no histórico seriam criados 2 vectores, em que num dos vectores, cada elemento seria atrasado 1 unidade e no outro vector cada elemento seria atrasado 2 unidades. Assim com este processo consegue-se adicionar histórico aos dados que serão introduzidos na rede neuronal. Assim, o módulo ao receber o vector v11 agrega as amostras e adiciona histórico, criando a matriz Input1, seguidamente o módulo agrega as amostras do vector v21 e adiciona ao histórico, por fim concatena a matriz criada à matriz Input1. Efectua a mesma operação para os vectores v12 e v22, só que desta vez cria a matriz Input2. A matriz Input1 será utilizada no treino da rede neuronal e a matriz Input2 será utilizada na simulação. No sistema proposto o número de amostras a agregar tem a designação Hn e o número de entradas do histórico tem a designação de Th. Foram feitas várias simulações com os dados do tráfego capturado para compilar uma lista de valores para Th (número de entradas do histórico) e Hn (número de amostras agregadas). Para cada simulação foram utilizadas as seguintes combinações valores de Th (número de entradas de histórico) com valores de Hn (número de instantes agrupados). Este valor Hn é igual ao número de neurónios da camada intermédia da rede neuronal. Estes foram os valores escolhidos, com base nas melhores probabilidades de acerto que foram obtidas à saída da rede neuronal.

Os conjuntos de valores estão representados na seguinte tabela.

Hn	Th
16	6
16	8
16	10
16	12
16	14
4	16
8	16
10	16
12	16
16	20
20	16

Tabela 1 – valores de histórico (Hn) e valores de instantes agregados (Th).

O módulo de treino recebe duas entradas, a rede neuronal a treinar e uma matriz de entrada (Input1). Antes dos dados serem utilizados para treinar a rede neuronal, estes são normalizados com a função *prestd,* para ficarem com a média zero e desvio padrão 1. Seguidamente, a rede neuronal é treinada com os dados da matriz Input1 e obtém-se a saída deste módulo a rede neuronal treinada e um vector de saída (*Train Output*).

O módulo seguinte é o módulo *Simulator, que* corresponde a fase em que os dados são simulados na rede neuronal, este módulo têm como entrada, a rede neuronal treinada e uma matriz (Input2), com metade dos dados obtidos do módulo *Shapper*. Antes dos dados serem utilizados para simular na rede é necessário normalizar os dados com média zero e desvio padrão 1. À saída da rede neuronal obtêm-se um vector (Output), ao qual será feito a operação inversa à operação de normalização que foi feita com a função *prestd*, neste caso é utilizada a função *poststd*. À saída deste módulo obtêm-se o vector Output.

No módulo *Comparator* é feita a comparação entre o vector obtido na fase de treino (Train Output) e o vector obtido na fase de simulação (Output). Assim, se o valor obtido no vector Train Output para a mesma posição *i,* for igual ao valor do vector Output, então o tráfego identificado foi o tráfego do serviço utilizado, senão for igual, então a rede neuronal não conseguiu identificar o tráfego nessa amostra e é assumido que estamos na presença de tráfego zombie, o tráfego zombie é detectado por falso positivo. O valor obtido à saída deste módulo é a probabilidade de acerto à saída da rede neuronal

(Pr). Nesta arquitectura os dados são divididos em metade para treino e metade para simulação. Na arquitectura foi utilizada uma rede neuronal de três camadas (*layers*), que está representada na figura 11.

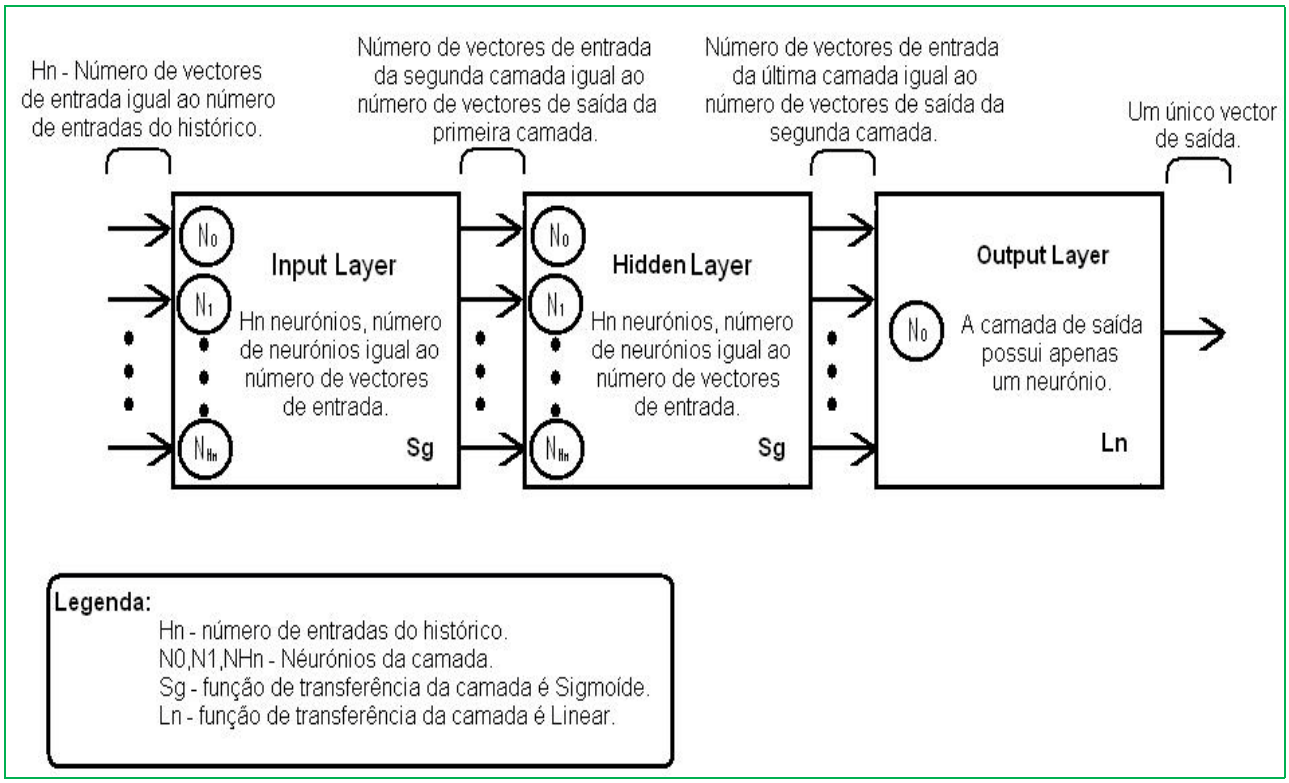

Figura 11 – Esquema da rede neuronal utilizada na simulação.

Pela figura 11, podemos observar que a rede neural utilizada possui três camadas, uma camada de entrada, em que o número de vectores de entrada é igual no nosso caso ao número de entradas do histórico, esta camada possui Hn neurónios, o número de neurónios desta camada é igual ao número de entradas. A camada escondida (Hidden Layer) tem Hn entradas que é igual ao número de saídas da camada de entrada, que por sua vez é igual ao número de entradas do histórico. As funções de transferência usadas nestas duas camadas são funções de transferência *sigmoid*. Estas funções de transferência são *sigmoid,* para limitar as saídas destas camadas a valores entre zero e um. A camada de saída possui 1 único neurónio, e a sua entrada tem Hn vectores, que são os vectores de saída da camada escondida. Esta camada possui um único vector de saída. A função de transferência da camada de saída é linear para limitar as saídas aos valores zero ou um. A técnica de aprendizagem utilizada pela rede neuronal é a técnica *BackPropagation,* é uma técnica de Aprendizagem supervisionada, usada para treinar redes neuronais, e que é apresentada abaixo.

- É apresentada uma amostra à rede neuronal.
- A saída da rede é comparada com a saída desejada da amostra. É calculado o erro em cada neurónio de saída.
- Para cada neurónio, calcular a saída que devia ser obtida, e um factor de escalonamento de quanto mais a saída deve ser ajustada para obter a saída desejada, este é o erro local.
- Ajustar o peso de cada neurónio para baixar o erro local.
- "Culpar" os neurónios do nível anterior pelo erro local, atribuir maior responsabilidade aos neurónios ligados por pesos maiores.
- Repetir os passos anteriores nos neurónios do nível anterior, usando a "culpa" de cada um como o seu erro.

Como o nome do algoritmo sugere os erros (e logo a aprendizagem) propagam-se para trás, dos nós de saída para os nós de entrada. Assim, redes neuronais de *backpropagation* são necessariamente *multi-layer*. Redes neuronais *multi-layer* têm uma função de activação não linear. Uma rede neuronal *multi-layer* usando apenas funções de activação lineares é equivalente a uma rede neuronal linear de uma única camada. Se a última camada de uma rede neuronal *multi-layer* tiver neurónios *sigmoídes*, então as saídas da rede estão limitadas a uma pequena gama. Se ao invés forem usados neurónios lineares as saídas podem tomar qualquer valor. O treino incremental pode ser utilizado em redes estáticas e dinâmicas, embora seja mais usado em redes dinâmicas. Por exemplo se tivermos, uma rede com uma entrada de atraso, se inicializamos os pesos a zero e inicializarmos a taxa de aprendizagem a 0.1 temos.

```
net = newlin([-1 1],1,[0 1],0.1);
net.IW{1,1} = [0 0];
net.biasConnect = 0;
```

Para treinar esta rede incrementalmente temos as entradas e os *targets* como elementos dos *arrays*:

```
Pi = {1};
P = {2 3 4}; T = {3 5 7};
```

Assim treinamos a rede com a soma da actual entrada e a anterior para criar a saída actual, é usada a função *adapt*.

```
[net,a,e,pf] = adapt(net,P,T,Pi);
a = [0] [2.4] [ 7.98]
e = [3] [2.6] [-0.98]
```

A primeira saída é zero porque os pesos ainda não foram actualizados. Os pesos são alterados a cada passo. Se treinarmos a rede em *batch mode* as entradas são convertidas para vectores concorrentes (colunas da matriz). Se usarmos *adapt* o formato das entradas, determina o método de treino. Se as entradas são passadas como uma sequência então a rede é treinada de modo incremental, se as entradas são passadas de modo concorrente, então o modo *batch* é usado.A taxa de aprendizagem utilizada na seguinte rede neuronal é de 0.02.

```
net = newlin([-1 1],1,[0 1],0.02);
net.IW{1,1}=[0 0];
net.biasConnect=0;
net.trainParam.epochs = 1;
Pi = {1};
P = {2 3 4}; T = {3 5 6};
```

Queremos treinar a rede com a mesma sequência usada no modo de treino incremental, mas queremos actualizar os pesos depois de todas as entradas serem aplicadas. A rede é simulada em modo sequencial, porque à entrada é uma sequência, mas os pesos são actualizados em modo *batch*. No modo incremental os pesos são actualizados 3 vezes durante uma passagem pelo conjunto de treino. No treino *batch* os pesos são actualizados apenas uma vez em cada época. A nossa função de treino usa o algoritmo de Lenvenberg-Marquardt que é descrito nas linhas logo abaixo. Os parâmetros de treino a inicializar são *epochs, goal, time, min_grad, Max_fail*. Inicialmente foi utilizada a função *adapt* para treinar a rede, mas posteriormente foi mudada para a função *trainlm*, visto que obteve-se melhores resultados com esta função de treino. A rede neuronal é uma rede de reconhecimento de padrões (*pattern recognition*), visto que tem como objectivo reconhecimento de padrões, que neste caso são perfis de tráfego. Na fase de treino a função de cálculo do erro usada para redes *feedforward* é a função *mse – average squared error* entre os outputs *a* e os *targets t*. O gradiente é calculado usando a técnica chamada *backpropagation* que envolve efectuar cálculos para trás nos nós da rede. A implementação mais simples de *backpropagation learning* actualiza os pesos e a polarização da rede na direcção em que a função de performance decresce mais rapidamente – o negativo do gradiente. Xk+1=Xk – alpha(k)*g(k) Xk – vector com os valores correntes de pesos e polarização. Gk – Gradiente corrente. Existem duas maneiras que o gradiente pode ser implementado, no modo *batch* e modo incremental. No modo *batch* todas as entradas são aplicadas a rede antes de actualizar os valores dos

pesos. No modo incremental o gradiente é calculado e os pesos são actualizados depois de cada entrada ser aplicada a rede.

Na rede neuronal foi utilizado um histórico que varia entre 4 a 16 valores de instantes anteriores utilizados à entrada da rede neuronal, estes valores foram escolhidos com base nas simulações iniciais efectuadas e foram os valores para os quais foram obtidos melhores resultados. Também foi decidido agrupar os instantes de tempo, visto que para 1 hora existem 3600 entradas mais o número de valores do histórico, o que torna a simulação lenta, por isso foram testados vários valores para verificar qual o melhor número de instantes a agrupar, os valores a testar foram entre 4 e 20, dos valores testados foram escolhidos os que foram obtidos melhores resultados. No script criado para treinar e simular a rede neuronal foi utilizado um histórico (definido pela variável Hn), os valores de histórico utilizados foram aqueles valores para os quais obtivemos melhores resultados. O número de épocas definido foram 500 épocas, porque nos testes iniciais foi testado para 1000 e 2000 épocas, e nestes testes verificou-se, que não existia uma alteração muito significativa dos resultados à saída da rede neuronal, ao ser utilizado um valor para o número de épocas maior que 500 épocas. O valor recolhido à saída da rede neuronal è o número de vezes que a rede neuronal acertou no tipo de tráfego. Os valores a saída da rede neuronal são arredondados para zero, ou para um. Depois estes valores são comparados com os resultados obtidos durante a fase de treino. Inicialmente, a rede neuronal foi testada com a função de treino *adapt*, mas os resultados obtidos com esta função de treino eram inferiores aos resultados obtidos com a função de treino *trainlm* que utiliza o algoritmo *Levenberg-Marquardt*. A rede neuronal escolhida possui três camadas, as duas primeiras camadas possuem funções de transferência *logsig*, enquanto a função de transferência da última camada é do tipo *purelin,* para limitar a saída. Assim, as duas primeiras camadas limitam os vectores entre [0,1], enquanto a última camada limita os valores entre {0,1}. Antes de introduzir os dados na rede neuronal, estes são normalizados pela função *prestd* que pré-processa os dados para ficarem com média 0 e desvio padrão 1. No final da simulação é necessário pré-processar os dados obtidos antes de analisar estes, para isso foi utilizada a função *trastd*, que utiliza a média pré-calculada e desvio padrão *standard*. Os dados que foram utilizados foram o número de bytes que saiu da interface por segundo em upload e download. Assim, para cada perfil de tráfego, temos um vector com 3600 entradas, dividimos essas 3600 entradas em 2 vectores de 1800. Um dos vectores é utilizado para treinar a rede neuronal, o outro é utilizado para simular. Em todas as

simulações o *target* utilizado foi de $1x10^{-7}$, visto que nas simulações foi o valor com que foram obtidos melhores resultados, para qualquer objectivo abaixo deste valor não é obtida uma melhoria significativa dos resultados à saída da rede neuronal.

4. Teste e Validação

Neste capítulo são descritos os perfis de tráfego gerados pelas aplicações utilizadas ao longo desta dissertação. Na secção 2 é apresentado o perfil de tráfego gerado pelas funcionalidades do *rootkit* utilizado (*sub7*). Finalmente são apresentados os cenários criados para gerar tráfego. Na secção 4 são apresentados os resultados obtidos à saída da rede neuronal, para cada cenário criado.

