The Modeling and Detection of Distributed Port Scans: A Thesis Proposal

Carrie Gates

Technical Report CS-2003-01

Jan 17, 2003

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

The Modeling and Detection of Distributed Port Scans
A Thesis Proposal

Carrie Gates

01/2003

Contents

Chapter 1

Introduction

1.1 Problem Description

The hypothesis of this thesis is that distributed port scans can be represented by a formalized model. The key problem to be addressed is how can a distributed port scan be recognized as being one co-operative port scan, as opposed to multiple independent port scans?

A port scan consists of connection attempts to hosts on a target network in an effort to determine what services are available on what machines. A scan will often be performed by an attacker in order to map a target network, which will help in determining potential weaknesses.

There are a number of variables that an adversary can modify in order to avoid having their port scan detected. These variables include the time between each scan packet (which can be random), the destination IP address and TCP port of the scan (which can be randomly chosen from a list), and the flags within the TCP packet itself. A sophisticated adversary is likely to employ multiple, complementary methods to avoid detection.

It is suspected that sophisticated attackers have started using distributed port scans to gather information. In a distributed port scan, many source machines are used to perform a port scan of the target, where each source is given a separate, although possibly overlapping, piece of the target to scan. If the intrusion detection system on the target network recognizes the port scans, it will not recognize that they are controlled by one attacker, but rather log that many port scans occurred from many hosts. The goal of this thesis is to determine how a distributed port scan can be detected and recognized as such.

The motivation for researching this problem follows, while a more detailed description of the research problem can be found in Section 1.3.

1.2 Motivation

In 2001, the Computer Emergency Response Team/Co-ordination Center (CERT/CC) reported 52,658 computer security incidents. These security incidents were executed against networks potentially containing hundreds of computers[59]. The security incidents reported range from loss of sensitive data to modification of data to rendering computers or their network unusable for a period of time.

The Computer Security Institute (CSI) and the FBI conduct surveys annually of information security professionals in an effort to determine the extent of computer security crimes. In the 2002 survey, they had 503 respondents representing both corporations and government[64]. This survey found that the losses from theft of proprietary information amounted to $170.8 million (41 respondents), and that the losses from fraud totaled $115.8 million (40 respondents). The losses in the United States in 2000 resulting from economic espionage were estimated by the Office of the National Counterintelligence Executive at $100-250 billion USD in lost sales[58]. Their report recommended that "substantial resources" be allocated to the security of systems connected to the Internet, especially given the "considerable effort that is under way in the cyber attack and exploitation arenas."[58] As more government services, industries and individuals rely on these computer networks, and as electronic commerce continues to grow, the need for comprehensive, proactive security will increase.

A taxonomy of attackers has been developed[73], where their skill levels was defined as being between very low (newbie/tool kits) to very sophisticated (professional criminals and cyber-terrorists). Large computer networks, such as those maintained by the military and by large corporations, are subject to intrusion attempts by all levels of attackers. It is assumed that these targets have invested in some measures to protect themselves, and therefore are not likely to encounter breaches by those with lower technical skill levels. However, given that espionage is a concern, large corporations and the military will likely also need to defend against sophisticated attackers, such as those classified as professional criminals and cyber-terrorists. Due to the high level of technical ability these attackers possess, being able to recognize and defend against them is important for any organization.

Jonsson and Olovsson[37, 38] identified three phases in the security intrusion process — the learning phase, the standard attack phase and the innovative attack phase. In the learning phase, a potential attacker is learning about computer security through websites, books, and various other resources. This is followed by the standard attack phase, where attackers use scripts and exploits from other sources in an attempt to breach the security of a system. In the final phase, an attacker has exhausted the standard methods for breaching computer security and must now develop new methods for circumventing the standard security mechanisms.

This model represents the phases in learning how to attack a system, rather than how such attacks are performed. Skoudis[77] presents a model of an attack, using five phases:

reconnaissance, scanning, gaining access, maintaining access, and covering tracks. Reconnaissance and scanning both refer to information gathering efforts aimed at learning about the target system. In this thesis, we combine these two phases into the information learning phase, expand the gaining access phase to include both the standard attack phase and the innovative attack phase as defined by Jonsson and Olovsson, and combine the maintaining access phase with the covering tracks phase into the post-access phase.

Any military attack begins with reconnaissance missions in order to determine the resources of the enemy, and is described as "a precursor to maneuver and fire" in the United States Army's Field Manual 100-5 (cited from [51]), and in fact, the success of an attack has a high correlation with the thoroughness of the reconnaissance[51, 70]. Similarly, before an adversary makes an attack against a target computer or network, they will usually perform some form of information gathering in an effort to determine potential vulnerabilities on a target computer or network.[50] As stated by the HoneyNet Project[66, p. 77], "Most attacks involve some type of information gathering before the attack is launched." While this particular project concentrated primarily on those attackers with low skill levels ("script kiddies" according to the project, who fit in Rogers' taxonomy as newbie/tool kits and cyberpunks[73]), it is believed that information gathering techniques are also used by more sophisticated attackers. In fact, according to the Annual Report to Congress on Foreign Economic Collection and Industrial Espionage 2001[58], the "majority of Internet endeavors are foreign probes searching for potential weaknesses in systems for exploitation."

This information gathering can take several forms, not all of which are technical. For example, adversaries will often use social engineering approaches, where they call a secretary or computer operator pretending to be someone high in the company or a technical support person, requiring particular information immediately[60]. Other information gathering techniques include "dumpster diving" or "trashing", where an attacker searches through a target's physical garbage for items such as computer listings and floppy disks.

While the social information gathering procedures are interesting, in this thesis we concentrate on a particular technical information gathering technique, that of port scans. The other technical techniques, such as performing web searches on a company, browsing their web site for information, and searching DNS information, can not be detected as either no packets actually enter the target's network, or those that do are legitimately browsing the target web site. However, a port scan consists of packets that do traverse the target's network, and can be logged as unusual activity. A port scan occurs when the adversary sends one or more network packets, which may be specially formed in order to avoid detection, to particular ports on particular IP addresses in an effort to map the target network or computer.

Legitimate monitoring is often performed on the target networks and computers by the network administrator in an effort to detect intrusions. However, network monitoring activities need to be more sophisticated in order to detect attackers. Unfortunately, as our monitoring techniques become more sophisticated, so will those techniques used by an attacker in order to avoid being caught by any monitoring practices.

For example, one method employed by attackers to elude monitoring and bypass firewalls is to employ malformed packets and incomplete connections. However, this alone is insufficient to prevent detection, as some monitoring software is configured to detect these forms of scans and as some networking logging software will still detect and log the traffic. To further avoid detection, an adversary can ensure that their scanning is fewer than X packets within time frame Y, as monitoring software often has some threshold which must be exceeded before an alert is generated. By inserting delays between each scan, attackers can avoid detection by such simple algorithms. The software may log that one port was scanned (due to the logging of malformed packets and incomplete connections), but will not associate it as being part of a larger port scan and so not generate an alarm.

While it is possible to evade detection by inserting time delays between port scans, this slows down the information gathering process considerably, particularly on large networks. In response, it is suspected that the more technically sophisticated adversaries have developed more sophisticated methods to elude detection by monitoring software. One such method is the use of a distributed port scan. In this case, the adversary may in fact employ more than X scans in Y time frame, however they will be from different sources. If an attacker employs multiple source computers from which to scan a network, the software can not recognize that the different sources are collaborating as part of one large scan. While distributed port scanning does not yet appear to be common practice among those attackers who typically gain media attention, at least three freeware tools — dps[26], dscan[32] and Rivat[84] — are in development (and have been released in beta versions) that will provide this utility.

Using Brinkley and Schell's classification scheme[3], port scans are direct probes, so are not in and of themselves dangerous. As we are interested in more professional attackers, it can be assumed that they have the understanding of the tools at their disposal to know how to best hide their tracks, as well as the capability to build their own customized tools, which makes detection of their actions by a network administrator more difficult. Additionally, they are motivated to remain undetected. To quote Brinkley and Schell [3, p. 38]:

> "We must assume that a serious penetration attempt will be indirect in nature, will not require direct physical access by the penetrator and/or operator to the penetrated target, and will not advertise its presence or cause easily observable disturbances to the system's behaviour. In short, a serious penetration attempt will be quite unlike those of amateur penetrators (hackers) occasionally receiving media publicity...."

It is suspected that the more skilled an attacker is technically, the more likely he is to attempt to hide his information gathering activities. The more skilled an attacker, the more dangerous the threat, as he is more likely to be able to penetrate a target system without detection. If he is likely to hide his initial attempts to gather information, he is likely to employ port scanning in the manners described above, including distributed port scanning where the adversary employs more than one host to gather the desired information.

1.3 Research Goal

Hypothesis: We contend that distributed port scans can be represented by a formalized model. This model will be created through the identification and quantification of the properties of such scans. Verification of the model will be performed against real data through the use of an advanced prototype.

The focus of my research is to develop a model that represents distributed port scans. It is anticipated that the model will evolve over time. As new forms of distributed port scans are discovered, the model will need to be modified to represent these previously unseen forms, if the model is not sufficiently general to already represent them.

The initial approach to creating a model will be an analysis of the tools that currently perform distributed port scanning. Three such tools have been identified — Rivat, dps and dscan — and are freely available on the Internet. The code for these tools will be analysed to determine precisely how the software performs a distributed port scan, such as the options available for configuring TCP flags, the source ports used, the destination ports scanned, and how the targets are distributed amongst the scanning agents. The programs will also be run on a test network, and the resulting log files analyzed for this same information. Based on these two analyses, the characteristics of the three distributed port scanners will be identified. Any similarities in the characteristics between the port scanners will form the initial basis of a model that represents distributed port scans.

The Computer Emergency Response Team (CERT) located in the Software Engineering Institute at Carnegie Mellon University has a real dataset collected from a very large network (more than ten thousand computers) that was in active use. Summarized, uni-directional data is available for a period of approximately two years, while more complete, bi-directional data is available for a period of a few months. This data has already been analyzed for port scans by CERT, resulting in a separate database consisting solely of port scans. This data set is large enough that it can be partitioned into two sets, one to be used for data exploration and one to be used for model validation.

Data exploration of real data will be used to complement the information gathered through the analysis of current tools. Several data exploration methods have been described in the machine learning and data mining communities. The initial approach will be the use of a self-organizing map, a method known to be valuable for data exploration[40]. Other data exploration methods may also subsequently be employed. The results obtained from the data exploration phase will be used in the creation of a model of distributed port scans.

Following the development of a model, a validation phase is required. Tools for detecting distributed port scans will be implemented using the model as a basis. These tools will be deployed against real data, such as the portion of the CERT data set not previously used in the data exploration phase. The results from this deployment will be used to validate that the model performs as expected.

One issue to be addressed during the data validation stage is that, if some portion of the real data set is to be used, the existence and location of distributed port scans amongst the data is not known. Thus a domain expert will validate each set of port scans that are flagged as a distributed port scan by the prototype. Additionally, the number of false negatives can not be calculated without knowing apriori the location of distributed port scans.

Solving this problem will result in advancing the state of the art in computer security by providing methods for the recognition of distributed port scans, moving beyond the current state where only single-source scans can be recognized. It can provide a basis for the development of tools for network administrators to use on large networks, such as national and international corporations and government organizations, to detect distributed port scanning against their networks. It is also anticipated that lesser contributions may be made in the areas of clustering or set covering. It is currently assumed that these techniques will provide a basis for the tools developed to detect distributed port scans, but that modifications to these techniques will be required, possibly resulting in a contribution to that literature.

As stated by McCarthy[51] and alluded to by Jonsson and Olovsson[37, 38], the chance of success of an attack is related to the quality of the learning or reconnaissance phase of the attack. It is anticipated that sophisticated attackers will employ distributed port scans to ensure a high-quality learning phase. Therefore the detection of port scans, and in particular stealthy or distributed port scans, allows for the early detection of and reaction to potential intruders. Cohen[8] reports from simulations of computer attacks and defences that a quick detection and response to an attack is a more successful strategy for a network administrator than having more more defences but with a slower response time. It is also possible that the sophistication level of the scan will enable organizations to determine the level of the threat, and hence the amount of resources to put into ensuring a swift reaction to any future penetration attempts.

Chapter 2

Literature Review

This chapter will provide background on both the people involved in computer attacks and the tools they use, providing an introduction to some of the taxonomies that have been developed. When looking at people, we focus on the motivation for their attacks to provide some context for this thesis, however it is not the focus of the thesis. Following the motivation, the tools employed by these attackers are described, as well as methods commonly employed by network administrators for detecting attacks. The focus will then shift to a discussion of port scans, and distributed port scans in particular. Distributed computing, and in particular distributed denial of service attacks, will be reviewed to provide some context for the discussion of distributed port scans. Finally, a review of some common clustering techniques will be presented as these techniques may be employed in the development of tools to recognize a distributed port scan.

2.1 Taxonomies of Attacks and Attackers

In the early days of computers, people who were exceptionally proficient at solving computer problems using elegant techniques became known as hackers. Being labeled a hacker was a compliment to a person's ability with computers. Additionally, hackers of this time largely believed that software should be free, resulting later in the formation of the Free Software Foundation[71]. At this time, operating systems were being developed, however security was not considered as being very important. Computers were not interconnected, so physical security tended to be sufficient. With the development of ARPANet, only a few computers were connected, and these were research computers at research institutions, so again security was not considered an issue.

