UNIVERSITY OF SOUTHAMPTON

An Intelligent Firewall Architecture Model To Detect Internet-Scale Virus Attacks

by

Vicky InSeon Yoo

A thesis submitted in fulfillment for the
degree of Master of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

February 2004

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Master of Philosophy

by *Vicky* InSeon Yoo

My MPhil Thesis is based on research in progress concerning an Intelligent Firewall Architecture Model against Internet-scale viruses. An Internet-scale virus is defined to be a piece of code or a program that performs unintended tasks and brings unintended side effect. The Intelligent Firewall focuses on risk management against novel attacks. A main purpose of this project is to integrate a packet-based classification engine and a smart detection engine into a firewall.

Classification is based on finding proper information and establishing links between data, on the other hand, recognition is based on making a decision about the information after classifying the data. I would like to use these terms with these concepts in my thesis. The packet-based classification engine aims at classifying Internet-scale virus packets apart from normal packets using packet header and payload, and then the smart detection engine deals with the stream of filtered packets from the classification engine which selected them as having a high probability of containing malicious content. To classify and recognize malicious packets from normal packets, I surveyed statistics of current Internet-scale viruses and analyzed malicious packets.

I describe current Internet-scale viruses' effects on the Internet and security systems' problems. I analyse features of present network security systems: firewalls, intrusion detection systems, and anti-virus servers and examine related work to cope with disadvantages of the systems. I disuss the current Internet-scale virus trend through Internet-scale viruses' statistics. I will present the concepts of the Intelligent Firewall by discussing statistics and a survey on several current Internet-scale virus attacks. Detailed studies concerning the analysis of the virus infection processes and security holes are beyond the scope of this thesis. Using this analysis and investigation, I propose an Intelligent Firewall model which has several packet-based components, especially the packet-based classification with Bayesian Networks, and the smart detection engine with a Self-Organizing Map. This thesis will be beneficial to other security systems, including router parts and anti-virus detection systems.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, thanks to my Lord, Jesus Christ. Whenever I am feeling down, he holds my hand to steady me. When I am suffering from tiredness, sickness and numerous difficulties, he cures me and gives me more strength to keep on with my work. He stays with me constantly, as my companion, when I feel lonely and afraid, In contrast with my younger Christian years, these challenging experiences have brought about a new closeness with Jesus Christ. Living and studying alone in England has encouraged me to grow stronger and to gain maturity. With Jesus Christ at my side, I have received many blessings, among them my friendships with generous people.

You will be made rich in every way
so that you can be generous on every occasion,
and through us your generosity will result in thanksgiving to God.
- 2 Corithians 9:11.

I would like to thank my supervisor, Dr. Ulrich Ultes-Nitsche. I am very glad to have met him; his guidance encourages me to do as I want, yet steers me to do as I should. I am certain that this dissertation is a far better one through his continuous advice. While preparing this thesis, my parents, my two younger sisters - HyoSeon and DuSeon - and my friends - SooHee Ryu, JeeSun Kim, Muan Yong NG, April Lloyd, HyunSang Kang, Liu Shu Hui, Rod Rumble, EunHee Park, KyungEun Lee, Tony & Anne Smallwood, Anne Newton, Rachel Trost, JungMin Kim, YoonYoung Hur, Clare Best - have prayed and supported me in various ways. Moreover, thanks to the DSSE group colleagues, especially Professor Peter Henderson, Professor Michael J. Butler, Dr. Juan C. Augusto and Yvonne M. Howard. I will have unforgettable memories of DSSE, in ECS.

Whoever sows sparingly will also reap sparingly,
and whoever sows generously will also reap generously.
- 2 Corithians 9:6.

Vicky, InSeon Yoo
Southampton in England
December 2003

To my mother, and those cherished memories of Korea...

Chapter 1

Introduction

Today, the Internet is used for many purposes. In accordance with this wide-spread Internet usage, Internet security breaches are growing. Internet-scale viruses are one of the major causes of the rising number of security breaches. A serious security risk is the propagation of Internet-scale viruses through email attachments. An Internet-scale virus is defined as a piece of code or a program that performs unintended tasks and brings unintended side effects such as damaging a system or network, obtaining secure information without permission, putting a system or network under heavy load, and so on. Nowadays some companies, which recognize the importance of security, adopt security systems such as firewalls, intrusion detection systems, and anti-virus servers. It is not easy to use all security systems, owing to high cost. Furthermore, it is not enough to protect a company's system with only a single security system, because each security system has different features, in which particularly protection against Denial of Service (DOS) and Internet-scale virus attacks is increasingly important.

1.1 Increasingly Serious Attacks

Internet-scale viruses include file viruses, file worms, and network worms. These Internet-scale viruses are being spread via systems' security holes, emails, messengers, etc. A virus is a piece of code that adds itself to other programs and cannot run independently. As Microsoft Windows became popular, Windows viruses and Windows-application-derived viruses using Visual Basic for Applications (VBA) spread widely. A common way of Windows virus dissemination is through emails. In addition, a worm is a program that can run by itself and propagate a fully working version of it to other machines. A network worm is a worm, which copies itself to another system by using common network facilities, and causes execution of the copy on that system. A

recent serious attack was Code Red. The Code Red worm is a malicious self-propagating code (CERT/CA-2001-23 2002) that spreads surreptitiously through a hole in certain Microsoft software. The Code Red, which left computers open to hijacking, has caused a lot of traffic being sent, clogging the bandwidth on the Internet. An infected system will show an increased processor and network load. The worm could easily permit hackers to take control of hundreds of thousands of infected machines.

The biggest impact of these worms is that their propagation creates a DOS attack in many parts of the Internet, because of the huge amount of traffic generated. DOS attacks can interrupt services by flooding networks or systems with unwanted traffic. A service will be denied, because the network or system is overwhelmed. Distributed systems based on the client/server model have become increasingly popular. Therefore Distributed Denial of Service (DDOS) attacks are also getting escalated. In an DDOS, an attacker controls a number of handlers. A handler is a compromised host with a special program running on it. Each handler is capable of controlling multiple agents. An agent is a compromised host, which is responsible for generating a stream of packets that is directed towards the intended victim.

1.2 Classification and Recognition

In my MPhil thesis, I am considering classification and recognition. We can give the following situation: We may be given a set of observations with the aim of establishing the existence of classes or clusters in the data. Or we may know for certain that there are so many classes, and the aim is to establish a rule whereby we can classify a new observation into one of the existing classes.

The task of classification could cover any context in which some decision or forecast is made on the basis of currently available information, and a classification procedure is then some formal method for repeatedly making such judgments in new situations.

If we create some classifier, the classifier should be considered of accuracy, speed, comprehensibility and time to learn (D.Michie 1994).

1. Accuracy. There is the reliability of the rule, usually represented by the proportion of correct classifications, although it may be that some errors are more serious than others, and it may be important to control the error rate for some key class.

2. Speed. The speed of the classifier is a major issue in real critical circumstances. 90% accurate may be preferred over one that is 95% accurate if it is 100 times faster.

3. Comprehensibility. If it is an administrator that must apply the classification procedure, the procedure must be easily understood else mistakes will be made in applying the rule. It is important also that the administrators believe the system.

4. Time to Learn. Especially in a rapidly changing environment, it may be necessary to learn a classification rule quickly, or make adjustments to an existing rule in real time.

I shall assume that the problem concerns the construction of a procedure that will be applied to a continuing sequence of cases, in which each new case must be assigned to one of a set of predefined classes on the basis of observed attributes or features.

The other is Pattern Recognition. There are many kinds of patterns; visual patterns, temporal patterns, logical patterns. Using a broad enough interpretation, we can find pattern recognition in every intelligent activity. No single theory of pattern recognition can possibly cope with such a broad range of problems. There are several models, statistical pattern recognition, syntactic or structural pattern recognition, knowledge-based pattern recognition and so on. These pattern recognitions could be viewed as a classification.

However, I would like to define both of them like this. Classification is based on finding proper information and establishing links between data, on the other hand, recognition is based on making a decision about the information after classifying data. I would like to use these terms with these concepts in my thesis. The packet-based classification engine classifies packets into packet classes such as HTTP traffic, SMTP traffic, and FTP traffic. In addition, it makes a decision whether the packet class is filtered into the smart detection engine or dropped according to its probability of containing malicious content, which is deduced by Bayesian Networks. On the other hand, the smart detection engine deals with the filtered packets, which have a high probability of being malicious, from the packet-based classification engine. The smart detection engine deals with virus infected file to detect the virus part within the file. The infected file has a particular type of file structure one can recognize, but the file does not have a specific border line to distinguish the virus part from the remaining content of the file. Using a Self-Organizing Map, the smart detection engine aims at recognizing categorized patterns.

1.3 Reader's Guide

My MPhil Thesis is based on research in progress concerning an Intelligent Firewall Architecture Model against Internet-scale viruses. The Intelligent Firewall focuses on risk management against novel attacks. This packet-based recognition against Internet-scale viruses is one research part of the Janus (U.Ultes-Nitsche and IS.Yoo 2002) (IS.Yoo and

U.Ultes-Nitsche 2002) Project. A main purpose of this project is to integrate a packet-based classification engine and a smart detection engine into a firewall. The packet-based classification engine aims at classifying Internet-scale virus packets apart from normal packets using packet header and payload, and then the smart detection engine deals with the stream of filtered packets from the classification engine which selected them as having a high probability of containing malicious content. To classify and detect malicious packets from normal packets, I surveyed statistics of current Internet-scale viruses and analysed malicious packets. I will present the concepts of an Internet-scale virus and the Intelligent Firewall by discussing statistics and a survey on several current Internet-scale virus attacks. Detailed studies concerning the analysis of the virus infection processes and security holes are beyond the scope of this thesis. In this MPhil thesis, I have developed the general concepts that the Intelligent Firewall will comprise. My architecture decisions were based on a thorough analysis of available data about viruses as well as a literature study of suitable approaches for the detection of novel viruses. This led to the Intelligent Firewall architecture, which contains the packet-based classification engine and the smart detection engine. The considered approaches will be beneficial to other security systems, including router parts and anti-virus detection systems.

The rest of this dissertation is organized as follows. In Chapter 2, I describe current Internet-scale viruses' effects on the Internet and security systems' problems. In Chapter 3, I analyse features of present network security systems: firewalls, intrusion detection systems, and anti-virus servers and examine related work to cope with disadvantages of the systems. In Chapter 4, I discuss current Internet-scale virus trends through Internet-scale virus statistics. In addition, I classify Internet-scale viruses to analyse, and examine malicious virus patterns. Using this analysis, I propose an intelligent firewall model in Chapter 6.

Chapter 5 addresses potential research models for classification and recognition. Several research models are possible. However, in this dissertation, I explain Bayesian Networks, Neural Networks, and Data Mining Techniques. I discuss what Bayesian Networks are, what kinds of features the networks have, and why they are suitable for my project. Moreover, I address what the main components of Neural Networks are, how they are trained, and what Self-Organizing Maps and Probabilistic Neural Networks are. In Chapter 6, I show what kind of architecture is needed for an Intelligent Firewall, what the components are, and how the packet-based classification engine and the smart detection engine work with Bayesian Networks and Neural Networks. Finally, in Chapter 7, I make a conclusion and mention future work.

Chapter 2

Motivations

2.1 Internet-Scale Viruses

Internet security breaches are growing. Viruses are the major cause of the rising number of security breaches. With the advantage of high-speed Internet access, many organizations and companies put their websites and their confidential information at high risk when establishing a connection to the Internet that is not securely implemented. Nowadays companies which recognize the seriousness of security adopt security systems such like firewalls, intrusion detection systems, and virus scanning proxy servers. It is not easy to take all security systems owing to high cost. However, it is also not enough to prevent their system with one security system because each security system has different features in present days; particularly Denial of Service (DOS) attacks and virus attacks are becoming serious.

As I defined in the Introduction, an Internet-scale virus is a piece of code or a program that performs unintended tasks and brings unintended side effects. Internet-scale viruses are more than 90% of detected malicious occurrences, at least in the University of Southampton at the time of writing this thesis (see Figure 2.1). I present statistics about viruses which were detected while entering the ECS department in Appendix B.

2.2 Current Network Security Systems' Problems

DOS attacks are easy to perpetrate and almost impossible to defend against even whilst firewalls and Intrusion Detection Systems (IDSs) are installed. Even if the Intrusion Detection Systems generate alerts, log packets, send emails, and call pagers, the attacker could still get in, and by the time somebody could respond the damage would be done. A

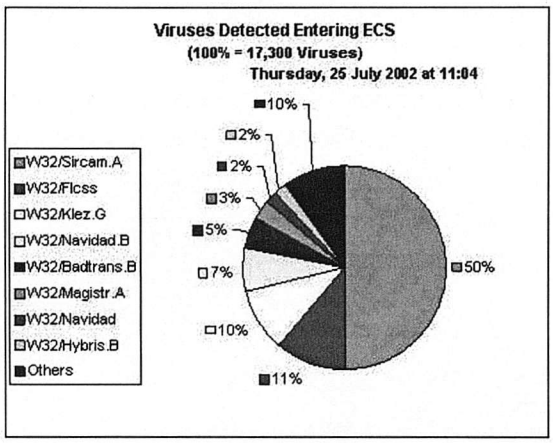

FIGURE 2.1: Viruses Detected Entering ECS

malicious attacker could spoof attacks from many sources and effectively deny everybody access to the server. This is considered to be an unacceptable risk. A firewall would be of no help either. The web server has to remain available to the public and the vulnerability is in the web server software such as IIS. A firewall has no way of determining if a request being sent to a web server is benign or malicious. While the firewall could stop traffic to ports that do not need to be publicly accessible, it is useless in this situation. Increasingly complex security scenarios and incorrect configurations contribute to a firewall's inability to provide gateway security. A firewall is also only able to deal with traffic that passes through the firewall, with all internal traffic completely unchecked.

When a virus associated with DOS spreads through the Internet, virus scanning proxy servers and IDSs or Firewalls must cooperate to prevent these attacks. Although anti-virus servers and software have served users well for a lengthy period, today's fast-paced technology means that viruses travel much faster than signature updates can keep up with. This kind of software often relies on databases containing these virus signatures, which catch and define viruses. It is therefore essential to ensure that the database of signatures is as up-to-date as possible. This implies that a mechanism guarantees that the latest signatures are updated, as and when new viruses are detected. Therefore a process of automatic updating of signatures, a built-in feature found in most anti-virus products, as well performing the necessary upgrade maintenance are critical actions. These are steps that cannot be left up to human processing, but must be automated to be kept up-to-date at all times. This type of solution has become vital due to the different ways that a virus can enter an organisation. This means that protection is needed at each of the levels, stopping a virus where it enters rather than having to clean up after it has spread.

Considering weaknesses of current security systems, the Intelligent Firewall focuses on risk management against novel attacks. The packet-based classification engine classifies

malicious packets from normal packets using packet header and payload. Then the smart detection engine deals with the stream of filtered packets from the classification engine which selected them as to have a high probability of malicious contents. This novel firewall model will enable one to detect novel attacks.

Chapter 3

Analysis of Present Network Security Systems

Anti-virus servers examine network traffic, aiming to prevent malicious code from entering network nodes by detecting known malicious-code patterns. Apparently, they can detect only known viruses. New viruses will only become detectable after their pattern characteristics have been analyzed and are made available. Current intrusion detection systems (IDSs) do not prevent an intrusion from happening; they only detect and report it. When a virus associated with a DOS attack spreads through the Internet (e.g. the CodeRed), anti-virus servers and IDSs should co-operate to prevent such an attack. However, although anti-virus servers and IDSs are installed, new virus information needs to be updated constantly. Furthermore, firewalls are used to guard and isolate connected segments of inter-networks. Inside network domains are protected against outside un-trusted networks, or parts of a network is protected against other parts (Schuba 1997). These firewalls use only TCP/IP headers - no payload information is used - to decide whether data packets are safe or not. I will discuss the different network security systems in this chapter.

3.1 Firewalls

Firewalls are used to guard and isolate connected segments of inter-networks. Inside network domains are protected against outside un-trusted networks, or parts of a network are protected against other parts (Schuba 1997). There are three types of Firewalls. I will try to explain the strengths and weaknesses of each firewall, then introduce existing approaches for this.

3.1.1 Types of Firewalls

First is the packet-filter firewall. Packet filtering is mainly focused on the ability to accept or deny packets based on analysing the packets' heades. It is not suitable for defence means against intruders, just appropriate as another security measure. In addition, large lists of rules can be difficult to manage. Packet filters by themselves are not adequate to secure a complex network from attack. Main strengths of packet filter firewalls are speed and flexibility (John Wack and Pole 2002). These systems can be used to secure nearly any type of network communication or protocol. They are easy to be deployed into nearly any enterprise network infrastructure. However, they cannot prevent attacks and their logging functionality is very limited, because they do not examine upper-layer data. They do not support advanced user authentication schemes. They cannot, for instance, detect a network packet in which the OSI Layer 3 addressing information has been altered.

Second is the stateful inspection firewall. Stateful inspection firewalls add Layer4 awareness to the standard packet filter architecture (John Wack and Pole 2002). Stateful inspection adds some intelligence to packet filtering. With stateful inspection, the firewall keeps track of a session so it knows if the session is already active. It uses this information plus a list of rules to determine whether a packet should be blocked or forwarded. Stateful inspection is a combination of packet filtering and a circuit level gateway. There is no examination of the application information. These systems share the strengths and weaknesses of packet filter firewalls. The actual stateful inspection technology is relevant only to TCP/IP. Moreover it has very high costs because the state of connection is monitored at all times.