4.1. Descrição das Aplicações e do Perfil de Tráfego Gerado

O tráfego HTTP é caracterizado pela utilização elevada da largura de banda com duração não periódica. Nesta experiência foi criado um script que vai buscar uma página a um URL que consta de uma lista, de 30 em 30 segundos. Na figura 12 está representado o *throughput* de um serviço HTTP ao longo de 30 segundos.

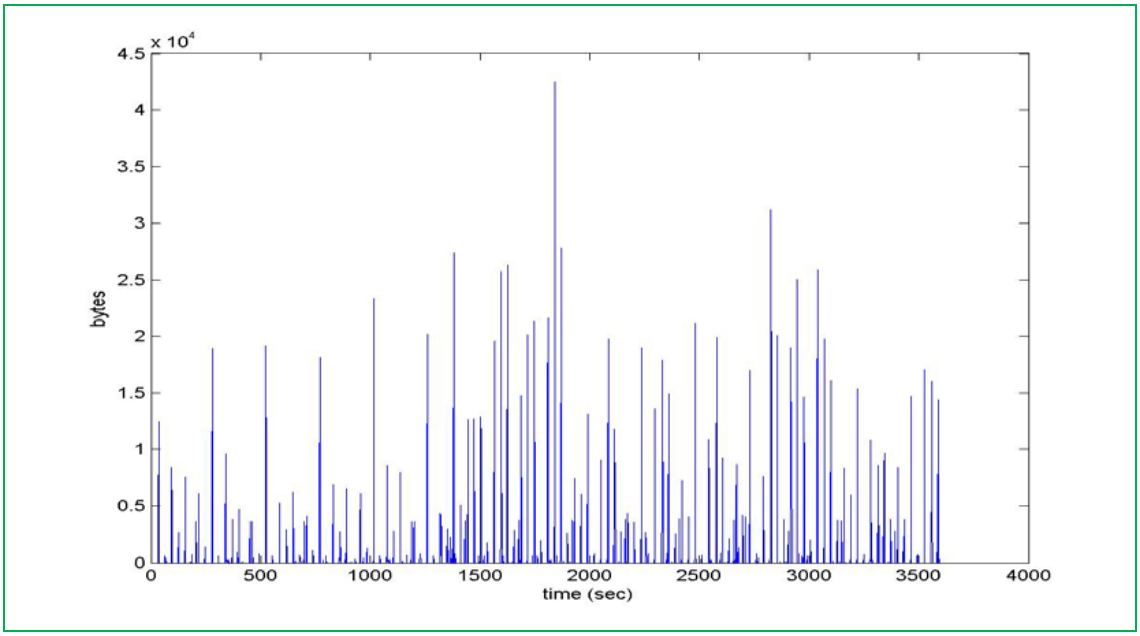

Figura 12 – Tráfego gerado quando apenas temos HTTP (download).

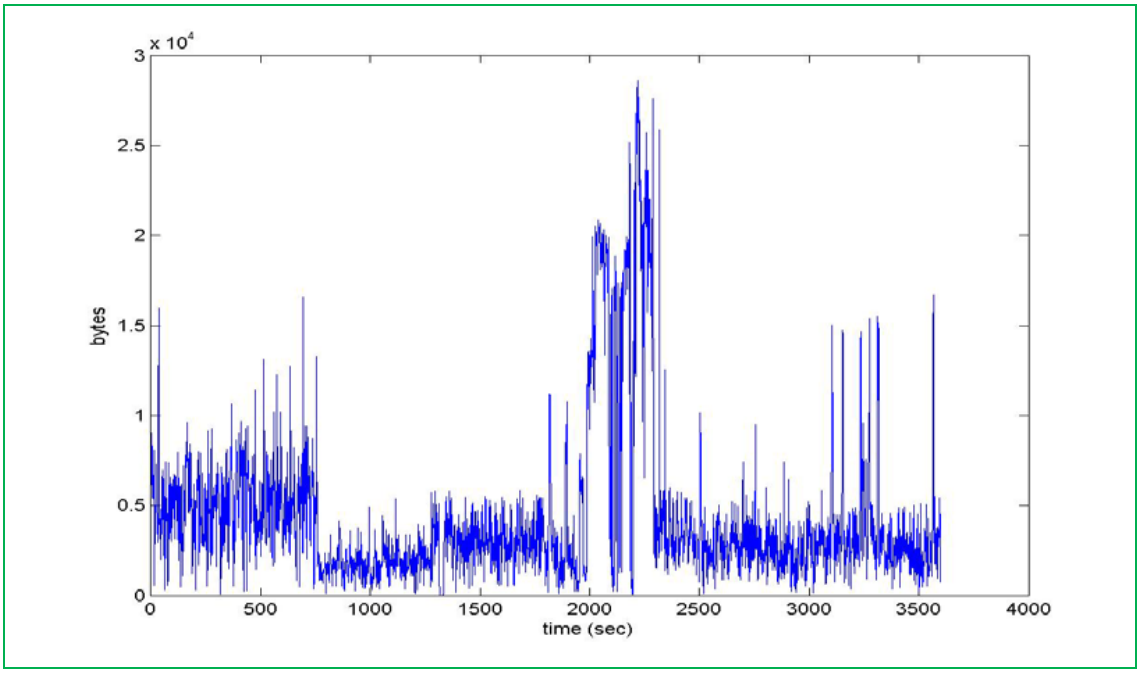

Figura 13 – Tráfego gerado ao longo do tempo quando apenas temos tráfego FTP (download).

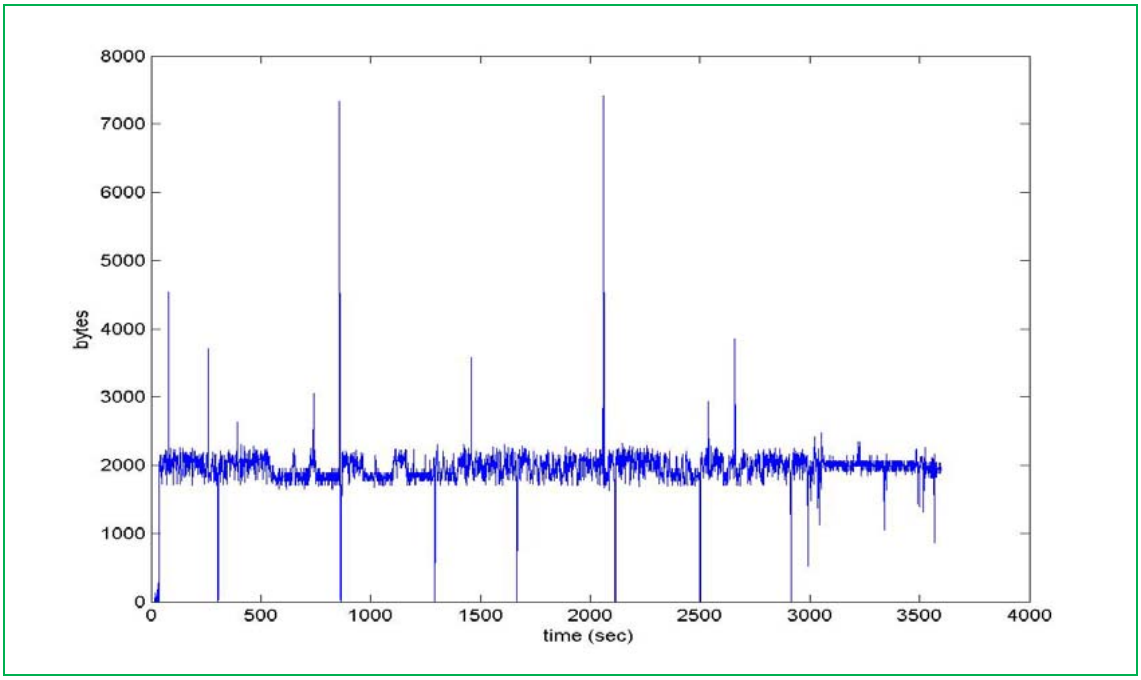

Figura 14 – Tráfego gerado ao longo do tempo quando apenas temos tráfego Jogos (download).

Na figura 13 temos ilustrado o *throughput* gerado pelo serviço FTP, podemos verificar que existe uma utilização elevada da largura de banda, com grande variabilidade devido ao elevado número de sessões TCP/IP abertas/fechadas. As sessões estabelecidas tendem a ocupar toda a largura de banda disponível. Os jogos são caracterizados por uma utilização periódica da largura de banda (picos periódicos), pela figura em 14 pode-se verificar que a utilização média da largura de banda anda a volta dos 2000 bytes/s.

Na figura 15 podemos verificar que o tráfego gerado pelo Skype é aproximadamente constante. O Skype utiliza tecnologia VoIP, podemos verificar que existe uma certa variabilidade, isto acontece porque o skype ajusta os parâmetros de transmissão, para reduzir aos atrasos induzidos pela rede onde circulam os pacotes e reduzir também a perda de pacotes.

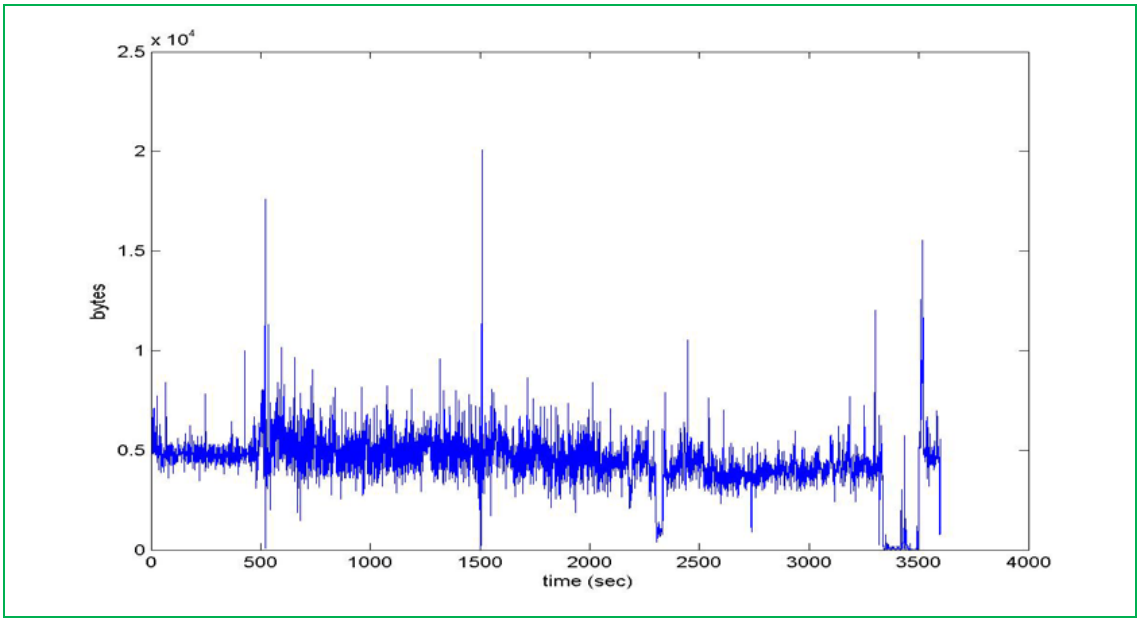

Figura 15 – Tráfego gerado ao longo do tempo quando apenas temos tráfego Skype (download).

O tráfego de *stream* Tv é caracterizado por uma utilização média e constante da largura de banda, exemplos de tráfego de *Stream* são as Rádios e a TV que existem On-line na internet. Na figura 16 temos ilustrado o tráfego gerado pela visualização de uma estação de televisão online durante o período de 1 hora.

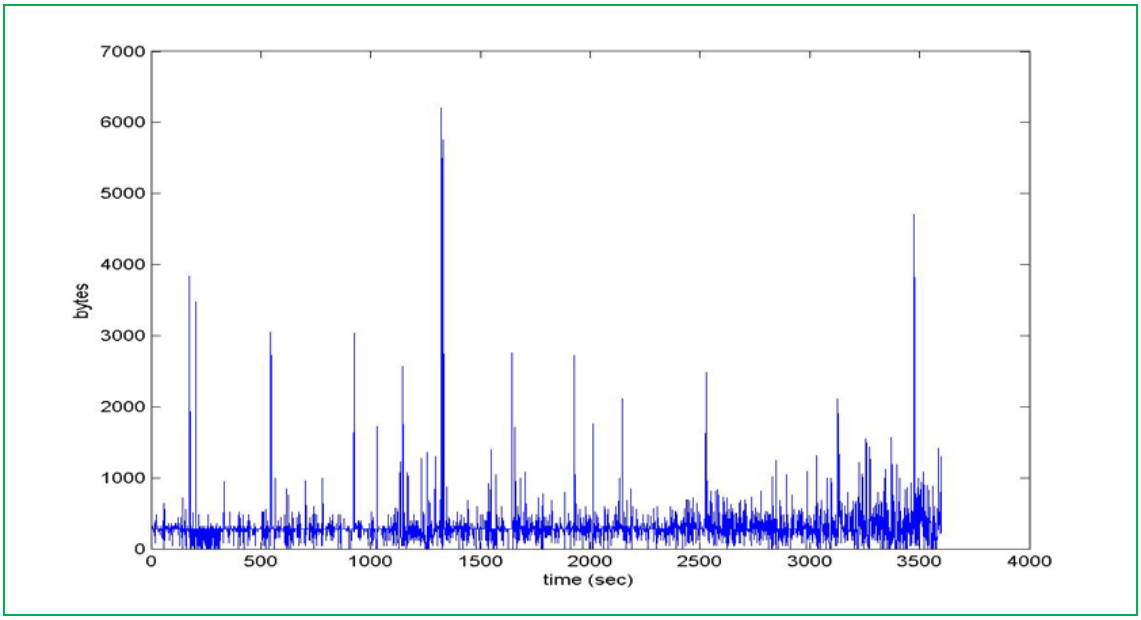

Figura 16 – Tráfego gerado ao longo do tempo quando apenas temos tráfego Stream Tv (download).

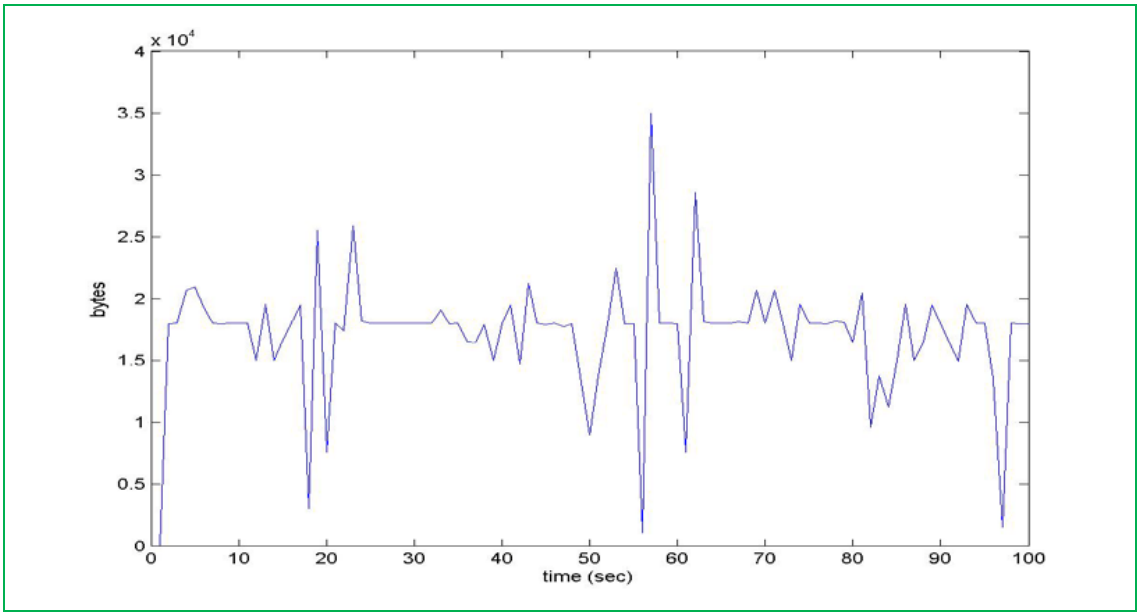

Figura 17– Tráfego gerado ao longo do tempo quando apenas temos tráfego Stream Tv (download) Zoom da figura.