In the 1980s the term hacker began to mutate into describing a person whose interests involved breaking into computer systems. Concurrent with this was a shift in the availability of computing power, as personal computers came onto the market, and communications via modems began to proliferate. This shift allowed external attacks against computer systems to be performed more easily. External attacks were still considered infrequent enough at this point that creating and deploying appropriate security mechanisms were still not adequately addressed.

Today, the term hacker is used almost exclusively in the negative sense of its meaning, referring to a person who attacks computer systems. Personal computers have become widespread. The World Wide Web (WWW) was developed, causing an explosion in Internet access. This has resulted in the use of the WWW for financial gain through electronic commerce. However, security as it is currently implemented on commercial systems and home computers can not meet the expectations that external attackers will not be able to penetrate systems. As a result, there are now numerous attempts at security breaches, and numerous successful security breaches. Due to the proliferation of computer systems and accessibility, along with the uses for which systems are now employed, security is required for two main reasons: for privacy of personal information and for protection of economic viability.

At the same time that the meaning of the word hacker was evolving, so were the motivations for "hacking". As personal computers started to enter the market, people who were fascinated by this technology learned how to obtain unauthorized access to other systems, such as mainframes. Computer attacks of this form were seen as a game[45]. However, as the usage of computers evolved, so did the motivations for attacking them. Today we see that, while some people still view this as a form of game or entertainment, the more sophisticated computer attackers are interested in financial profit or advancing their political agendas[34, 60, 73]. This section reviews the literature on computer attackers and motivations, and shows how it has changed over time.

Very few taxonomies of computer attackers have been published. The earliest taxonomy that can be found is by Landreth in 1985[45]. Landreth, a convicted hacker, gave a description of the different types of hackers, consisting of novice, student, tourist, crasher and thief, where his divisions are based loosely on the motivation of the hackers. The novice is someone who is new to hacking. The student is a hacker who is more knowledgeable than the novice, and who breaks into a system in order to learn about not only the system, but from any information housed in any of the files on the system. Landreth describes the ethics of this group, stating that they are against the deletion of files or any malicious activity performed against the system. The tourist category consisted of hackers who were interested in the challenge of breaking into a system, but who would perform no further actions once a system had been compromised. The crasher category consisted of hackers who had malicious intents, and would perform acts such as deleting files. The thief category contained those hackers who compromised systems to obtain information that could be sold to competing companies. Landreth alleges that the thief category consists primarily of people who are internal to the company from which they are stealing. It should be noted that this taxonomy was developed in 1985, and does not reflect the current body of hackers, as shown by more recent taxonomies.

By 1995, national interests were recognized as a motivation in attacking computer systems. For example, Hundley and Anderson, in [34], provided a taxonomy that consisted of five categories, based on motivation. Hackers, zealots and disgruntled insiders were lumped together as their motivation was personal gain. Criminals were a separate category, whose

goal was financial gain. Advancing a particular cause was another motivation, that of terrorists and other such organisations. Espionage was a further motivation, which was divided into national (for warfare) and industrial (for disadvantaging a competitor). Thus nations and commercial organisations, respectively, were listed as two separate attacker categories. Later, in a taxonomy published in 1998 by Parker[60], national interests ("extreme advocates") and financial gain ("career criminals") again figured prominently in the motivations of computer attackers.

Rogers argues in [74] that in order to empirically study the psychology of hackers, a generally-accepted taxonomy of hackers needs to be developed. The taxonomies previously published did not provide categories that allowed psychological profiles to be assigned. To address this issue, Rogers developed a new taxonomy[73]. Here he defines the categories of hackers as newbie/tool kit, cyber-punks, internals, coders, old guard hackers, professional criminals, and cyber-terrorists, where these levels are listed in increasing order of technical sophistication and ability. **Newbie/tool kit** refers to people with very little actual programming or technical knowledge, who download toolkits written by other people and use these to break into other systems. **Cyber-punks** have a bit more computer knowledge, and sometimes write their own limited software. They often engage in malicious activity, and are the group of hackers on which the media tends to focus. Their psychological profile indicates that this group consists primarily of Caucasians between 12 and 30 years of age, who are described as having limited social skills. They often perform poorly in school and are classified as loners.

Internals refer to criminals who perpetrate their crimes against the company for which they work, exploiting their privileges as employees or contractors. This group is technically literate and may even hold IT-related positions. They already have knowledge of the internal network, and can exploit their legitimate access to that network for illegitimate purposes. This group can also work for or with external adversaries. According to Power in [62], the majority of computer crime falls under this category. However, according to the surveys performed by the Computer Security Institute (CSI) in conjunction with the FBI, the majority of computer attacks are now perpetrated through the external Internet connection, and not through internal systems. In 2002, the CSI/FBI survey found that 74% of 481 respondents cited their external connection to the Internet as the most frequent point of attack, versus internal systems and remote dial-in. This survey continued to highlight a trend noticed over the previous six surveys (conducted annually) that showed an increase in external attacks and a decrease in internal attacks[64]. However, in 2000, 81% of respondents surveyed by the CSI/FBI still felt that disgruntled employees were a likely source of attack, whereas 77% were concerned with independent hackers, 44% with US competitors, 26% with foreign corporations and 21% with foreign government[62].

Coders were not defined in Rogers' paper, but likely refers to that group of computer attackers who write code that exploits system vulnerabilities. This code can have very advanced properties, and is often made publicly available.

Old guard hackers refers to a group who relate to what is considered to be the old-style hacker ethic, believing that software should be free to all. They are more interested in hacking for the intellectual challenge, and not inclined towards malicious acts. This group has a social conscience, believing in the freedom of information, and that corporations and governments should be scrutinized to ensure that they are not violating the rights of any citizens.

The **professional criminals** and **cyber-terrorists** groups are considered to be the most technically proficient and the most dangerous, and often have access to state-of-the-art equipment. Individuals in these two groups can be hired for acts such as corporate espionage. According to Dr. Eugene Schultz (as cited in [63]), "But what I am saying is that it is important to avoid underestimating the external threat. It is not only growing disproportionately, but it is being fueled increasingly by organized crime and motives related to espionage."

Marc Rogers has started to investigate the psychology of hackers in [74], where he argues that Kohlberg's moral development theory, differential reinforcement theory, and social learning theory can all be used in explaining hacker psychology, such as why people start hacking and why this behaviour persists. Kohlberg's moral development theory provide levels of moral development, and Rogers argues that in some cases hackers have attained the hedonism stage, which is one of the pre-morality stages. This explains hackers who attack systems for personal gain only. Differential reinforcement theory argues that a behaviour is learned, and then maintained through reinforcement. This fits the hacker who learns how to hack through various hacker groups, and then has this activity reinforced through acceptance into the hacker subculture. Social learning theory is very similar to differential reinforcement theory, and states that the initial behaviour is acquired through observation, and then reinforced from both external and internal sources.

Some work has also been pursued by Rogers[72] regarding the justifications used by computer criminals for the activity. He presents a theory on moral disengagement, where the criminal justifies his behavior through a variety of mechanisms, such as saying it was a group decision (and therefore removing personal responsibility), or by viewing themselves as victims who were provoked into acting, or by blaming the system administrators at a hacked site stating that they were lax in security and therefore deserved to be attacked. Some hackers even justify their acts as required to demonstrate to victim sites where their security needs to be improved.

Eugene Spafford has also given consideration to the motivations of computer hackers for their activities. In [78] he identifies (and refutes) commonly-used justifications employed by hackers. These justifications include what he describes as the hacker ethic, which is the belief that all information should be free. This relates to Rogers' descriptions of old guard hackers[73]. There is also the student hacker, who uses breaking into computer systems as a form of personal education. This motivation has also been identified by Landreth[45]. Arguments that hackers break into computer systems to show system administrators and/or vendors where the weaknesses are in their systems, thus improving their security, have also

been voiced. Related to this is the argument that hackers are acting as social protectors by breaking into systems to ensure that personal data is not being kept or misused. Finally, some hackers argue that they only compromise systems in order to use their idle cycles, therefore affecting no one.

The tools employed to attack a computer system can vary based on both the skill level and motivation of the attacker. For example, Moyer, as cited in [62], divides the tools used by attackers into categories based on the type of attacker — sport intruder, competitive intelligence or foreign intelligence. Moyer claims that the sport intruder uses probes, exploits and deceptions. Probes are tools that allow an attacker to find out more about your network (such as port scans). Exploits are methods for gaining unauthorized access to a computer network by using weaknesses in the code for the services offered, such as buffer overflows against printing and web services. Deceptions are those tools that allow an attacker to not be noticed once a successful intrusion has occurred, and includes tools such as rootkit (which replaces standard system binaries with hacked versions to help prevent detection). By comparison, competitive intelligence focuses more on legal arenas in which to gain information. Tools employed by this category of attacker include public records, patent records, search engines, and the target company's web site. Foreign intelligence takes competitive intelligence even further, employing information collected from satellite imagery, for example, as well as placing a wire tap on phone or fax lines, breaking into hotel rooms and deploying listening devices.

Brinkley and Schell, by comparison, have developed a taxonomy based on vulnerability versus threat. They distinguish between threats to computers, which includes theft or disruption of computational resources, and threats to information, such as unauthorized information disclosure or modification. They contend that threats to information are fundamentally different than threats to computers, and are more difficult to protect against as information requires protection from both legitimate users and non-users. Based on this, they concentrate on threats to information, defining six subcategories:[3]

1. human error,
2. user abuse of authority,
3. direct probing,
4. probing with malicious software,
5. direct penetration, and
6. subversion of security mechanism.

Human error refers to the ability of attackers to take advantage of the mistakes of computer users, such as a user setting the wrong permissions on their files, thus allowing others to read them. In this case, the attacker needs to wait for someone else to make a mistake to gain access to information that they should otherwise not be able to view.

User abuse of authority, by contrast, refers to users using their legitimate access to particular systems to their own advantage. An example of this would be someone who works for a bank performing a monetary transaction from one account to another that was not actually authorized by the owner of the first account, and where the second account belongs to the employee. The employee uses their legitimate access in an abusive manner. Such a user is equivalent to an internal in Rogers' classification scheme.

Direct probing refers to attackers who probe systems, but do not actually perform any malicious actions. An attacker may gain unintended access through a flaw in a system, and from there continue to look at files in the system, potentially gaining greater access to the system while doing so. Port scans are one technique for information gathering that fall under this category.

Probing with malicious software is the next iteration to direct probing, where instead of a manual method for examining a system, the probing is automated. Brinkley and Schell classify viruses, worms, trojan horses, and logic/time bombs in this category.[3] A virus attaches itself to executable programs and, when executed, spreads by replicating itself, or attaching its code to other executable programs. By comparison, a worm does not require attaching itself to legitimate code, but rather spreads through the use of flaws in a computer system, or through legitimate means such as e-mail. A trojan horse is a legitimate program that a user might execute, that also performs some other legitimate action that was unintended by the user. For example, the user might be using a compiler that has a trojan horse set that changes the permissions on all of the user's files. Logic bombs and time bombs are programs that are set to execute given a particular circumstance. In the case of a time bomb, the program will execute on a particular date, while a logic bomb executes given a particular set of actions or conditions.

Direct penetration is the gaining of unauthorized access to a computer system by bypassing the security mechanisms in place, whereas the previous four security breaches do not require this unauthorized access. An example of direct penetration is the gaining of superuser privileges through the use of buffer overflow vulnerabilities in system functions.

Subversion of security mechanisms is the most insidious of the six categories of security breaches. In this case, somewhere during the design, building, installation or use of a system, a trap-door is inserted in the system that can be used at a later date. No one other than the perpetrator is aware of the existence of the trap-door, which may or may not be later employed. The trap-door allows the subversion of all security mechanisms as it is part of the system itself, allowing the perpetrator superuser access, and thus allowing the access required to remove any traces of this activity from the audit logs.

Lindqvist and Jonsson have developed a taxonomy of intrusion techniques[49]. They categorize an intrusion from the system administrators perspective, and break it down along two dimensions: intrusion techniques and intrusion results. The results are based on the first, direct result of the intrusion, and do not take into account any later misuses the

attacker may perform given the new privileges gained through the intrusion. Indeed, in another paper, Jonsson states that "there exists no established method to rank breaches with respect to their seriousness, and it not even clear that it is possible to define such a measure unambiguously."[38]

Intrusion techniques are broken down coarsely into the following areas: password attacks, spoofing privileged programs, utilizing weak authentication, exploiting inadvertent write permission, resource exhaustion, manual browsing and automated searching. Intrusion results are broken down coarsely into disclosure of confidential information, service to unauthorized entities, denial of service (selective, unselective, and transmitted), and erroneous output (selective, unselective, and transmitted). These classifications are based on experimental results, as described in Section 2.2, from a paper by Jonsson and Olovsson[38]. Due to the limited nature of this experiment, it is likely that some classifications have been missed. One example that does not fall into any of the techniques given is that of social engineering, where a user is tricked into providing information such as a password. The techniques listed in [49], however, concentrate on technical breaches.

2.2 Anatomy of Attacks

As mentioned in Section 1.2, Jonsson and Olovsson have developed a model of the security intrusion process consisting of three phases — the learning phase, the standard attack phase and the innovative attack phase[37, 38]. This model is based on the results obtained from a controlled experiment, where 24 students were divided into 12 groups and given four weeks to find all possible security breaches in a school computer lab that consisted of 24 diskless Sun workstations and a file server.