Third are application-proxy gateway firewalls. Application gateways are software based. All the data in a packet is examined, including the application layer, for content. If a field meets a set of predefined rules, the gateway creates a path between two protocols between the remote host and the internal host. Since every detail of the packet is examined, application gateways are the most secure type of firewall. These tools also have very sophisticated logging facilities. The disadvantage of an application gateway is that a proxy application must be created for each networked service. Some application gateways require modified clients. Application gateways also have the lowest throughput. According to (John Wack and Pole 2002), application-proxy gateway firewalls have more extensive logging capabilities, are capable of authenticating users directly, and can be made less vulnerable to address spoofing attacks. These systems are not generally well suited to high-bandwidth or real-time applications.

3.1.2 Related Work

Related to our concept of an intelligent firewall, there are two existing approaches I would like to discuss here. The first one in (Xu and Singhal 1999) proposes an ATM firewall using a proxy cache, which uses a QoF (Quality of Firewalling) scheme. Its main components are call screening, proxy, traffic monitoring service, packet filtering service, and firewall management. These combined components determine a packet's safeness. The packet-filtering service inspects the headers of IP packets to block unsafe packets, while allowing safe packets to pass. The traffic-monitoring service checks the packet headers against the traffic-monitoring rules, which are similar to the packet-filtering rules.

The second approach in (J.Hughes 1996), which was referred to in (Xu and Singhal 1999), proposes a policy cache architecture. To determine whether or not a packet is safe, only the first cell will be checked, which contains the IP header, protocol, TCP/UDP ports and TCP flags. However, there are limitations: IP packets with IP option fields are not accepted, because IP options can be as large as 40 bytes and may push the TCP headers to the second cell. Using CAM (Content Addressable Memory) to cache a safe header is not a scalable solution. CAM cannot scale to a large size due to technological constraints and is extremely expensive.

Both of these papers use only TCP/IP headers - no payload information is used - to detect whether data packets are safe or not, even though they aim to develop a new firewall architecture. It does not seem that inspecting only header information is sufficient to overcome weaknesses of firewalls.

3.2 Intrusion Detection Systems

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network (Bace and Mell 2001). IDS (Intrusion Detection Systems) are software or hardware products that automate this monitoring and analysis process. The definition of IDS does not include preventing the intrusion from occurring, only detecting it and reporting the intrusion to an operator (Jai Sundar Balasubramaniyan and Zamboni 1998).

3.2.1 Types of Intrusion Detection Systems

First is the network-based IDS. This monitors network traffic on the wire for specific activities/signatures that represent an attack (Richards 1999). Strengths of this are to

monitor a large network and to be little impact upon an existing network (Bace and Mell 2001). Moreover, it detects malicious and suspicious attacks as they are occurring in true real-time and provides faster response and notification to the attack at hand (Laing 2000). It examines all packet headers for signs of malicious and suspicious activity and can also investigate the content of the payload. It uses live network traffic for its attack detection in real-time and a hacker cannot remove this evidence once captured. However, weaknesses of this are to have difficulties in processing all packets, it is possible for the system to fail to recognize an attack during high traffic and there is a need to analyze packets quickly (Bace and Mell 2001). It is also possible to misunderstand normal traffic as a malicious traffic. It is easy to trigger numerous false positives because of normal traffic looking very close to malicious traffic. In addition, the approach faces performance problems, especially with increasing network speeds, and resource exhaustion problems could occur (John Mchugh and Allen 2000). IDSs can suffer from resource exhaustion problems when they must maintain attack-state information for many attacked hosts over a long period of time. A network-based IDS does not control the network or maintain its connectivity (H.Ptacek and N.Newsham 1998). It follows that these systems are vulnerable to DOS.

Second is the host-based IDS. This focuses on a server and monitors specific user and application actions and logs entries (Richards 1999). Strengths referred by (Laing 2000) are that it could be possible to verify attacks; actual attacks or exploit's detection has been deemed as more accurate and less prone to false positives. It suits for system specific activity and monitoring key components. Weakness summarized by (Bace and Mell 2001) are that it is hard to manage, easy to be attacked and disabled as part of the attack, does not suit for detecting network scans and can be disabled by DOS attacks.

Third is the application-based IDS. This monitors interaction between users and applications, but is more vulnerable than host-based IDSs (Bace and Mell 2001).

3.2.2 Classification of Intrusion Detection

First is misuse detection (signature-based detection (Bace and Mell 2001) or signal detection (John Mchugh and Allen 2000)). It recognizes known "bad" behaviour (Sundaram 1996). Specific pattern matching is applied to a portion of a network packet (John Mchugh and Allen 2000). However, it needs to update signatures of new attacks constantly because it is unable to detect truly new attacks (Bace and Mell 2001).

Second is anomaly detection (anomaly-based detection; noise detection; unusual or abnormal with intrusions (John Mchugh and Allen 2000)). This detection is able to catch "bad" behaviour (Sundaram 1996) and unusual behaviour (Bace and Mell 2001). It produces information that can be used to define signatures for misuse detection (Bace and Mell 2001). However, it produces a large number of false alarms and its costs are high

because it requires extensive 'training sets' (Bace and Mell 2001). It is computationally expensive because of the overhead of keeping track of and updating several system profile metrics (Sundaram 1996).

3.2.3 Related Work

Intrusion represents both intrusion and misuse in (Jai Sundar Balasubramaniyan and Zamboni 1998). This paper dealt with intrusion detection systems using autonomous agents. An agent is an independently-running entity that monitors certain aspects of a host and reports abnormal or interesting behaviour. To perform data collection and analysis, agents send all their messages to the transceiver. However, this research focused on distributed architectures and it did not mention about how agents detect anomalous behaviour.

The other research (Kumar and H.Spafford 1995) proposed using a pattern matching approach to the representation and detection of intrusion signatures. One of interesting part of their server architecture model is that intrusion patterns are placed in queues having an appropriate priority level, and patterns in each queue are dealt with in a round robin fashion. This approach, which focused on software architecture and pattern matching, is not enough to prevent new attacks from happening as they change their attack pattern very rapidly.

3.3 Anti-Virus Servers

According to NUA (Online Internet Survey Company) Research, email is responsible for the spread of 80 percent of computer virus infections (Postini 2000). Various estimates place the cost of damage to computer systems by malicious email attachments in the rangeof 10-15 billion dollars last year alone. Many commercial systems are used in an attempt to detect and prevent these attacks. The most popular approach to defend against malicous program is through anti-virus scanners such as Symantec (Symantec 2002) and McAfee (McAfee 2002), as well as server-based filters that filters email with executable attachments or embedded macros in documents.

Currently monitoring systems exist through organizations such as WildList (WildList 2001), and the Trend Micro (Micro 2002). WildList is an organization consisting of 64 virus information professionals, who report all computer programs that they have received and positively identified as malicious. This list does not include those cases where an attachment is considered suspicious but not yet classified as malicious, or include any viruses not specifically reported by these 64 participants. This leaves computer systems

vulnerable to attack from unreported viral incidents (WildList 2001). Since the process of reporting is not automated, malicious program, especially the self-replicating program, can spread much faster than the warnings generated by WildList.

Trend depends on a proprietary virus scanner, HouseCall (HouseCall 2002) which integrates with the Trend Micro Control Manager to report information about actual virus infections. It attempts to predict virus outbreaks and prevent them pro-actively with the use of a dynamic map to analyze worldwide virus trends in real time (Micro 2002). However, since HouseCall is not widely used, Trend's data is incomplete. Furthermore, if Trend's database is not updated at the time that a virus infects a system, then the virus remains unreported.

An anti-virus server is defined as a server-side virus-checking programs. Its specific name depends on the company that produes it; for example, V3Netscan and V3VirusWall in Ahnlab.Inc.(Ahnlab 2002), and MailMonitor in Sophos (Sophos 2002). Anti-virus servers examine network traffic, aiming to prevent malicious code from entering network nodes by detecting known malicious-code patterns, for instance in an email attachment. Apparently, they can detect only known viruses. All of the major anti-virus vendors have produced networked products and systems that scan incoming email. However, because Trojan horses, worms and viruses can spread through local networks, shared hard drives and individual document files, as well as through the Internet, it is always necessary to have virus checking available on each client machine as well as on Internet gateways. Too often, patterns that identify new malware are not ready until days or even weeks after serious damage has been done. New viruses will only become detectable after their pattern characteristics have been analysed and are made available. Looking at techniques applied by other security systems, in our case intrusion detection systems, seems to benefit virus detection (M.Swimmer 2000).

There approaches have been successful in protecting computers against known malicious programs usually employing signature-based methods. Almost all anti-virus products claim that they can detect 100% of known viruses. However, we realize that hundreds of new viruses are created every month, they have not yet provided a means of protecting against unknown viruses, nor do they assist in providing information that may help trace those individuals responsible for creating viruses.

3.3.1 Related Work

Recently there have been approaches to detect new or unknown malicioius program by analyzing the payload of an attachment. The methods used include heuristics (S.R.White 1998), neural networks (J.O.Kephart 1994), and data mining techniques

(Matthew G.Schultz and Zadok 2001), (Matthew G.Schultz 2001). However, these methods in general do not perform well enough to detect malicious programs.

IBM researchers (O.Kephart and C.Arnold 1994) developed a statistical method for automatically extracting malicious executable signatures. Their research was based on speech recognition algorithms and was shown to perform almost as good as a human expert at detecting known malicious executables. Their algorithm was eventually packaged with IBM's anti-virus software.

(R.W.Lo and R.A.Olsson 1995) presented a method for filtering malicious code based on telltale signs for detecting malicious code. These were manually engineered based on observing the characteristics of malicious code.

Unfortunately, a new malicious program may not contain any known signatures so traditional signature-based methods may not detect a new malicious executable. In an attempt to solve this problem, the anti-virus industry generates heuristic classifiers by hand (Gryaznov 1999). This process can be even more costly than generating signatures, so finding an automatic method to generate classifiers has been the subject of research in the anti-virus community. To solve this problem, different IBM researchers applied Neural Networks to the problem of detecting boot sector malicious binaries (G.Tesauro and G.B.Sorkin 1996). A Neural Network is a classifier that aims to explore in human cognition. Because of the limitations of the implementation of their classifier, they were unable to analyse anything other than small boot sector viruses which comprise about 5% of all malicious binaries.

In similar work, (Arnold and Tesauro 2000) applied the same techniques to Win32 binaries, but because of limitations of the Neural Network classifier, they were unable to have the comparable accuracy over new Win32 binaries.

Chapter 4

Analysis of Internet-Scale Viruses

4.1 Internet-Scale Virus Statistics

According to Computer Virus Incident Reports (IPA/ISEC 2002) for May 2002 compiled by the Information-technology Promotion Agency Security Center (IPA/ISEC), the total number of reports for the first half of 2002 was 1.2 times greater than that of the year before. Moreover, the major reported viruses propagated via email, and the top 2 viruses were spread by exploiting security holes. Even though I am not satisfied with the categories of this survey, the results are quite interesting to look at more closely. I have produced charts based on the survey.

In Figure 4.1 and Figure 4.2, I use two terms: a security-hole virus and an email virus. A security-hole virus exploits a security hole. Even if distribution is via email, the way to infect a system is because of a security-hole, the virus belongs to the security-hole viruses. An email virus is distributed as an email attachment and does not exploit a security hole.

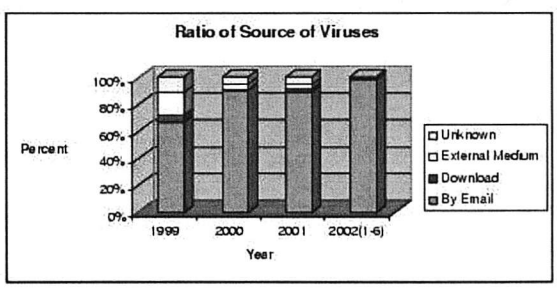

FIGURE 4.1: Ratio of Source of Viruses. As the Internet became popular, email attachments have been the major source of viruses since 1999.

Figure 4.2 shows new Internet-scale viruses' trends using security holes and emails. In 1999, macro viruses were the biggest portion, but from 2000, the ratio of email viruses has

increased significantly. Complex Internet viruses exploiting security holes have got escalated in 2002.

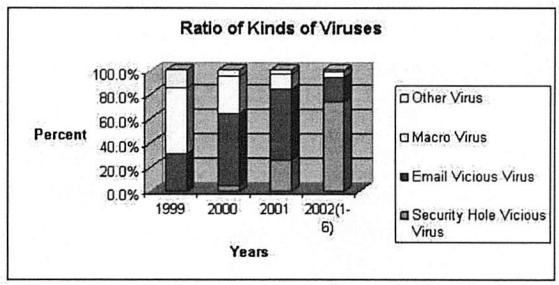

FIGURE 4.2: Ratio of Kinds of Viruses. The trend of virus is shown in this chart. In 1999, macro viruses were the biggest portion, but from 2000, the ratio of email virus has increased. Complex Internet viruses exploiting security holes have got escalated in 2002.

Figure 4.3 shows the percentage of several file extensions among Windows file worms. As one can see, most Internet-scale viruses [1] have Win32 executable format (about 86%) among the file/network worms.

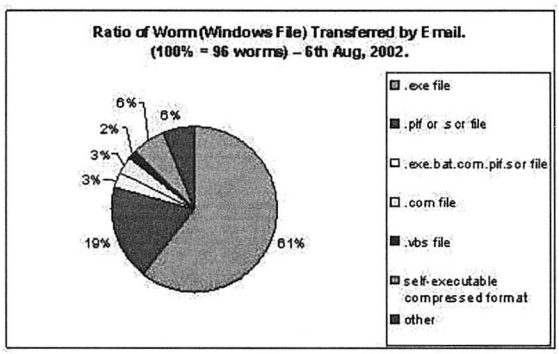

FIGURE 4.3: Ratio of Windows File Worms transferred by Email. This pie chart shows the percentage of several file extensions among Windows file worms. Win32 executables are about 86%, among these approx. 80% are transferred by email.

According to my survey into Internet-scale virus statistics based on the Virus Information of the Ahnlab (Ahnlab 2002), about 80% of Windows file worms are transferred via email, and approx. 61% have an .EXE file extension. I present this statistics survey record in Appendix A.

[1] Internet-scale viruses include file viruses, file worms and network worms (ref. Introduction). Figure 4.3 presents file worms or network worms, but the division is not clear in this chart, therefore, I use the term Internet-scale viruses rather than worms.

4.2 Classification of Internet-Scale Viruses

Most Internet-scale virus attacks have unselective targets. In this section, I discuss two types of blind targeting: social engineering attacks and security vulnerabilities attacks. I focus on the deployment of Internet-scale viruses rather than the detection of viruses in infected systems; before a machine is being infected, how do they spread, and after a system is infected, how do viruses spread from there to another machines. Internet-scale viruses show some characteristics, whilst their deployment is being processed which are useful in order to detect them. I analyze the deployment of Internet-scale viruses subsequently.

4.2.1 Social Engineering Attacks

Social Engineering is the hacker term for tricking unaware users into downloading or executing malicious software via email, Internet relay chat (IRC) or instant messaging (IM).

4.2.1.1 W32/SirCam Malicious Code

It spreads through email and potentially through unprotected network shares (CERT/CA-2001-22 2001). Once the malicious code has been executed on a system, it may reveal or delete sensitive information. The virus appears in an email message written in either English or Spanish with a seemingly random subject line. The email message contains an attachment whose name matches the subject line and has a double file extension (e.g. subject.ZIP.BAT or subject.DOC.EXE). The second extension is .EXE, .COM, .BAT, .PIF, or .LNK. The attached file contains both the malicious code and the content of a file copied from an infected system.

In addition, this worm includes its own SMTP client capabilities, which it uses to propagate via email. It determines its recipient list by recursively searching for email addresses contained in all *.WAB (Windows Address Book) files. As a result, its propagation via mass emailing can cause DOS conditions.

4.2.1.2 W32/Gibe Malicious Code

W32/Gibe is a Windows binary executable written in Visual Basic that is disseminated via email (CERT/IN-2002-02 2002). The email appears to be a patch from Microsoft updating the security system. The email message created by W32/Gibe tries to convince

users that the attached file is a patch supplied by Microsoft. In fact, the attached file is a copy of the malicious code.

This virus installs a backdoor(GFXacc.exe), which listens on port 12378/tcp. This may allow an intruder to gain access to the system and execute arbitrary commands. In addition, W32/Gibe mass-mails copies of itself to addresses found on the victim host. The victim and targeted sites may experience an increased load on the mail server when the malicious code is propagating.

4.2.1.3 W32/MyParty Malicious Code

This virus is written for the Windows platform. It spreads as an email attachment (CERT/IN-2002-01 2002). The attached file name is "www.myparty.yahoo.com.", which causes the web browser to run unexpectedly. '.com' is both an executable file extension in Windows and a top-level domain. The payload contained in W32/Myparty is non-destructive.

When this virus is executed, an email message is sent to a predefined address with a subject line of the folder where the W32/MyParty malicious code was stored on the victim's host. When it sends this message, it uses the SMTP statement HELO HOST to identify itself to the SMTP server. Meanwhile, the hard drive is scanned for *.WAB files, Outlook Express indexes and folders(.DBX) in order to harvest email addresses. Then copies of the malicious code are emailed to all the email addresses it could find. This step of mass mailing may be time-dependent. W32/MyParty may cause the default web browse to run unexpectedly. Likewise, the victim and targeted sites may experience an increased load on the mail server when the malicious code is propagating.

4.2.1.4 W32/Goner Malicious Code

This worm is a malicious Windows program distributed as an email file attachment and via ICQ [2] file transfers (CERT/IN-2001-15 2001). To a user, the file (gone.scr) appears to be a Windows screen saver. When the file is executed, the worm displays a splash screen then a false error message "Error While Analyze DirectX!" in an attempt to get an OK click on the error windows to launch the worm. It copies itself to the Windows system folder and modifies the Windows registry to execute itself. W32/Goner propagates by sending itself to all addresses listed in the Microsoft Outlook address book.

This code may also propagate via the ICQ messaging program. W32/Goner initiates a file transfer with any online users in the infected user's contact list. If the user on the

[2]ICQ stands for "I Seek You".

receiving end approves the transfer, the worm sends a copy of itself.