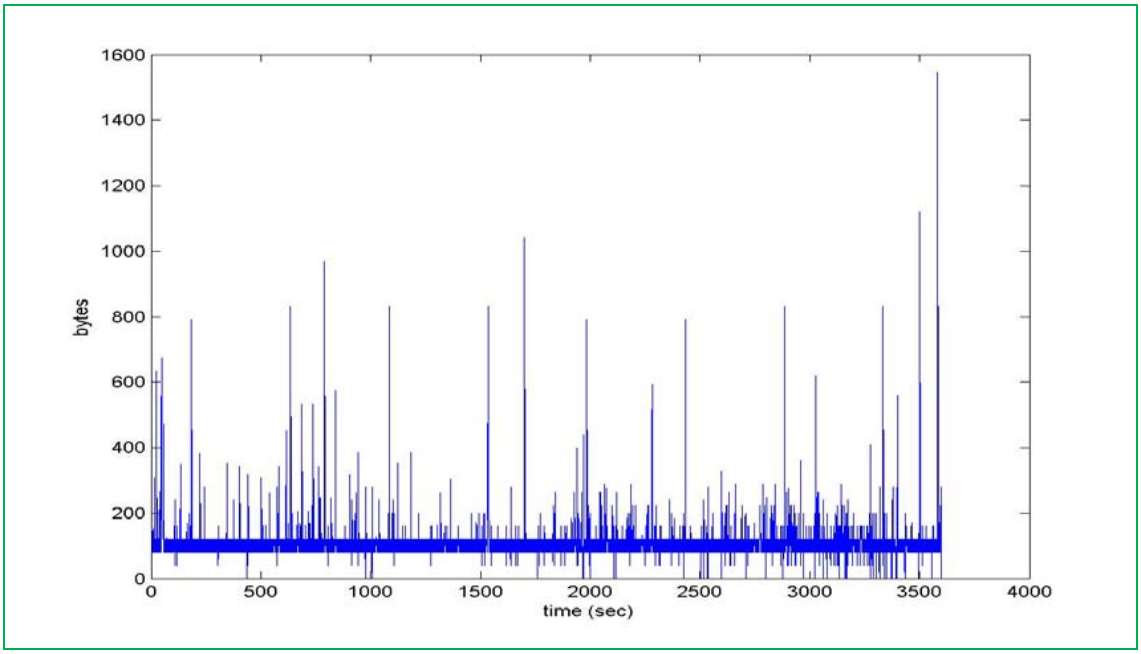

Figura 18 – Variação largura de banda ao longo do tempo quando apenas temos tráfego Stream Rádio (download).

Na figura 17 temos um zoom do figura 14 nos primeiros 100 S, pode-se ver que a utilização da largura de banda anda volta dos 20000 bytes/s que são aproximadamente 160Kbps.

Na figura 18 está representado o tráfego gerado pela utilização online de uma rádio (TSF). Pela figura podemos verificar que a utilização da largura de banda é muito baixa e constante. Tipicamente a transmissão de áudio de boa qualidade é de 128 Kbps, mais o *"overhead"* dos pacotes. Na figura 19 temos um zoom dos primeiros 100 segundos,

pode-se verificar que a largura de banda utilizada, anda a volta dos 4000 bytes/s que corresponde a 32Kbps, também é possível verificar que a utilização da largura de banda é aproximadamente constante ao longo do tempo.

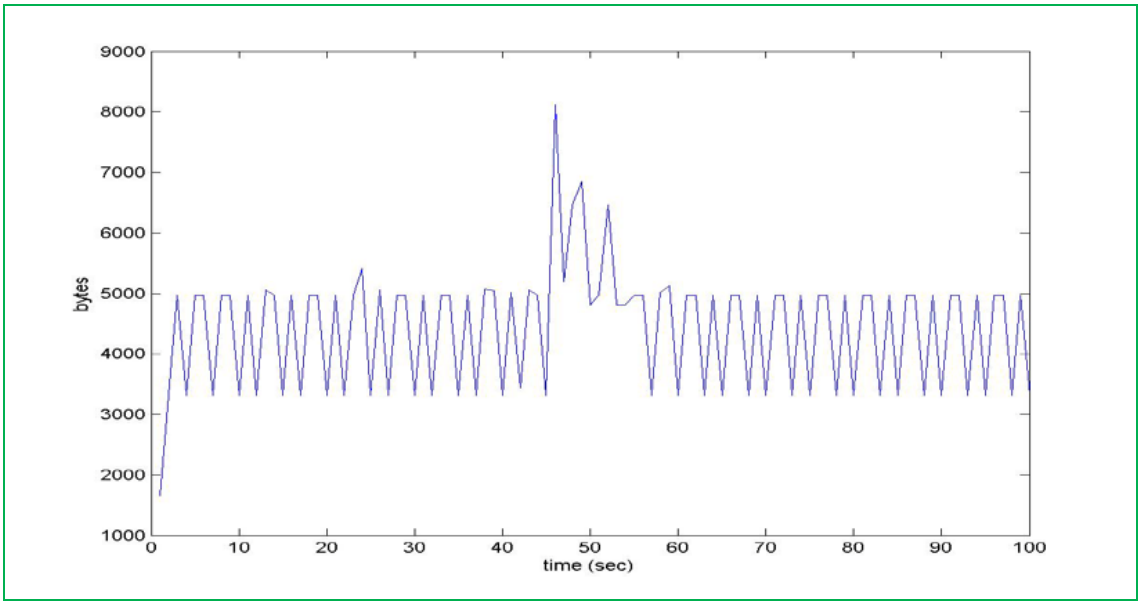

Figura 19 – Tráfego gerado ao longo do tempo quando temos tráfego Stream Rádio (download) Zoom da figura 16.

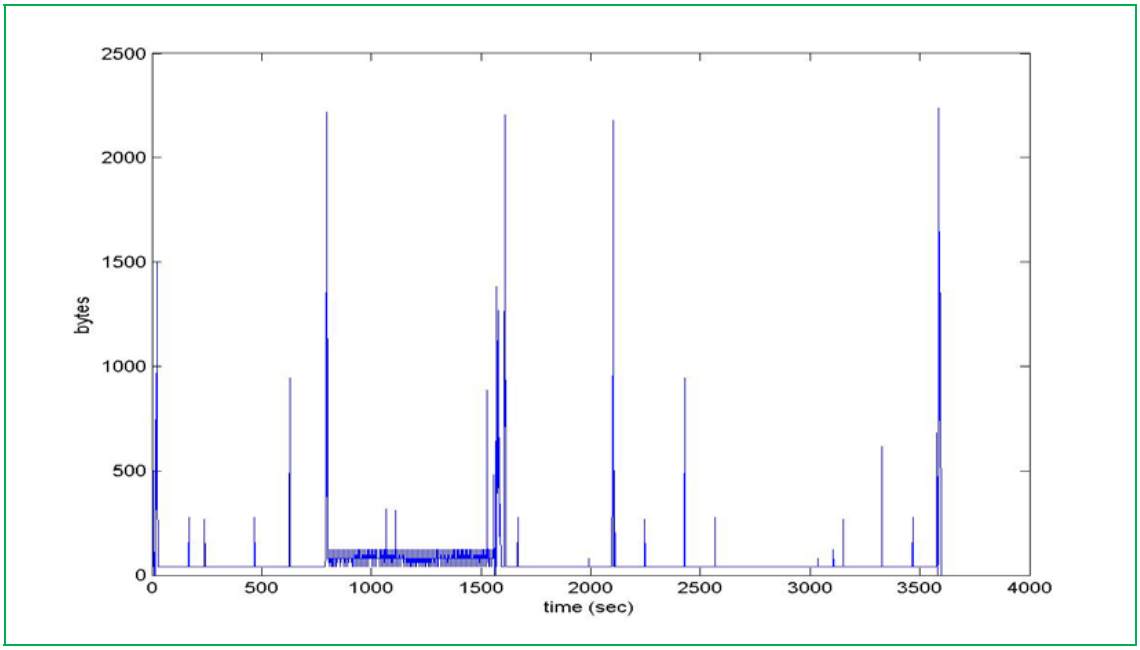

Figura 20 – Tráfego gerado ao longo do tempo quando apenas temos tráfego RDC (download).

Na figura 20 temos um gráfico que representa o tráfego gerado pelo serviço RDC do Windows XP, pode-se verificar que a utilização da largura de banda é aproximadamente constante, com alguns picos.

4.2. Descrição do Trojan, Funcionalidades e perfil de Tráfego Gerado

A figura 21 mostra como evolui a taxa de transferência (download) ao longo do tempo, quando é utilizada a funcionalidade *Port scan* do *subseven.*

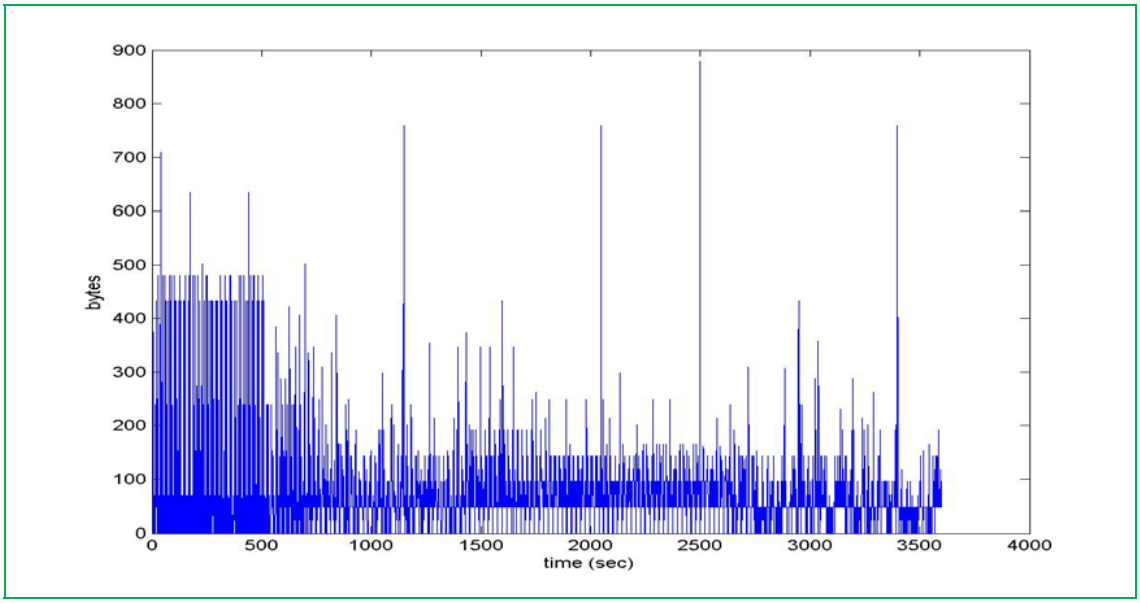

Figura 21 – Funcionalidade de Port Scan do *subseven*.

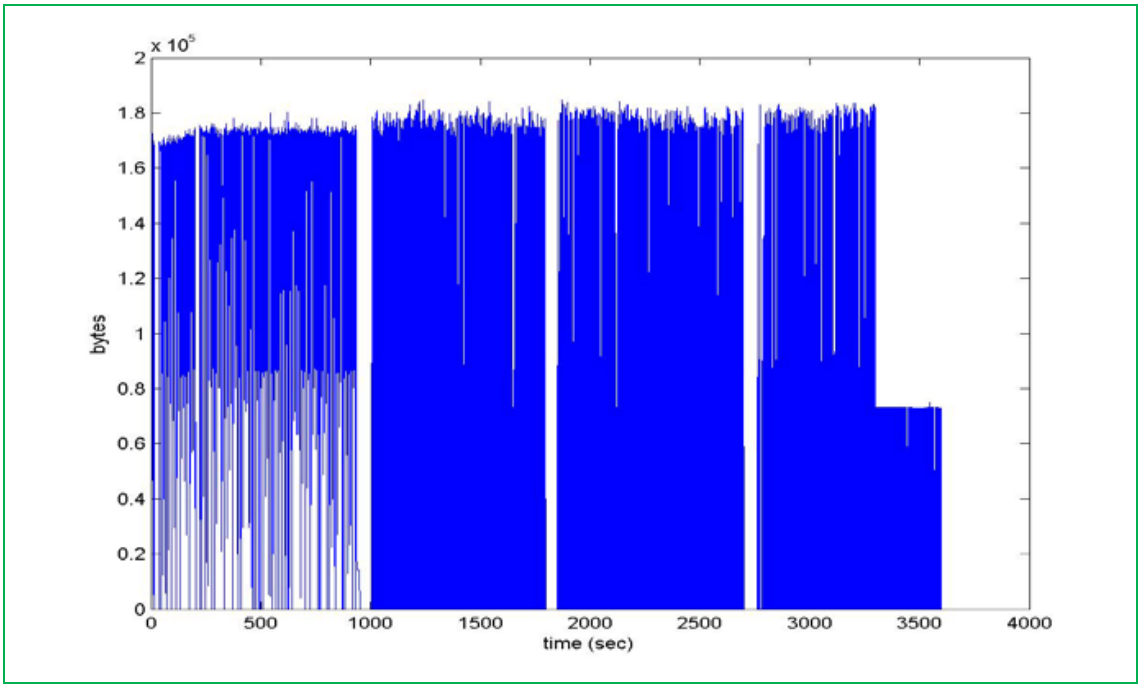

Figura 22 – Funcionalidade de *Snapshot* em que foi alterado o intervalo entre envio de *snapshots*.

Nesta experiência (figura 21) com a funcionalidade de *Port scan* do subseven, foi feito nos primeiros 15 minutos um *Port scan* com um delay de 1s entre envios de pedidos, entre 15-30 minutos o delay foi aumentado para 2s, dos 30-45 o *delay* passou para 3s e

dos 45-60 minutos o *delay* já foi de 5s. Se for feito um zoom aos primeiros 100 segundos é possível verificar que o tráfego é caracterizado por picos periódicos de 1 segundo. Se for feito um zoom aos últimos 100 s verifica-se que o período é de 5 segundos. Isto porque a experiência no inicio tem um *delay* de 1 segundo e no fim tem um *delay* de 5 segundos.