Unfortunately, this experiment is very limited and the results can not be generally applied. The students appear to most closely represent the skill level of the cyber-punk group (using Rogers' classification[73]) and the motivation of the old guard hackers group. However, it is expected that those groups with a higher skill level will spend increasingly less time in the learning phase. The model developed represents the phases for learning to attack a system, rather than the phases of an attack. Thus the model was based on the behaviour of these students and not the technologies employed. For example, the learning phase refers to the attackers learning how to breach computer security systems, as opposed to doing reconnaissance on the systems themselves.

In contrast, Skoudis defines five phases to the security intrusion process, based on the technical steps employed — reconnaissance, scanning, gaining access, maintaining access, and covering tracks and hiding[77]. He defines reconnaissance as using all those techniques to acquire information that can not be detected by standard auditing (e.g. such as searching an organization's web site and searching the who-is databases), whereas scanning concentrates on the actual mapping of a target network (such as through port scanning to find available services and war dialing, that is dialing all phone numbers in a target organization, to find modems). Gaining access is divided into three approaches — attacks against the operating

system, attacks against applications, and attacks against the network. The purpose of the gaining access phase is to find an exploit that will allow unauthorized access to the systems and networks of interest, such as through stack buffer overflows or sniffing the network for passwords. Once access has been gained, it will need to be maintained. Thus the fourth phase is maintaining access, and refers to techniques such as installing trojans or backdoors so that the attacker can later access the system. Finally, once access has been gained and a program installed that guarantees access will still be granted at a later date, the attacker will enter the final phase, that of hiding the intrusion. This will involve activities such as altering the log files on the invaded system to remove any indication of the intrusion.

Some papers mention classification schemes for determining the amount of damage caused by an attack. For example, Hundley and Anderson present four types of consequences in [34]: minor annoyance or inconvenience, limited misfortune, major or widespread loss, and major disaster. The majority of attacks can be classified in the first two categories. The first category requires no significant effort to recover from the attack and causes no damage or loss of information. The second category requires that some effort be expended to repair the damage caused by the intrusion, however limited damage was caused and the repair effort does not require unreasonable amounts of time given the resources of the organisation. A number of newbie/toolkit attacks fall in this category. The third category, that of major or widespread loss, refers to events that have significant financial impact on the organisation or significant impact on society. Such events have occurred in the past, such as in January 1990, when a large portion of the nation-wide long distance telephone network in the United States was unavailable for approximately nine hours due to a software error (cited from [34]). The last category, that of major disaster, is a damage or loss that is irrecoverable or nearly so.

In addition to research papers, companies, such as Symantec and Microsoft, have severity ratings. In the case of Symantec, the severity ratings refer to the impact of a virus, based on the distribution of the virus or number of machines infected, the damage caused by the virus, and how the attack is distributed (e.g. virus versus worm versus Trojan)[27]. Microsoft, in contrast, rates the severity of vulnerabilities discovered in its products. It rates the severity based on the results of exploiting the vulnerability (such as root access versus disclosure of web script code) and the type of system the vulnerability affects (internet server versus internal servers versus desktops).

However, there is no generally agreed-upon severity classification scheme. Companies develop schemes specific to their products and focus. Classifications presented in research papers are not the focus of the paper, but are presented as a side issue.

While there are papers that present taxonomies of attacks, these papers do not relate the attacks to the amount of damage typically caused. Also, there is very little research showing the relationship between the information gathering techniques employed, the attack methods employed and the damage caused.

2.3 Reconnaissance Techniques

There are many reconnaissance techniques that can be employed to gather information about a target network or company, not all of which are technical in nature. In the technical realm are networking mapping, war dialing and vulnerability scanning. Merging technical and non-technical is the search through publicly-available information, while the non-technical approaches include "dumpster diving" and "social engineering"[77].

Technical reconnaissance takes the form of mapping the target network, employing tools such as traceroute. This tool details the path taken from the source machine to the target machine, outlining each "hop" or machine between the source and destination. By tracing to various target machines, the source can determine information such as the location of border routers and firewalls, thus starting a map of the target network. War dialing can also be deployed, which involves using a modem and calling every phone number on an internal network during the late night hours searching for modems. Employees may have a modem connected to their workstation to allow them to work from home, however these modem connections are often not very secure. Some attackers may employ vulnerability scanners, such as SAINT[9] or Nessus[28], which provide automated scanning of a target network, reporting on services offered that are known to be vulnerable.

Combining technical and non-technical approaches involves searching for publicly-available information about the target. This can involve searching through a target's web site for information regarding their network and facilities and through on-line databases such as the American Registry for Internet Numbers. These databases contain domain registration information, such as the name of the target's mail server and technical contacts.

Dumpster diving is the term used to refer to searching through the physical garbage of a target company[77]. This technique can result in information such as sticky notes containing passwords, discarded printouts containing network information, and even discarded floppy disks which may contain sensitive information. This technique has been known to be employed by attackers intent on industrial or economic espionage[6].

Social engineering consists of infiltrating a target by extracting information from its employees. In some cases, an attacker will call an employee pretending to be another employee needing assistance. For example, the attacker could call a help desk needing their password reset. In other cases, the attacker can pretend to be part of the help desk, calling a secretary about fixing a reported problem with his machine, and requiring his username and password. The attacker can also pretend to be someone with a higher status in the company. In this instance, the attacker can intimidate an employee into divulging sensitive information about the network[77]. If the attacker has more time, he can even go so far as to befriend employees of the target company. For example, he could meet an employee at a conference, then keep in touch casually, perhaps meeting again at another conference. Through these casual meetings, the attacker gains the trust of the employee, and can then guide conversation so that the employee reveals sensitive information[62].

2.4 Port Scans

Using Brinkley's classification scheme[3], port scanning would be considered direct probing. Port scans allow an attacker to work from a remote location to determine if a computer is located at a particular address, what services it is running, and even what operating system it is likely using. A port scan consists of a connection attempt to each port on a target computer to determine which ports are open, and thus what services are running.

Given an intranet, computers can be located in this address space through a ping sweep, which means pinging every address in the address space to determine if a computer is alive at the other end. A ping request consists of an Internet Control Message Protocol (ICMP) packet encapsulated inside an Internet Protocol (IP) packet. As ICMP packets are handled at the network layer by the operating system kernel, no service ports on the destination machine are accessed and so the echo request/echo response is not logged by any application. However, a ping sweep is only able to determine that there is a response from a particular address. It can not determine what services are available on the machine[19].

Given a target computer or network, a port scan can be used to determine the ports that are open, and hence the services that are likely active. Port scans can be classified by the target of the scan, and by the Transport Control Protocol (TCP) flags set. There are four types of scans, which are defined by the target information to be gathered[79]:

- vertical scans,
- horizontal scans,
- strobe scans, and
- block scans.

A vertical scan refers to scanning all of the ports on a single system to determine what services that one system is running. In contrast, a horizontal scan is one where a single port is checked across a large portion of the systems on a network. A strobe scan is similar to a horizontal scan, however it checks a few ports rather than a single port. This is named after the software Strobe, which probed commonly-used ports such as 21, 22, 23, and 80. A block scan combines vertical and horizontal scans into one large scan that checks all ports on all systems.

Port scans can also be classified by the TCP flag settings used in the scan. That is, any of the port scans listed above (vertical, horizontal, strobe and block) can be defined instead based on the flag settings. The more common of these scans are:[19]

- TCP connect() scanning,
- TCP SYN (half open) scanning,
- TCP FIN (stealth) scanning,

- SYN/FIN scanning using IP fragments,
- UDP raw ICMP port unreachable scanning,
- XMas scans and Null scans.

Each of these types of scans have been developed in an effort to prevent the target from detecting the scan. For more detail, see Appendix A.

The results returned by these scans can be used to identify the remote operating system. Each operating system has implemented the TCP/IP stack slightly differently, and these differences are evident in the responses to various packets. For example, some operating systems have a pattern in the initial sequence numbers chosen, such as IRIX, which increases initial sequence numbers in increments of 800. By sending enough carefully crafted packets, the remote operating system, including in many cases the version number, can be determined[20]. This process is known as TCP fingerprinting.

The HoneyNet Project[66] has noticed a change in the information gathering stage performed. It used to be that an attack was preceded by port scans to determine if the service was available, as well as the operating system, and sometimes the version of the service in use. However, lately the trend has been to not perform port scans, but to deploy an attack directly against a system to see if it works. It should be noted that the HoneyNet Project focuses on the newbie/toolkit group, using Rogers[73] classification, and so does not necessarily apply to the group in which I am interested.

2.4.1 Stealth Port Scans

An attacker can model the scan performed based on a balance between the information he wishes to obtain and the desire to not be discovered. For example, if the attacker is interested in a complete intranet, he will likely start with a ping sweep to determine the addresses of machines currently on that network. This would be followed by a port scan. If the attacker has an exploit for a particular service, a horizontal scan will be used to determine what machines offer that service. If he is interested in attacking a particular machine, a vertical scan may be performed to determine the services offered by that machine[79]. It is also likely that TCP fingerprinting will be performed concurrently to determine the remote operating system.

However, attempts at information gathering through port scans is often tempered by the desire to not be discovered. One method for avoiding detection is to use modified TCP packets as described in Appendix A. However, as scans of these types are now well-known in the security community, they may still be logged by the destination through tools such as Snort. To prevent the scan from being detected, other methods commonly employed to hide the activity include:[35, 29]

- randomize order in which IPs are scanned,

- insert time delay between scans and/or randomize the length of the time delay between scans,
- randomize TCP packet field values (e.g. sequence numbers, acknowledgment numbers, source ports, etc.),
- hide real scan within scans from spoofed IPs, and
- distribute port scan amongst multiple sources.

Intrusion detection systems often recognize port scans by the patterns they generate. For example, by changing the order in which IPs are scanned, so that they are not scanned in increasing order of IP, some systems may not detect the scan because it is watching for the increasing IP order. Similarly, by inserting a delay between scans, a system may not detect the scan as it expects to see some minimum number of scans within a certain time frame. The length of this delay can be randomized to defeat any system that checks for a consistent delay between connection attempts from a source. Another method for evading detection is to hide the real scan within a number of scans, all sent with spoofed IPs (that is, using source IP numbers of machines other than the true source). Assuming that the spoofed IPs are still valid IP addresses, the target system will know that it was scanned, but be unable to determine the actual source of the scan.

2.4.2 Distributed Port Scans

A relatively new method for evading detection is to use distributed port scans. A distributed port scan, modeled after distributed denial of service attacks, employs multiple sources, each of which performs a portion of the scan. The results are then collected at a single system. By employing distributed port scans, the target system believes that multiple sources were performing small scans, if these scans are detected at all. The large, co-operative nature of the scan is not known.

One of the first occurrences of distributed port scans in the literature occurs in September 1998, in a paper released by the Naval Surface Warfare Center, USA[57]. In this paper, examples of co-ordinated scans of target systems are presented. It is asserted that there are three purposes to co-ordinating scans:

1. "stealth": By probing a system from multiple sources, the target is less likely to detect the scans or respond to them.
2. "firepower": The attacker(s) will be able to collect a static amount of information in a shorter amount of time.
3. "more data": By co-ordinating information gathering efforts, more information can be gained by an attacking person or group given a static amount of time.

The paper goes on to describe five examples of co-ordinated activity detected by the author, however the reasons for believing that this activity is indeed co-ordinated are not well articulated. The main criteria appears to be similarities in the signatures (e.g. same ack numbers, same type of scans or malformed packets) and that the scans occur during roughly the same time frame. The similarity in the signatures could indicate that the same software with the same parameters was being used to generate the scan packets. This would be likely if it was one person generating the scans, versus multiple people, who would be more likely to use at least different parameters, if not different software. Also, given that two of the reasons for employing a distributed port scan is firepower and for more data, the scans from multiple sources will likely take place within the same time frame, otherwise these two advantages no longer hold true.

One of the first occurrences of distributed port scans in the common hacker literature happens one year later, in September 1999. An article in Phrack Magazine described the reasons for performing distributed information gathering[35], citing the 1998 paper from the Naval Surface Warfare Center. At that time, there were no known programs available to perform this function. The purpose of a distributed port scan, as given by this article, is to perform information gathering activities while avoiding detection. Other advantages to distributed port scanning mentioned included the acquisition of multiple "points-of-view" about a network topology, and the ability to circumvent the countermeasures employed by some network intrusion detection systems (e.g. some systems block network access to IPs that appear to be hostile).

According to hybrid[35], "It is likely that detection of distributed information gathering will be available only as a retrospective function .... Logs from multiple N-IDS agents must be centralized and cross-correlated before distributed information gathering attacks can be detected." This article goes on to state "In a large enterprise (for example a military, government, or large corporation installation) this process of event consolidation must be considered a non-trivial task."

2.4.3 Publicly-Available Tools

The most widely used tool in the hacker community for performing port scans is nmap[29]. However, nmap does not perform distributed port scans. Other tools need to be employed in this instance. Several tools have been developed and are freely available on the Internet. One of the earliest and perhaps most widely distributed of these is called dps and was published on Phrack in May, 2000[76]. Also available is Rivat (2000), dscan (2001), another tool that was also named dps (2000), and a php-based distributed port scanner (2002).