The worm looks for and terminates processes associated with many popular anti-virus and security programs. Furthermore, this worm may install DOS scripts for the mIRC Internet Relay Chat client. The worm may disable anti-virus and security software which are installed on the system. During propagation, sites may experience DOS conditions on hosts or email systems through which the worm is sent.

4.2.1.5 VBS/LoveLetter

This worm is created in VBS (Visual Basic Script language) and spreads in a variety of ways; email propagation, widows file sharing, IRC, USENET news, and possibly via WebPages (CERT/CA-2000-04 2000).

It arrives via email and is activated by a double click on the message attachment called LOVE-LETTER-FOR-YOU.TXT.vbs. This worm attempts to send copies of itself using Microsoft Outlook to all the entries in all the address books. This worm uses email as the primary spreading channel.

It is also able to use mIRC clients as secondary distribution channels. When the worm executes, it attempts to create a script file to send a copy of the worm via DCC (Direct Client Communication) to other people in any IRC channel joined by the victim.

This worm also uses windows file sharing systems. When the worm executes, it searches for certain types of files and replace them with a copy of the worm. It may also be applied to users reading messages in USENET newsgroups.

4.2.2 Security Vulnerability Attacks

Currently security holes which Internet-scale viruses misuse are related to Microsoft Software, such as Internet Information Server (IIS), Windows NT, Windows 2000, Outlook Express, and Windows Internet Explorer.

4.2.2.1 Code Red/Code Red II

The Code Red/Code Red II is a malicious self-propagating worm (CERT/CA-2001-23 2002) misusing Microsoft's Internet Information Server (IIS), which affects network performance.

The Code Red worm attempts to connect to port 80/tcp on a randomly chosen host assuming that a web server will be found. Upon a successful connection to port 80, the

attacking host sends a crafted HTTP GET request to the victim, attempting to exploit a buffer overflow in the Indexing service. And the same HTTP GET request is sent to each of the randomly chosen hosts due to the self-propagating nature of the worm. However, depending on the configuration of the host which receives this request, IIS 4.0 and 5.0 servers with Indexing service installed almost certainly are compromised by the Code Red worm. Unpatched CISCO 600-series DSL routers process the HTTP request thereby triggering an unrelated vulnerability which causes the router to stop forwarding packets. Moreover, systems which are not running IIS, but with an HTTP server listening on port 80/tcp accept the HTTP request, return with an "HTTP 400 Bad Request" message, and potentially log this request in an access log.

On the other hand, the Code Red H worm attempts to connect to port 80/tcp like the Code Red did. Then the HTTP GET request is sent to each of the randomly chosen hosts. Unpatched Windows 2000 servers running IIS 4.0 or 5.0 with Indexing service installed are likely to be compromised by the Code Red II worm. Unpatched Windows NT servers running IIS 4.0 or 5.0 with Indexing Server 2.0 installed could experience crashes of the IIS server. With unpatched CISCO 600-series DSL routers, it is same as with the Code Red worm. Patched systems, or systems which are not running IIS with an HTTP server listening on TCP port 80 accept the HTTP request, return with an "HTTP 4xx" error message, and potentially log this request in an access log.

If the HTTP request is successful, the worm begins executing on the victim host. The Code Red II copies CMD.EXE to root.exe in the IIS scripts and MSADC folders. Placing CMD.EXE in a publicly accessible directory may allow an intruder to execute arbitrary commands on the compromised machine with the privileges of the IIS server process. Then the worm creates a Trojan horse as copy of explorer.exe and copies it to the C: and D: drives. On systems which are not patched against the relative shell patch (MS00-052 2000) [3] vulnerability, this Trojan horse runs every time when a user logs in the system.

The beginning of Code Red's attack packet looks like the following (CERT/CA-2001-19 2001), (CERT/IN-2001-09 2001):

```
GET/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3
```

[3] Microsoft has released a patch that eliminates this security vulnerability in Microsoft Windows NT 4.0 and Windows 2000. Under certain conditions, the vulnerability could enable a malicious user to cause code of his choice to run when another user subsequently logged onto the same machine.

4.2.2.2 Nimda Worm

The Nimda worm affects both user workstations (clients) running Windows 95, 98, ME, NT or 2000 and servers running Windows NT and 2000 (CERT/CC 2002). The worm modifies web documents (e.g., .htm, .html, and .asp files) and certain executable files found on the systems it infects, and creates numerous copies of itself under various file names. One part of the Nimda Worm's attack packets looks like the following (CERT/CA-2001-26 2001):

```
GET /scripts/root.exe?/c+dir
GET /MSADC/root.exe?/c+dir
GET /c/winnt/system32/cmd.exe?/c+dir
GET /d/winnt/system32/cmd.exe?/c+dir
.....
```

The Nimda worm has three types of propagation. First is email propagation. This worm propagates through email messages consisting of two sections; a blank message, and an executable attachment. The first section is defined as MIME (Multipurpose Internet Mail Extensions) type "text/html", but it contains no text, so the email appears to have no content. The second section is defined as MIME type "audio/x-wav", but it contains a base64-encoded attachment file "readme.exe", which is a binary executable. Due to a vulnerability of Microsoft Internet Explorer to start the HTML mail automatically, the enclosed attachment is executed and, as result, infects the machine with the worm. Even though this worm is promulgated through email, the infected machine provides a copy of the worm via a web server or the file system because the executable file modifies all web content files in the system.

Second is browser propagation. Nimda modifies all web content files it finds. As a result, any user browsing web content on the system may download a copy of the worm.

The third way of propagation is file system propagation. The Nimda worm creates numerous copies of itself in all writable directories to which the user has access. If a user on another system subsequently selects the copy of the worm file on the shared network drive, the worm may be able to compromise that system. This worm may also cause bandwidth DOS conditions on networks with infected machines.

4.2.2.3 W32/BadTrans

This is a mass mailing worm which uses Outlook to reply to unread email messages. It also drops a remote access Trojan Horses to the infected computer. This malicious

Windows program distributes as an email file attachment. Using a known vulnerability in Internet Explorer, Outlook Express and Outlook email programs may execute the malicious program as soon as the email message is viewed. The format of the MIME headers in an email containing W32/BadTrans attempts to exploit a vulnerability in Internet Explorer. The filename in the email attachment of a W32/BadTrans infected email varies from message to message but always has two file extensions.

The beginning email message of the worm looks like the following (CERT/IN-2001-14 2001).

The MIME headers contain:
Mime-Version: 1.0
Content-Type: multipart/related;
type="multipart/alternative";
boundary="====_ABC1234567890DEF_====="

The body of the MIME message contains:
-====_ABC1234567890DEF_====
Content-Type: multipart/alternative;
boundary="====_ABC0987654321DEF_====="

-====_ABC0987654321DEF_====
Content-Type: text/html;
charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

<HTML><HEAD></HEAD><BODY bgColor=3D#ffffff>
<iframe src=3Dcid:EA4DMGBP9p height=3D0 width=3D0>
</iframe></BODY></HTML>
-====_ABC0987654321DEF_====-

-====_ABC1234567890DEF_====
Content-Type: audio/x-wav;
name="filename.ext.ext"
Content-Transfer-Encoding: base64
Content-ID:

4.2.2.4 W32/Klez-H

This worm contains a compressed copy of the new variant of the W32/Elkern virus, which is dropped and executed when the worm is run. It is quite similar to the other variants of this dangerous virus. This worm searches for email address entries in the Windows address book, in ICQ list and in the files on the disk. It uses its own mailing routine.

The worm attempts to use the well known MIME security hole in MS-Outlook, MS-outlook Express, and Internet Explorer to run the attachment automatically. Infected emails have some characteristics; the subject line is either random or is composed from several strings, the body text is either empty or composed randomly, and the attached file has a random name with extension .PIF, .SCR, .EXE or .BAT.

4.3 Malicious Virus Patterns in Infected Files

I address infected files' format patterns rather than the virus itself in this section. As I mentioned, a virus is a piece of code, however it infects several files and changes their forms as an effect of the infection. By and large Internet-scale viruses consist of a virus program and several auxiliary files and information which support the virus program to spread smoothly. Once the virus program infects several files in one system, existing files contain a piece of virus code and are spread as another infected virus program. The following will examine the virus-infected programs' structure.

4.3.1 Parasitic Viruses

Parasitic Viruses are all the file viruses which have to change the contents of target files while transferring copies of themselves, but the files themselves remain to be completely or partly usable. The main types of these viruses are the prepending viruses which are storing themselves at the top of a file, the appending ones which store themselves at the end of a file, and the inserting ones which insert themselves somewhere in the middle of a file. The insertion method may also be different by moving a fragment of the file towards the end of file or by copying its own code to such parts of the file which are known to be unused.

The most common method of virus incorporation into a file is by appending the virus to the end of file. In this process the virus changes the top of file in such way that the virus code is executed first.

This kind of appending is simple and usually effective. The virus writer does not need to know anything about the program to which the virus will append and the appended program simply serves as a carrier for the virus (P.Pfleeger 1997).

In DOS COM files, in most cases this is achieved by changing the first three or more bytes in the instruction code into "JMP LOC_Virus" or into the address of the routine passing control to the body of the virus like in Figure 4.4.

DOS EXE files are converted to the format of a COM file and then infected as a COM file or the head of the file is modified. In the DOS EXE file header, the starting address is

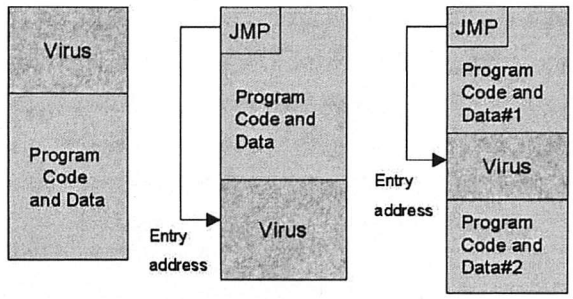

FIGURE 4.4: Virus positions in COM file.

changed (CS:IP), and the length of the executable module is changed. Or less often the stack pointer registers are changed (SS:SP), then the virus goes to change the CRC(Cyclic Redundancy Checksum) part of the file and so on.

In the Windows and OS/2 executables (newEXE - NE, PE, LE, LX) the fields in the NewEXE header are changed. The structure of this header is much more complicated than that of a conventional DOS EXE file, so there are more fields to be changed; the starting address, the number of sections in the file, properties of the sections etc. In addition to that, before infection, the size of the file may increase to a multiple of one paragraph (16 bytes) in DOS or to a section in Windows and OS/2. The size of the section depends on the properties of the EXE file header. Figure 4.5 shows this case.

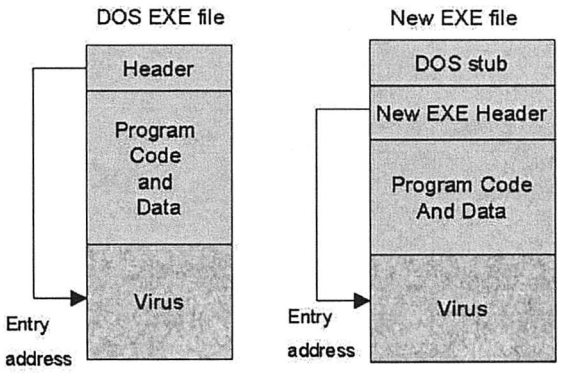

FIGURE 4.5: Virus positions in DOS EXE/New EXE file.

4.3.2 File Worms

File worms are a modification of companion viruses [4], but unlike them they do not connect their presence with any executable file. When they multiply, they just copy their

[4]These viruses do not change the infected files. Their operation is to create a clone of the target file, so that when the target file runs, its clone virus gets the control instead.

code to some other disk or directory hoping that these new copies will someday be executed by a user. Sometimes these file worms give their copies some special names in order to push user into running the copy, for example, INSTALL.EXE or WINSTART.BAT. There are worm viruses using rather unusual techniques, for instance, to add their copies to archives (ARJ, ZIP and others). Such viruses are ArjVirus and Winstart. Some other viruses insert the command starting the infected file into BAT files.

4.3.3 Macro Viruses

Macro viruses are in fact programs written in macro languages, built into some data processing systems such like Microsoft Word, and Microsoft Excel spreadsheet. To propagate, such viruses use the capabilities of macro languages and with their help transfer themselves from one infected file, e.g. document or spreadsheet, to another. Macro viruses for Microsoft Word, Microsoft Excel and Office97 are most common. Figure 4.6 shows the case.

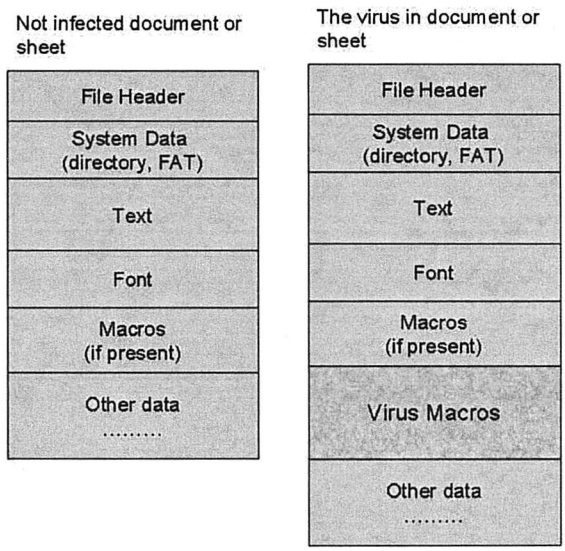

FIGURE 4.6: Macro virus' position in an infected document

File Worms are able to infect Word documents, and Excel viruses can infect Excel spreadsheets. The same is true for Office97(Kaspersky 2000). Because of the complicated format of the Word, Excel and Microsoft Office file, the mechanism of spreading of macro viruses in a file can only be presented approximately here.

I have to mention that Microsoft Word Version 6 and 7 allows to encrypt macros in documents (Kaspersky 2000). Therefore some word viruses are present inside the infected documents in an encrypted and execute only form.

Most of the known viruses for Word are incompatible with native language versions and fail under the English version. However viruses inside documents remain active and may infect other computers with the corresponding version of Word.

Chapter 5

Potential Research Models for Classification/Recognition

A wide variety of approaches has been taken toward classification. Three main historical strands of research can be identified (D.Michie 1994): the statistical approach, machine learning and neural networks. Statistical approaches are generally characterised by having an explicit underlying probability model, which provides a probabiliy of being in each class rather than a classification. Machine learning is generally taken to encompass automatic computing procedures based on logical or binary operations, that learn a task from a series of examples. Machine learning aims to generate classifying expressions simple enough to be understood easily by human beings. Like statistical approaches, background knowledge may be exploited in development, but operation is assumed without human intervention. The field of neural networks has arisen from diverse sources, ranging from the fascination of mankind with understanding and emulating the human brain, to broader issues of copying human abilities, to the practical commercial and scientific disciplines of pattern recognition, modelling, and prediction. These have largely involved different professional and academic groups, and emphasised different issues. However, all groups have had some objectives in common. They have all attempted to derive procedures that would be able:

- to equal, if not exceed, a human decision-maker's behaviour, but have the advantage of consistency and, to a variable extent, explicitness.
- to handle a wide variety of problems and, given enough data, to be extremely general.
- to be used in practical settings with proven success.

As I mentioned in the Introduction, there are several models for recognition; statistical pattern recognition, syntactic or structural pattern recognition, knowledge-based pattern recognition and so on. For these pattern recognition models, there are three main research areas: Neural Network, Fuzzy Inference Systems, and Data Mining.

In this thesis, I selected three main models for classification and recognition. Even though historically the main models are statistical, machine learning and neural networks, this range is too wide to deal with. Furthermore, each area is connected with each other. For instance, machine learning can include neural networks, the machine learning and neural networks combine the complexity of some of the statistical techniques.

Therefore, I chose three potential approaches in this big area after investigating and surveying them : Bayesian Networks, Neural Networks, and Data Mining.

5.1 Bayesian Networks

A Bayesian network is a directed and acyclic graph that compactly represents a probability distribution (Pearl 1988). Each variable X_i is represented as a node in the network. A directed edge between two nodes indicates probabilistic dependency from the variable denoted by the parent node to that of the child.

Conditional independence appears in the cases of serial and diverging connections. Figure 5.1 where A and C are conditionally independent given B.

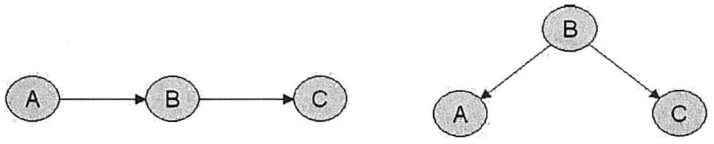

FIGURE 5.1: Conditional Independent.: A and C are conditionally independent given B.

Consequently, the structure of the network denotes the assumption that each node X_i in the network is conditionally independent of its non-descendants given its parents. To describe a probability distribution satisfying these assumptions, each node X_i in the network is associated with a conditional probability table, which specifies the distribution over X_i given any possible assignment of values to its parents. If X_i has no parents, it simply contains a prior probability distribution over X_i's values. The network structure and the associated parameters uniquely define a probability distribution over the variables in the network.

5.1.1 Subjective Probability and Objective Probability

Bayesian probability is interpreted as degree of belief (J.M.Bernardo and A.F.M.Smith 1994). Bayesian probability is also known as subjective probability, personal probability, or epistemic probability. Beliefs are always subjective, and all the probabilities appearing in Bayesian probability theory are conditional. In particular, under the belief interpretation probability is not an objective property of some physical setting, but is conditional to the prior assumptions and experience of the learning system.

There is a long standing debate as to whether the subjective or the objective approach is the most appropriate. Objective may seem more plausible at first, but does require lots of data and is prone to experimental error. Therefore, some people say the Bayesian approach must be subjective. This is because there were no statistics when Bayes published his theorem, and so he would have considered a probability as purely subjective.