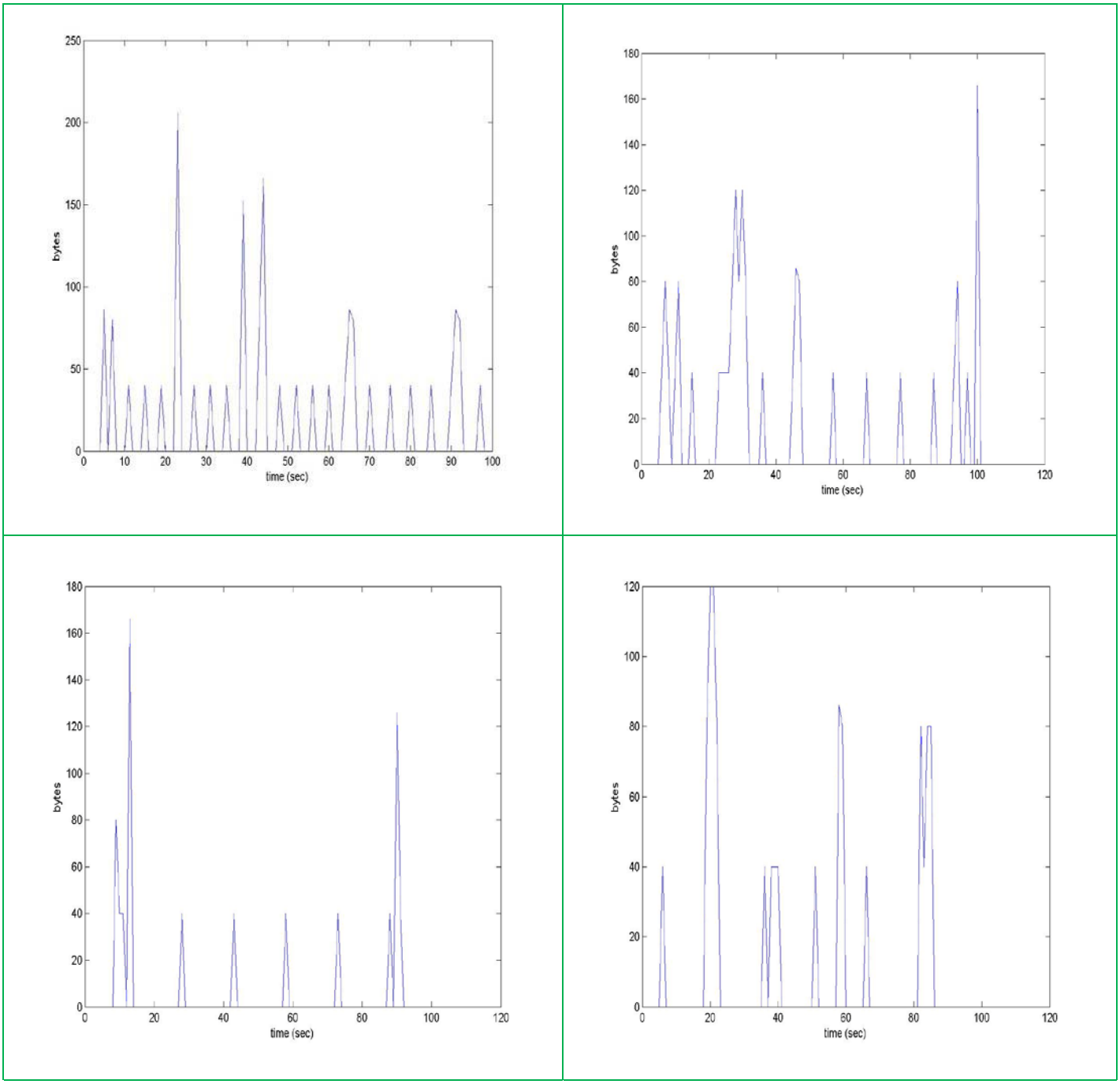

Figura 23 – Zoom da figura 21, funcionalidade *subseven* de port scan com *delay* 1 s (canto superior esquerdo), com um *delay* de 2s (canto superior direito), com *delay* de 3 s (canto inferior esquerdo) e com um *delay* de 5 s (canto inferior direito).

Nesta experiência (figura 23) foi utilizada a funcionalidade de *snapshot* do *subseven*. Assim, nos primeiros 15 minutos o intervalo entre envio de *snapshots* foi de 1s, dos 15-30 minutos o período de envios dos *snapshots* passou para 2s, dos 30-45 minutos foi de 3s e por último dos 45-60 minutos o período foi de 4s, a esta experiência foi dado o nome de *snapshot interval*, que será a referência utilizada neste documento.

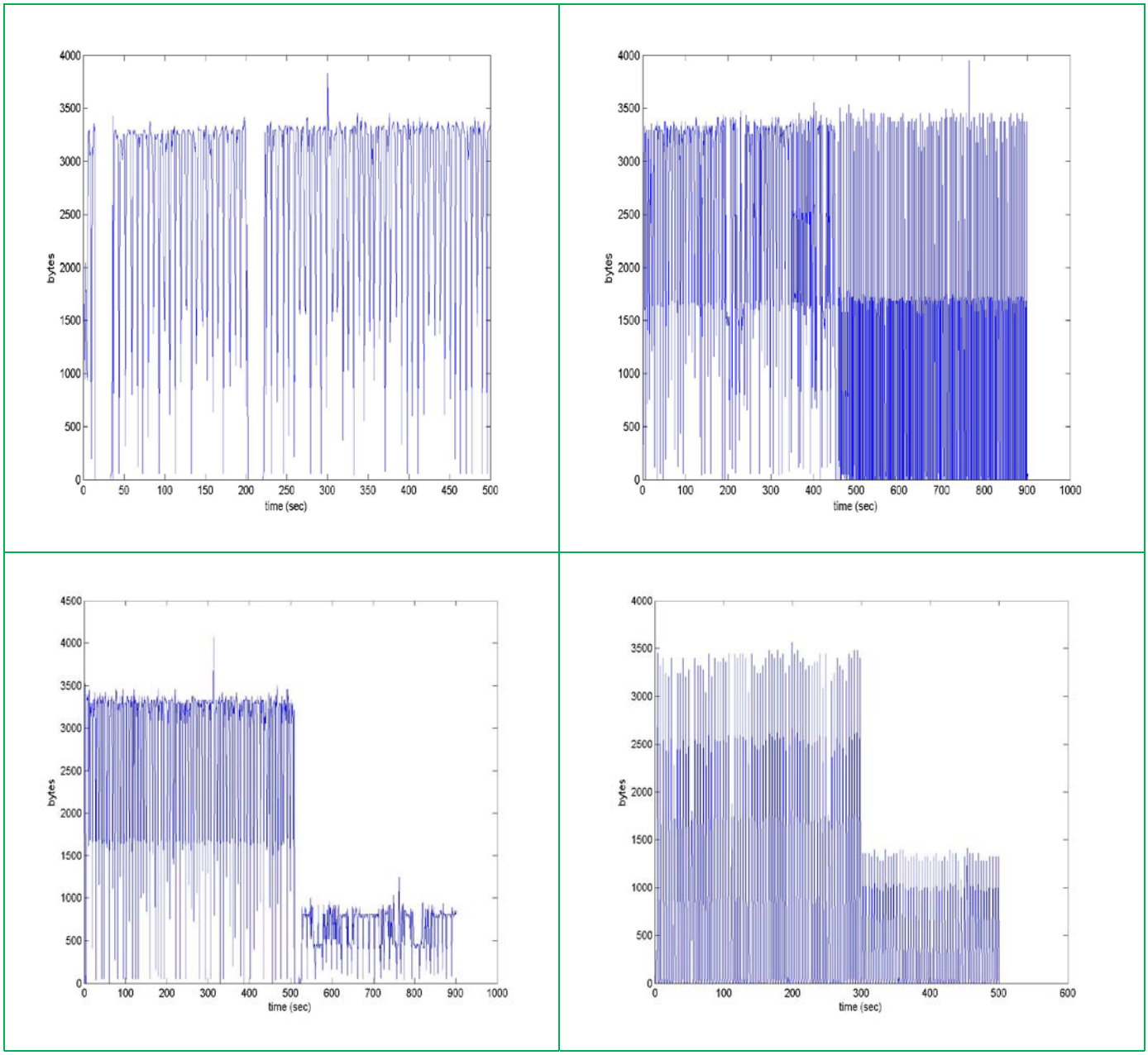

Figura 24 – Zoom da figura 23, funcionalidade *subseven* de *snapshot* com período 1 S (canto superior esquerdo), com um período de 2 S (canto superior direito), com período de 3 S (canto inferior esquerdo) e com um período de 4 S (canto inferior direito).

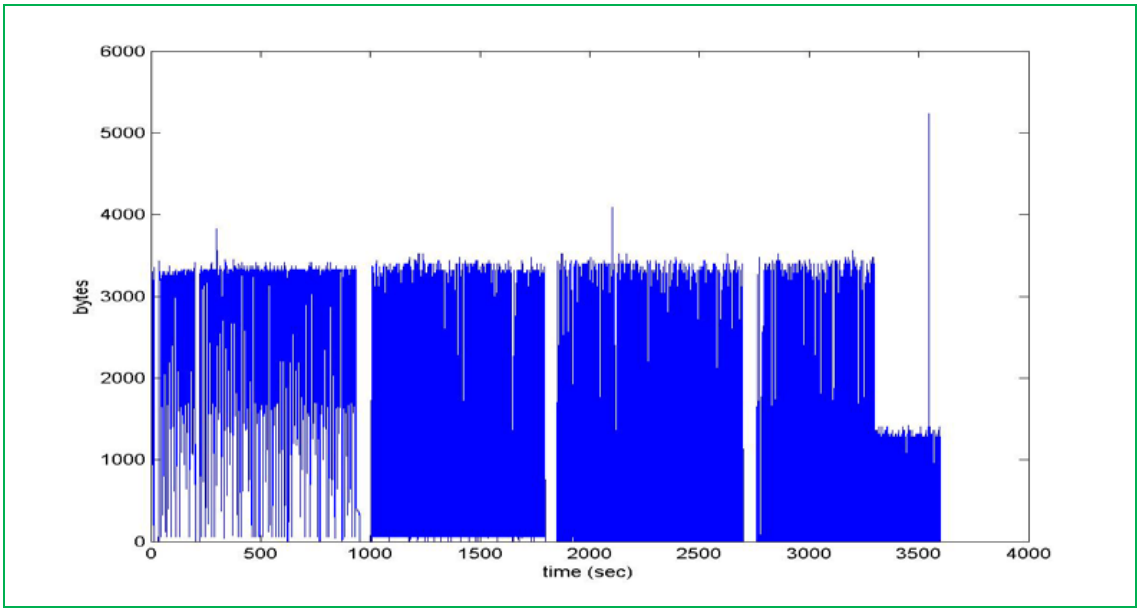

Figura 25– Funcionalidade de *Snapshot* em que foi alterada a qualidade do *snapshot*.

Nesta experiência (figura 25), variou-se a qualidade dos *snapshots*, o intervalo entre envio de *snpashots* foi constante. Nos primeiros 10 minutos o *snapshot* foi tirado com a melhor qualidade, dos 10-20 minutos a qualidade do *snapshot* foi reduzida para metade e dos 20-30 minutos o *snapshot* foi tirado com a pior qualidade. Dos 30-40 voltou-se a colocar na melhor qualidade, dos 40-50 foi colocado na qualidade média e dos 50-60 foi tirado na pior qualidade. Neste documento é utilizada como referência a esta experiência a designação *snapshot quality*. Na figura 26 estão representadas várias figuras que são um "Zoom" da figura 25 que melhor mostram qual a forma do perfil de tráfego representado na figura 25. Na figura 27 está representado o tráfego gerado pelo *rootkit subseven* quando é utilizada a funcionalidade de transferência de ficheiros. Inicialmente existe uma elevada utilização da largura de banda, visto que foi feita a transferência várias vezes de um ficheiro de 20 M. Podemos verificar que a largura de banda utilizada nos primeiros 30 minutos é elevada, isto porque foi feito download do ficheiro, nos últimos 30 minutos a utilização da largura de banda e mais reduzida isto porque foi feito upload do ficheiro várias vezes e o que ficou registado foi a confirmação da recepção dos pacotes por parte do PC comprometido.

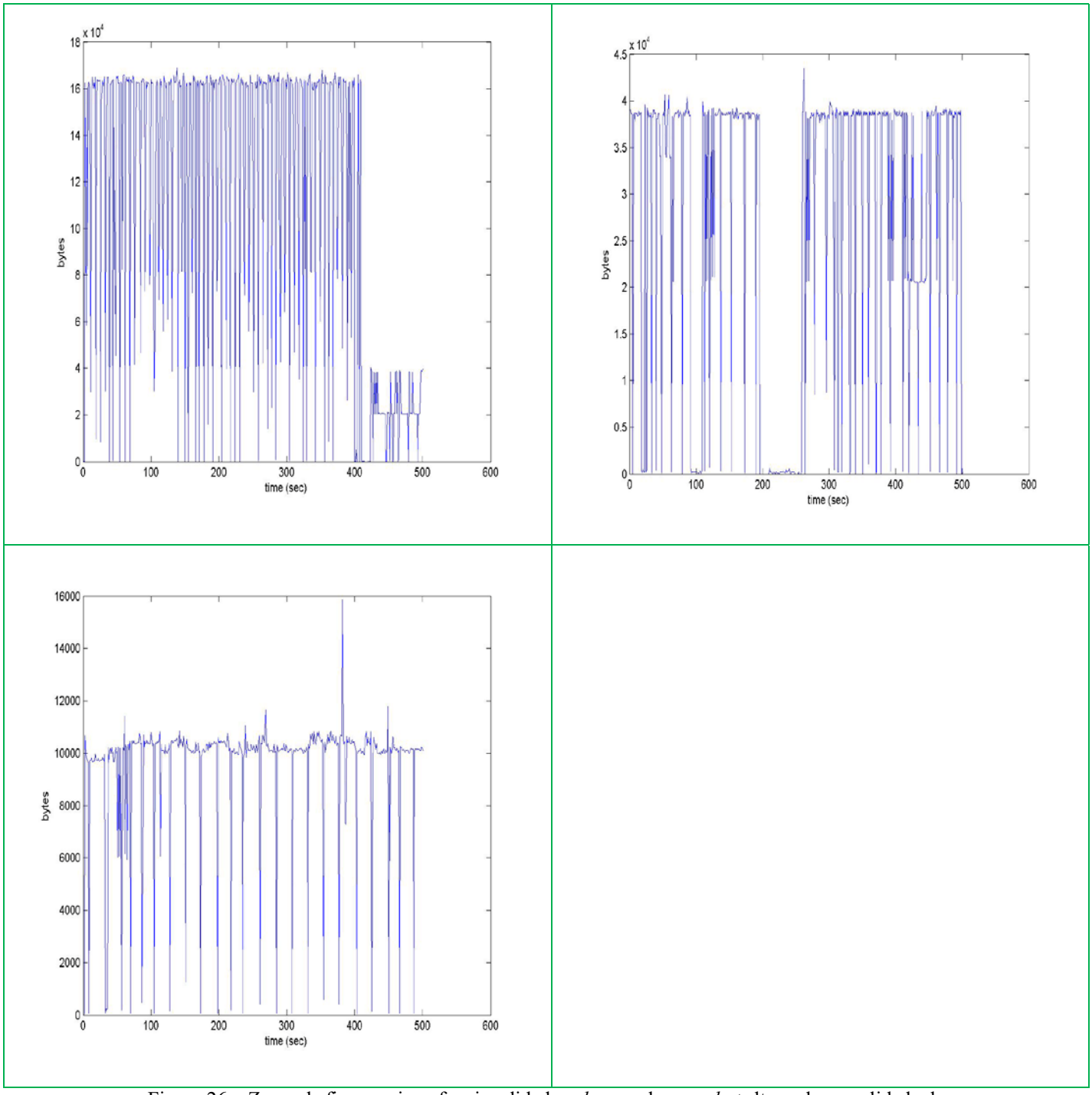

Figura 26 – Zoom da figura acima, funcionalidade *subseven* de *snapshot* alterando a qualidade do *snapshot*, com melhor qualidade (canto superior esquerdo), com metade da qualidade (canto superior direito), com a pior qualidade (canto inferior esquerdo).