Rivat, written by X-tremist, is a distributed port scanning tool available as freeware. It is written in perl, and consists of one script for the server, which is any machine that performs a port scan, and one for the client, which is the central co-ordinator that contacts the servers to perform scans. The server is distributed across multiple hosts, and listens on port 2345 by default (although this is easily changed). A client can then contact each server and ask

```
# perl client.pl hosts log

---129.173.66.235---
SCANNING STARTED for rex.cs.dal.ca
Vulnerabilities:
Other Information:
FINGERD - [Present]
FTPD - [220-]
HTTPD - [Apache/2.0.16 (Unix)]
Possible ROOTSHELL - [1524]
Possible ROOTSHELL - [31337]
SCANNING FINISHED for rex.cs.dal.ca

---129.173.66.237---
SCANNING STARTED for locutus.cs.dal.ca
Vulnerabilities:
Other Information:
FINGERD - [Present]
FTPD - [220 locutus FTP server ready.]
SSHD - [SSH-1.99-OpenSSH_3.1p1]
HTTPD - [Apache/1.3.12 (Unix) PHP/4.0.4]
SCANNING FINISHED for locutus.cs.dal.ca
```

Figure 2.1: Output from Rivat using two servers and one target per server

them to perform a scan. Authentication is achieved via a cleartext password embedded in each of the scripts. The distribution of scans is determined by a configuration file used by the client, although the only configuration available is the ability to specify what targets will be scanned by which servers. The software scans 14 specific ports (checking for finger, ftp, ssh, mail, http, pop and imap, pcserver, and common exploit ports such as 1524 and 31337). All port scans for a specific target come from a specific host — that is, it is not possible to distribute the ports to be scanned across multiple servers, only the hosts to be scanned. Only SYN scans are performed. Finally, the server uses a new source port for each port scanned on the target, where the source ports are used in increasing sequential order, and the target ports are always in the same order. Once a scan has been performed, the server exits and needs to be restarted if another scan is to be performed. Figure 2.1 shows the output where two servers were used to scan two targets (one target per server). Figure 2.2 shows the log file resulting from the scan for one of the targets.

The tool dps was written by Chris Bechberger and released in the year 2000. A home page can no longer be found for the author, however the software itself can still be downloaded from PacketStorm[26].

```
Jul  7 15:51:07 129.173.66.235:47044 -> 129.173.66.66:79 SYN ******S*
Jul  7 15:51:07 129.173.66.235:47045 -> 129.173.66.66:21 SYN ******S*
Jul  7 15:51:07 129.173.66.235:47046 -> 129.173.66.66:22 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47047 -> 129.173.66.66:25 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47048 -> 129.173.66.66:80 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47049 -> 129.173.66.66:109 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47050 -> 129.173.66.66:110 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47051 -> 129.173.66.66:143 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47052 -> 129.173.66.66:600 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47053 -> 129.173.66.66:1524 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47054 -> 129.173.66.66:2222 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47055 -> 129.173.66.66:31337 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47056 -> 129.173.66.66:65535 SYN ******S*
Jul  7 15:51:08 129.173.66.235:47057 -> 129.173.66.66:65534 SYN ******S*
```

Figure 2.2: Output from Rivat using two servers and one target per server

DScan, written by Anthraxx (Dobin Rutishauser)and Kolrabi (Björn Paetzel), is a distributed port scanning tool available as freeware, although it is still in a beta version[32]. This software consists of a client and multiple servers. Each server performs the intended port scans, returning their results back to a central client. The features incorporated into DScan include the ability to scan single or multiple hosts, or entire networks, all distributed amongst various servers. The scans available include full connect() scans, syn, fin and xmas scans. Communication between client and servers is authenticated via des, md5 or cleartext. The time between scans can be specified when the scan is initiated. The authors intend for their next version of the software to include more of the features that are available in nmap, such as the ability to use decoy hosts.

2.5 Intrusion Detection Systems

Intrusion detection systems analyze the activities on a computer or network system for signs of intrusion or misuse. Described in a well-known paper by Dorothy Denning in 1987[11], intrusion detection consists of two broad categories: signature detection and anomaly detection.

Many intrusion attempts produce identifiable "signatures" that indicate the activity being performed. A signature is a series of system calls or events that are particular to the attack being performed, and that do not occur during normal system usage. By searching for these signatures in the system audit logs, intrusions can be detected. The limitation of this method is the intrusion signature needs to be known. Thus new attacks for which a signature has not yet been identified will be undetected by this method. This system also requires continual updating as new signatures become available.

Anomaly detection, by comparison, determines anomalous activity by comparing all activity to some baseline of what is considered normal system behaviour. The intrusion detection system is first trained on the system's activities, using methods such as building statistical models of what is normal activity for this system. This can be further broken down into system/network models, which indicate normal activity for each system or network, and user models, which indicate normal usage for each user of the system. Once these models are put into use, the system will alert administrators of any anomalous behaviour within these models. The threshold for behaviour can be tuned in order to adjust the number of false alarms generated.

Anomaly detection suffers from two drawbacks that are characteristic of learning algorithms, and that are not present in signature-based methods. The first is that any intrusive activity or misuse that occurs during the initial training phase will be incorporated into the model as normal. The second is that the model might not incorporate legitimate behaviour that is sporadic as normal. Thus a large number of false alarms may be generated. Adjusting the threshold to reduce the number of false alarms can also have the effect of then not generating an alarm when an intrusion occurs, as the anomalous behaviour of the intrusion may no longer be anomalous enough to generate a warning.

Ye et al.[85] contend that intrusion detection is currently measured in terms of three properties:

1. the frequency of events,
2. the duration of events, and
3. the ordering of events.

Their experiments considered the frequency and ordering of events, and determined that the frequency of events (based on probabilistic measurements) can provide reasonable intrusion detection capabilities. The ordering of events, when combined with frequency, was shown to improve the detection capabilities over frequency alone. However, no experiments were performed to compare ordering to frequency to show which of these performs best when used alone. Also, this paper did not address the duration of events as a factor.

Axelsson[2] has developed a taxonomy of intrusion detection systems, based on his analysis of 20 different systems (see Table 2.1). First he breaks the systems down into signature detection and anomaly detection, which is the most widely employed distinction in intrusion detection systems. He further subdivides anomaly detection into self-learning systems and programmed systems, while signature detection only had programmed systems as a subcategory. A third category, called signature-inspired, contains a self-learning system, Ripper, which is a product developed by Lee[46].

anomaly	self-learning	non time series
		time series
	programmed	descriptive stat
		default deny
signature	programmed	state-modeling
		expert-system
		string-matching
		simple rule-based
signature-inspired	self-learning	automatic feature selection

Table 2.1: Taxonomy of Intrusion Detection Systems, adapted from Axelsson[2]

The self-learning systems (anomaly detection) refer to systems that are able to "learn" what is normal for a system and hence what should be flagged as unusual or anomalous. These systems were divided into non time series and time series based, depending upon if the calculation of normal took into account behaviour over time. Both non time series and time series were further refined in Axelsson's taxonomy[2], however this refinement was based on the intrusion detection systems surveyed, as opposed to what systems may be possible. One example here is that artificial neural networks have been employed by Hyperview, which uses a recurrent neural network to provide a time series analysis. Therefore neural networks were classified as a time series based method. However, they can also be employed in a non time series approach. Axelsson's taxonomy[2] does not take this into consideration. Due to this limitation, only the first three levels of this taxonomy are presented.

The programmed anomaly systems are divided into descriptive statistics and default deny approaches, where both approaches have their rules explicitly programmed by the user. Descriptive statistics involves the collection of statistical information regarding specified events, such as the number of failed logins, or the amount of network traffic a particular host receives. This statistical information defines what is average for a system. Once this information has been collected, any event that falls outside a particular probability, or exceeds a user-defined threshold, generates an alarm. In the default deny approach, the allowed behaviours are explicitly defined, such that no deviation from those behaviours is allowed.

In contrast to the anomaly detection systems, signature detection systems, which have no self-learning features, only programmed approaches, have been divided into four subcategories: state-modeling, expert system, string-matching, and simple rule-based systems. Signature detection systems require that signatures be explicitly defined and matched exactly. State-modeling signature detection systems model the intrusion as a series of states.

Each of these states must be present in order for the intrusion to have occurred, where the states may require a particular order (e.g. state transition models) or may be order independent (e.g. Petri nets). Simple rule-based systems (where signatures are examined to see if they follow simple rules) have been employed as a signature-based intrusion detection method, as have the more powerful expert systems approach (where signatures are compared against more complex rule structures). However, the expert systems approach, while providing greater power and flexibility, tends to be slower than the simple rule-based systems. Simple string matching has also been employed as an intrusion detection method, where the audit logs or incoming packets are examined for the occurrences of specific strings of characters, which is the least powerful and flexible.

In addition to dividing intrusion detection systems into signature-based versus anomaly-based, using their detection method to form the initial basis of a taxonomy, they can also be divided based on the systems being monitored. More specifically, intrusion detection systems can be considered as either host-based or network-based. Host-based intrusion detection employs the high-level audit logs generated on an individual host (or multiple hosts in those systems configured to co-ordinate information from multiple sources). As the granularity of the information contained in these logs is relatively coarse, low-level network-based intrusive activity will not be detected. Network intrusion detection systems address this short-coming by passively monitoring a network connection and analyzing all packets that utilize that network segment. However, this introduces new concerns, such as the inability to determine exactly how the target interprets the packets it receives. Network-based systems are also incapable of detecting attacks deployed by those internal to the system who either log into the console of a target computer, or use only those portions of the internal networks that are not monitored by any network-based intrusion detection systems.

2.5.1 Approaches to Intrusion Detection

A number of approaches to intrusion detection have been taken. These approaches vary from modeling the immune system[16], to using distributed agents[33], to using data mining techniques[47], to strict signature-based[31], to the use of agents[39]. While some experiments have been performed to provide some useful measurement of the effectiveness of the proposed systems, none of the approaches provides a complete solution to the difficult problem of detecting the occurrence of an intrusion.

In [33], Huang, Jasper and Wicks present a new approach to distributed intrusion detection systems. They argue that, rather than having a centralized system where a single node collects information from distributed nodes, an intrusion detection system can be viewed as a battlefield. In a battlefield there is a central command who gathers intelligence information from the troops that have been deployed, rather than gathering all information they collect. This intelligence information is then used to determine the enemy's attack strategy. Huang, Jasper and Wicks argue that a similar approach can be taken with intrusion detection systems, where a central system collects only intelligence information from deployed local intrusion detection systems. The local systems can process their data and alert the

central system of any suspicious behaviour. The central system can use this information to request local systems to modify their logging and audit for particular information, based on attack strategy knowledge that indicates how suspicious behaviour can be linked together towards a particular goal. The approach suggested in this paper, however, has not yet been implemented. The system design itself is vulnerable due to the requirement of a central control topology — if an attacker can take out the central control then the intrusion detection system in its entirety is effectively nullified.

The use of intelligent agents in an intrusion detection system in order to provide a response to the intrusion has been suggested by Carver et al[39]. Their proposed system (which is as yet unimplemented) uses multiple programs that each perform a function towards responding to a suspected intrusion attempt. Multiple intrusion detection systems will be deployed over multiple hosts. In response to an intrusion attempt, they will notify a central authority (master analysis agent) who will determine if this is part of an already established intrusion attempt or a new attack. This output will be fed to a tactics agent who will determine an appropriate response. Fuzzy logic will be employed to make many of the decisions required by this system. For example, a response plan will be developed while considering items such as the reliability of the reporting intrusion detection system (is it a system that is known to generate a large number of false alarms?) and the success rate of the tactics under consideration when deployed against an intrusion. The system described in this paper is very large and complex, and employs a great deal of fuzzy logic to determine appropriate responses. Given the large number of false positives intrusion detection systems tend to generate, it is likely that the tactics to prevent intrusions (which are stated as including steps as drastic as shutting down the host system) will likely be deployed largely against legitimate users. Attackers will also be able to use this system to create denial of service attacks.

Intrusion systems have also been developed based on an analogy with the human immune system[17, 16], where a computer system is given a sense of self and anything that is not-self (such as a virus or intrusion) can be responded to (similar to white blood cells). Forrest et al. have experimented with creating such a sense of self in unix processes, based on these immune system analogies[18, 22]. The sequences of system calls used during typical executions of unix processes (tested primarily on sendmail and lpr) were recorded, using a sliding window. Comparing the system call sequences of different processes indicated that processes could be distinguished from each other. Using processes in an unusual fashion (such as attempting a buffer overflow attack, for example) also resulted in sequences of system calls that differed from those found during normal execution. Thus this method shows promise as an intrusion detection method (as well as detecting errors generated by unusual circumstances, such as swap becoming full and thus not printing a very large document). However, this has only been tested on a very small subset of processes on a live system, and more experiments need to be performed to determine how well this approach works with large number of processes, how efficient it is when run on-line, and how to optimally train this system so that it is trained using only clean data.