I think subjective probability is more or less the feel for how probable something is, the degree to which a person's belief is probably true given the total evidence that the person has. I will use the term Bayesian to describe methods based on the calculus that develops out of Bayes' theorem, regardless of whether the probabilities are estimated by subjective or objective methods. However, I would like to say that the way to get a probability from total evidence and to deal with the probability for my project is subjective.

5.1.2 Bayes' Theorem and Bayesian Inference

Here, we use two variables which are related with the next chapter. To get probability of malicious packets, we are getting some specific evidence from data.

For independent events E and M, M represents a group of malicious packets, E represents specific evidence about these packets.

$$\mathrm{P}(M \wedge E) = \mathrm{P}(M) * \mathrm{P}(E)$$

However, in cases where E and M are not independent. We must write:

$$\mathrm{P}(M \wedge E) = \mathrm{P}(M) * \mathrm{P}(E \mid M)$$

Where, $\mathrm{P}(E \mid M)$ is the probability of the specific evidence given the malicious packets have occurred.

5.1.2.1 Conditional Probability

The conditional probability of event E given event M, denoted $\mathrm{P}(E \mid M)$, is given by,

$$\mathrm{P}(E \mid M) = \frac{\mathrm{P}(E \wedge M)}{\mathrm{P}(M)}$$

Now since conjunction is commutative,

$$\begin{aligned} \mathrm{P}(M \wedge E) &= \mathrm{P}(M) * \mathrm{P}(E \mid M) \\ &= \mathrm{P}(E) * \mathrm{P}(M \mid E) \end{aligned}$$

and by rearranging we get:

$$\mathrm{P}(M \mid E) = \frac{\mathrm{P}(M) * \mathrm{P}(E \mid M)}{\mathrm{P}(E)}$$

5.1.2.2 Bayes' Theorem as an Inference Mechanism

$$\mathrm{P}(M \mid E) = \frac{\mathrm{P}(M) * \mathrm{P}(E \mid M)}{\mathrm{P}(E)}$$

We write,

- $\mathrm{P}(M \mid E)$: Posterior probability, the probability of malicious packets given the specific evidence, which is what we wish to infer.
- $\mathrm{P}(E)$: The probability of the specific evidence; this is a measurable quantity that we get from existing data.
- $\mathrm{P}(E \mid M)$: The probability of the specific evidence given the malicious packets. We can measure this from the case histories of the malicious packets.
- $\mathrm{P}(M)$: Prior probability, the probability of malicious packets which we get from existing data.

5.1.2.3 Prior and Likelihood Information

$$\mathrm{P}(M \mid E) = \alpha * \mathrm{P}(M) * \mathrm{P}(E \mid M)$$

We write,

- P($M \mid E$): Posterior Information.
- $1/\mathrm{P}(E) = \alpha$: It is a normalized value [1] by measurable quantity. We represent this value as α.
- P(M): Prior information, since we knew it before we made any measurements.
- P($E \mid M$): Likelihood information, since we gain it from measurement of evidences.

Prior information should be subjective. It represents our belief about the domain we are considering, even if data has made a substantial contribution to our belief. Likelihood information should be objective. It is a result of the data gathering from which we are going to make an inference. It makes some assessment of the accuracy of our data gathering. In practice either or both forms can be subjective or objective.

In some cases, we obtain the prior probability from statistics. For example, we can calculate the prior probability as the number of instances of a disease divided by the number of patients presenting for treatment. However in many cases this is not possible since the data is not there, and there may also be prior knowledge in other forms.

5.1.3 Simple Bayesian Classifier

The most straighforward and widely tested method for probabilistic induction is the simple Bayesian Classifier (Langley and Sage 1994) [2]. A simple Bayesian Classifier is a simple structure in which nodes that show the same parent node cannot have a connection between them (i.e. it is a tree-like structure). A simple Bayesian Classifier is illustrated in Figure 5.2.

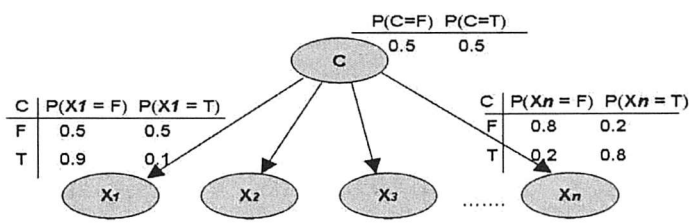

FIGURE 5.2: A Structure of a simple Bayesian Classifier

We assume each node as a class. This simple Bayesian Classifier represents each class with a single probability table. In particular, each table has an associated class probability P(C_i), which specifies the prior probability that one will observe a member of class C_i.

[1] The goal of data normalization is so that none of components of input vectors has an overwhelming influence on the training result. There are several methods for data normalization.

[2] Bayesian classifier has a number of names: the *simple Bayesian Classifier(P.Langley 1993), the Naive Bayesian Classifier(I.Kononenko 1990) and idiot Bayes(W.Buntine 1990). These names are commonly used.*

Each table also has an associated set of conditional probabilities, specifying a probability distribution for each attribute. However, Bayesian Inference is inadequate to deal with more complex models of prior knowledge. Once several factors affect a decision they need to be combined somehow.

The simple Bayesian classifier relies on two important assumptions (Langley and Sage 1994). First is a single probability table. Instances in each class can be summarized by a single probability table, and these are sufficient to distinguish the classes from one another.

Another assumption is independence of attributes. The simple Bayesian classifier requires that the probability distributions for attributes are independent of each other within each class. One can model attribute dependence within the Bayesian framework (Pearl 1988). But determining such dependencies and estimating them from limited training data is much more difficult. Thus, the independence assumption has clear attractions. It is applicable in many cases.

When we use Bayes' theorem we have just one hypothesis and one piece of evidence. However, we have evidence from more than one source in the real world. We get:

$$\mathrm{P}(M \mid E_1 \wedge E_2 \wedge \cdots \wedge E_n) = \frac{\mathrm{P}(M) * \mathrm{P}(E_1 \wedge E_2 \wedge \cdots \wedge E_n \mid M)}{\mathrm{P}(E_1 \wedge E_2 \wedge \cdots \wedge E_n)}$$

The term $\mathrm{P}(M) * \mathrm{P}(E_1 \wedge E_2 \wedge \cdots \wedge E_n \mid M)$ is of little use for inference since for large n we are unlikely to be able to estimate it. Hence we normally make the assumption that the E_i are independent given M, this allows us to write:

$$\mathrm{P}(E_1 \wedge E_2 \wedge \cdots \wedge En \mid M) = \mathrm{P}(E_1 \mid M) * \mathrm{P}(E_2 \mid M) * \cdots * \mathrm{P}(E_n \mid M)$$

The term $\mathrm{P}(E_1 \wedge E_2 \wedge \cdots \wedge E_n)$ can be eliminated by normalisation, therefore the inference equation we can obtain from Bayes' theorem is:

$$\mathrm{P}(M \mid E_1 \wedge E_2 \wedge \cdots \wedge E_n) = \alpha * \mathrm{P}(M) * \mathrm{P}(E_1 \mid M) * \mathrm{P}(E_2 \mid M) * \cdots * \mathrm{P}(E_n|M)$$

where

$$\alpha = \frac{1}{\mathrm{P}(E_1 \wedge E_2 \wedge \cdots \wedge E_n)}$$

5.1.4 Bayesian Decision Tree

There is a large number of conditional probabilities. We need a very large data set to make a reasonable estimate. In this point, the use of a tree gives us a much more accurate

and easy-to-read way of expressing how each term relates to one another in a given context, such as the analysis of SMTP packets.

SMTP (Simple Mail Transfer Protocol) (RFC0821 1982) (RFC2821 2001) is for the Internet electronic mail transport. Based on this protocol, I can get SMTP's semantic entities, which might be present in a mail packet. The Mail Packet Decision Tree in Figure 5.3 shows us a model which is better to understand. The "header" node represents considering a mail's subject and "body" represents the mail's data part. The nodes "header" and "body" of a mail packet can be seen as a common, cause of the "MIME type" and "Plain Text Type" nodes (e.g. "MIME type" and "Plain Text type" are children of "header" meaning that a mail's subject is either plain text or MIME encoded.). It is important to note here that the MIME (Multipurpose Internet Mail Extensions) type of the header node and that of the body node are very different. The header node's MIME is followed by (RFC2047 1996), on the other hand, the body node's MIME is followed by (RFC2045 1996), (RFC2046 1996), and (RFC2048 1996).

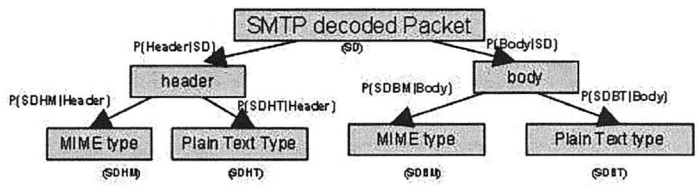

FIGURE 5.3: Mail Packet Decision Tree

If we use each nodes' symbols like *SD* for the root node(SMTP decoded Packet), *Header* for the header node, *Body* for the body node, *SDHM* for the MIME type node, *SDHT* for the Plain Text Type node, *SDBM* for the MIME type node inherited from the body node, and *SDBT* for the Plain Text Type node which is again inherited from the body node, as we can see in Figure 5.3, the probability of header node of a mail packet could be denoted by P(*Header* | *SD*), because it is the conditional probability that an observed packet is of type *Header* given that it is an SMTP packet (*SD*). That means that it is the probability of a data packet to be part of a mail's header under the assumption that we already know that is it a SMTP packet. Like this, the probability of the body node of a mail packet is P(*Body* | *SD*). I will call this kind of probabilities a link matrix. This Bayesian Decision Tree presents relations among nodes and gives us accurate estimation of conditional probabilities.

5.1.5 Related Work

The simple Bayesian classifier gives remarkably high accuracies in many natural domains (G.Cestnik and I.Bratko 1987). The simple probabilistic method outperformed a

decision-tree algorithm on four out of five natural domains (P.Langley and K.Thompson 1992). They compare the method to IND's emulation [3] of C4 algorithm [4] and a frequency-based algorithm that simply guesses the modal class. The five domains include the small soybean dataset (classifying soybeans into different categories based on their appearance), chess end games involving a king-rook-king-pawn confrontation, cases of lymphography diseases, and two biological datasets. For each domain, they randomly split the data set into 80% training instances and 20% test instances, repeating this process to obtain 50 separate pairs of training and test sets. In four of the domains, the simple Bayesian classifier is at least as accurate as a C4 reimplementation. The result shows that it behaves well across a variety of domains.

5.2 Neural Network Models

Neural networks are a wide class of flexible nonlinear regression and discriminant models, data reduction models, and nonlinear dynamical systems (Haykin 1999). They consists of an often large number of neurons, i.e. simple linear or nonlinear computing elements, interconnected in often complex ways and often organized into layers.

When neural networks are used for data analysis, it is important to distinguish between neural network models and neural network algorithms (S.Sarle 1994). Many neural network models are similar or identical to popular statistical techniques such as generalized linear models, polynomial regression, nonparametric regression and discriminant analysis, and cluster analysis, especially where the emphasis is on prediction of complicated phenomena rather than on explanation. These neural network models can be very useful. There are also a few neural network models, such as counter propagation, learning vector quantization, and self-organizing maps, that have no precise statistical equivalent but may be useful for data analysis.

Standard neural network learning algorithms e.g. perceptron, multi-layer perceptron, back propagation, radial basis function networks, and feed-forward network, are inefficient because they are designed to be implemented on massively parallel computers but are, in fact, usually implemented on common serial computers such as ordinary PCs. On a serial computer, neural networks can be trained more efficiently by standard numerical

[3] IND is a tree calssification software. IND does supervised learning using classification trees. It was developed as part of a NASA project to semi-automate the development of data analysis and modeling algorithms using artificial intelligence techniques.

[4] C4 is a well-known learning algorithm developed by Quinlan. This type of algorithms tries to fit a tree to a training sample using recursive partitioning. This means that the training set is split into increasingly homogeneous subsets until the leaf nodes contain only cases from a single class. An important problem in learning classification trees is overfitting on the training set. To this end, pruning strategies can be adopted, whereby the classification tree is simplified by discarding one or more subtrees and replacing them with leaves.

optimization algorithms such as those used for nonlinear regression. Nonlinear regression algorithms can fit most neural network models orders of magnitude faster than the standard neural network algorithms. Another reason for the inefficiency of neural network algorithms is that they are often designed for situations where the data are not stored, but each observation is available transiently in a real-time environment. Transient data are inappropriate for most types of statistical analysis. In statistical applications, the data are usually stored and are repeatedly accessible, so statistical algorithms can be faster and more stable than neural network algorithms. Hence, for most practical data analysis applications, the usual neural network algorithms are not useful.

Because all neural networks are based on the concept of neurons, connections, and transfer functions, there is a similarity between the different structures, or architectures, of neural networks. The majority of the variations stems from the various learning rules and how those rules modify a network's typical topology. I am considering the Self-Organizing Map (SOM) and the Probabilistic Neural Network (PNN) in this thesis.

5.2.1 Major Components of an Artificial Neuron

These components (Anderson and McNeill 1992) are valid whether the neuron is used for input, output, or is in one of the hidden layers.

5.2.1.1 Weighting Factors

A neuron usually receives many simultaneous inputs. Each input has its own relative weight which gives the input impact that it needs on the processing element's summation function. These weights perform the same type of function as do the varying synaptic strengths of biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response. Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules.

5.2.1.2 Summation Function

The first step in a processing element's operation is to compute the weighted sum of all of the inputs. Mathematically, the inputs and the corresponding weights are vectors which can be represented as $(i_1, i_2, \ldots, i_n)$ and $(w_1, w_2, \ldots, w_n)$. The total input signal is the

dot, or inner, product of these two vectors. This simplistic summation function is found by multiplying each component of the i vector by the corresponding component of the w vector and then adding up all the products. $input_1 = i_1 * w_1$, $input_2 = i_2 * w_2$, etc., are added as $input_1 + input_2 + \cdots + input_n$. The result is a single number, not a multi-element vector. The inner product of two vectors can be considered a measure of their similarity. If the vectors point in the same direction, the inner product is maximum; if the vectors point in opposite direction (180 degrees out of phase), their inner product is minimum.

The summation function can be more complex than just the simple input and weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. In addition to a simple product summing, the summation function can select the minimum, maximum, majority, product, or several normalizing algorithms. The specific algorithm for combining neural inputs is determined by the chosen network architecture and paradigm. Some summation functions have an additional process applied to the result before it is passed on to the transfer function. This process is sometimes called the activation function. The purpose of utilizing an activation function is to allow the summation output to vary with respect to time. Activation functions currently are pretty much confined to research. Most of the current network implementations use an identity activation function, which is equivalent to not having one. Additionally, such a function is likely to be a component of the network as a whole rather than of each individual processing element component.

5.2.1.3 Transfer Function

The result of the summation function, almost always the weighted sum, is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal. If the sum of the input and weight products is less than the threshold, no signal or some inhibitory signal is generated. Both types of response are significant. The threshold, or transfer function, is generally non-linear. Linear functions are limited because the output is simply proportional to the input. Linear functions are not very useful.

5.2.1.4 Scaling and Limiting

After the processing element's transfer function, the result can pass through additional processes which scale and limit. This scaling simply multiplies a scale factor times the transfer value, and then adds an offset. Limiting is the mechanism which insures that the

scaled result does not exceed an upper or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed.

5.2.1.5 Output Function (Competition)

Each processing element is allowed one output signal which it may output to hundreds of other neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify the transfer result to incorporate competition among neighbouring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both of two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process.

5.2.1.6 Error Function and Back-Propagated Value

In most learning networks, the difference between the current output and the desired output is calculated. This raw error is then transformed by the error function to match the particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, some cube the error, other paradigms modify the raw error to fit their specific purposes. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error. The current error is typically propagated backwards to a previous layer. Yet, this back-propagated value can be either the current error, the current error scaled in some manner (often by the derivative of the transfer function), or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied against each of the incoming connection weights to modify them before the next learning cycle.

5.2.1.7 Learning Function

The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the adaption function, as well as the learning mode. There are two types of learning: supervised and unsupervised. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the

network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network. This is learning by doing.

5.2.2 Training a Neural Network

There are two approaches to training; supervised and unsupervised. Supervised training involves a mechanism of providing the network with the desired output either by manually grading the network's performance or by providing the desired outputs with the inputs. Unsupervised training is where the network has to make sense of the inputs without outside help.

5.2.2.1 Supervised Training

In supervised training, both the inputs and the outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are continually tweaked. The set of data which enables the training is called the training set. During the training of a network the same set of data is processed many times as the connection weights are ever refined.

5.2.2.2 Unsupervised Training

In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization.

5.2.3 Self-Organizing Map (SOM)

The Self-Organizing Map (T.Kohonen 1995) is a neural network model for analyzing and visualizing high dimensional data. It belongs to the category of competitive learning network. The Self-Organizing Map is based on unsupervised learning map and nonlinear statistical relationships between high-dimensional input data into two-dimensional lattice (Nguyen 2002). The self-organizing map defines a neighbourhood relation between prototype vectors - also called codebook vectors. This neighbourhood relation can be a rectangular or hexagonal lattice of map units. (see Figure 5.4)

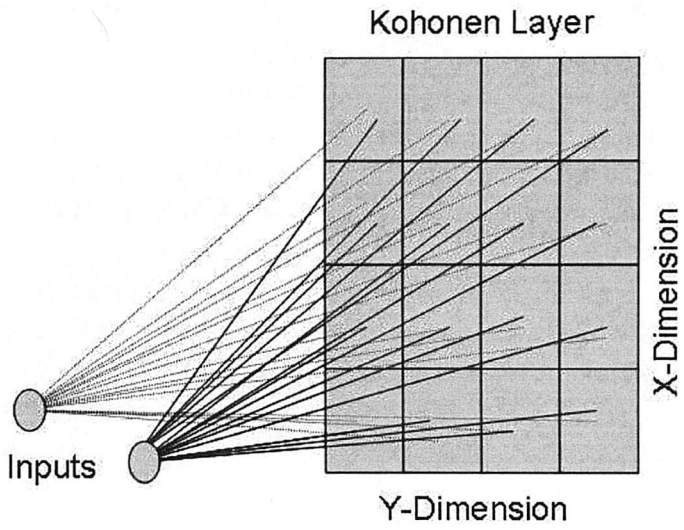

FIGURE 5.4: The Kohonen Layer

The primary use of the self-organizing map is to visualize topologies and hierarchical structures of higher-order dimensional input spaces. The self-organizing network has been used to create area-filled curves in two-dimensional space created by the Kohonen layer. The Kohonen layer can also be used for optimization problems by allowing the connection weights to settle out into a minimum energy pattern.