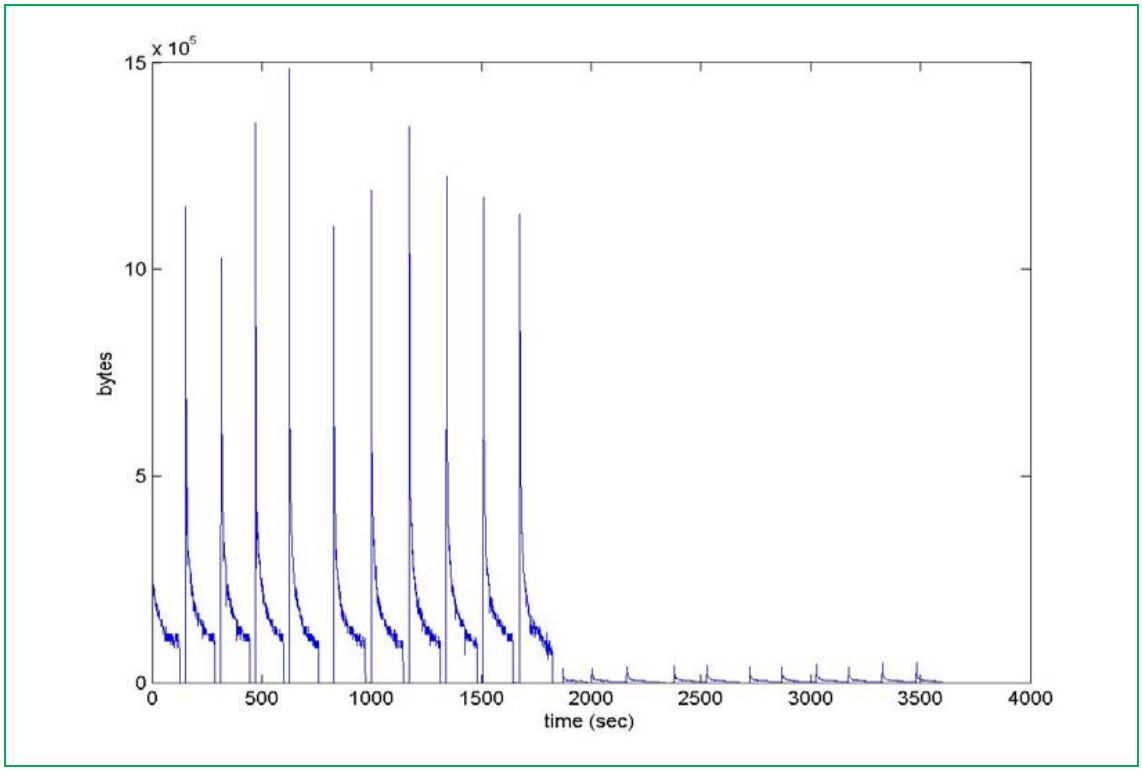

Figura 27– Largura de banda gerada utilizando o *rootkit subseven* com a funcionalidade de transferência de ficheiros.

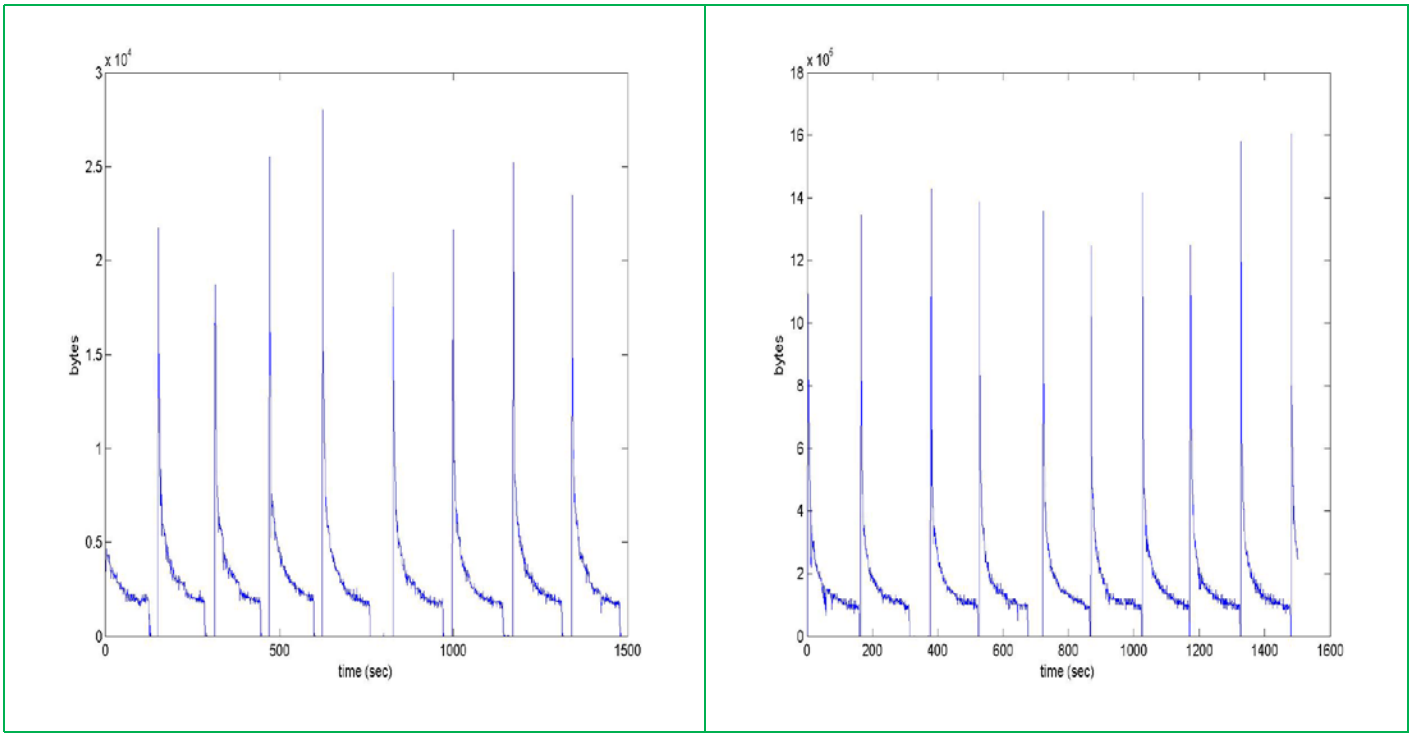

Figura 28 – Zoom da figura acima, funcionalidade *subseven* de transferência de ficheiros, a esquerda temos zoom dos primeiros 1700 s em que foi feito download e a direita temos a parte em que foi feito upload de um ficheiro.

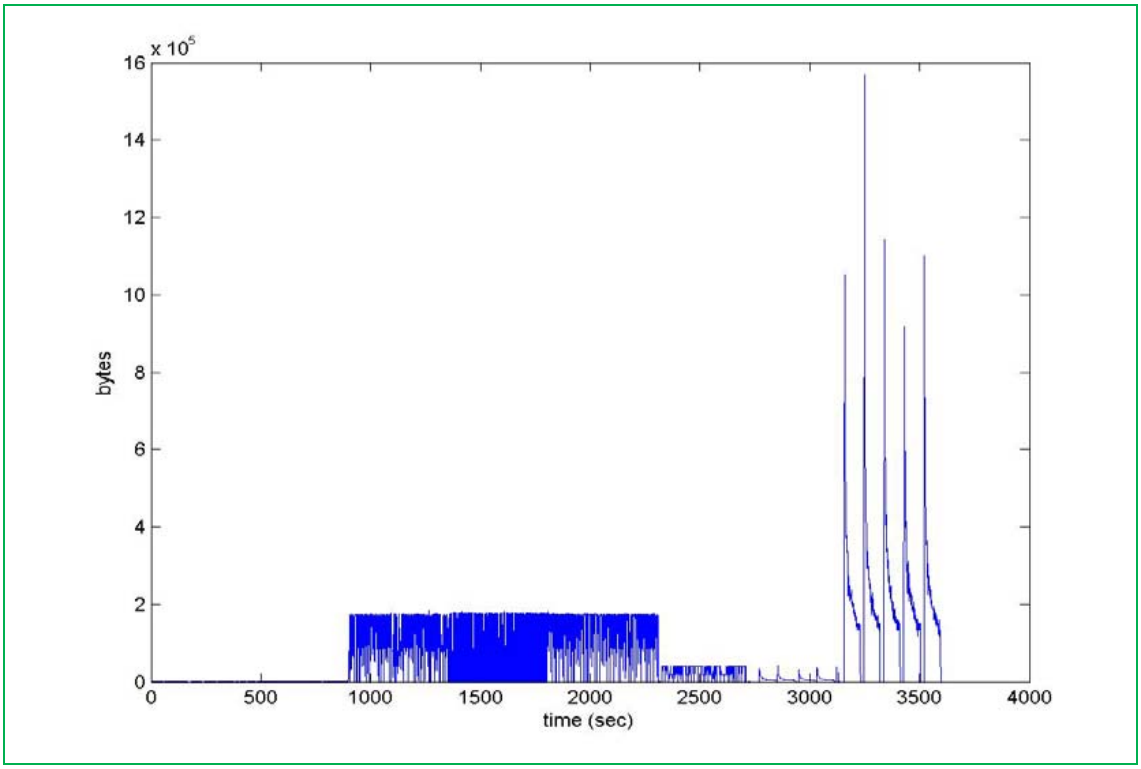

Figura 29 – Utilização da largura de banda quando são utilizadas todas as funcionalidades do *subseven*.

A figura 29 representa a utilização sequencial das três funcionalidades do *rootkit subseven*. Inicialmente foi feito um *port scan*, seguido pela utilização da funcionalidade de *snapshot interval*, *snapshot quality* e transferência de ficheiros. Assim, nos primeiros 15 minutos foi feito um *port scan* com um *delay* de 2s, dos 15-23 minutos foi utilizada a funcionalidade de *snapshot interval* com um período de 1s e na melhor qualidade, dos 23-30 minutos, foi utilizada a funcionalidade de *snapshot interval* na melhor qualidade com um período de 3s, 30-38 foi utilizada a funcionalidade de *snapshot quality* com o período de 1s e na melhor qualidade, dos 38-45 minutos foi utilizada a funcionalidade de *snapshot quality* com o período de 1s e na qualidade média. Dos 45 aos 53 minutos foi utilizada a funcionalidade de transferência de ficheiros em que foi feito o upload de um ficheiro, dos 53-60 minutos foi feito o download desse mesmo ficheiro.

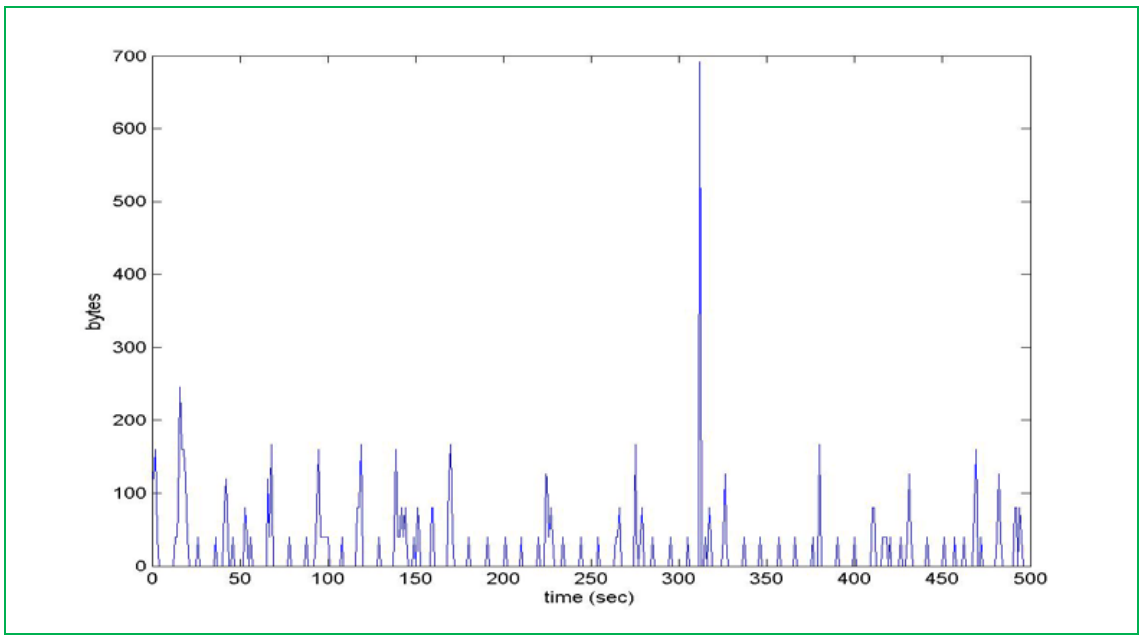

Figura 30 – Zoom dos primeiros 500 s em que foram utilizadas todas as funcionalidades do *subseven*, aqui está representada a funcionalidade de *port scan* com um *delay* de 3 S.

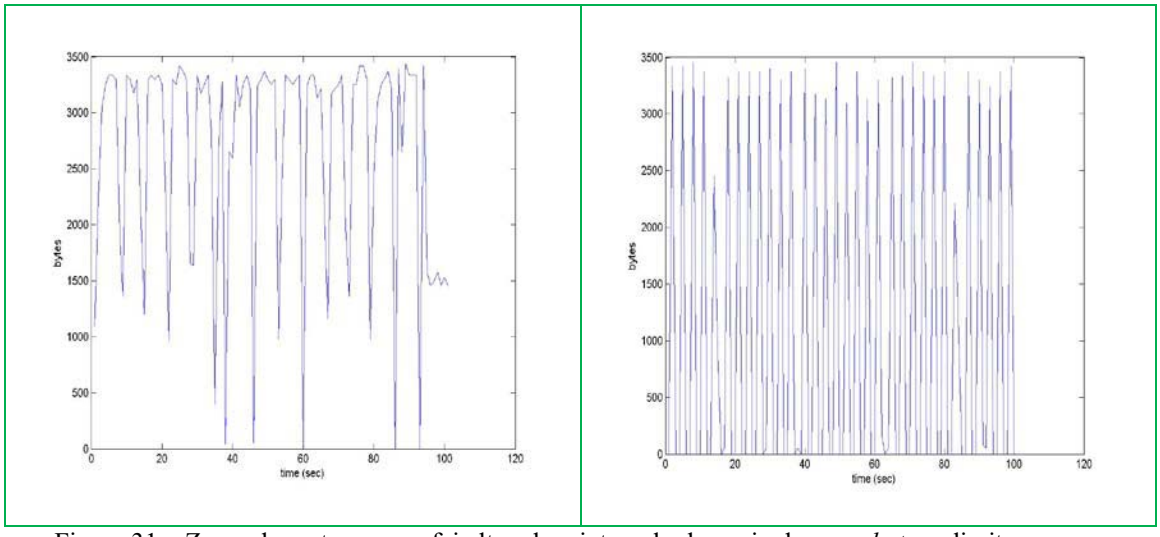

Figura 31 – Zoom da parte em que foi alterado o intervalo de envio de *snapshots*, a direita com um intervalo de envio de 1 S e a esquerda com um intervalo de envio de 3 S.