The data set developed by Hofmeyr et al. in [22] to test the immune system analogy was later used in [86] by Zhu et al., who compare three data mining techniques to determine which provides the best results for network intrusion detection. This experiment compared inductive learning, rough sets and neural network techniques to determine which best predicted normal versus intrusive activity. The data set from Hofmeyr et al. consisted of the system calls made by the sendmail program, both when executed normally and when used to compromise the system. Each invocation of sendmail was divided into sequences of seven system calls, where this number of system calls was based on the results of [22]. The data sets used for training the three prediction methods were divided into balanced (equal numbers of normal and abnormal sequences) and unbalanced (approximately 60% normal versus 40% abnormal sequences). The data set was further represented as binary data and integer data to determine if the data format affected the results.

The data set was split into three equal portions, which were combined in three different ways to generate the training and testing data. Additionally, a three-fold cross-validation approach was used, generating three different partitions of the data set. This resulted in nine total data sets used for each experiment, and 12 experiments (three methods and two representation formats and two data proportions). The results from these experiments showed that rough sets outperformed both neural networks and inductive learning. The best results were generated when a balanced data set was used for training, while the data representation did not affect the results. However, the best results still only recognized normal data with 86% accuracy and abnormal data with 72% accuracy.

2.5.2 Effectiveness of Intrusion Detection Systems

Network intrusion detection systems can be eluded by dedicated attackers who need only a minimal amount of information about the network and system, such as the presence of a network intrusion detection system and the operating system of the target system. Assuming there is no host-based intrusion detection systems deployed — only network-based intrusion detection systems — detection can be avoided by understanding how various operating systems handle malformed packets. Ptacek and Newsham present three methods for evading detection by a network intrusion detection system in [67]: insertion, evasion, and denial of service. Insertion refers to the attacker inserting packets onto the network that are destined for the target, however are malformed and so will be dropped by the target system. The network intrusion detection system, however, will still analyze the packet as part of the session if it is not configured similarly to the target operating system. In this manner simple string matching algorithms can be thwarted. For example, if the network intrusion detection system watched for the string "phf" directed to the web server (phf is a well-known cgi-bin vulnerability that was present by default in older versions of the Apache web server) then inserting packets "ackets", "ave", and "un" (with properties that will cause these packets to be dropped by the target system) between packets with "p", "h", and "f" would cause the intrusion detection system to receive "packetshavefun", once all the packets were put together. If the packets "ackets", "ave", and "un" had properties that would cause them to be dropped by the target system, then the target would see the phf attack without any alarms having been raised by the intrusion detection system.

Opposite to insertion is evasion, where packets are formed so that the network intrusion detection system will drop them, however the target system will not. For example, if the intrusion detection system reassembles packets differently than the target operating system, then it is open to evasion attacks, where the attacker can craft a packet with a sequence number that is ignored by the target system, but causes the intrusion detection system to become desynchronized (e.g. through the use of a new SYN packet in the middle of a connection, where the SYN packet also has a new sequence number).

The insertion and evasion attacks described in [67] appear less than one year later in an article in Phrack magazine, a magazine dedicated to hacking. This article outlines 11 different insertion and evasion hacks, and provides source code for performing these attacks. These attacks all utilize changes in the packet header that cause the target to drop the packet, but which are not checked by the intrusion detection system and are thus recorded. For example, sending a SYN with new sequence numbers in the middle of a session may cause an intrusion detection system to reset their connection information with the new sequence values, allowing the original session to continue unrecorded[24].

Network intrusion detection systems are also vulnerable to denial of service attacks[67]. These systems tend to be "fail open", meaning that if they stop functioning, the network itself remains functional. Thus it is possible to launch a denial of service against the intrusion detection system solely and disable it, preventing it from logging any future intrusion attempts against the other systems on the network. A denial of service can be launched against the CPU, memory or disk restrictions on an intrusion detection system. For example, opening a large number of connections to other machines requires the intrusion detection system to maintain state information on each of these connections, and to process the packets it receives to determine if they are part of an established connection or a new connection. All of this state information requires memory, and the processing of new packets requires CPU resources. Thus opening enough connections and flooding the network can force the network intrusion detection system to drop packets.

Intrusion detection systems need to be measured in terms of effectiveness, which is the ability to recognize intrusions and avoid false alarms.[2] This has also been described as "accuracy" and "completeness", where accuracy indicates the number of false positives (the number of non-intrusive events that generated an alarm) and completeness indicates the number of false negatives (the number of intrusive events that were not flagged by the intrusion detection system)[67]. Axelsson argues in [1] that effectiveness, or accuracy, will be difficult to achieve due to the base-rate fallacy. This is due to the large amount of legitimate traffic, compared to the relatively infrequent intrusive or anomalous activity. As a result, even a system that generates very few false alarms (e.g. 1% or less), will generate considerably more false alarms than true alarms due to the relative amounts of legitimate versus anomalous activity. As a result, the systems administrator is less likely to react quickly to an alarm.

2.5.3 Testing Intrusion Detection Systems

Puketza et al. presented a methodology for testing intrusion detection systems in 1996[69], however they concentrated on testing the resilience of the software to adverse conditions (such as stress testing), and spent very little time comparatively on how to properly test the effectiveness of the software in correctly detecting intrusions with a minimal number of false alarms. Their approach to this form of testing was to use the taxonomies of intrusions developed by others, and to choose representative intrusion techniques from each class within each taxonomy. These representatives were then used for testing to determine the detection accuracy of the software. No discussion on choosing appropriate non-intrusive behaviour was given. This work was continued in 1997[68], where the authors identified three performance objectives for intrusion detection systems. The first was that a system should be able to detect many types of intrusions. The second was that a system should not require a large amount of CPU, memory, disk or other resources. The third was that a system should be resilient to stress, continuing to function even under conditions such as high load. However, the authors concentrated primarily on the second and third objectives. While the authors acknowledge that effectiveness at detecting intrusions is important, measures such as percentage of correctly identified intrusions and the number of false positives are not addressed.

The Lincoln Laboratory at MIT later developed a controlled test environment with funding from the Defense Advanced Research Projects Agency (DARPA), and experiments were performed in 1998 and 1999. The purpose of the DARPA environment was to evaluate intrusion detection systems and provide recommendations for future research directions based on the results. In order to perform this evaluation, a simulation of a typical network was created. The traffic from this simulation resembles that of a typical United States Air Force Base. The simulation is described by Lippmann et al. in [50], and consists of network traffic captured by a border router, as well as Solaris BSM audit data. Four types of attacks were injected into the network: denial of service, remote to local (where a remote user gains local access to a machine), user to root (where a local user gains superuser privileges) and surveillance/probing, which consisted of network probes from tools such as nmap. Training data was provided to six groups, who used this for training their intrusion detection systems. These systems were then tested by the Lincoln Labs, with the results presented as a Receiver Operating Characteristic (ROC) curve, graphing the number of false alarms per day versus the percentage of attacks detected. The testing data included both known intrusions found in the training data, and new intrusions that were not in the training data. As expected, all of the intrusion detection systems performed reasonably at detecting known attacks, and poorly at detecting novel attacks.

The DARPA simulation and evaluation was later critiqued by McHugh, with the goal of providing insights on how the process could be improved[52]. Much of the critique centered around how the data was generated, about which very little information is available in the public record. For example, Lippmann et al. describe how realistic email and web browsing information is generated (using wordspotting), however comparatively little information is presented on background noise (e.g. network traffic caused by packet storms, and other

Internet phenomenon) is generated, if at all. McHugh goes on to critique the amount of data generated, and if any consideration was given to generating realistic amounts of data, including the hourly patterns that can be seen in office traffic. McHugh also critiqued how the results of the various intrusion detection systems were analyzed, as well as the lack of detail in how various calculations (such as the number of false positives) were calculated. Despite these criticisms, the DARPA evaluation is still the best approach to testing the effectiveness of various intrusion detection systems to date.

2.6 Distributed Denial of Service

Distributed computing has had a wide influence on computer science, touching on a number of areas such as distributed operating systems[81], distributed database systems and query processing[43], and distributed mobile agents[80]. However, in 1999, distributed technology came to be used for malicious purposes with the creation and use of distributed denial of service attacks. These attacks became highly publicized in February 2000 when large sites such as Yahoo and eBay became the targets[75].

A denial of service attack is an attack against a target where the aim is to deny legitimate access to a particular service[5]. Denial of service attacks can be designed to deny specific services (e.g. telnet) or can consume excessive bandwidth, effectively disconnecting the target from the Internet. Such attacks tended to be from a single source to a single destination. As a result, attacks that consumed excessive bandwidth, for example, were difficult unless the source had significantly more resources than the target.

In 1999, distributed denial of service networks, such as Trinoo and the Tribe Flood Network (TFN), started to become widespread[25]. The distributed denial of service (DDoS) was different in that it employed a number of host computers (called agents) to perform a coordinated attack against a target. The host computers were controlled by one of more handler machines, which are often machines that have also been compromised by the attacker. A DDoS had the advantages that it was much more difficult to trace back to the source(s), and while the target could block incoming traffic from a single, known source, they could not effectively block traffic from many different sources without also denying service to legitimate users.

In August, 1999, a successful DDoS attack was performed against the University of Minnesota, as well as several other systems worldwide. This attack employed a Trinoo network of 2200 systems[13]. In order to perpetrate an attack using such a large number of systems, automated tools were used to compromise remote systems. Based on attacks such as this, and later modifications to the automation of these attacks (such as using IRC channels to deliver attack instructions to compromised machines, thus adding another level of indirection in attempting to identify the attacker), a taxonomy of DDoS attacks has been developed. This taxonomy is based on the degree of automation, exploited vulnerability, attack rate dynamics, and impact[53].

The tools employed to generate DDoS attacks provide some degree of control to the attacker in terms of forming the packets. For example, the tool Shaft allows the attacker to specify the duration of the attack, the size of the packets, and the type of flooding (TCP SYN, ICMP, UDP). Additionally, Shaft will hide the source of the attacking systems by generating a random source IP. Random source ports are used, and the ACK (acknowledgment) and URG (urgent) flags are randomly set for TCP attacks. However, the sequence number is fixed, and flooding occurs in bursts of 100 packets[12]. Other tools have been found to randomize nearly all IP header options so that only the destination IP is the same for each packet[25].

It is interesting to note that, while the automation of intrusions and DDoS tool installation is increasing, these tools are not widely shared. In fact, it is noted by Dittrich that "these automated tools are held closely by those groups who wrote them"[14]. This trend is also identified by Houle and Weaver who state that these tools are not published so as to prevent rival groups from using them[25].

2.7 Clustering

A number of clustering techniques have been developed, such as statistical methods (e.g. k-nearest neighbour), neural networks (e.g. self-organizing maps), evolutionary techniques (e.g. genetic algorithms)[36], decision trees, rough sets[86], and support vector machines[10]. Many of these techniques have been applied to the area of intrusion detection research[86].

Many of these clustering techniques (e.g. decision trees and neural networks) are inductive learning methods. Inductive learning is a generic term that refers to making assumptions from current knowledge and applying those assumptions to unseen data. The belief behind inductive learning is that any hypothesis that approximates a sufficiently large data set will generalize to previously unseen examples[54]. One of the issues faced by inductive learning methods, given that the initial assumption holds, is that the hypothesis can learn a data set too well and thus overfit the training data. That is, as the hypothesis becomes better at representing the training set, its ability to generalize to previously unseen data diminishes. There are currently very few guidelines on how large the training data set needs to be, and how to determine that a hypothesis sufficiently approximates the training data without having sacrificed its generalizability to unseen data.

All clustering techniques are sensitive to the representation of the input pattern and the similarity measure used for generating clusters[36], so careful preprocessing of the data set is required before any clustering can be performed. In terms of pattern representation, the features to be used in clustering need to be determined. This step can greatly influence the results, as excluding a particular feature from the clustering algorithm can generate incorrect or misleading results, as that feature may be pertinent to generating the ideal clusterings. Conversely, including an irrelevant feature may unduly influence the final result, again resulting in clusters that do not represent the ideal solution.

The similarity measure is another important characteristic of clustering algorithms. This measure is used to determine if an item belongs to a particular cluster. The most commonly employed similarity measure is the Euclidean distance[36]:

$$d_p(x_i, x_j) = (\sum_{k=1}^{d} |\chi_{i,k} - \chi_{j,k}|^2)^{1/2} = ||x_i - x_j|| \tag{2.1}$$

where x_i is the feature vector for pattern i, $\chi_{i,k}$ is the individual feature k in pattern i, d represents the dimensionality of the feature vector. The drawback to using this method is that the largest-scaled features tend to dominate over the other features as they will generate the largest differences. Some features may be very far apart in their feature space, but due to the small scale this distance will be overwhelmed by larger-scaled features.

2.7.1 k-Nearest Neighbour

The best known statistical clustering technique is the k-means clustering approach, or k-nearest neighbour[36]. In this approach the user provides the number of clusters desired, k. Random cluster centers (centroids) are then chosen from the pattern set K. Patterns are assigned to clusters based on the distance between the pattern and the centroid, such that the distance is minimized. Once all patterns have been assigned to a cluster, the centroids for each cluster are recalculated. Patterns may then be moved to other clusters based on the distance between them and the centroid. This process continues until all of the clusters are stable, or the squared error no longer decreases significantly between iterations.

The distance for the k-nearest neighbour approach is calculated using the squared error criterion:

$$e^2(\mathcal{K}, \mathcal{L}) = \sum_{|=\infty}^{\mathcal{K}} \sum_{\rangle=\rangle}^{\backslash_|} ||\S_{\rangle}^{|} - \rfloor_{|}||^{\in} \tag{2.2}$$

This defines the squared error for a clustering $\mathcal{L}$ of pattern set $\mathcal{K}$, where x_i^j is the i^{th} pattern in the j^{th} cluster and c_j is the centroid of the j^{th} cluster.