A key difference between this network and many other networks is that the self-organizing map learns without supervision. However, when the topology is combined with other neural layers for prediction or categorization, the network first learns in an unsupervised manner and then switches to a supervised mode for the trained network to which it is attached.

5.2.3.1 Training Structure

The self-organizing map has typically two layers. The input layer is fully connected to a two-dimensional Kohonen layer. The output layer is used in a categorization problem and represents classes to which the input vector can belong. There are two steps in the training process for SOM (T.Kohonen 1995). First, a winner is searched, which minimizes the Euclidean distance measured between input data sample x and the map unit m_i. The smallest of the Euclidean distances $||x - m_i||$ can be made to define the best-matching node, signified by the subscript c.

$$c = \text{argmin}_i ||x - m_i||$$

The Kohonen layer's processing elements measure the Euclidean distance of its weights from the incoming input values. During recall, the Kohonen element with the minimum distance is the winner and outputs one to the output layer. This is a competitive win, so all other processing elements are forced to zero for that input vector. Thus the winning processing element is, in a measurable way, the closest to the input value and thus represents the input value in the Kohonen two-dimensional map. So the input data, which may have many dimensions, comes to be represented by a two-dimensional vector which preserves the order of the higher dimensional input data. This can be thought of as an order-perserving projection of the input space onto the two-dimensional Kohonen layer.

Second, the map units in the neighbourhood of the winner unit are updated according to a neighbourhood function $h_{ci}(t)$ centered on the winner unit c. This update step can be carried out by applying the following formula:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)]$$

Where t = 0, 1, 2, ... is an integer, and the last term in the square brackets is proportional to the gradient of the squared Euclidean distance. A widely applied neighbourhood function is the the Gaussian function defined below.

$$h_{ci} = \alpha(t) \cdot \exp(-\frac{||r_c - r_i||^2}{2\sigma^2(t)})$$

The SOM defines a mapping from input data space R^n onto a two-dimensional array of nodes. $r_c \in R^2$ and $r_i \in R^2$ are the location vectors of nodes c and i, respectively, in the array. The value of $\alpha(t)$ is identified with a learning-rate factor ($0 < \alpha(t) < 1$), and the parameter $\sigma(t)$ defines the width of the kernel.

During training, the Kohonen processing element with the smallest distance adjusts its weight to be closer to the values of the input data. The neighbours of the winning element also adjust their weights to be closer to the same input data vector. The adjustment of neighbouring processing elements is instrumental in preserving the order of the input space.

The problem of having one processing element takes over for a region and representing too much input data exists in this paradigm. As with counter-propagation, this problem is solved by a conscience mechanism built into the learning function. The conscience rule depends on keeping a record of how often each Kohonen processing element wins and this information is then used during training to bias the distance measurement. This conscience mechanism helps the Kohonen layer achieve its strongest benefit. The processing elements naturally represent approximately equal information about the input data set. Where the input space has sparce data, the representation is compacted in the

Kohonen space, or map. Where the input space has high density, the representative Kohonen elements spread out to allow finer discrimination. In this way the Kohonen layer is thought to mimic the knowledge representation of biological systems.

5.2.4 Probabilistic Neural Network (PNN)

Probabilistic Neural Networks (PNN) are a class of neural networks which combine some of the best attributes of statistical pattern recognition and feed-forward neural networks (Specht 1990). The PNN is based on well established statistical principles rather than heuristic approaches. Heuristic approaches usually involve making many small modifications to the system parameters which gradually improve system performance. The Multilayer Perceptron (MLP) neural network is typical of the heuristic approach and it is associated with long training times with no guarantee of achieving a suitable solution within a reasonable training time. The PNN on the other hand is derived from the Bayes' decision strategy and nonparametric [5] kernel [6] based estimators of probability density functions. It is guaranteed to approach the Bayes' optimal decision surface as the number of training samples increase provided the class probability density functions are smooth and continuous.

The probabilistic neural network uses a supervised training set to develop distribution functions within a pattern layer. These functions, in the recall mode, are used to estimate the likelihood of an input feature vector being part of a learned category, or class. The learned patterns can also be combined or weighted with the a priori probability, also called the relative frequency, of each category to determine the most likely class for a given input vector. If the relative frequency of the categories is unknown, then all categories can be assumed to be equally likely and the determination of category is solely based on the closeness of the input feature vector to the distribution function of a class.

5.2.4.1 Training Structure

This network has three layers. The network contains an input layer which has as many elements as there are separable parameters needed to describe the objects to be classified. It has a pattern layer, which organizes the training set such that each input vector is represented by an individual processing element. And finally, the network contains an

[5] In pattern recognition, density estimation without any prefererence model assumptions is called non-parametric density estimation. And one of non-parametric techniques used for density estimation is Parzen. Parzen density estimation method (de Ridder 1998) is a superposition of Gaussian Kernel located on many objects. These kernels are combined by linear summation to find a probability density function of the objects. This method is very accurate but requires a huge amount of objects, especially when estimating in a high dimensional space.

[6] A local region size and shape is called a kernel.

output layer, called the summation layer, which has as many processing elements as there are classes to be recognized. Each element in this layer combines via processing elements within the pattern layer which relate to the same class and prepares that category for output. Sometimes a fourth layer is added to normalize the input vector, if the inputs are not already normalized before they enter the network. The input vector must be normalized to provided proper object separation in the pattern layer.

The pattern layer represents a neural implementation of a version of a Bayes classifier, where the class dependent probability density functions are approximated using a Parzen estimator (Richard Duda and Stork 2001) [7]. This approach provides an optimum pattern classifier in terms of minimizing the expected risk of wrongly classifying an object. With the estimator, the approach gets closer to the true underlying class density functions as the number of training samples increases, so long as the training set is an adequate representation of the class distinctions.

In the pattern layer, there is a processing element for each input vector in the training set. Normally, there are equal amounts of processing elements for each output class. Otherwise, one or more classes may be skewed incorrectly and the network will generate poor results. Each processing element in the pattern layer is trained once. An element is trained to generate a high output value when an input vector matches the training vector. The training function may include a global smoothing factor to better generalize classification results. In any case, the training vectors do not have to be in any special order in the training set, since the category of a particular vector is specified by the desired output of the input. The learning function simply selects the first untrained processing element in the correct output class and modifies its weights to match the training vector.

The pattern layer operates competitively, where only the highest match to an input vector wins and generates an output. In this way, only one classification category is generated for any given input vector. If the input does not relate well to any patterns programmed into the pattern layer, no output is generated.

The Parzen estimation can be added to the pattern layer to fine tune the classification of objects. This is done by adding the frequency of occurrence for each training pattern built into a processing element. Basically, the probability distribution of occurrence for each example in a class is multiplied into its respective training node. In this way, a more accurate expectation of an object is added to the features which make it recognizable as a class member.

[7] The objective of pattern recognition problems is to minimize the number of classification errors, also called error rate. The minimum error rate over the set of all classifiers is defined as the Bayes error. Parzen fixes the volume of the local region (of Photogrammetry and Sensing 2001). Estimating the Bayes error using the Parzen estimator is done by forming the log likelihood ratio functions and then using resubstitution and leave-one-out methodologies to find an optimistic and pessimistic value for error estimate. However, Parzen estimators are not known to bound the Bayes error (William E.Pierson 1998).

5.2.4.2 Pros and Cons

Training of the probabilistic neural network is much simpler than with back-propagation. However, the pattern layer can be quite huge if the distinction between categories is varied. There are many proponents for this type of network, since the groundwork for optimization is founded in well known, classical mathematics.

PNNs' features are very fast training times and the production of outputs with Bayes posterior [8] probabilities. These useful features come at the expense of larger memory requirements and slower execution speed for prediction of unknown patterns compared to conventional neural networks. Testing can be slower than for backpropagation for software implementation as the computation time for a classification is proportional to the size of the training set.

The main drawback of PNNs is that, like kernel methods [9] in general, it suffers badly from the curse of dimensionality. A PNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. But if all inputs are relevant, PNN has the very useful ability to say whether a test case is similar (i.e. has a high density) to any of the training data; if not, the output classification must be checked with scepticism. This ability is of limited use when inputs are irrelevant, since the similarity is measured with respect to all of the inputs, not just the relevant ones.

5.2.5 Related Work

Nowadays almost all practical IDSs are signature-based systems. These systems work based on predefined descriptions of attack signatures. Various data source and type-of-pattern recognition techniques are still used. However, many known attacks can be easily modified to present many different signatures. If not all variations are in the database, a known attack may be missed. Moreover, early systems were constructed around concepts of statistical anomaly detection. These systems faced practical and theoretical difficulties, such as performance and creation of false positives. Some more modern approaches try to exploit neural-network-based techniques.

- SOM and BP

  The research presented in (C.Lee and V.Heinbuch 2001) is based on a simulated network and IDS. The IDS is composed of a hierarchy of back propagation(BP)

[8] The revised values of prior probabilities after receiving additional information.

[9] In non-parametric density estimation, there are three main methods; histograms, k-nearest neighbours methods, kernel methods (Iliescu). Kernel methods use density function as sums of common point spread interpolation functions.

neural networks. The experimental IDS focuses on the protocol. The transfer control protocol (TCP) has a rich repertoire of well-defined behaviours that can be monitored by the experimental IDS. But, ill-formed packets could not be produced by the network simulation; therefore the experimental IDS did not monitor packet formation. A packet formation check remains necessary.

To explore attack spaces, they use a hierarchy of Back Propagation (BP) neural networks for the protocol, and the Self-Organizing Map (SOM) technique for the anomaly classification. However, they did not regard the volume of traffic, packet size distribution, inter-arrival rates, login rates, the number of and ports for services (which could well be doable for real systems). For anomaly detection, they choose a hierarchy of back propagation. To test it, every packet is sent and received. Writing such a program would be akin to rewriting the TCP network software. They aggregated statistics to detect anomalies. This includes a lot of costs. The BP needs to know the output form. The BP is initialized randomly and must undergo supervised learning before its use as a detector. This requires knowledge of the desired output for each input vector. Often, obtaining training data with known content is difficult. Furthermore, if the input representing an anomaly represents a known attack, the neural network will learn to recognize those particular signatures as bad, but may not recognize other, novel attack signatures. The simulation of data traffic is rather artificial. Since true network activity does not follow the normal distributions they used. And the output for the SYN flood and fast scan attacks are well separated from the nominal output.

For anomaly classification, they use an SOM. The SOM provides a two dimensional mapping of n-dimensional input data into unique clusters. Using the SOM, they can classify an anomaly, but not a malicious attack. Their training data itself is already quite anomalous. That is why they can classify many anomalies. But real data packets are hard to classify using their SOM, because the data parts of real packets must be grouped together in order for the SOM to analyse it.

- TDNN and BP

  Research in (K.Ghosh and Schwartzbard 1999) is based on a classical feed-forward multi-layer perceptron network: a Back-Propagation neural network and Time Delay Neural Network (TDNN) to program-based anomaly detection. For a long time, such a research approach was focusing on user behaviour, but this paper dealt with the behaviour of software. It is therefore a novel and different approach to anomaly and misuse detection. The TDNN to program-based anomaly detection has proved to be more successful than using the BP for the same prupose.

- MLP and SOM

  Lastly, to identify and classify network activity based on limited, incomplete, and

nonlinear data sources, (Cannady and Mahaffey 1998) present an analysis of the applicability of neural networks. They use the RealSecure network monitor from Internet Security Systems,.Inc. In their neural network architecture, Multi-Layer Perceptron (MLP) and hybrid forms (MLP + SOM) are adapted. The MLP was conducted using a backpropagation algorithm. The MLP result demonstrates the potential of neural networks to detect individual instances of possible misuse, on the other hand, Hybrid MLP + SOM result in correctly classifying each of the test cases and provide a positive demonstration of the ability of the hybrid neural network architecture to detect complex instances of misuse.

5.3 Data Mining Techniques

Data mining is the analysis of large observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner (David Hand and Smyth 2001). The data mining approach describes the discovery of useful summaries of data based on relations, patterns, and rules that exist in the data. Pattern extraction and discovery as well as feature extraction capabilities of data mining processes seem to offer interesting possibilities for our detection engine.

Data mining takes advantage of advances in the field of Artificial Intelligence(AI) and statistics. Both disciplines have been working on problems of pattern recognition and classification (Crows 1999). Data mining is, in some ways, an extension of statistics, with a few artificial intelligence and machine learning twists thrown in. Data Mining is about finding understandable knowledge, on other hand, machine learning is concerned with improving performance of an agent, for instance, training a neural network. Moreover, efficiency of the algorithm and scalability is more important in data mining. Because data mining is concerned with very large, real-world databases and machine learning typically looks at smaller data sets. However, data mining does not replace traditional statistical techniques. Rather it is an extension of statistical methods that is in part the result of a major change in the statistics community (Crows 1999).

Data mining creates classification models by examining already classified data cases and inductively finding a predictive pattern. In the case where data mining is applied to detection (Matthew G.Schultz 2001), data mining methods detect patterns in large amounts of data, such as byte code, and use these patterns to detect future instances in similar data. Their classifier is a rule set, or detection model, generated by the data mining algorithm that was trained over a given set of training data. This needs to gather many kinds of information which is helpful to detect attacks. For an IDS, it has good

advantages. In the firewall case, the data which we need to monitor is very restricted. That is why it is not suitable. Data mining programs are applyed to computing patterns for feature constructions; for computing activity patterns from audio data, constructing features from the patterns and learning classifiers for intrusion detection from audit records processed according to the feature definitions (Wenke Lee 2000). Data mining-based approaches to building detection models for IDSs are discussed in (Wenke Lee 2001). These models generalize from both known attacks and normal behaviour in order to detect unknown attacks.

5.3.1 Related Work

Data mining methods detect patterns in large amounts of data, and use these patterns to detect future instances in similar data. These data mining techniques have already been applied to IDSs (W.Lee and P.K.Chan 1997), (Wenke Lee and Mok 1999). Their methods were applied to system calls and network data to learn how to detect new intrusions. They reported good detection rates as a result of applying data mining to the problem of IDS.

Data mining-based IDSs are only useful if their detection rate is higher than a hand-crafted method's detection rate with an acceptably low false positive rate. (Wenke Lee 2001) developed a data mining-based IDS that is capable of outperforming hand-crafted signature-based systems at a tolerated false positive rate. They have applied a number of algorithm-independent techniques to improve the performance of data mining-based IDSs.

Another applicative part is detecting new malicious executables (Matthew G.Schultz 2001). They designed a framework that uses data mining algorithms to train multiple classifiers on a set of malicious and benign executables to detect new examples.

Chapter 6

An Intelligent Firewall Architecture Model

In various ways, we can create intelligent engines. All I focus on in my thesis is about two points of risk management in the Intelligent Firewall. One is increasing the detection ratio, the other is reducing false positives. Especially Internet-scale viruses are increasingly serious and have various formats. In infected data packets, establishing existences of classes or establishing a rule is necessary, even though it is really hard to find. As I defined before, classification is based on finding proper information and establishing links between data, on the other hand, recognition is based on making a decision about the information after classifying the data. For intelligent engines to do these classification and recognition roles, two engines are needed, each to handle one of the two mentioned abilities.

The Intelligent Firewall has packet-based detection components (see Figure 6.1). To overcome weaknesses of a traditional firewall, this Intelligent Firewall contains packet classification and smart detection features. These detection components deal with not only a packet's header but also with its payload. To capture packets from the datalink layer I have used libpcap (LIBPCAP 2002), and to decode data packets I have modified parts of snort (SNORT 2002). These packet-based detection components will be useful if their detection rate is reasonably high with an acceptably low rate of false positive. Moreover, the packet-based classification engine and the smart detection engine will not only detect anomalous network traffic as in IDSs, but also detect unusual data packets potentially belonging to Internet-scale viruses.

The main role of the packet-based classification engine is to classify packets which could be malicious and estimate the probabilitiy of them being malicioius, on the other hand, the main role of the smart detection engine is recognizing malicious patterns in data packets that have a certain probability of being malicious. The packet-based classification engine

deals with investigating relations of packets based on certain parts of evidences, on the other hand, the smart detection engine recognizes patterns which represent maliciousness based on reasoning under uncertainty (the probability of being malicious data is high). By and large, the smart detection engine will deal with attached malicious files.

I apply these concepts to establish a rule for the packet-based classification engine, and to identify the classes needed in the smart detection engine. To reduce error, the smart detection engine will deal with malicious packets which are filtered by the packet-based classification engine and have a high probability of being malicious.