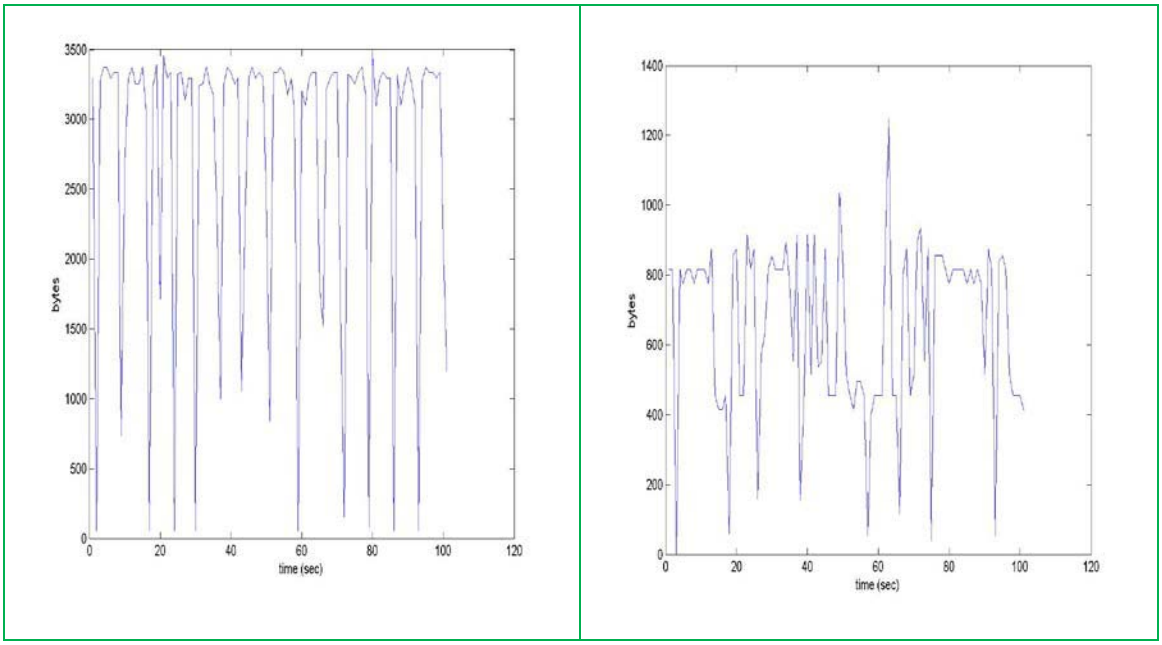

Figura 32 – Zoom da parte em que foi alterando a qualidade de envio de *snapshots*, a direita *snapshot* com a melhor qualidade e a esquerda com metade da qualidade.

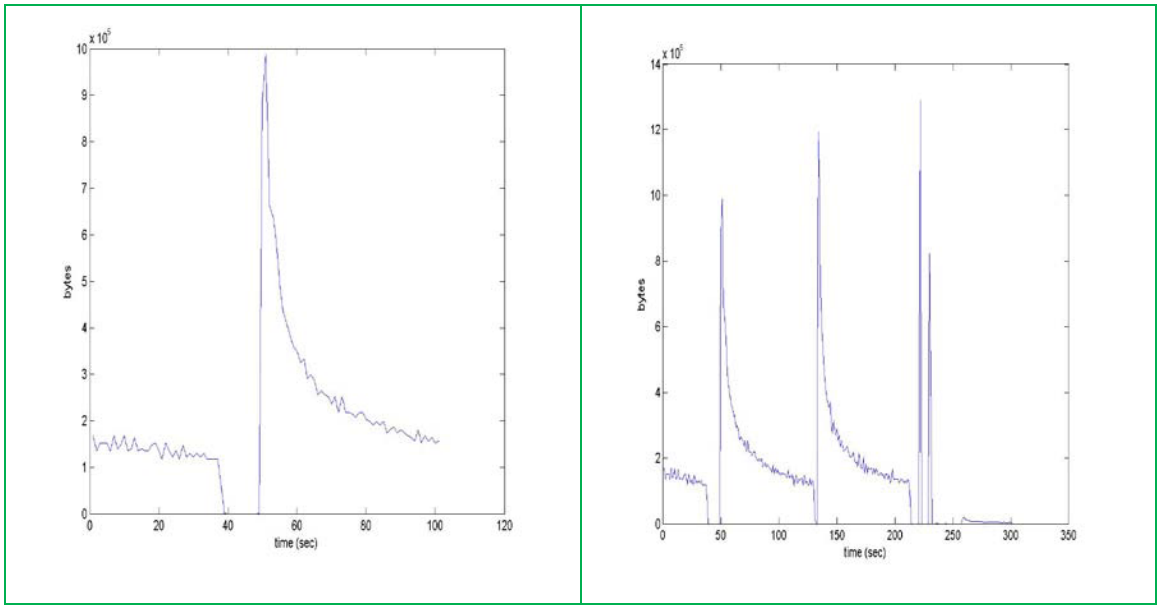

Figura 33 – Zoom da parte em que foi feita a transferência de ficheiros, a esquerda a parte de upload de um ficheiro e a direita a parte de download de um ficheiro.

4.3. Cenários Criados

A referência que é feita ao conceito experiência, neste documento, aplica-se a uma experiência realizada durante o período de 3600 segundos e engloba a captura de tráfego e dados estatísticos durante esse intervalo de tempo. Nas experiências o intervalo de amostragem é de 1 segundo, assim cada linha do ficheiro de saída possui os valores correspondentes a 1 segundo. Foram criados vários cenários possíveis para os

utilizadores. Foi gerado tráfego HTTP, simulando o acesso de um utilizador a diversas páginas na Internet. Para simular este acesso foi feito em script que vai buscar uma página a um URL de 30 em 30 segundos. Para simular a transferência de ficheiros por FTP foi utilizado o FTP da UA, em que foi feito o download de um ficheiro.

Na simulação de tráfego *Stream,* foi utilizado a Rádio On-line da TSF (Stream áudio) em http://www.tsf.pt/online/emissao/default.asp, e a televisão On-line da SIC (Stream áudio e video) em http://www.tsf.pt/online/emissao/default.asp. Para a simulação de *Instant messaging* com suporte para áudio e vídeo foi utilizado o SKYPE que utiliza tecnologia VoIP.

A tabela 2 mostra os vários cenários criados.

Tipo de tráfego	Funcionalidade do subseven utilizada				
	Port scan	Snap interval	Snap quality	File transfer	Sub all
HTTP	X				
		X			
			X		
				X	
					X
FTP	X				
		X			
			X		
				X	
					X
Streaming Video (SIC)	X				
		X			
			X		
				X	
					X
Streaming Audio (Radio TSF)	X				
		X			
			X		
				X	
					X
Skype	X				
		X			
			X		
				X	
					X
Jogos	X				
		X			
			X		
				X	
					X
RDC	X				
		X			
			X		
				X	
					X

Tabela 2 – Cenários criados com as aplicações utilizadas e a utilização do *rootkit subseven*.

Foram criados nove cenários possíveis para os utilizadores que são os que estão na tabela 3:

Tipo de tráfego	Funcionalidade do subseven utilizada				
	Port scan	Snap interval	Snap quality	File transfer	Sub all
HTTP + FTP	X				
		X			
			X		
				X	
					X
HTTP + STREAMING video (sic)	X				
		X			
			X		
				X	
					X
HTTP+SKYPE	X				
		X			
			X		
				X	
					X
Skype + Streaming radio (Tsf)	X				
		X			
			X		
				X	
					X
HTTP + Streaming video	X				
		X			
			X		
				X	
					X
FTP + streaming video	X				
		X			
			X		
				X	
					X
Streaming radio + RDC	X				
		X			
			X		
				X	
					X
Skype + Jogos	X				
		X			
			X		
				X	
					X

Tabela 3 – Cenários de utilizador criados com as aplicações utilizadas e a utilização do *rootkit subseven*.

Por último foram criados vários cenários possíveis de um utilizador, para simular um utilizador que têm várias aplicações a executar no seu *desktop*. Por exemplo, o *subseven* poderia estar a correr em background sem o conhecimento do utilizador, enquanto o utilizador executa serviços como por exemplo Skype, Jogos, FTP e HTTP. Um exemplo de um cenário possível seria, o utilizador estar a aceder a uma página Web, enquanto

em background está a efectuar uma transferência de ficheiros por FTP, e colocar um *Stream* de uma Rádio on-line a tocar. Outro cenário possível seria o utilizador estar a ver televisão on-line, e em background estar a fazer a transferência de ficheiros. Foi também gerado tráfego de jogos com tráfego transferência de ficheiros (FTP) em background. Outro cenário utilizado foi utilizar o Skype com os serviços de voz e vídeo, aceder a uma página Web, e em background efectuar a transferência de ficheiros por exemplo com FTP.

Ao executarmos diversas aplicações numa rede como por exemplo transferência de ficheiros, *Stream*, HTTP, jogos, entre outros torna-se difícil para um gestor de redes conseguir monitorizar que aplicações estão a ser executadas na sua rede e assim mais difícil é a tarefa de proteger a rede. Para cada experiência foi criado um ficheiro que consistia na soma dos tráfegos constituintes dessa experiência, que foram recolhidos em experiências separadas, isto é, se por exemplo tínhamos a experiência HTTP+Sub7 com a funcionalidade Port scan, foi feita a soma dos dados da experiência HTTP, com os dados da experiência com sub7 com a funcionalidade port scan. Com os dados resultantes da soma foi simulado na rede neuronal, para posterior comparação com os resultados obtidos com o agregado de tráfego.

4.4. Resultados

Depois de obter os dados (número de bytes capturados por segundo e número de pacotes/s), das diversas simulações mencionadas nos cenários, foram introduzidos os dados na rede neuronal criada. A rede neuronal é treinada com metade dos valores de cada simulação, isto é, 1800 s, que equivale a 1800 amostras. De seguida os outros 1800 segundos são usados, para simular na rede neuronal. Após obter os resultados, depois da fase de simulação, estes resultados foram comparados com o que deveria ser obtido à saída da rede neuronal. Com base no número de vezes que o tráfego foi identificado correctamente pela rede neuronal (simulação) foi calculada a percentagem de vezes que a rede neuronal acertou na identificação do tráfego.

Nas secções seguintes são apresentadas as percentagens de acerto à saída da rede neuronal para os cenários que foram criados. No primeiro caso temos a experiência em que foi feita a simulação com os dados pertencentes aos vários cenários, com as funcionalidades do subseven. Na segunda experiência tentou-se verificar, se ao somar os tipos de tráfegos constituintes do cenário, mas capturados isoladamente, à saída da

rede neuronal eram obtidos os mesmos resultados, tanto para download como para upload. A primeira coluna de cada uma das tabelas seguintes, é feita referência ao tipo de tráfego que foi simulado e a funcionalidade do rootkit utilizada.

A segunda coluna representa a percentagem de acerto obtida à saída da rede neuronal, que foi calculada no final de cada simulação, aqui nestas tabelas apenas estão representados os melhores valores obtidos.

Na terceira coluna está representada a percentagem de acerto obtida à saída da rede neuronal, quando são somados os dados referentes aos tipos de tráfego constituintes de cada cenário, que foram obtidos separadamente e será designada por Soma de Tráfegos.

Serviço HTTP

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	95,79	98,26	80	97,86
Snap Interval	99,29	100	95,21	100
Snap Quality	99,29	100	100	50
File Transfer	97,02	94,49	98,40	93,94
Sub All	93,52	97,39	93,20	98,40

Tabela 4 – Resultados obtidos com o serviço HTTP.

Analisando a tabela 4, pode-se verificar que a probabilidade de acerto para download foi melhor no caso em que foi simulado os dados capturados no cenário, já no caso da soma dos tipos de tráfegos constituintes do cenário, a probabilidade de acerto foi inferior. Para upload a probabilidade de acerto da rede neuronal também foi inferior no caso da soma de tráfegos. Na figura 34 está representado o perfil de tráfego do serviço HTTP utilizando a funcionalidade de *snapshot* do *subseven* fazendo variar o tempo de envio destes.

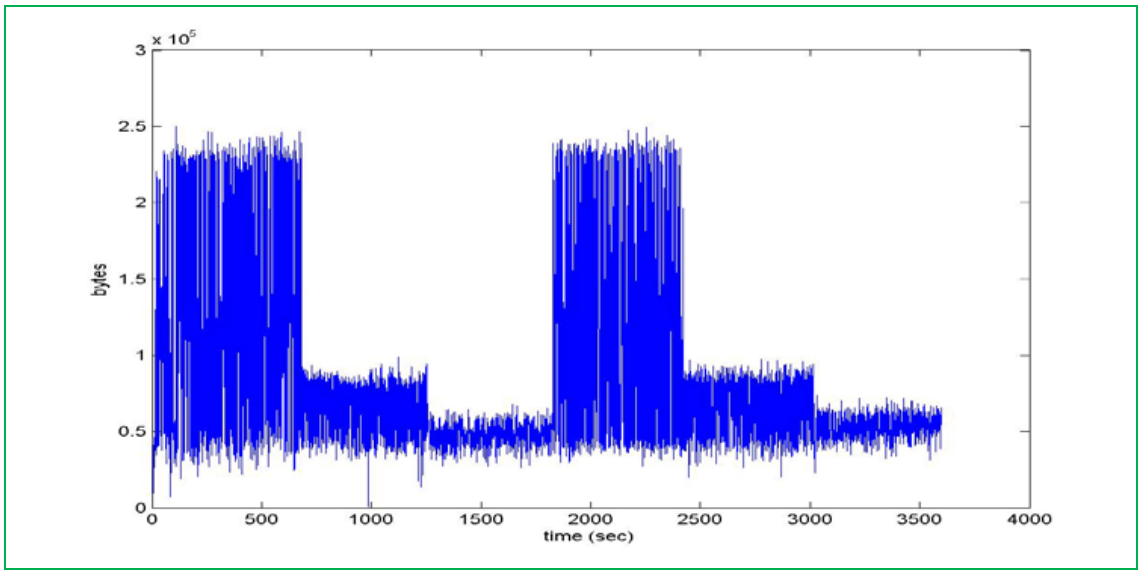

Figura 34 – Perfil de tráfego do serviço HTTP.

Serviço FTP

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	88,40	86,08	52,75	80,05
Snap Interval	90,54	100	76,10	100
Snap Quality	95,36	96,18	88,55	50,00
File Transfer	90,29	90,76	80,05	92,22
Sub All	92,37	95,94	72,28	93,18

Tabela 5 - Resultados obtidos com o serviço FTP.

No caso do serviço FTP, a probabilidade de acerto à saída da rede neuronal no caso em que temos os dados capturados no cenário, foi inferior à probabilidade de acerto à saída da rede neuronal quando temos HTTP.

Para download no caso em que foi feita a simulação com os dados do cenário a probabilidade de acerto á saída da rede neuronal, foi superior à probabilidade obtida quando utilizamos os dados obtidos da soma dos tipos de tráfego constituintes do cenário, obtidos separadamente. Para upload no caso da soma a probabilidade de acerto, foi inferior a probabilidade de acerto quando temos os dados do cenário. No experiencia com a funcionalidade do *rootkit* de transferência de ficheiros, devido a grande variabilidade do número de pacotes e do número de bytes tanto em upload como em download a rede neuronal tem por vezes dificuldade em diferenciar este tipo de tráfego.