The problem with this approach is that the final number of clusters needs to be determined before the algorithm is run, as this is the stopping condition. Often, however, it is desirable to have a clustering algorithm that determines the optimal number of clusters during execution based on some other criteria. Otherwise, the human operator needs to be a domain expert in order to reasonably predict the desired number of clusters.

A second problem with this approach is its sensitivity to the starting conditions. The algorithm generates different clusters based on different starting states. Thus in practical use, this algorithm is run multiple times, with the best output chosen as the final clustering.

2.7.2 Self-Organizing Maps

Neural networks have been used for clustering as well. The most commonly used network is Kohonen's self-organizing map (SOM). This method is particularly appropriate for exploratory data analysis as it is an unsupervised learning method requiring no previous assumptions about the distribution of the data. Additionally, it allows visualization of the data by reducing the dimensionality of the data to two-dimensions, while still largely preserving the distance between elements[40].

Each element in the data set is represented by a vector x_k where $k = 1, 2, ..., n$ of features or observations. The SOM can be viewed as a two-dimensional grid of points, where each point has been randomly initialized with a vector of the same dimensionality as the input data set. When an input vector is read, the unit in the network with the smallest distance to the input vector is chosen, based on the Euclidean distance

$$c(x) = \arg\min_i \{||x - m_i||^2\} \tag{2.3}$$

where x is the input vector and m_i is the is the i^{th} unit in the SOM. Thus the unit chosen most closely resembles the input vector (by choosing the unit that minimizes the Euclidean distance). The values in this unit are then adjusted, moving them closer to the input values, using the equation

$$m_i(t+1) = m_i(t) + \alpha(t)[x(t) - m_i(t)] \tag{2.4}$$

where t is time, $m_i(t)$ is the value of the i^{th} unit in the SOM at time t, $x(t)$ is the input vector at time t, and $\alpha(t)$, called the adaptation gain, is $0 < \alpha(t) < 1$. The adaption gain decreases over time, meaning that the largest changes to the units are made in the initial stages of the algorithm, with the magnitude of the changes decreasing over time[41].

The network described so far does not take advantage of the relation between nodes in the network. For this relation to be exploited, related nodes need to be located near each other. In order to achieve this, the SOM defines a neighbourhood surrounding the chosen node, c. As the node is adjusted to more closely resemble the input vector, so too are all neighbouring nodes. The size of the neighbourhood is initially large, and is slowly reduced as the network continues to be trained. Thus the equation representing the training at each step is:

$$m_i(t+1) = \left\{ \begin{array}{ll} m_i(t) + h_c(t)[x(t) - m_i(t)] & \text{if } i \in N_c(t) \\ m_i(t) & \text{if } i \ni N_c(t) \end{array} \right\} \tag{2.5}$$

where $N_c(t)$ is the neighbourhood set, and all the other variables are as described previously. This equation has also been modified so that the effect of the input vector is greater the closer the neighbouring node is to the chosen node, while the modifications taper off as the distance from the chosen node increases, represented by $h_c(t)$. This function is often shaped like a bell curve[41].

One of the advantages of self organizing maps is that they are an unsupervised learning method, not requiring labels on the data. Thus the structures formed will indicate relations in the data which can later be labeled[40].

However, the self organizing map is very sensitive to the relative scales of the features in the elements it is clustering. A feature with a relatively large scale can disproportionately affect the generated map. Changing the relative scales of the features can result in completely different maps. Thus the selection and preprocessing of the data may be of equal or greater importance to actual generation of the map itself[40].

2.7.3 Genetic Algorithms

Genetic algorithms are modeled after biological systems, employing crossover and mutation in a manner similar to DNA recombination[54]. In a genetic algorithm, hypotheses are represented as strings of bits. For example, the attributes sunny, cloudy or rainy could be represented by the strings 001, 010 and 100 respectively. Thus a string can represent a series of attributes, with the ending substring representing the predicted result (e.g. play tennis could be represented by 10, while do not play tennis could be 01). An example hypothesis would be 00110, representing the hypothesis "if it is sunny, play tennis". This representation has the additional benefit of being able to represent a "don't care" state, using 000, for example, to represent that the weather (sunny, cloudy or rainy) is not relevant to the hypothesis.

An initial population is created randomly consisting of strings that represent the target hypothesis. Genetic algorithms then employ a beam search (where the k best candidates are kept, as opposed to the single best candidate) through the hypothesis space. In order to determine if a hypothesis should be kept, a fitness function is employed, such as $fitness(h) = (correct(h))^2$, where the fitness is the square of the number of training examples correctly classified by the hypothesis. Often the ratio of the fitness of an individual hypothesis to the average fitness of the population (called fitness proportionate selection) is used as a probability for selecting the current hypothesis.

The search through the hypothesis space is performed primarily by a process called crossover, which occurs on a user-specified percentage of the current population. Crossover is modeled after its biological counterpart, where two pieces of DNA are recombined to form a new structure. In a single-point crossover for a genetic algorithm, two hypothesis are chosen from the current population. A random point within the string representing the hypothesis is chosen as the crossover point, so that the first half of hypothesis x is combined with the second half of hypothesis y, and vice versa, generating two new strings. For example, given strings 00110 and 01001, and the crossover point between bits two and three, the new hypothesis would be 00001 and 01110. Crossover can also be performed on multiple points. For example, using two-point crossover would result in two strings swapping a substring in the middle of the hypothesis. Using the previous two strings 00110 and 01001 to illustrate, assume crossover points are between bits one and two, and between bits three and four. The resulting hypotheses would be 01010 and 00101, where bits two and three have been exchanged.

Similarly, mutation occurs within the current population based on a user-specified probability. Mutation, also modeled after biological evolution, refers to the changing of a single, randomly chosen, bit within a hypothesis string. This feature, employed with a very low probability, enables genetic algorithms to avoid becoming trapped in local minima.

The advantages of genetic algorithms include their ability to avoid becoming trapped in local minima. Additionally, they can be easily parallelized, allowing for faster convergence to an optimal solution, and they have been successfully applied to a number of optimization problems. However, they require careful consideration to the representation of the hypothesis. Also, genetic algorithms suffer a condition called "crowding", where an individual hypothesis that is particularly fit will reproduce quickly, resulting in offspring that represent the majority of the population. This reduces the genetic diversity of the resulting populations which can have the effect of slowing down the search of the hypothesis space[54].

2.7.4 Decision Trees

One of the earliest and best known approaches to inductive learning is that of decision trees. Decision trees are built using a set of attribute-value pairs that can be used to predict some outcome. At each node in the tree, a decision based on the value of a particular attribute is made, and the appropriate branch followed. The leaves of the tree represent the desired outcome given specific values of the various attributes.

The best known decision tree approach is ID3, developed by Quinlan in the early 1980s[54]. Given a set of data, ID3 chooses which attribute should be placed at the root node based on the amount information gain provided by that attribute. Information gain measures the expected reduction in entropy, where entropy refers to the how many bits are required to represent information. In terms of building a decision tree, each node consists of the attribute that best distinguishes between the different possible outcomes based on its own possible values. ID3 is one example of Top Down Induction of Decision Trees (TDIDT) algorithms. Other examples, such as ID5R and AQR, are similar in concept, however employ different algorithms[15].

Decision tree learning is best suited to solving problems that contain the following five characteristics:[54]

1. The data used for making a decision consists of attribute-value pairs. For example, attributes such as temperature (hot, warm, cold), blood type (O, A, B), and colour (red, blue, green, etc.) can be used by ID3. Learning based on relations (e.g. is address A close to address B?) is not suited to decision tree learning.

2. The desired output consists of discrete values, such as yes/no answers. Decision trees can not represent continuous valued results.

3. The output rules take the form of disjunctive conjunctions. That is, a decision tree is a representation of a set of if-then rules such as "if it is hot and sunny, or if it is warm and sunny, or if it is warm and overcast, then play tennis".

4. The data set contains missing values. There are variations on the ID3 algorithm that allow a decision tree to deal with missing data, such as in the case of medical data where information such as a person's blood type may not be known. There are various methods for dealing with missing data, most consisting of calculating some form of an average or expected value for the missing piece.

5. The data set contains errors. Decision trees are robust to small errors in the training set as their results are based on the information gain for each attribute. Thus if there are small errors, the impact on the information gain will be negligible. While this may result in a tree that does not represent every training example exactly, it still results in a tree that approximates the desired function.

2.7.5 Rough Sets

Rough set theory refers to a set theoretic method for clustering labeled data. Rules can then be extracted so that future events can be categorized. All input records are considered to represent the universe U, while each attribute a is an element of the set of all attributes A. Let $\mathcal{A} = (U, A)$ represent the complete information system. Let $B \subseteq A$ and $X \subseteq U$. The information in the set B can be used to generate two different approximations to X. The first, called *B-upper approximation*, is $\overline{B}X = \{x|[x]_B \cap X \neq \emptyset\}$, where x is an element of X. This set contains all items which might be elements of X based on knowledge of B. A subset of this, called the *B-lower approximation* contains all elements that can be classified with certainty as being a member of X. This is represented as $\underline{B}X = \{x|[x]_B \subseteq X\}$. As a result, $U - \overline{B}X$ consists of all elements which are with certainty not elements of X. Thus those elements that are in $\overline{B}X$ but not in $\underline{B}X$ ($\overline{B}X - \underline{B}X$) are in the boundary region between those elements that with certainty belong to X, and those that with certainty do not belong to X. If this set is empty, the set is considered to be crisp, otherwise the set is said to be rough. In practical terms, this boundary region represents those elements with similar or identical attributes, but different labels[42, 61, 82]. In order to speed the process of generating rough sets, the data sets used can be reduced. One method of reducing data is to use equivalence classes. That is, have all elements with the same values for both attributes and decision be represented by only one element. A second method of reducing data is to calculate the minimal reduct, which is the minimal number of attributes required to partition the universe. If there are attributes which contribute no information to the generation of the sets, then they do not need to be considered. This in turn speeds the process of generating the rough sets and allows for simplified decision rules.

One of the advantages of rough sets is that the input data set can be used as is, without requiring any preprocessing or additional information, such as various probabilities. Additionally, this method has the ability to to work with missing values, and the results it generates are in the form of decision rules which allow for easy classification of new data elements[86]. However, the reduction of data through the computation of minimal reducts is known to be an NP-hard problem and is currently one of the challenges facing rough set theory[42].

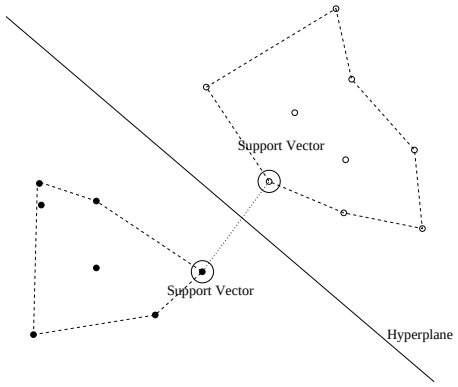

Figure 2.3: A data set consisting of two classes, represented here as solid circles and clear circles, can be separated using a hyperplane. The dashed line shows the convex hulls for the two sets. The double circles represent the support vectors, and the dotted line shows the shortest distance connecting the two hulls.

2.7.6 Support Vector Machines

Support Vector Machines (SVMs) is a supervised training algorithm for classifying data into one of two possible outputs. Given a data set consisting of two classes, the set is divided using a hyperplane. If the set is not linearly separable, it is mapped using a non-linear function (called the kernel function) to a higher dimension feature space, Z, so that it can be linearly separated. The input vectors that are closest to the hyperplane are called the support vectors (see Figure 2.3). The support vectors can be calculated by first finding the convex hulls for the two sets of data points, where it is known that these hulls will not overlap because the input vectors are linearly separable. The chosen hyperplane is orthogonal to the shortest line connecting the two hulls and has the additional property that it is maximally distant from each hull, and is called the maximum margin hyperplane[83] or optimal hyperplane[10].

One of the difficulties with this is that the feature space can have a very high dimensionality (for example, a polynomial of degree 4 or 5 in a 200 dimensional space may result in a feature space of a billion dimensions before a hyperplane can be constructed[10]). However, the optimal hyperplane can be calculated using only the support vectors, thus reducing the computation complexity of the problem. Additionally, the optimal hyperplane provides an upper bound on the probability of errors — the ratio between the number of support vectors and the number of training vectors. Thus by decreasing the number of support vectors and increasing the number of training vectors, the generalizability of the function can be increased[4].

Having found the optimal hyperplane separating the two classes, where the values of the two classes are in the set $-1, 1$, the expected output value of a new input vector z is:

$$f(x) = sign(\sum_{i=1}^{n} \alpha_i z_i \cdot z + b) \tag{2.6}$$

where the sum is over the n support vectors, $z_i \cdot z$ is the dot product between the support vectors z_i and the input vector z, α_i are weights associated with each support vector, which were calculated based on the optimal hyperplane, and b is the bias[4].

One of the advantages of support vector machines is that they do not become trapped in local minima, but that they always find a global solution. Nor do SVMs overfit the data as only the support vectors are used in determining the output function, not the entire set of training vectors[4].