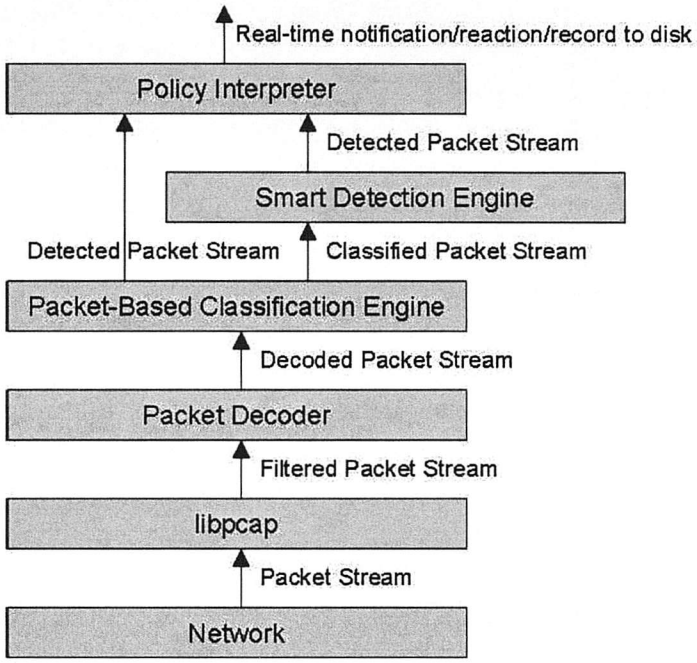

FIGURE 6.1: Packet-Based Detection Components in the Intelligent Firewall. After decoding packets, the classification engine and the smart detection engine deal with not only packet headers but also packet payload to detect malicious packets.

6.1 Data Packet Detection

As I have surveyed social engineering attacks and secure vulnerability attacks, attack trends of Internet-scale viruses are that they are automatic and sophisticated, and intruders misuse infrastructure for their own purpose. To identify Internet-scale viruses, in addition to the usual network control ability of a firewall, data packet detection is compulsory. A packet-header check in a firewall is not sufficient to detect infrastructure based attacks, for example, to identify multiple file extension, domain-name-style file names and long subjects in email attachments. Win32/SirCam, Win32/Gibe,

Win32/Myparty, Nimda, and Win32/BadTrans misuse this point. Moreover, I believe that attack packets are peculiar. Likewise anti-virus softwares can detect viruses in programs using a unique character, which is called a virus signature, a peculiar attack pattern will also appear in the data packets such as Code Red's packet. IDSs run a process known as anomaly detection. An IDS constantly monitors network traffic and compares the stream of network packets with what it perceives as normal network traffic. Anomaly detection appears to be applicable not only to intrusion detection but also to virus monitoring (M.Swimmer 2000), now not being applied to the level of the full network traffic, but to single data packets. The improved virus monitor will examine data packets as usual. Besides checking against known malicious-code patterns, it will check whether it sees a pattern that it perceives as potentially malicious and will react accordingly, e.g. by creating some sort of warnings. However, during my investigation of data packets in some of the good packets, I could identify very similar patterns in packets which seem to contain malicious code, e.g. the BAT911/Chode worm. These were packets sent by Microsoft Servers to NetBIOS and DNS lookup services. For example, port 137 is reserved for the NetBIOS name service and port 138 is reserved for the NetBIOS datagram service. The subsequent packet was assumed to contain the signature of the "BAT911/Chode" worm even though it was a benign packet[1]:

```
05/24-13:10:13.082716 152.78.70.46:137 -> 152.78.70.127:137
UDP TTL:128 TOS:0x0 ID:47635 IpLen:20 DgmLen:78
Len: 58
.....
0x0030: 00 00 00 00 00 00 20 45 45 46 44 46 44 45 46 43 .....EEFDFDEFC
0x0040: 41 43 41 43 41 43 41 43 41 43 41 43 41 43 41 43 ACACACACACACACAC
0x0050: 41 43 41 43 41 42 4C 00 00 20 00 01            ACACABL .. ..
```

6.2 Dynamic Packet Handling Ability

Although a firewall is able to control a network and maintain its connectivity, it handles packets only statically. Through open ports, a firewall would not inspect/control packets willingly. According to the analysis of distributed denial of service attack tools, it is well known how to use tools such as TFN (D.Dittrich 1999c), TFN2K, Trinoo (D.Dittrich 1999a) and Stacheldraht (D.Dittrich 1999b). These programs use not only TCP and UDP packets but also ICMP packets. Moreover, because the programs use ICMP_ECHOREPLY packets for communication, it will be very difficult to block attacks without breaking most Internet programs that rely on ICMP. Since TFN, TFN2K and Stacheldraht use ICMP packets, it is much more difficult to detect them in action, and

[1]It should be noted that such packets are rare.

packets will go right through most firewalls. The current only sure way to destroy this channel is to deny all ICMP_ECHO traffic into the network. Furthermore, the tools mentioned above use any port randomly; it is hard to prevent the port from an attack in advance using the fixed port close scheme in current firewalls. Therefore, to prevent degradation of service on the network and to deny this kind of malicious packet, dynamic packet handling on the level of firewalls is crucial.

6.3 Packet-Based Classification Engine Using Bayesian Networks

The purpose of the packet-based classification engine is to determine whether a packet header is valid or not, to classify the packets into packet classes such as HTTP traffic, SMTP traffic, and FTP traffic, and to make a decision whether the packet classes are filtered into the smart detection engine or are dropped according to their probabilities of being malicious, which is deduced by a Bayesian Network. The packet-based classification engine will aim to cover the TCP/IP/ICMP protocols. Before classifying each packet, the engine checks the protocol header part of packets. In the IP protocol, according to the Internet Protocol Standard (RFC791 1981), an IP header length should always be greater than or equal to the minimal Internet header length (20 octets) and a packet's total length should always be greater than its header length. IP address checks are also important since land attacks [2] use the same IP address for source and destination. According to the TCP standard (RFC793 1981), neither the source nor the destination TCP port number can be zero, and TCP flags, e.g. URG and PSH flags, can be used only when a packet carries data. Thus, for instance, combinations of SYN and URG or SYN and PSH become invalid. In addition, any combination of more than one of the SYN, RST, and FIN flags is also invalid. The classification engine will use a probability table for protocols. If an incoming packet does not satisfy the protocol standard, its probability of being malicious will increase. In this thesis, I consider only mail packets.

6.3.1 Classifying Mail Packets

Many Internet-scale viruses consist of a virus program and several auxiliary files which support the virus program to spread smoothly. If we can classify these auxiliary files, the

[2]The land attacks are also known as IP DOS (Denial of Service). The land attack involves the perpetrator sending spoofed packets with the SYN flag set to the victim's machine on any open port that is listening. If the packets contain the same destination and source IP address as the host, the victim's machine could hang or reboot. In addition, most systems experience a total freeze up, where as CTRL-ALT-DELETE fails to work, the mouse and keyboard become non operational and the only method of correction is to reboot via a reset button on the system or by turning the machine off.

virus parts of a program could be estimated as to have a high probability of being malicious. The packet-based classification engine classifies Internet-scale virus packets from normal packets using a packet's header and payload. Classification of payload depends on packet classes. For example, if incoming packets are mail packets which use the SMTP protocol, the engine classifies this packet as SMTP traffic and examines the mail subject, attachment files, body and so on, after checking whether an SMTP packet overflow occurs or not. In HTTP traffic, the engine checks URL, HOST, MIME type, and so on. The decision of the packets' maliciousness depends on a probabilistic analysis of data packets.

Classified packets will be classified additionally to pass through either the smart detection engine or the policy interpreter, which deals with the packets following a security policy.

6.3.2 A Probabilistic Analysis of Mail Packets

According to SMTP (Simple Mail Transfer Protocol) Standard (RFC0821 1982) (RFC2821 2001), I can get SMTP's semantic entities and create a Bayesian decision tree to present probabilistic dependencies related to the maliciousness of a data packet, like in Figure 6.2. This shows us a better model to understand mail packets. It includes filenames as a semantic entity, as well as a substructure that I have created to analyse mail packets, and dependencies between entities in this substructure of mail packets.

Moreover, this Bayesian decision tree represents relations among nodes and gives us accurate estimation of conditional probabilities. Furthermore, I connect each terminal node with the bottom node "malicious packet" to represent that this Bayesian decision tree is used to estimate the probability of packets to be malicious.

The header node's "MIME type" is followed by Message Header Extensions for NON-ASCII Text (RFC2047 1996), on the other hand, the "MIME type" of the body node is followed by several standards, eg. Format of Internet Message Bodies (RFC2045 1996), Media Types (RFC2046 1996) and Registration Procedure (RFC2048 1996). Each node is represented by a symbol and there is a probability between nodes, for instance, the "MIME type" of the "body" node is denoted by $SDBM$, the MailContent of the "MIME type" node is $SDBMC$. The probability $P(SDBMC \mid SDBM)$ between the two nodes $SDBM$ and $SDBMC$ represents the probability of "MailContent" having a particular structure given the particular structure "MIME type" in the body content of a mail packet. That is why I call this kind of probability a link matrix. Like this, the other parts are also following the same notation except for the file extension parts.

As Figure 6.2 shows, the file extension part is more complicated than the other parts. Because of multiple parents, pairs of link matrices need to be created. I will deal with

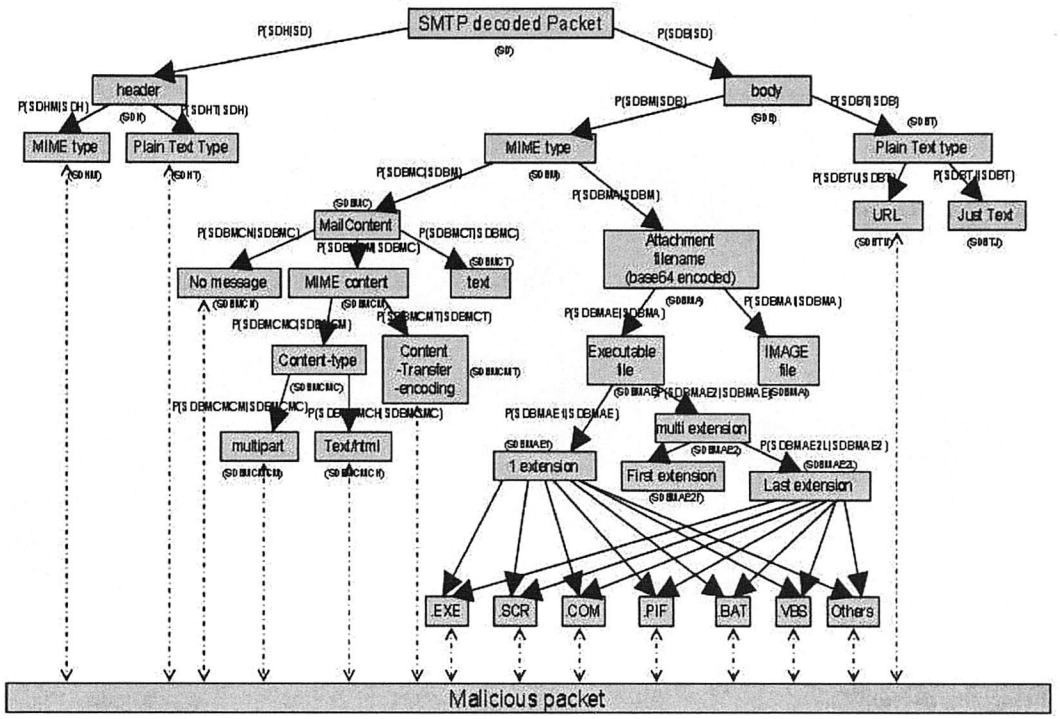

FIGURE 6.2: For malicious packet classification, Probabilistic Dependency Diagram of mail packets

more details in the next section using real data. Using the Bayesian decision tree, we can get the probabilities of being malicious for each node from a virus database. Using this information, the packet-based classification engine will make a decision whether packets could be malicious or not, based on a probabilistic analysis using the decision tree of Figure 6.2.

6.3.3 Applying Real Data to a Bayesian Decision Tree

In most situation, we do not focus on defeasible assumptions, but rather on their logical consequences. Especially when this logical consequences come from reality, the assumption is supported strongly. I would call this logical consequences of defeasible assumptions 'beliefs'. As time goes by and new evidence is oberved, changes in defeasible assumptions lead to changes in beliefs. Belief change leads to understand systems over time. This belief can be formed as a Bayesian Decision Tree. In additon, I believe that applying real data to a Bayesian Decision Tree represents the systems at the moment time of real data.

In order to apply real data, I have used MRTG (Multi Router Traffic Grapher) [3] to check

[3] The Multi Router Traffic Grapher (MRTG) is a tool to monitor the traffic load on network-links. MRTG

TABLE 6.1: Email messages and Virus numbers based on Mail Entering Numbers [4]Note for reading this chart.

	Email numbers	in a Year	Email numbers	in a Month	Email numbers	in a Week
Type	Messages	Viruses	Messages	Viruses	Messages	Viruses
Max	215000	4957 (2.3%)	215000	395 (0.2%)	109000	361 (0.3%)
Average	8339	165 (2.0%)	8995	201 (2.2%)	8542	293 (3.4%)
Current	9797	274 (2.8%)	9633	223 (2.3%)	9633	223 (2.3%)

email messages entering ECS network during past few years, and have analysed whole virus database from Ahnlab, then built a database of email packets. Through data from an existing database (see TABLE A.3, TABLE A.4, TABLE B.1, TABLE B.2, and TABLE B.3), we can get the probability of malicious mail packets, denoted P(SD) (see Figure 6.2), the probability of malicious executable files, denoted P($Exec$), and the probabilities of malicious executable-formatted files, denoted P($format$): P($.EXE$), P($.SCR$), P($.COM$), P($.PIF$), P($.BAT$), P($.VBS$), and P($Others$).

TABLE 6.1 shows email messages and virus numbers among the messages, this information is built by MRTG based on mail entering numbers on ECS. With this TABLE 6.1, I can make a matrix for the probability of malicious mail packets. This data came from average virus messages in tables (Ref. Details in TABLE B.1, TABLE B.2, and TABLE B.3.) I use three average data values, not just one, for accuracy reasons.

$$\mathbf{P(SD)} = \left(\ 0.020, 0.022, 0.034 \ \right)$$

Even though P(SD) is called a prior information in a particular model, it is the probability of some event prior to updating the probability of that event, within the framework of that model using new information. It does not mean a probability prior to any information.

The other probabilities, which I can get from tables: TABLE A.3 and TABLE A.4, are P($Exec$) = 0.8, P($.EXE$) = 0.64, P($.SCR$) = 0.22, P($.COM$) = 0.06, P($.PIF$) = 0.22, P($.BAT$) = 0.03, P($.VBS$) = 0.02, and P($Others$) = 0.12. These probability values are computed by summation of each extension's percentage which is displayed by terminal

generates HTML pages containing graphical images which provide a live visual representation of this traffic.

[4]Email messages are measured by MRTG every second. To display the visual representation of this traffic, MRTG uses the weekly representation with 30 minute average data, the monthly representation with 2 hour average data, and the yearly representation with 1 day average data. Each number of messages represents a max, an average, and a current in a year, a month, and a week. MRTG displays this number which received data during a year, a month, and a week, in current time per day. For example, a max in a year is chosen by the maximum received number per day during one year, in a certain time when we measure. Please note that the year data is based on the 1 day average, the month data is based on 2 hour average data, and the week data is based on 30 minute data. That is the reason that the current data of a year is different from that of a week and a month.

nodes of Figure 6.3. I applied these probabilities to the Bayesian decision tree in Figure 6.3, which I created for a probabilistic analysis of mail packets.

Included in Figure 6.3 is that, if an executable filename has a multi-extension, this file is almost 100% abnormal i.e. it has an extremely high probability of being malicious. Therefore, we do not need to calculate this part of probability individually. The packet-based classification engine should deal with these parts without calculating probability of being malicious for the obvious abnormal parts. In addition, the smart detection engine will recognize patterns, which represent packet maliciousness, based on reasoning over the probability of malicious data in the packet-based classification engine. If the maliciousness probability is high based on the probabilty in the packet-based classification engine, it means the protocol specification part of current packets is odd-looking or the protocol behaviour does not follow standards. However, to reduce error, the role of smart detection engine is to recognize and locate malicious patterns in the packets, which are filtered by the packet-based classification engine and are having high probability of being malicious. To deal with it, the smart detection engine will deal with mail attached documents and network traffic packets. Basically all packets with a maliciousness probability above a certain threshold will be filtered into the smart detection engine for examination. The threshold has to be set in a relatively arbitrary fashion first and then be adapted when fine-tuning is applied to the decision procedures. Some odd-looking packets can ,however, include legitimate traffic. For example, storms of FIN and RST packets and fragmented packets with the 'don't fragment packet' fiag set. Then, the policy interpreter will analyse the information it gets from the two engines and will decide whether to drop a packet or let it pass through the firewall, based on a packet's probability to contain malicious content and a specific security policy that rules the policy interpreter's decisions.

In this point, one part of the future work could be this kind of problem, how to deal with this in a Bayesian decision tree, when the real situation reflects almost 100% abnormality. Furthermore, the current database is not sufficient to calculate all needed probabilities precisely, because of a lack of data. Moreover, all the way to calculate the probability in each node looks very cumbersome, I need to find a more intelligent way to calculate probabilities in the Bayesian networks. However, this is a part of the future work.