Serviço Stream Tv

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	93,91	95,94	62,02	99,41
Snap Interval	97,98	100	97,50	100
Snap Quality	95,36	100	100	50,00
File Transfer	98,40	96,48	99,27	98,86
Sub All	94,20	96,08	90,72	98,98

Tabela 6 - Resultados obtidos com o serviço Stream TV. Pela tabela 6

Pode-se verificar que em todos os casos a probabilidade de acerto à saída da rede neuronal foi elevada, excepto no caso da experiência com a funcionalidade de *port scan* e download (soma dos tipos de tráfego) que foi de 62%, isto poderá dever-se a elevada frequência com que são feitos os *port scans*, em que o perfil de tráfego é parecido ao perfil de tráfego de *stream*, como pode-se verificar na figura 35. Outro caso em que a probabilidade de acerto foi inferior, foi no caso em que temos a funcionalidade de *snapshots* do *rootki*t, e é alterada a qualidade dos *snapshots* no caso em que foi feita a soma dos tipos de tráfego (50%), isto acontece porque, o *stream* e a funcionalidade de *snapshots* são ambos tráfegos de *stream* logo tem as mesmas características.

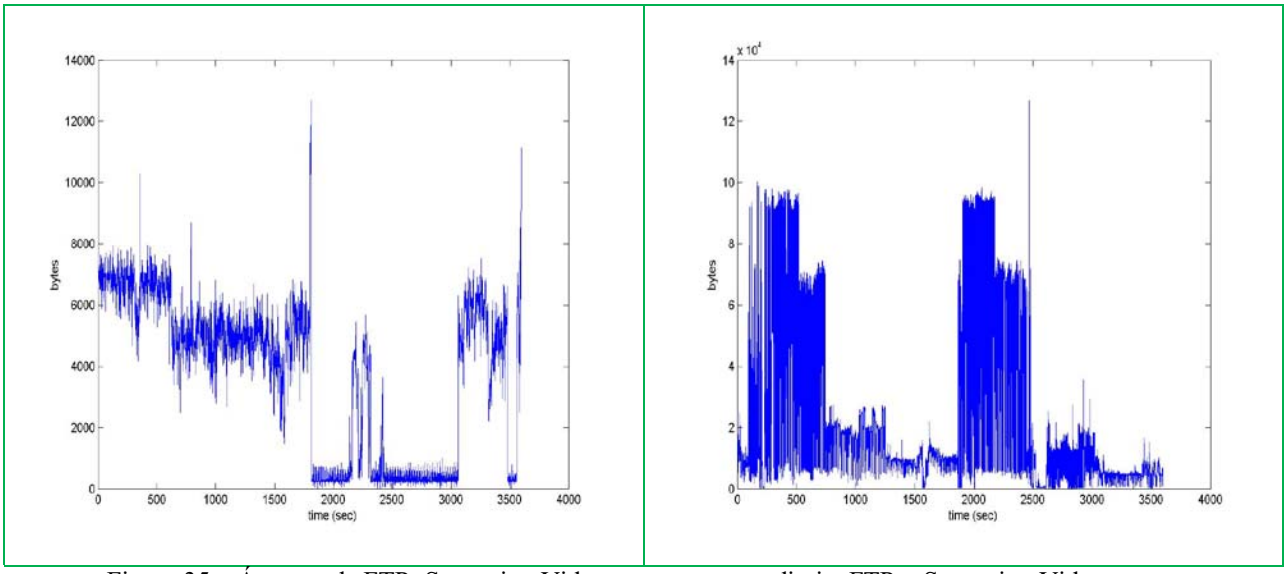

Figura 35 – Á esquerda FTP+Streaming Video + port scan e a direita FTP + Streaming Video + snap quality (download).

Serviço Stream Rádio

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	92,89	100	89,56	100
Snap Interval	97,68	100	98,84	100
Snap Quality	97,82	100	100	50,00
File Transfer	98,98	97,13	100	96,76
Sub All	95,36	100	97,82	100

Tabela 7 - Resultados obtidos com o serviço Stream Rádio On-line.

Neste tipo serviço para download os resultados são muito bons, já no caso de upload anda a volta dos 100%, tanto na soma dos tipos de tráfego como no caso em que temos os dados do perfil capturados. Só no caso do upload (soma), em que foi utilizada a funcionalidade de *snap quality*, é que a probabilidade de acerto à saída da rede neuronal foi de 50%, isto deve-se a esta funcionalidade do *subseven* ter características de *stream*.

Serviço Skype

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	93,12	96,77	76,83	85,65
Snap Interval	93,04	100	90,46	100
Snap Quality	92,17	100	100	50,0
File Transfer	94,49	100	99,85	92,22
Sub All	93,33	100	93,33	94,33

Tabela 8 - Resultados obtidos com o serviço Skype.

No serviço Skype, à saída da rede neuronal os resultados obtidos foram inferiores aos dos serviços anteriores. No caso de download, verifica-se que a probabilidade de acerto anda a volta dos 93% para o tráfego capturado no cenário, no caso da soma os

resultados obtidos são inferiores excepto no caso em que temos a funcionalidade de transferência de ficheiros, que a percentagem de acerto à saída da rede neuronal foi superior.

Serviço Jogos

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	94,90	91,59	80,05	90,31
Snap Interval	99,13	98,69	98,84	100
Snap Quality	98,69	100	100	50
File Transfer	99,71	99,27	100	92,31
Sub All	96,66	93,33	92,21	92,07

Tabela 9 - Resultados obtidos com o serviço JOGOS.

No serviço de jogos a probabilidade de acerto à saída da rede neuronal no caso de download foi superior ao de upload, tanto no caso em que foi simulado o perfil de tráfego, como no caso em que foi feita a soma dos tipos de tráfego.

No caso em que foi feita a simulação do serviço de jogos com a funcionalidade de *snap quality* do *subseven* (soma), a probabilidade de acerto à saída da rede neuronal foi de 50%, muito inferior aos outros resultados obtidos, isto poderá dever-se ao perfil de tráfego obtido com a soma dos tipos de tráfego constituintes do cenário ser diferente do perfil de tráfego dos jogos.

Serviço RDC

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	99,14	99,12	98,72	100
Snap Interval	99,57	100	99,71	100
Snap Quality	99,57	100	99,85	100
File Transfer	100	99,71	100	99,71
Sub All	99,57	100	99,15	100

Tabela 10 - Resultados obtidos com o serviço RDC.

No serviço de RDC a probabilidade de acerto foi a volta dos 99%, tanto no caso em que temos o perfil de tráfego gerado pelo RDC, como no caso da soma dos tipos de tráfego constituintes das experiências.

Cenário FTP + Streaming Video

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	99,14	90,49	52,93	83,04
Snap Interval	99,57	100	75,65	99,71
Snap Quality	99,57	100	86,66	50,00
File Transfer	100	97,68	82,99	92,17
Sub All	99,57	100	71,40	100

Tabela 11 - Resultados obtidos com o Cenário FTP + Stream Vídeo.

Na tabela 11 pode-se verificar que a probabilidade de acerto no caso de download (cenário) foi a volta de 100%, quando temos a soma dos tipos de tráfego constituintes (download) do cenário a probabilidade de acerto diminuiu consideravelmente. O mesmo aconteceu para o caso de upload, os resultados no caso em que foi simulado os dados do cenário, são superiores aos resultados obtidos com a soma dos tipos de tráfego constituintes do cenário. Isto poderá ser porque a soma dos tipos de tráfego não têm um perfil de tráfego igual ou parecido com perfil de tráfego simulado no cenário, logo temos uma probabilidade de acerto menor.

È de salientar que no caso em que é simulado a soma dos tipos de tráfego, constituintes do cenário (download), utilizando a funcionalidade de *port scan* do *subseven,* a probabilidade de acerto foi de 52%. Isto poderá ser porque o tráfego gerado pelo *rootkit* influenciar no perfil de tráfego, sendo por isso mais difícil identificar o tráfego ilícito.

Cenário HTTP + FTP

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	98,00	98,38	56,15	79,32
Snap Interval	94,75	100	79,32	100
Snap Quality	97,24	100	87,60	50
File Transfer	97,65	95,07	78,73	92,17
Sub All	92,89	97,42	65,24	96,81

Tabela 12 - Resultados obtidos com o Cenário FTP + HTTP.

Os resultados obtidos neste cenário foram bons, tanto em download como em upload.

Pode-se verificar que os resultados obtidos à saída da rede neuronal, com a soma dos tipos de tráfego constituintes do cenário, são inferior aos resultados obtidos com o tráfego capturado na simulação do cenário. Na figura 36 temos o perfil de tráfego do cenário.

Cenário HTTP + Streaming Rádio

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	88,96	90,59	75,65	97,16
Snap Interval	95,36	100	96,37	100
Snap Quality	96,86	100	100	50
File Transfer	97,56	95,41	99,42	92,63
Sub All	95,65	96,60	88,71	95,18

Tabela 13 - Resultados obtidos com o Cenário HTTP + Stream Rádio On-Line.

Na tabela 13 está representada a percentagem de acerto obtida à saída da rede neuronal, pode-se verificar que em geral os resultados obtidos foram bons tanto em download

como em upload. É no entanto de verificar que a probabilidade de acerto foi inferior no caso da simulação com a soma dos tipos de tráfego constituintes do cenário.

Na figura 37 está representada uma figura com o número de bytes em download ao longo do tempo (3600 s).

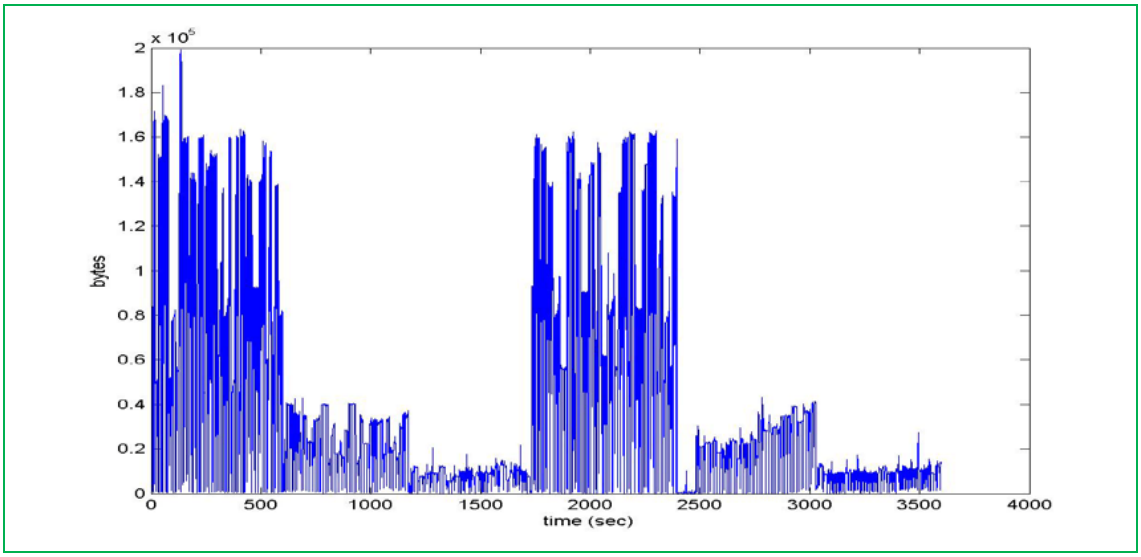

Figura 36 – HTTP + FTP + snap quality (download).

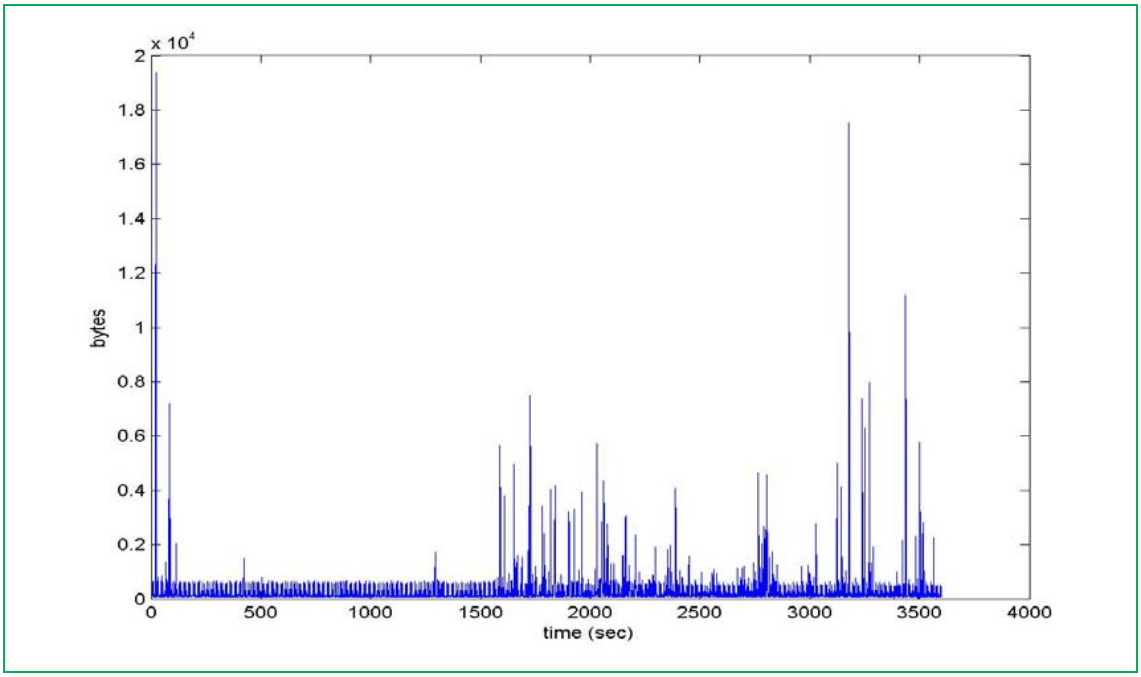

Figura 37 – HTTP + Streaming Radio + port scan (download).

Cenário HTTP + streaming video

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	91,88	93,62	77,71	97,9
Snap Interval	99,85	100	95,45	100
Snap Quality	100	100	100	50
File Transfer	98,98	93,10	99,13	94,08
Sub All	96,95	99,15	89,13	98,55

Tabela 14 - Resultados obtidos com o Cenário HTTP + Stream Vídeo.

No caso do cenário com *stream* de vídeo e HTTP, pode-se verificar que os resultados obtidos foram muito bons, tanto em download como em upload, tanto no caso em que foi feita simulação com o tráfego capturado no cenário, como no caso em que temos a soma dos tipos de tráfego constituintes do cenário. È de notar que no caso em que foi utilizada a funcionalidade de *snapshot* do subseven e foi alterada a qualidade dos *snapshots* (soma dos tipos de tráfego, em upload), a probabilidade de acerto foi de 50%, um valor muito abaixo das outras probabilidades de acerto obtidas neste cenário.

Cenário Stream Rádio + RDC

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	99,13	100	86,45	100
Snap Interval	97,29	100	98,69	100
Snap Quality	100	100	100	50
File Transfer	99,85	99,85	100	99,42
Sub All	98,86	100	95,50	99,42

Tabela 15 - Resultados obtidos com o Cenário Stream Rádio + RDC.

Neste cenário (*stream* rádio + RDC) pode-se verificar que as probabilidades de acerto obtidas à saída da rede neuronal foram muito boas para todas as experiências realizadas com as diversas funcionalidades do *subseven*.

Na experiência em que é utilizada, a funcionalidade de *port scan* do *rootkit* e no caso em que foi feita a soma dos tipos de tráfego constituintes do cenário, a probabilidade de acerto foi de 86%, um bocado inferior a obtida com o resto das simulações.

Cenário HTTP + Skype

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	93,47	97,10	58,06	68,18
Snap Interval	89,13	96,95	86,52	97,10
Snap Quality	91,69	92,60	94,78	50
File Transfer	87,82	95,50	89,13	92,37
Sub All	93,54	90,87	79,13	97,70

Tabela 16 - Resultados obtidos com o Cenário HTTP + Skype.