The Support Vector Machine described here requires that the training data be linearly separable with no errors. This has been extended to allow the calculation of the optimal hyperplane given the presence of some errors in the training data by Cortes and Vapnik[10]. Extending SVMs to perform classification in the case where there are more than two possible outputs has been addressed[48], however is still in the early stages of research[4].

Additionally, SVMs are limited by the choice of kernel (the non-linear mapping function that transforms the input space to the higher-dimension feature space) — if a user chooses their kernel poorly, then the SVM will perform poorly. Additionally, very large problems (e.g. >> 100,000 support vectors) remain intractable[4].

2.8 Set Covering

The set covering problem (SCP) is: given a set of points, and a collection of sets of points, find the smallest subset of these sets such that all of the points are "covered" (contained in one of the chosen sets). A variation on this is the weighted SCP, where weights are assigned to points in a set. This problem is NP-hard.

Grossman and Wool[21] compare various algorithms for solving the set covering problem, such as the greedy algorithm, approximation algorithms and neural networks. Their results indicated that the randomized greedy algorithm obtained the best solution in the majority of cases, given constrained test sets. The randomized greedy algorithm is a standard greedy algorithm, where the set chosen at each point is the largest set currently available, however ties are broken by randomly choosing one of the candidate sets. The majority of test sets were two-dimensional graphs consisting of up to 500 rows by 5000 columns.

2.9 Related Research

In those intrusion detection systems which have been programmed and configured to report port scan activity, the most common method employed to detect port scans is to check that for some minimum number of port scans (such as miscellaneous SYN packets, where the connection has not been completed) to some minimum number of different ports occurring within a fixed amount of time. For example, the intrusion detection software Snort[31] is configured by default to generate an alarm only if it has detected SYN packets sent to at least four different ports in less than three seconds.

With regards to detecting distributed port scans, Carver, who suggests employing intelligent agents for intrusion detection, only addresses this briefly, stating:[39]

> Attacks such as a distributed port scan from multiple IP addresses over several days or weeks would be much more difficult to detect. In determining if an attack is a continuation of the same attack or a new attack, the Master Analysis agent uses two-level fuzzy rule set which considers the time between the incident reports, IP addresses, the user name, and the program name to determine whether an incident is a continuation of an existing attack or a new attack.

Although the details of the fuzzy logic employed are not stated in the article, it is likely that using just those variables will not detect a distributed port scan. User name will not be available for a port scan. Interpreting program name to mean port number results in only three variables available to be considered by the system — destination port number, source IP address and time between incidents. This does not consider additional available information, such as source port number and destination IP address, and so will likely have difficulties detecting distributed port scans.

Staniford et al.[79] present a method for the detection of stealth port scans, where stealthy port scans are defined as port scans with properties that would elude traditional IDS systems. Examples of such properties are long delays between scans and using numerous source IPs in a distributed port scan.

The method employed by Staniford et al. can be divided into two distinct sections: the network anomaly detector (called Spade) and the correlation engine (called Spice). The network anomaly detector estimates the current probability distribution of normal traffic, and based on the entropy in this measurement assigns an anomaly score ($A(x) = -log(P(x))$ where x is the packet, $A(x)$ is the anomaly score, and $P(x)$ is the probability of the packet occurring) to an incoming packet. The more unlikely it is that a particular packet will occur (e.g. based on source and destination port/IP combinations), the higher its anomaly score.

Any packet flagged as anomalous is passed to the correlation engine, where it is inserted into a graph, where the nodes represent packets and the connections between nodes contain weights indicating the strength of the relationship between the two nodes (packets). The weights are based on a combination of the following items:

feature equality : two features have the same value (e.g. source IP).

feature proximity : two features have values that are near to each other. For example, two event times may be similar, or two destination ports may only differ by one.

feature separation : two features may exhibit a well-known separation between values. For example, they may occur exactly five minutes apart, or the time gap between event A and event B may be the same as the time gap between event B and event C.

feature covariance : this indicates that there is a relationship between the change in one feature and the change in a second feature. An example of this would be both the source port and destination IP increasing at the same rate.

Each item produces a value between 0 and 1, which is then multiplied by a co-efficient and summed, generating the following equation:

$$f(e_1, e_2) = c_1 h_1(e_1, e_2) + c_2 h_2(e_1, e_2) + ... + c_k h_k(e_1, e_2) \tag{2.7}$$

where e_1 and e_2 are two network events, $h_k(e_1, e_2)$ is the function comparing a feature k of e_1 to e_2, c_k is a weighting factor for the function h_k, and $f(e_1, e_2)$ is the function generating a weight that represents the relationship between e_1 and e_2. In the final graph, all edges with weights less than a certain threshold are dropped, and the remaining subgraphs represent port scans.

The graph is not fully connected, but rather as each node is inserted, it is initialized with four connections. Nodes are inserted into the graph using a simulated annealing procedure for clustering. That is, a new node is randomly connected to another node in the graph, and the weight for that connection is calculated. One of the neighbours of the node is then chosen at random, and a connection is made to it from the new node. If the weight of this new connection is greater than the weight of the previous connection, the new connection is maintained and the older one dropped. To lessen the likelihood of getting trapped in a local minima, there is some probability associated with connecting to the node with the greater weight. A probability that the node with the smaller weight will be chosen is maintained, and decreased over time until the probability is zero. This process is repeated another three times, resulting in each node having four connections.

The correlation engine described in Staniford et al.[79] has recently been implemented, and is available as part of Sentarus CounterStealth from Silicon Defense[30]. However, no further details of the correlation engine, Spice, have been released. It is known that Spice is not designed to specifically detect distributed port scans. Rather, it clusters network packets that have similar properties together. Each cluster can represent various events, such as distributed port scans, port scans with long delays between each scan, distributed denial of service attacks, and network misconfigurations.

2.9.1 SOMs and IDS

A few researchers have looked into applying self-organizing maps (SOMs) to the issues that arise in intrusion detection. Höglund et al.[23] have applied self-organizing maps to anomaly detection, where the behaviour of each user is modeled with a self-organizing map, based on the behaviour for the past 90 days. If the current behaviour deviates from the established norms for that user, an alarm is generated identifying the anomalous behaviour and allowing the system administrator to investigate further. In this instance, the SOMs are retrained each day using the data from the previous 90 days. While this approach alleviates the issues that arise from users changing their behavioural patterns over time, there is considerable overhead from recalculating the SOMs each day. Additionally, while anomalous behaviour appears to be predicted using this method, no results were presented that indicates the false positives and negatives that this method produces, nor were any results presented that indicate that this approach actual detects successful intrusions.

Chapter 3

Approach to Research Problem

3.1 Research Objectives

Hypothesis: We contend that distributed port scans can be represented by a formalized model. This model will be created through the identification and quantification of the properties of such scans. Verification of the model will be performed against real data through the use of an advanced prototype.

Section 1.2 presents three phases of an attack — the information gathering phase, the attack phase, and the post-access phase — based on the results from Jonsson and Olovsson[37, 38] and Skoudis[77]. It is during the information gathering phase that the attacker tries to map the target network, often employing methods such as port scans to determine what services are available and on what machines.

While every port scan, whether stealthy or naive, represents a potential threat, it is likely that a stealthy or distributed port scan represents a more technically sophisticated attacker. This is because performing a distributed port scan requires access to a large number of machines from which to launch the scan, which likely implies having first compromised those machines without being detected. Also, as no easily usable software is currently available for performing distributed port scans, an attacker will likely need to have developed their own. This implies a certain level of technical ability in terms of network programming. Finally, those adversaries who are less technically capable (and whose motives focus more on conquest than the malicious or financial use of information found on the compromised systems) are less likely to employ even stealthy scans, prefering to try to connect outright to systems to determine the versions of services running[65].

The sophistication of the information gathering phase and the resources employed to perform the information gathering is an indicator of the overall technical sophistication of the attacker. The target can use this perceived level of sophistication to determine the priority to be placed on responding to the port scan, and the subsequent security and monitoring to be deployed.

Software to detect distributed port scans is not currently known to exist, thus some of the sophistication of the adversary is not necessarily detected by the target. There is very little research to date on the characteristics of distributed port scans, and indeed very little on port scans in general. The first contribution of this research will therefore be to develop a general model of port scans, followed by a model of distributed port scans. A second contribution of this research will be the development of a suite of tools for systems administrators to allow them to detect distributed port scans performed against their networks, where these tools are based on the models developed. These tools will be developed with the aim of verifying that the properties identified in the model are correct. It is expected that contributions will also be made in the area of clustering during the development of these tools, as it is expected that modifications to current clustering methods will need to be made.

According to Rogers[73], those adversaries who are the most technically competent are also those who are likely to be motivated by greed or nationalism. It is assumed that this class of attacker will also cause the most damage. Thus by recognizing a distributed port scan, an early warning can be issued regarding a sophisticated adversary intent on a very destructive act.

If it is true that distributed port scans are used, then they should be found in network audit logs. More specifically, if it is true that distributed port scans are used by sophisticated attackers such as professional criminals and cyber-terrorists, then evidence should be found in the network logs of large corporations (e.g. IBM, Microsoft), government and the military. To confirm this requires two distinct steps: the development of a general model to detect distributed port scans, and then the verification of this model.

3.2 A General Model

The initial approach to the detection of distributed port scans will be the investigation into the properties required to generate a formal model of port scans. Currently, very little has been published identifying the unique characteristics of port scans. Thus the first step will be the analysis of the footprints left by distributed port scans. An analysis of one distributed port scanner was already presented in Section 2.4.3, however the other port scanners, dps, DScan and the scanner from Phrack, should also be analyzed. If necessary, a wrapper can be written around nmap to provide it with the capability to perform distributed scans, and the results from this tool can also be analyzed. Chapter 4 presents an initial investigation into the characteristics of port scans that can be used in the creation of a general model.

The second step will be to perform data exploration of a real dataset of port scan information using various data mining and machine learning tools. It is anticipated that this phase will provide information about port scans, and that this information can then be applied to the model of distributed port scans.

The last step will be to develop a tool that detects distributed port scans based on the model developed. This tool will then need to be validated against a real data set to evaluate the correctness of the model.

This process gives rise initially to two issues: what to use for a data set, and what techniques to employ in the implementation.

3.2.1 Data Sets

There are four data sets that can be used. The first data set can be generated locally on a network testbed. This data set will be small and well controlled and can be used for initial hypothesis testing. The second data set is the set available from Lincoln Labs, although this set will require some modification. This will provide a larger data set with real data for hypothesis testing. The third data set is a simulated data set that can be used for testing. The last data set is available from the Computer Emergency Response Team Co-ordination Center (CERT/CC), and will be reserved for performing model verification.

The data sets to be used, at least initially, will consist only of suspected scans, such as connections with a very low number of packets. This is a valid approach because connections where a large number of packets were sent likely represent legitimate connections. In those cases where the activity was intrusive (for example, through the execution of an exploit, or through the successful installation of a virus), it was still not a port scan and therefore not relevant to detecting distributed port scans. Further, using an approach where the data set only contains suspected port scans can be extended as intrusion detection systems, for example, will return just this type of information. Thus using a dataset of this format should be easily extensible later to combining the results from multiple intrusion detection systems to search for collaborative information gathering efforts.

This type of approach — that of aggregating and extracting only relevant data — is not new. A similar approach is taken by Staniford et al.[79] in which they first deploy Spade to determine what packets in an input stream are anomalous. It is these anomalous packets, similar to the database of suspected scans described above, that will be used by their correlation engine Spice. This is also similar to the approach suggested by Huang, Jasper and Wicks[33] where multiple deployed intrusion detection systems aggregate information, passing only relevant information on to a central control center.

Local Data Set

A data set can be generated locally by performing logging on one machine (or, preferably, one router serving multiple machines), and performing distributed attacks against that machine or network using one of the publicly-available distributed port scanners. The advantage with this data set is that the attacks will be known, so the number of false positives and false negatives can be measured. Additionally, the log results will contain valid signatures as actual distributed port scans have occured and public tools have been used to generate those

scans. In the event that the publicly-available scanners do not provide sufficient flexibility, a wrapper can be written for a single-source scanner such as nmap, allowing a distributed approach to mapping the target network. However, there are two disadvantages to this data set. First, the data set is very small, containing only one or a few hosts, so any techniques that work well on this data set may not scale to larger data sets. Also, the tools used to generate the distributed port scans may not generate all of the possible signatures. Any technique that detects these signatures may not generalize to other signatures that it has not already encountered.

Lincoln Labs Data Set

A second data set that is available is that used by Lincoln Labs in the 1998 and 1999 DARPA evaluations of intrusion detection systems[50]. This data set consists of simulated network traffic captured by tcpdump, and contains 32 types of attacks across four broad categories. One of the categories is "surveillance/probing", and contains attacks such as ip sweep, nmap and satan (a vulnerability scanner available freely on the Internet), across four platforms (Solaris, SunOS, Linux and Cisco router). These types of attacks will need to be extracted from the data set to create the scan database format being used by the implementation in this thesis. It should also be noted that this data set does not contain any distributed port scans, so this type of information will need to be injected into the data set. The advantage of this data set is that it has been used for intrusion detection evaluation in the past. However, as the distributed port scans would need to be added, it runs into the same problem as the small data set, where the tools used to generate the port scan data may not be representative of all possible distributed port scan techniques. Using this data set will still be valuable, however, as it contains single-source port scans. This will be helpful in ensuring that the model does not detect distributed port scans that are not present.