6.4 Smart Detection Engine Using Neural Networks

The smart detection engine deals with the filtered packets from the packet-based classification engine, which have a high probability of being malicious. I will train the smart detection engine to distinguish anomalous data packets from normal packets

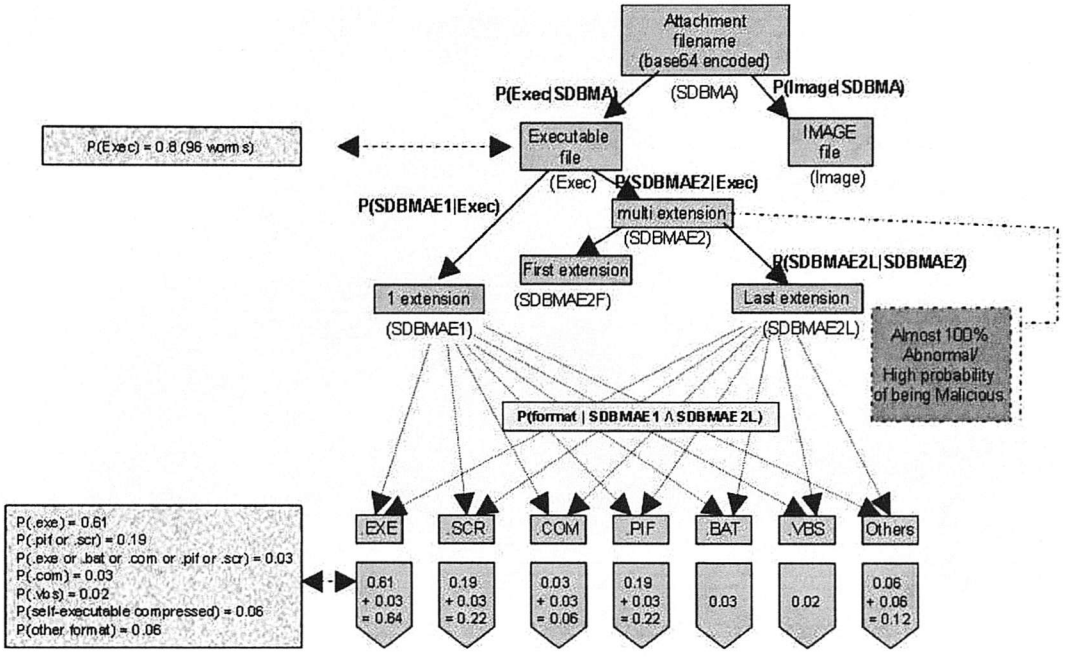

FIGURE 6.3: Executable files' Probability based on Bayesian Decision Tree

(Cannady and Mahaffey 1998), (C.Lee and V.Heinbuch 2001). I expect the smart detection engine will be able to detect malicious executable packets such like file viruses and file worms. In the file virus case, the infected data packet will be different from the original data packet. However, unlike anti-virus software, this engine does not need to match the infected part of a program exactly (G.Tesauro and G.B.Sorkin 1996). Detecting viruses in a system or file is a role of anti-virus software. This engine will help to reduce the risk from Internet-scale viruses entering a protected sub-network by identifying potentially malicious packets at the network boundary. The methodology to be applied to this engine are neural networks. To detect bad patterns, I take five aspects into account: pattern classification, competitive learning, unsupervised learning, [5] good performance, and flexible time delay. Possession of such features can be found in neural-network-based detection.

Note that the smart detection engine deals with virus infected files rather than file worms. In a file worm's case, the packet-based classification engine can classify this file worm based on the context information as depicted in Figure 6.2.

[5] Bad patterns to be detected do not produce only obvious output. Considering this fact, competitive learning and unsupervised learning are the only concepts which can be suitably applied to the smart detection engine. Without observing target output, this kind of learning is able to provide self-organisation.

6.4.1 Mail Attached File's Sequence Similarity

In this section, I discuss how to classify mail attached files, what kind of sequence similarity exists, and what kind of case the smart detection engine deals with. Figure 6.4 represents how to decide the maliciousness of a mail attached file using a flow chart. This decision occurs in the packet-based detection engine then the engine will pass sequenced packets which need to be checked for abnormal patterns into the smart detection engine.

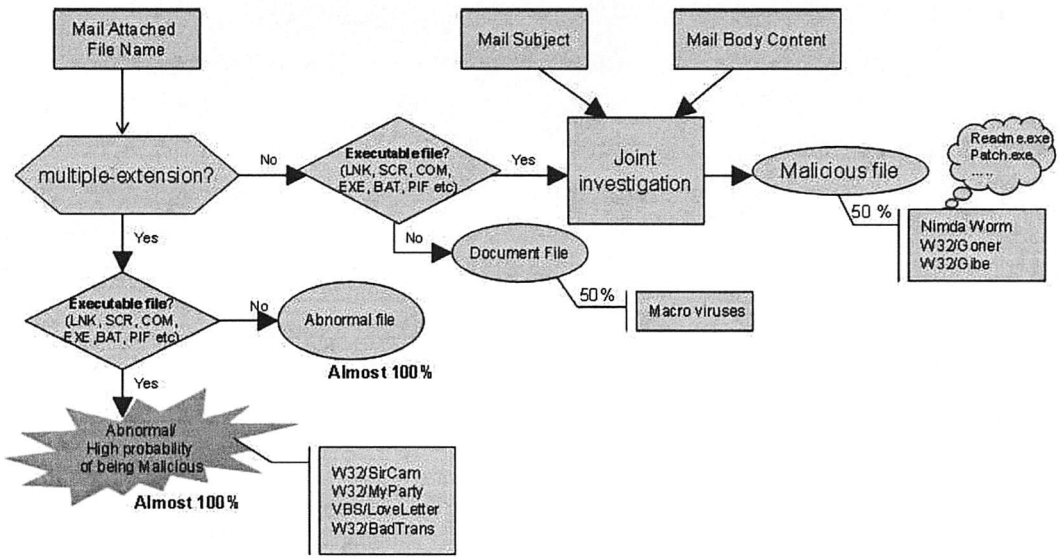

FIGURE 6.4: Mail Attachment Sequence Similarity and Basic probability of Being Malicious.

As Figure 6.4 shows, if a mail attachment has multiple file extensions, and is executable, then it contains with a very high probability abnormal and/or malicious code. W32/SirCam, W32/MyParty, VBS/LoveLetter, and W32/BadTrans are in this category. Otherwise, even if the file is not executable, the multi-extension makes it appear abnormal. Without considering file's contents, because of multi-extension filename, we assume this file is almost 100% abnormal. So if the packet-based classification recognizes such a file structure, the packets which contain this file will be filtered or dropped based on a policy in place. In this case, the file does not need to be sent it to the smart detection engine.

If the attached file has a single extension, then, if the file is executable, the packet-based classification engine will investigate the packets' other information, e.g. mail subject and mail body contents, and decide whether to pass the packet into the smart detection engine or not. Nimda Worm, W32/Goner and W32/Gibe are covered by this case. If the packet-based classification engine cannot make an informed decision due to a lack of data, the engine assumes that the case it currently deals with 50% probability of being malicious.

However, if the file has a single extension and is not executable, then the file is a document. In this case, if the packet-based classificaiton engine cannot get the file's probability, the packet-based engine assumes this document has 50% probability of being malicious. Therefore, the file is sent to the smart detection engine to be checked further for macro viruses.

Subsequently I summarise how the flow-graph of Figure 6.4 describes how to estimate the probability of a file to be malicious. Note that without any specific probabilities for a case, the packet-based classification engine uses default probabilities. Moreover, the two cases of 50% maliciousness (the "document file" and "malicious file" nodes of Figure 6.4), are sent to the smart detection engine.

- Almost 100% probability of being abnormal/malicious executable file: Before the process of the smart detection engine, the packet-based classification engine filters or drops the packets which contain this file.
- At least 50% probability of being malicious document file: This file could contain macro viruses, the packet-based classification engine sends it to the smart detection engine to check for macro virus patterns.
- At least 50% probability of being malicious executable file: If it is likely that this is a file worm (i.e. it comes with auxiliary files), the packet-based classification engine will investigate the packets' auxiliary information together with the other evidences. If it is likely that this is a virus-infected file, the smart detection engine will deal with it, aiming at recognizing the infection pattern.

6.4.2 Recognition of File Structure Pattern Using Self-Organizing Map

As I analyzed malicious virus patterns in Chapter 4, there are infected file structure patterns to recognize parasitic viruses. However, there is no precise border between the virus part and the proper file content. Self-Organizing Maps (SOM) are suitable to be applied to recognizing virus patterns. There are different types of virus-infected files (see Figure 6.5): These types are based on my analysis of malicious virus patterns in Chapter 4.3.

- Prepending Viruses: The virus part is located at the top of a file.
- Appending Viruses: The virus part is appended at the end of a file.
- Inserting Viruses: The virus part is somewhere in the middle of a file.
- Macro Viruses Pattern: The virus part is more specific than the other viruses.

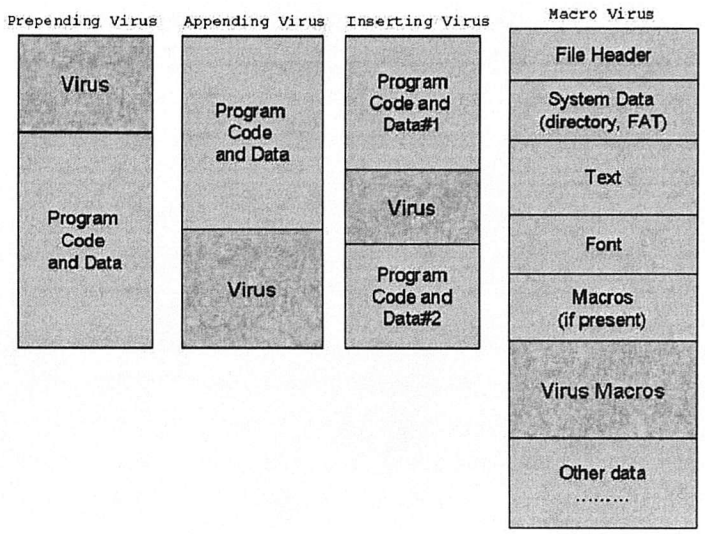

FIGURE 6.5: Virus Patterns based on the position of virus in an infected file.

Subsequently I discuss how SOMs operate and how they can be applied to detect virus patterns. SOM does not refer to any ideal outputs for learning. Moreover, it classifies the input patterns into arbitrary categories by using the Kohonen Layer (see Figure 6.6).

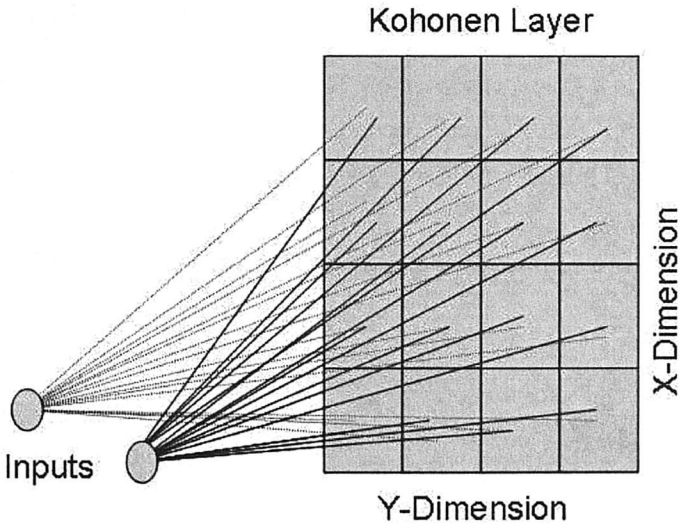

FIGURE 6.6: The Kohonen Layer

In the Kohonen Layer, the learning process occurs as cells compete to get the input patterns which are appropriate for the winning cells. Different categories of input patterns are won by different cells, therefore, categorical outputs are given by distinctive (x, y) coordinates of the winning cells.

To motivate that SOMs can be applied to detect attack patterns, it is worth mentioning the following related work. For instance, a SOM with six-dimensional input vectors, and a 30 X 25 Kohonen Layer (Nguyen 2002), was used to detect the mailbomb [6] attack. They examine a computer network data stream captured by real-time TCPTrace. This TCPTrace reports three different types of messages which are created by analyzing the headers of packets ; open messages, close messages, and update messages. Using these messages, they can get 6 statistics : interactivity, average size of connection request, average size of connection reply, sum of idle request-reply time, sum of idle reply-request time, and number of connections. Data for these 6 dimensions was normalized and was used to train the SOM. After training with normal packets for 4 weeks, they detected the mailbomb attack using this SOM. According to their result, their program not only detected all mailbomb attacks but it could also detect errors in the input data.

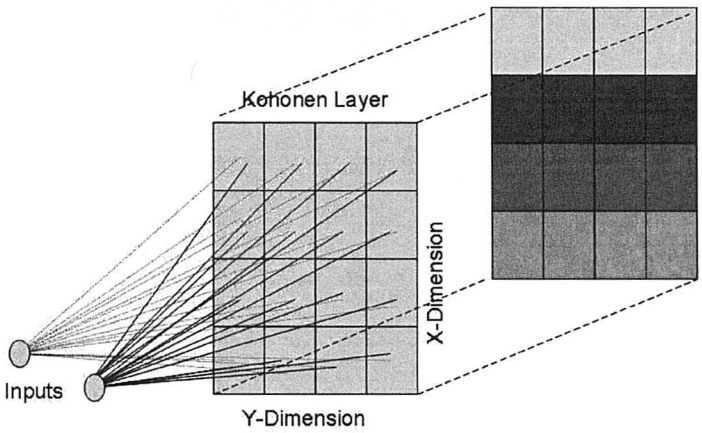

FIGURE 6.7: 4-type of outputs categorized by the Kohonen Layer

If we used, for example, a (4 X 4) 2-dimensional Kohonen Layer, the Kohonen Layer will categorize 4 types of inputs like in Figure 6.7. The four rows present the top of a file, two middle parts of a file, and the end of the file. So based on the position of a virus part in a file, categorical outputs are given, then according to these outputs, we can analyse the file. My goal to use SOM is to identify the position of the virus data and to identify that there is virus data in the file.

However, if we use an X-dimension of 7, the Kohonen layer will categorize 7 types of inputs like in Figure 6.8. I prefer to stick to Y-dimension as 4, because packets are sent by bytes. In general we will need a finer granularity of the Kohonen layer. For implementation purposes the granularity of each dimension should be a multiple of 4. Hence I expect to use $4n$ X $4n$ Kohonen layer for a suitably chosen n.

[6] A mail bomb is the sending of a massive amount of email to a specific person or system. A huge amount of mail may simply fill up the recipient's disk space on the server or, in some cases, may be too much for a server to handle and may cause the server to stop functioning.

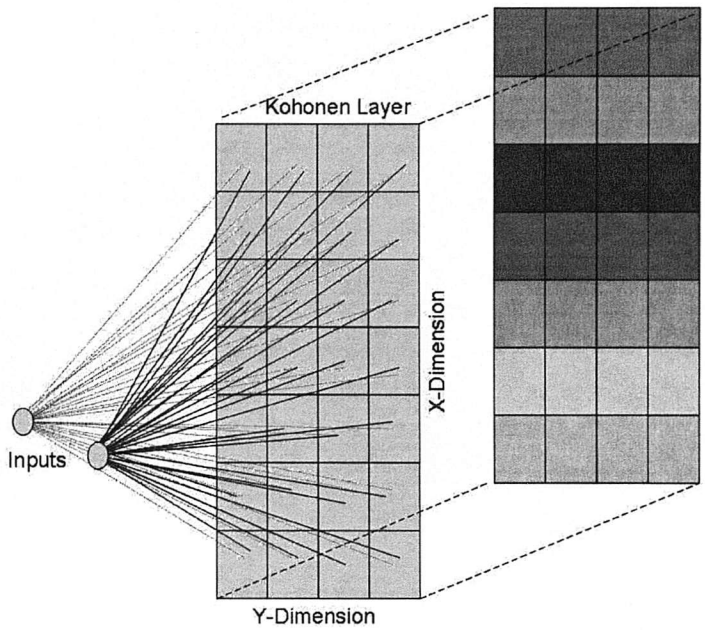

FIGURE 6.8: 7-type of outputs categorized by The Kohonen Layer

To categorise the file and to detect a virus part in the file precisely, Self-Organizing Maps should be one of solutions. Other possibilities will be examined in future work.

Chapter 7

Conclusions

7.1 Conclusion

Currently I have produced a rudimentary firewall prototype using libpcap (LIBPCAP 2002) and snort (SNORT 2002) modules. In order to capture packets from the datalink layer I have used libpcap, and to decode data packets I have modified parts of snort. My goal is to build a smart detection engine and integrate it into the prototype firewall which I have built. This prototype firewall runs like packet-filter firewalls, I referenced snort for packet decoding, however, eventually this firewall will be separated from snort part and libpcap to have a more light weight-implementation (only few features of snort are exploited by my implementation.) Meanwhile I built this prototype, I also captured and examined packets to design the packet-based classification engine and the smart detection engine. After training these engines, they will be included into the firewall. These engines will be useful if their detection rate is higher than that of traditional IDSs' anomaly detection rate with an acceptably low rate of false positives.

During my investigation of data packets, in some of the good packets I could identify very similar patterns in packets which seem to contain malicious code, e.g. the BAT911/Chode worm. These were packets sent by Microsoft Servers to NetBIOS and DNS lookup services. For example, port 137 is reserved for the NetBIOS name service and port 138 is reserved for the NetBIOS datagram service. Compared with benign intranet packets, Microsoft packets sent by running Microsoft servers look very distinguished and very many such intranet packets are sent per second. However, extranet packets are different from intranet packets, considering the engines I need to think of the network environment to train the engines. I also have created an isolated network to build the intelligent firewall and run tests with malicious packets.

The summary of this thesis is the following,

- Internet-Scale Viruses: An Internet-scale virus is a piece of code or a program that performs unintended tasks and brings unintended side effects such as damaging a system or network, obtaining secure information without permission, putting a system or network under heavy load, and so on.

  Approximately 86% of all Internet-scale viruses have a Win32 executable format. Moreover, about 80% of Windows file worms are transferred via email, and about 61% of these files have an .EXE extension.

  I have exhaustively analysed statistical information about viruses as well as the structures of viruses. This information has been used throughout this thesis.

- Present Network Security Systems: Current intrusion detection systems(IDSs) do not prevent an intrusion from happening. Although anti-virus servers and IDSs are installed to cooperate to prevent an Internet-scale virus attack, new virus information needs to be updated constantly. Because new viruses will only become detectable after their pattern characteristics have been analyzed and are made available. Furthermore, firewalls are used to guard and isolate connected segments of inter-networks. Inside network domains are protected against outside untrusted networks, or parts of a network is protected against other parts. These firewalls use only TCP/IP headers to decide whether data packets are safe or not.