Neste cenário a probabilidade de acerto foi inferior à dos cenários anteriores. Por exemplo no caso em que foi utilizada a funcionalidade de transferência de ficheiros do *subseven,* (dados obtidos do cenário download) a probabilidade de acerto foi de 87%, já no caso da soma dos tipos de tráfego constituinte deste cenário, pode-se verificar que a probabilidade de acerto à saída da rede neuronal foi inferior á probabilidade de acerto obtida com os dados do cenário. Na figura 38 está representado o *throughput* neste cenário ao longo dos 3600 segundos. Podemos verificar que este tráfego tem um perfil muito variável, sendo difícil para a rede neuronal identificar este perfil, daí as probabilidades de acerto à saída da rede neuronal serem inferiores aos cenários anteriores.

Cenário Skype + Jogos

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	87,24	100	54,05	80,20
Snap Interval	92,75	100	87,39	100
Snap Quality	100	100	98,98	50
File Transfer	94,86	94,72	92,65	92,17
Sub All	98,69	100	82,17	96,23

Tabela 17 - Resultados obtidos com o Cenário Skype + Jogos.

Neste cenário as probabilidades de acerto à saída da rede neuronal foram boas, a volta dos 90% para download (perfil de tráfego do cenário), para a soma dos tipos de tráfego constituintes do cenário as probabilidades de acerto foram inferiores. Na figura 39 pode-se visualizar, o *throughput* provocado por este cenário, podemos verificar que este cenário tem um perfil de tráfego muito variável induzido pelo *skype* que varia os seus parâmetros de transmissão de vídeo em função das condições de rede, torna-se assim difícil para a rede neuronal identificar este tipo de tráfego quando na presença das várias funcionalidades do *rootkit subseven*.

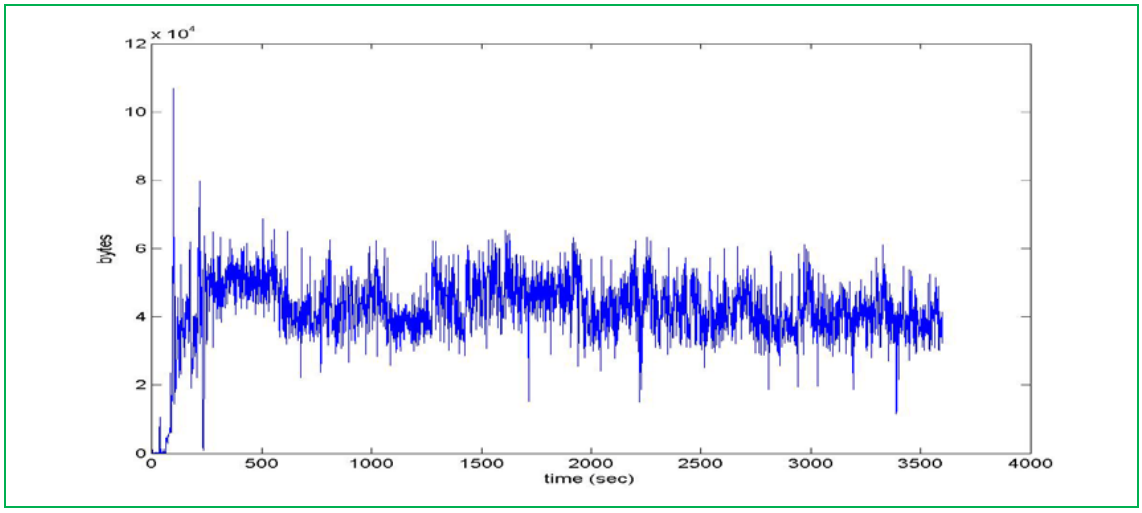

Figura 38 – Skype + HTTP + port scan (download).

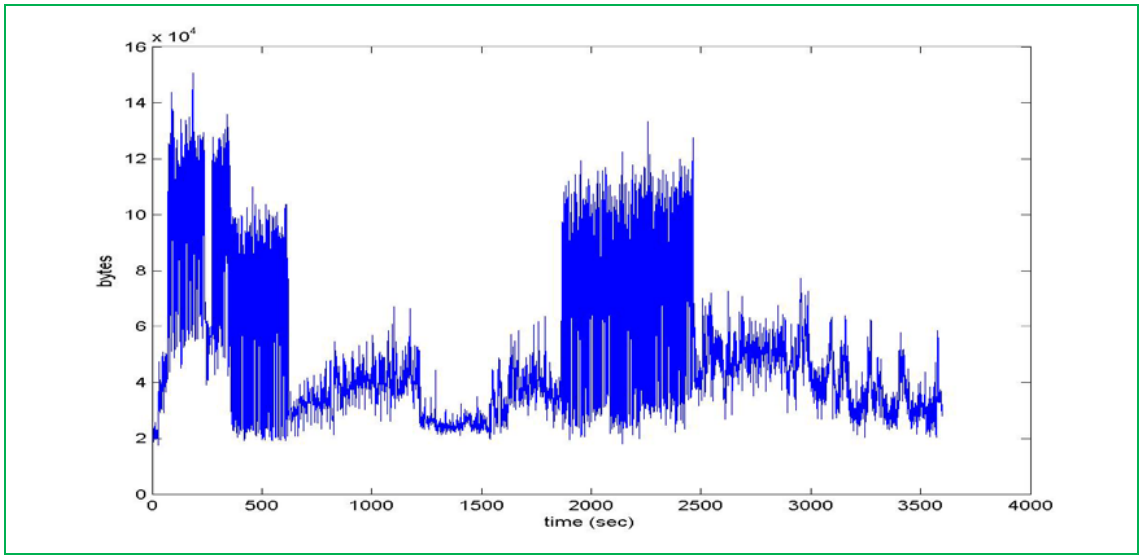

Figura 39 – Skype + Jogos + port scan (download).

Cenário Skype + Stream Rádio

Funcionalidade SubSeven	Acerto (%)		Acerto (%), Soma Tráfegos	
	Download	Upload	Download	Upload
Port Scan	97,50	85,38	56,59	78,88
Snap Interval	97,24	94,49	83,18	100
Snap Quality	97,39	97,97	96,95	50
File Transfer	87,09	92,37	83,87	92,37
Sub All	94,92	89,71	76,66	99,70

Tabela 18 - Resultados obtidos com o Cenário Skype + Stream Rádio.

Neste cenário os resultados obtidos foram bons. Pode-se verificar que a probabilidade de acerto para o caso em que foi simulado, com o tráfego capturado durante a simulação do cenário, foi inferior a probabilidade de acerto obtida com a soma dos tipos de tráfego constituintes do cenário. È também de salientar que a probabilidade de acerto de download é superior a probabilidade de acerto de upload.

5. Conclusão e Trabalho Futuro

5.1. Conclusão

O sistema desenvolvido e testado mostrou uma elevada eficácia na detecção de tráfego ilícito grado pelo *rootkit*. Na generalidade a probabilidade de acerto obtida à saída da rede neuronal foi muito boa. Assim, verificou-se que a probabilidade de acerto à saída da rede neuronal é inferior quando utilizamos para identificar o tráfego ilícito apenas tráfego de upload, quando é utilizado o tráfego de download esta probabilidade é superior. Verificou-se na experiência em que foi utilizado a soma dos tipos de tráfego constituintes de cada cenário para identificar o tráfego ilícito, que a probabilidade de acerto é inferior no caso da soma, logo superior no caso em que temos os dados capturados em cada cenário. Verificou-se também que a probabilidade de acerto foi superior, no caso em que temos apenas um serviço a correr na máquina de testes, enquanto a probabilidade de acerto obtida com os cenários, são constituídos com vários tipos de tráfego com diferentes características foi inferior. Isto deve-se a entropia adicionada por vários perfis de tráfego, que criam um novo padrão de identificação, o que torna difícil a identificação do tráfego ilícito, por parte da rede neuronal.

Relativamente a todas as experiências podemos concluir que a rede neuronal comporta-se muito bem e consegue identificar correctamente diversos tipos de tráfego ilícito que é gerado pelo *rootkit subseven*, o que pode ser muito útil no futuro uma vez que se existir um elemento com inteligência que consiga aprender o perfil de um utilizador, em tempo real, assim a rede neuronal poderá identificar quando um PC está comprometido. Poderá funcionar como mecanismo não só de detecção mas também como mecanismo de prevenção.

Os resultados obtidos permitem concluir que a utilização das redes neuronais para identificação de intrusões poderão ter grande utilidade no futuro, porque ultrapassam algumas das limitações dos IDS actuais, que perante técnicas como encriptação dos dados e multiplexagem de portos, tornam-se ineficazes. A utilização de redes neuronais para a análise e identificação de intrusões é de fácil implementação. Esta aproximação para identificar intrusões é baseada no perfil de tráfego de um determinado utilizador, ou de uma rede, e é capaz de categorizar o tráfego como lícito ou ilícito, mediante a identificação efectuada pela rede neuronal. Apesar do tráfego gerado nas simulações ter sido obtido num cenário controlado, os resultados obtidos mostram que esta é uma

solução que poderá ser utilizada no futuro em conjunto com outras técnicas de identificação de intrusões.

A utilização soma dos fluxos parciais para identificar o tráfego ilícito, não é uma solução tão boa como no caso em que temos os dados capturados no cenário, visto que os resultados obtidos nas experiências em que foi feita, a soma dos dados dos tipos de tráfego constituintes do cenário, foram em geral inferiores aos resultados que foi obtido com o tráfego capturado durante as experiências.

Um ponto fraco da utilização de redes neuronais para identificação de intrusões é que se a quantidade de tráfego gerada por uma intrusão for demasiado baixa, a rede neuronal não conseguirá distinguir o tráfego licito do tráfego ilícito, no entanto quando temos um cenário em que existe em operação uma *botnet*, aí a quantidade de tráfego gerada já é elevada e permite a rede neuronal identificar que existem maquinas comprometidas na rede. Outro ponto fraco desta solução é que necessita de um número elevado de amostras para funcionar correctamente, sendo que se o número de amostras for pequeno a probabilidade de acerto à saída da rede neuronal decresce drasticamente.

5.2. Trabalho Futuro

Uma experiência interessante a realizar no futuro seria criar várias simulações para o mesmo serviço, durante um intervalo de tempo grande para ter muitas amostras, ou então criar uma só simulação, mas de duração maior e dividir os dados em conjuntos, e simular na rede neuronal, de modo a obter várias probabilidades de acerto à saída da rede neuronal, e então calcular a probabilidade média de acerto. Com os dados capturados ainda tentou-se dividir os dados, em subconjuntos de modo a simular estes na rede neuronal, mas as probabilidades de acerto obtidas à saída da rede neuronal foram muito baixas, devido aos conjuntos possuírem poucas amostras, o que torna a identificação do tráfego ilícito difícil.

Outra experiência interessante a realizar seria utilizar como entradas da rede neuronal para identificação do tipo de tráfego, tanto tráfego de upload como tráfego de download. Poderá ser também utilizado o número de pacotes combinado com a quantidade de bytes/s, tanto em upload como em download para identificação do tráfego ilícito. Assim, poderiam ser utilizados mais factores na identificação do tipo de tráfego presente nas amostras. O próximo passo será implementar esta solução num cenário real

e capturar em *real time* os dados que comprovem, os resultados obtidos aqui nestas simulações.

6. Referências Bibliográficas

1) Paulo Salvador, António Nogueira, Ulisses França, Rui Valadas, "Detection of Illicit Traffic using Neural Networks", /International Conference on Security and Cryptography (SECRYPT 2008)/, 26-29 Julho, 2008, Porto, Portugal.

2) http://www.sans.org/reading_room/whitepapers/malicious/1299.php, Bots & Botnets, 2008. [cited 14 April 2008]; Available from: http://www.sans.org.

3) http://www.cert.org/archive/pdf/Botnets.pdf. Definition of Botnets, 2008. [cited 09 june 2008]; Available from http://www.cert.org.

4)http://www.answers.com/topic/malware. *WikiAnswers, definition of Malware*. 2008 [cited 23 September 2008]; Available from: http://www.answers.com/.

5) http://www.forum-intrusion.com/archive/Snort_20_v4.pdf. Snort analysis. 2008 [cited 20 September 2008]; Available from: http://www.forum-intrusion.com.

6)http://nsrc.org/workshops/2008/ait-wireless/kemp/nmap-nessus-snort.pdf. IDS analysis. 2008 [cited 20 November 2008]; Available from http://nsrc.org.

7)http://www.caida.org/workshops/dns-oarc/200507/slides/oarc0507-Dagon.pdf. Cooperative Association for Internet Data Analysis, *definition of botnet*. 2008 [cited 23 September 2008]; Available from: http://www.caida.org/.

8) http://lowkeysoft.com/proxy/server.php - *«Proxy Botnet Analisys»*. 2008 [cited 3 June 2008]; Available from: http://www. lowkeysoft.com/.

9) http://howto.wired.com/wiki/Build_your_own_botnet_with_open_source_software#Key_Components_include. *«Build your own botnet with open source»* [cited 23 January 2008]; Available from: http://www. howto.wired.com.

10) http://linfo.org/rootkit.html. *The Linux Information Project, definition of Rootkit*. 2008 [cited 23 January 2008]; Available from: http://linfo.org/ .

11) http://www.securityfocus.com/infocus/1214. Network Intrusion detection system. 2008 [cited 15 Sepetember 2008]; Available from: http://www.securityfocus.com.

12) http://www.snort.org/. *Geotumba - Temos Um Motor de Busca Geográfico Alternativo*. 2008 [cited 18 September 2008]; Available from http://www.snort.org/.

13) http://www.ossec.net/.*Host based Intrusion detection system*. 2008 [cited 2 September 2008]; Available from: http://www.ossec.net/.

14) http://www.cfengine.org/. *Suite of programs for maintaining and configuring unix-like machines.* 2008 [cited 10 September June 2008]; Available from: http://www.cfengine.org/.

15) http://www.nessus.org/nessus/. *Network vulnerability scan.* 2008 [cited 13 September 2008]; Available from: http://www.nessus.org/.

16) http://nmap.org/. *Free Tool for network exploration and security audinting*. 2008 [cited 23 september 2008]; Available from: http://nmap.org/.

17)Mehdi MORADI, Mohammad ZULKERNINE, http://research.cs.queensu.ca/~moradi/148-04-MM-MZ.pdf. *Neural network system for intrusion detection and classification of attacks*. (Paper) 2008 [cited 26 February 2008]; Available from: http://research.cs.queensu.ca.

18) Zheng Zhang, Jun Li, C.N. Manikopoulos, Jay Jorgenson, Jose Ucles, http://www.itoc.usma.edu/Workshop/2001/Authors/Submitted_Abstracts/paperT2A2(19).pdf. *HIDE: a Hierarchical Network Intrusion Detection System Using Statistical reprocessing and Neural Network* Classification. (*Paper)* 2008 [cited 2001]; Available from: http://www.itoc.usma.edu.

18) http://www.mathworks.com/. *Mathlab Reference guide*. 1998 [cited 24 May 2008]; Available from: - http://www.mathworks.com/.

19) http://www.softpanorama.org/Security/intrusion_detection.shtml.*The Official Microsoft ASP.NET* Architectural Issues of Intrusion Detection Infrastructure in Large Enterprises. 2008 [cited 13 February 2008]; Available from: http://www.softpanorama.org.

20) http://osiris.shmoo.com/. *Host integrity monitoring system*. 2008 [cited 20 September 2008]; Available from: http://osiris.shmoo.com/