  I have studied the characteristics of existing security systems, including their weaknesses, to come up with my prototype of an Intelligent Firewall.

- The Intelligent Firewall: The Intelligent Firewall focuses on high detection ratio with low false positives for risk management against novel attacks, in this thesis particularly, against novel Internet-scale viruses. The Intelligent Firewall has packet-based detection components: the packet-based classification engine and the smart detection engine.

  The firewall architecture is the result of an analysis of existing AI techniques, taking into account the gained knowledge about attack patterns and weaknesses in current security systems.

- Classification and Recognition: Classification is based on finding proper information and establishing links between data, on the other hand, recognition is based on making a decision about the information after classifying data. The Intelligent Firewall will classify packets into safe and unsafe packets.

- The Packet-Based Classification Engine with Bayesian Network: The packet-based classification engine has three goals. First is to determine whether a packet header is valid or not. Second is to classify packets into packet classes such as HTTP traffic, SMTP traffics and FTP traffic. Last is to make a decision whether the packet class is filtered into the smart detection engine or dropped according to its probability to

be malicious, which is deduced by analysing structural information using a Bayesian Network.

The Bayesian network presented in this thesis incorporates knowledge gained from the examination of past virus attacks.

- The Smart Detection Engine with Self-Organizing Map (SOM): The smart detection engine will deal with the filtered packets, which have a high probability of containing malicious content, from the packet-based classification engine.

  The smart detection engine deals with virus infected files rather than file worms, which can infect other files in a virus-like manner. The infected file then contains a pattern one can recognize, but there is no specific location within the file where this pattern will be found. Using a Self-Organizing Map, the smart detection engine can recognize these categorized patterns and their location in the file.

  The smart detection engine has only been developed conceptually in this thesis. Its concreate realization will be part of future work.

The Intelligent Firewall focuses on risk management against novel attacks. Recently, although new serious Internet attacks have increased, virus detection focuses on anti-virus software and intrusion detection systems (IDSs). However, it appears to be time also to focus on a new firewall to support the other security systems. There was no research about firewalls having the ability to make decisions by deduction, using e.g. a Bayesian Network and to recognize categorized patterns of virus structures in infected files. My goal is to build a prototype of such an Intelligent Firewall. In this MPhil thesis, I have developed the general concepts that the Intelligent Firewall will comprise. My architecture decisions were based on a thorough analysis of available data about viruses as well as a literature study of suitable approaches for the detection of novel viruses. This led to the Intelligent Firewall architecture, which contains the packet-based classification engine and the smart detection engine. Through the packet-based classification engine, the firewall can deal with file worms and infected virus files can be recognized by the smart detection engine. I expect that based on the firewalls' abilities, the number of serious Internet-scale viruses entering a network can be decreased. Furthermore, the Intelligent Firewall will increase the ability to detect novel network attacks. This novel firewall model will enable one to detect new malicious code entering a network already before its precise signature is known. The considered approaches will be useful not only to develop a new type of firewall, but also to exploit their use in other security systems.

7.2 Future Work

Future work will involve extending the architecture model and implementing a complete prototype. Currently, I decided to use Bayesian Networks for the classification engine and SOMs for the smart detection engine. These concepts form the basis of my future work and will possibly be refined during the course of their concrete realization. I am planning to implement the intelligent firewall on a network of computers to evaluate its performance after designing it in detail. One important area of future work will be the design of the classification engine and the smart detection engine to perform efficiently. The development of these engines requires many experiments using real attack packets to generate sufficient evidence of the applicability of the approach. Ideally, this work will be done in co-operation with industry or security sources to develop a standard data set consisting of infected data and many different sets of benign data. I also plan to extend this system to detect novel network attacks in other areas such as DOS(Denial of Service) and DDOS(Distributed Denial Of Service) attacks.

Appendix A

Virus Statistics based on Ahnlab database

I made virus statistics after surveying Ahnlab database, and then made these tables in accordance with our own purpose. The date of records in Virus Information is 06 August 2002.

In this database, the "virus" item includes only PC viruses, not Internet-scale viruses. Because of PC virus is originally called as a virus. Internet-scale viruses are contained in the "worm" item.

TABLE A.1: Distribution of Internet-Aware Viruses

Computer Virus Total Number : 989
(Current date: 06 August 2002.)

Type	Number	Percentage
Virus	776	78.5 %
Worm	139	14.1 %
Trojan	62	6.3 %
Hoax	4	0.4 %
Joke	4	0.4 %
Etc	3	0.3 %

TABLE A.2: Classification of Internet-Aware Viruses

Virus	Total: 776	Trojan Horse	Total: 62	Worm	Total:139
Windows File	58	Windows File	35	Windows File	120
DOS File	435	DOS file	24	DOS file	1
Boot Virus	101	Backdoor	0	Script	16
Boot/File Virus	9	Script	3	Macro	1
Script Virus	57			Other Executable File	1
Linux Virus	1				
Palm Virus	0				
Macro Virus	114				

TABLE A.3: Percent of Windows Worm

Windows Worm	120
Mail Transferred	96
Ratio	80%

TABLE A.4: Classification of Windows File Worms

Prevalence of Windows File Worm Via Email : total 96 worms

File Format	Number	Percent
.EXE file	58	60.4 %
.PIF or .SCR file	18	18.8 %
.BAT.COM.EXE.PIF.SCR file (among these, choose 1 format)	3	3.1 %
.COM file	3	3.1 %
.VBS file	2	2.1 %
Self-executable compressed format	6	6.3 %
Other	6	6.3 %

Appendix B

ECS Email Record in University of Southampton

I made these tables using ECS email record which is updated by MRTG.
The date of record of these tables is Thursday, 25 July 2002 at 11:04.

TABLE B.1: Email based Mail Entering Numbers into ECS in a Year

Type	Messages	Viruses	Spams	Type	Viruses	Spams
Max	215000	4957	1881	Max	2.3%	0.9%
Average	8339	165	706	Average	2.0%	8.5%
Current	9797	274	1482	Current	2.8%	15.1%

TABLE B.2: Email based Mail Entering Numbers into ECS in a Month

Type	Messages	Viruses	Spams	Type	Viruses	Spams
Max	215000	395	1881	Max	0.2%	0.9%
Average	8995	201	1383	Average	2.2%	15.4%
Current	9633	223	1586	Current	2.3%	16.5%

TABLE B.3: Email based Mail Entering Numbers into ECS in a Week

Type	Messages	Viruses	Spams	Type	Viruses	Spams
Max	109000	361	1710	Max	0.3%	1.6%
Average	8542	293	1383	Average	3.4%	16.2%
Current	9633	223	1586	Current	2.3%	16.5%

TABLE B.4: Viruses Detected Entering ECS

Total 17300

Name	Number
W32/Sircam.A	50
W32/Flcss	11
W32/Klez.G	10
W32/Navidad.B	7
W32/Badtrans.B	5
W32/Magistr.A	3
W32/Navidad	2
W32/Hybris.B	2
Others	10

Bibliography

Ahnlab (2002, August). Virus information.

Anderson, D. and G. McNeill (1992, August). Artificial neural networks technology. Technical report, Rome Laboratory, Griffiss AFB, NY, 13441-5700. A DACS state-of-the-art Report, Data & Analysis center for Software.

Arnold, W. and G. Tesauro (2000). Automatically generated win32 heuristic virus detection. Proceedings of the 2000 International Virus Bulletin Conference.

Bace, R. and P. Mell (2001, August). Nist special publication on intrusion detection system. Technical report, NIST (National Institute of Standards and Technology). Special Publication 800-31.

Cannady, J. and J. Mahaffey (1998). The application of artificial neural networks to misuse detection: initial results. the 1st International Workshop on Recent Advances in Intrusion Detection (RAID 1998).

CERT/CA-2000-04 (2000, May). Cert advisory ca-2000-04: Love letter worm. Online Publication.

CERT/CA-2001-19 (2001, July). Cert advisory ca-2001-19: Code red worm exploiting buffer overflow in iis indexing service dll. Online Publish.

CERT/CA-2001-22 (2001, July). Cert advisory ca-2001-22: W32/sircam malicious code. Online Publication.

CERT/CA-2001-23 (2002). Cert advisory ca-2001-23: Continued threat of the code red worm. Online Publication.

CERT/CA-2001-26 (2001, September). Cert advisory ca-2001-26: Nimda worm. Online publication.

CERT/CC (2002, April). Cert/cc, overview incident and vulnerability trends.

CERT/IN-2001-09 (2001, August). Cert incident note in-2001-09: Code red ii: Another worm exploiting buffer overflow in iis indexing service dll. Online publication.

CERT/IN-2001-14 (2001, November). Cert incident note in-2001-14: W32/badtrans worm. Online publication.

CERT/IN-2001-15 (2001, December). Cert incident note in-2001-15: W32/goner worm. Online Publication.

CERT/IN-2002-01 (2002, January). Cert incident note in-2002-01: W32/myparty malicious code. Online Publication.

CERT/IN-2002-02 (2002, March). Cert incident note in-2002-02: W32/gibe malicious code. Online publication.

C.Lee, S. and D. V.Heinbuch (2001, July). Training a neural network-based intrusion detector to recognize novel attacks. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 31*(4), 294–299.

Crows, T. (1999). Introduction to data mining and knowledge discovery. Two Crows Corporation, Third Edition, Online Publication.

David Hand, H. M. and P. Smyth (2001). *Principles of Data Mining.* The MIT Press, Cambridge, Massachusetts. Adaptive Computation and Machine Learning Series, A Bradford Book.

D.Dittrich (1999a, October). The dos project's trinoo distributed denial of service attack tool.

D.Dittrich (1999b, December). The stacheldraht distributed denial of service attack tool.

D.Dittrich (1999c, October). The tribe flood network distributed denial of service attack tool.

de Ridder, D. (1998). Unsupervised methods: Gaussian, parzen and 1-nearest neighbour methods. Online Publication.

D.Michie, D.J.Spiegelhalter, C. E. (1994, February). *Machine Learning, Neural and Statistical Classification.*

G.Cestnik, I. and I.Bratko (1987). A knowledge-elicitation tool for sophisticated users. In I.Bratko and N. Lavrac (Eds.), Progress in machine learning.

Gryaznov, D. (1999). Scanners of the year 2000: Heuristics. Proceedings of the 5th International Virus Bulletin.

G.Tesauro, J. and G.B.Sorkin (1996, August). Neural networks for computer virus recognition. *IEEE Expert 11*(4), 5–6.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation, International Edition/Second Edition.* Prentice Hall.

HouseCall (2002). Trend micro - free online virus scan. Online Publication.

H.Ptacek, T. and T. N.Newsham (1998, January). Insertion, evasion, and denial of service: Eluding network intrusion detection.

I.Kononenko (1990). Comparison of inductive and naive bayesian learning approaches to automatic knowledge acquisition. In B.Wielinga et al. (Eds), Current trends in knowledge acquisition.

Iliescu, S. Pattern recognition : Non-parametric techniques. Teaching COMP473 - Online Publication.

IPA/ISEC (2002, May). Computer virus incident reports. Online Publication.

IS.Yoo and U.Ultes-Nitsche (2002, November). Intelligent firewall: Packet-based recognition against internet-scale virus attacks. Conference on Communications and Computer Networks (CCN 2002).

Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, D. I. E. S. and D. Zamboni (1998, June). An architecture for intrusion detection using autonomous agents. Technical report. COAST Technical Report 98/05.

J.Hughes (1996). A high speed firewall architecture for atm/oc-3c.

J.M.Bernardo and A.F.M.Smith (1994). *Bayesian Theory.* Wiley.

John Mchugh, A. C. and J. Allen (2000). Defending yourself: The role of intrusion detection systems. *IEEE Software*, 42–51.

John Wack, K. C. and J. Pole (2002, January). Guidelines on firewalls and firewall policy. Technical report. Special Publication 800-41.

J.O.Kephart (1994). A biologically inspired immune system for computers. pp. 130–193. Artificial Life IV. Proceedings of the Fourth International Workshop on Synthesis and Simulation of Living Systems, Rodney A. Books and Pattie Maes, eds.

Kaspersky, E. (2000). Virus analysis texts - macro viruses. Online Publication.

K.Ghosh, A. and A. Schwartzbard (1999). A study in using neural networks for anomaly and misuse detection. Online Publication.

Kumar, S. and E. H.Spafford (1995, March). A software architecture to support misuse intrusion detection. Technical report, Department of Computer Science, Purdue University. Technical Report CSD-TR-95-009.

Laing, B. (2000). How to guide-implementing a network based intrusion detection system.

Langley, P. and S. Sage (1994). Induction of selective bayesian classifiers. Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence: Morgan Kaufmann.

LIBPCAP (2002). Tcpdump/libpcap. The Tcpdump Group.

Matthew G.Schultz, e. a. (2001, May). Data mining methods for detection of new malicious executables. IEEE Symposium on Security and Privacy (IEEE S & P 2001). Matthew G.Schultz, Eleazar Eskin, and Erez Zadok.

Matthew G.Schultz, E. E. and E. Zadok (2001, June). Mef: Malicious email filter, a unix mail filter that detects malicious windows executables. USENIX Annual Technical Conference - FREENIX Track.

McAfee (2002). Mcafee home page. Online Publication.

Micro, T. (2002). Trend micro virus protection products.

MS00-052 (2000, July). Microsoft security bulletin/ms00-052. Online Publication.

M.Swimmer (2000). Review and outlook of the detection of viruses using intrusion detection systems. the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000).

Nguyen, B. V. (2002). Self-organizing map(som) for anomaly detection. Spring.

of Photogrammetry, I. and R. Sensing (2001). Bayes decision theory. Online Publication.

O.Kephart, J. and W. C.Arnold (1994). Automatic extraction of computer virus signatures. pp. 178–184. 4th Virus Bulletin International Conference.

Pearl, J. (1988). *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann Publishers, Inc.

P.Langley (1993). Induction of recursive bayesian classifiers. pp. 153–164. Proceedings of the 1993 European Conference in Machine Learning.

P.Langley, W. and K.Thompson (1992). An analysis of bayesian classifiers. pp. 223–228. Proceedings of the 10th National Conference on Artificial Intelligence, AAAI.

Postini (2000). Postini press release. Online Publication.

P.Pfleeger, C. (1997). *Security in Computing*. Prentice-Hall International, Inc. International Edition, Second Edition.

RFC0821 (1982, August). Rfc 821, simple mail transfer protocol. Information Sciences Institute, University of Southern California.

RFC2045 (1996, November). Rfc 2045, multipurpose internet mail extensions (mime) part one: Format of internet messaged bodies. Network Working Group, Standards Track, The Internet Society.

RFC2046 (1996, November). Rfc 2046, multipurpose internet mail extensions (mime) part two: Media types. Network Working Group, Standards Track, The Internet Society.

RFC2047 (1996, November). Rfc 2047, multipurpose internet mail extensions (mime) part three: Message header extensions for non-ascii text. Network Working Group, Standards Track, The Internet Society.

RFC2048 (1996, November). Rfc 2048, multipurpose internet mail extensions (mime) part four: Registration procedures. Network Working Group, Standards Track, The Internet Society.

RFC2821 (2001, April). Rfc 2821, simple mail transfer protocol. Network Working Group, Standards Track, The Internet Society.

RFC791 (1981, September). Rfc 791, internet protocol. DARPA Internet Program Protocol Specification.

RFC793 (1981, September). Rfc 793, transmission control protocol. DARPA Internet Program Protocol Specification.

Richard Duda, P. H. and D. Stork (2001). *Pattern Classification.* John Wiley. 2nd edition, Chapter 4, ISBN: 0-471-05669-3.

Richards, K. (1999). Network based intrusion detection: A review of technology. *Computers & Security 18*, 671–682.

R.W.Lo, K. and R.A.Olsson (1995). Mcf: a malicious code filter. *Computers & Security 14*(6), 541–566.

Schuba, C. L. (1997). *On the Modeling, Design and Implementation of Firewall Technology.* Ph. D. thesis.

SNORT (2002). Snort:the open source network intrusion detection system.

Sophos (2002). anti-virus product.

Specht, D. (1990). Probabilistic neural networks. *Neural Networks 3*, 110–118.

S.R.White (1998). Open problems in computer virus research. IBM online publication.

S.Sarle, W. (1994, April). Neural networks and statistical models. Cary, NC, USA. the 19th Annual SAS Users Group International Conference.

Sundaram, A. (1996). An introduction to intrusion detection.

Symantec (2002). Symantec worldwide homepage. Online Publication.

T.Kohonen (1995). *Self-Organizing Maps.* Berlin, Heidelberg: Springer.

U.Ultes-Nitsche and IS.Yoo (2002, July). An integrated network security approach - pairing detecting malicious patterns with anomaly detection. the 2nd ISSA2002 Conferences.

W.Buntine (1990). A theory of learning classification rules. Dissertation, Department of Computer Science, University of Technology, Sydney.

Wenke Lee, Rahul A.Nimbalkar, K. K.-S. B. (2000). A data mining and cidf based approach for detecting novel and distributed intrusions. RAID 2000.

Wenke Lee, Salvatore J.Stolfo, P. K. (2001, June). Real time data mining-based intrusion detection. DARPA Information Survivability Conference and Exposition II (DISCEX II).

Wenke Lee, S. S. and K. Mok (1999). A data mining framework for building intrusion detection models. IEEE Symposium on Security and Privacy.

WildList (2001). Virus descriptions of viruses in the wild.

William E.Pierson, J. (1998). *Using Boundary Methods for Estimating CDlass Separability*. Chapter 3.

W.Lee, S. and P.K.Chan (1997). Learning patterns from unix processes execution traces for intrusion detection. pp. 50–56. AAAI Workshop on AI Approaches to Fraud Detection and Risk Management. AAAI Press.

Xu, J. and M. Singhal (1999, September). Design of a high-performance atm firewall. *ACM Transactions on Information and System Security 2*(3), 269–294.