There are some issues with this data set, as noted by McHugh in [52]. One issue is that very little detail is provided on how the network traffic was generated, and in particular, the characteristics of the background data (that is, non-attack data). The claim is that the background data is characteristic of the data from an Air Force base, however no comparisons are given to substantiate that claim. McHugh also notes that TCP/IP traffic is not well behaved and that TCP/IP stack implementations are not necessarily well done, resulting in odd traffic, such as unusual bursts (packet storms). It is not clear that such traffic has been included in the simulations.

Simulated Data Set

A third data set can be created through simulation. This data set should be modeled from a real data set, such as that available from CERT/CC. The simulated data set should contain the same fields as the CERT/CC data set, and should also be programmed to generate records with the same distribution as the CERT/CC data set. The simulated data set is explained in more detail below. The advantages of this approach are that different data sets, with different data distributions, can be created on an as-needed basis. Also,

if data is collected from a live network, then a program can be developed that recognizes distributed port scans, however there is no reliable method for validating that the program is correct. Therefore, a controlled environment is necessary for testing purposes. However, it is difficult to ensure that the generated data will be representative of actual data. Errors can arise in the model, such as the distribution of some event being was missed, and errors occuring in the program where the correct attributes are modeled but they are not handled correctly. Recognizing that these errors have occurred is difficult, and having these errors occur without being found and corrected can result in an incorrect distributed port scan detection model being generated.

It is important that any program that generates test data sets have two key properties: parameters that are easily modifiable, and be representative of true network data. In terms of configurability, there are a number of parameters that must be considered when generating a test data set. The values of these parameters should be easy to modify, thus generating test sets with different properties. Different test sets should also be generated even when the same parameters are used.

The parameters that need to be modeled by this program are:

- network range
- time period to model
- likelihood of a horizontal scan
- likelihood of a vertical scan
- likelihood of a strobe scan
- probability of backscatter, worms and web crawlers
- average amount of anomalous traffic (e.g. router misconfigurations)
- probability of distributed port scans

In this model, no differentiation is made between workstations, servers, or non-existent machines (that is IP addresses that have not yet been assigned to a particular computer). This is based on the assumption that port scans will occur indiscriminantly, rather than showing a preference for servers or workstations. This assumption should be confirmed before the simulation is written and employed.

The various scans (horizontal, vertical and strobe) will need to have their attributes further defined. For example, what port or ports are most likely to be scanned, how many machines are likely to be scanned, what is the time delay between scans, and what is the number of packets observed per scan. Other variables, such as source IP and source port, can be generated randomly. The characteristics of backscatter, worm behaviour and web crawlers will also need to be defined, and may require additional input parameters.

Distributed port scans will need to be defined in more detail. For example, the user should be able to specify parameters such as the number of source IPs employed, the number of destination IPs, the destination port or ports, the time delay between scans, if the source ports are the same across the source IPs, and if the destination IP/port pairs are randomly distributed across source IPs. Other parameters may need to be defined based on the model of distributed port scans that is developed.

This simulation focuses on TCP traffic, but may need to be extended to include UDP (some port scans use this) and ICMP (e.g. ping) traffic. This decision should be based on the type of data found in the real data sets, such as the Lincoln Lab and CERT/CC data sets.

Any generated data will need to have a pilot study conducted first "to validate the use of the artificial data or to ensure that the data generation process resulted in reasonably error-free data."[52] Thus the CERT/CC dataset should be analyzed for all of the above information, except for the items on distributed port scans, which will not be known. Ideally, other datasets can be obtained and similarly analyzed to provide a variety of live networks to be modeled.

CERT/CC Data Set

A fourth dataset is potentially available from the Computer Emergency Response Team / Co-ordination Center (CERT/CC), located at Carnegie Mellon University in Pittsburgh. This dataset was produced by a class A network using Cisco Netflow, and contains logs for all incoming traffic covering a two year time span. Unfortunately, this data set is limited in that only incoming traffic was logged, not outgoing, so it is not possible to reconstruct session information. Since then, logging in both directions has been started, however only a few months of information is available. The logs consist of tuples containing a minimum of <source IP, source port, destination IP, destination port, number of packets, timestamp>. The amount of data logged was limited in this instance by practical considerations such as disk space, long-term storage space, CPU and memory requirements, and personnel workload, and is likely indicative of the logging that would be performed on any large site.

The CERT/CC dataset contains approximately 8 Gb of data per day, or nearly 6 Tb of data in total. The vast majority of this information will contain legitimate traffic and complete sessions. As it is not possible to establish a session, due to the lack of logs for outgoing traffic, sessions are determined by the number of packets between a particular source IP/port and destination IP/port within a particular time span. All entries suspected of being legitimate sessions will be removed from the logs. The result will be just those entries that are suspicious. They will likely consist of traffic between particular source IP/port and destination IP/port combinations where the number of packets transmitted from the external source to the internal destination are unusually low (for example, five packets or less).

Low packets counts can have a variety of causes. Backscatter, which is a response by some external host to a connection request sent with a spoofed IP where the spoofed IP happens to be an IP on the monitoring network[56], will result in only one packet being logged. A router misconfiguration can result in erroneous packets. A connection attempt that is then disconnected (e.g. due to a network connection being lost) will result in only a small number of packets having been transmitted (depending upon when the connection was lost). And, port scans will result in only a small number of packets. While a number of stealthy port scans will consist of only one packet to have been received by the target (e.g. a SYN scan), there are scans that perform a complete connection before being reset, in order to determine the version of the service available on the target port. In this case, at least three packets will be sent to the target, so limiting the number of packets to a small number, but not necessarily just one packet, is a valid approach to catch port scans.

Based on standard logging procedures, and the available data set, this thesis will use a data set consisting of tuples containing at a minimum < source IP, source port, destination IP, destination port, number of packets, start time, end time >. This database will have already been cleaned to contain only those tuples where a small number of packets were transmitted.

3.2.2 Clustering

Using these data sets, one possible approach to an implementation to test the validity of the model developed is to use a clustering technique to recognize distributed port scans. The model would provide the basis for the similarity measure used by the clustering technique. The aim of the implementation would be to provide the groupings for each individual distributed port scan, so that a network administrator would know at any point the number and characteristics of the distributed port scans aimed at the network. Several basic techniques are available for clustering, including k-nearest neighbour, decision trees, self-organizing maps, rough sets, genetic algorithms and support vector machines.

The k-nearest neighbour approach involves creating clusters around the means of k nodes, where the means shift as nodes are added and removed from the clusters. The k nodes used are initially chosen randomly from the entire data set, which makes this method very sensitive to the initial choice of nodes resulting in different clusters being produced each time the algorithm is run. Also, the final number of clusters, k, is determined by the user before the algorithm is started[36]. To apply the k-nearest neighbour approach to distributed port scan detection, the intention would be to have each cluster represent a port scan, both distributed and non-distributed. However, in order to deploy the k-nearest neighbour approach, the number of clusters would need to be known a priori. While there are techniques that circumvent this, they tend to be based on hierarchical structures, which is not representative of port scans. Thus this is not a particularly promising technique for the problem of detecting distributed port scans.

Using decision trees, and specifically an ID3-like approach, is not likely to be applicable to the problem of detecting distributed port scans. The decision tree approach can essentially cluster into some known number of clusters. In this case, there would be two clusters — distributed port scans and non-distributed port scans. While this would alert an administrator that a set of scans were part of a distributed port scan, it does not cluster the scans into sets. Thus the administrator will not know how many distributed port scans there were, only the number of individual scans that were part of a distributed port scan. Additionally, detecting if a set of scans contain co-operative scans is also based more on the relationship between the scans and less on any particular characteristics of the scans themselves. Decision trees, however, are based more on differentiating based on the characteristics of individual occurrences, and can not represent relationships between items. While it is possible for decision trees to represent these concepts, they are extremely complicated for a system based on attribute-value pairs, and so the resulting tree tends to be very bushy[15]. Thus decision trees are not a promising direction to pursue.

Support vector machines are a statistical technique where the data set is mapped from its current input space into a higher dimensional feature space through a non-linear mapping. This is done so that the clusters in the data set become linearly separable, and a hyperplane is created separating the sets. However, support vector machines discriminate best between two clusters of data, and while some research has been performed on how to discriminate between multiple classes of data using support vector machines, this is still a very young area of research[4]. Additionally, support vector machines do not scale well to very large problems. Thus support vector machines are not a promising initial direction for creating a technique for detecting distributed port scans.

Genetic algorithms are based on biological functions of evolution. Bit strings are used to represent the characteristics of a problem, and the solution to the problem. Learning takes place by recombining strings (such as two strings exchanging bit patterns) and by bit mutation. Strings are chosen to continue on to the next generation based on a fitness function[54]. This is a potential approach to detecting distributed port scans. However, this method will not create clusters of distributed port scans initially, but rather needs to learn first what the characteristics are of a distributed port scan. This requires that labeled training data be available, whereas all data available currently is unlabeled. Thus, while this is a viable potential approach to this problem, it is not an ideal approach.

Rough sets are a set-theoretic approach to partitioning data. A set is considered rough if there exist contradictory elements. That is, the attributes of the elements can not uniquely determine the decision of the element. Those contradictory elements form a boundary between the sets, resulting in sets with rough edges. These sets can be used to generate rules in a similar fashion to decision trees[42]. The data reduction methods employed by this approach — that of calculating the minimal reducts of a data set so that attributes containing no discriminating information are not used — might be applicable to the detection of distributed port scans. However, rough sets are not an ideal approach to this problem as they are a supervised learning method requiring labeled data.

Self-organizing maps are a pattern classification method where there is a network of vectors. Each training example modifies the node whose vector has values closest to those of the example, called the chosen node. The nodes in the neighbourhood of the chosen node are also modified, using a scaling factor so that the further a node is from the chosen node the less its values are changed. This eventually results in a map where each node represents a cluster, and the related clusters are grouped near each other. The main advantage to self organizing maps is that they are an unsupervised learning method. That is, they do not require labeled data[40]. Thus it may be possible to use this method to cluster port scans where the map would learn the important similarities between packets. Further, assuming the map learns about distributed port scans, it may be possible to extract the rules it used to generate clusters. Thus the self organizing map appears to be a feasible approach to detecting distributed port scans.

3.2.3 Set Covering

Set covering is a mathematical approach to the problem of sampling as few items as possible while covering all possible sets where each set represents items with a common feature. In other words, choose the fewest number of sets such that the entire space is covered. Although this problem is known to be NP-hard[21], it may be possible to apply heuristics for this approach to the detection of distributed port scans, with some modifications. Each set can represent a single-source port scan, and can have some value or set of values that represents the characteristics of the port scan. Rather than choose the minimum number of sets that cover the entire space (which could be a single block scan), the problem is modified to choose the sets such that some large portion of the network is covered and that some minimum similarity between the sets is achieved. While the best heuristic for solving the set covering problem is the greedy approach, this will not be an appropriate algorithm for detecting distributed port scans as the largest set, likely representing the largest port scan, is not necessarily a part of a distributed port scan. Sets are a logical representation for port scans, and a variation of set covering is likely a feasible approach.

3.3 Proposed Approach

The first step in approaching this problem is to design a model of distributed port scans. However, before this can be achieved, a model of single-source port scans will need to be developed as none currently exist. This process has already been started, as can be seen in Chapter 4, with the first step being to analyze the characteristics of port scans, both distributed and single-source, from which a model will be developed.

Once a model has been developed, this will provide information on the characteristics of distributed port scans, including expected similarities between scan sources as well as expected differences. However, the model may not have quantified these differences, which will be needed for any implementations.

As self organizing maps (SOMs) have already proven themselves in the arena of exploratory data analysis[40], they are a logical first step in analyzing the data sets to determine how to detect distributed port scans. This method can initially be applied to the small, locally-generated, data set to determine how well SOMs cluster, especially as it relates to the detection of distributed port scans. The use of such a small, well-known, data set will allow for tweaking of the input data, such as determining how best to represent the data so as to avoid problems caused by the relative scales of different features. The important features will have already been identified by the model created above.

Once the self organizing map performs satisfactorily on the small data set, it should be tested in a controlled environment using the simulated data sets. This will allow further exploration of the data, and in particular the relationship between the features of a distributed port scan. Using this exploratory approach will also provide insights into the applicability of self organizing maps to detecting distributed port scans, as well as the scalability of the self organizing maps as a solution.

The data exploration phase should not be limited to self organizing maps, but should also take advantage of other data mining and machine learning techniques. A number of programs are available, both commercially and freely on the Internet.

Once the data has been explored and analyzed, an implementation can be developed. As discussed earlier in the chapter, there are two possible approaches to this problem: the use of clustering techniques, typically used in data mining settings, and set covering, typically applied to sampling problems.

The approach that appears to be most promising is the use of clustering methods. However, "Since similarity is fundamental to the definition of a cluster, a measure of the similarity between two patterns drawn from the same feature space is essential to most clustering procedures."[36] It is expected that the model that has been developed will provide a basis for the similarity measure used by the clustering method. This measure will need to not only generate clusters that represent distributed port scans, but also be justifiable in how it measures the cluster similarity.

3.3.1 Self Organizing Maps

Of the clustering methods reviewed, self organizing maps (SOMs) provide the most promising initial approach. As SOMs are an unsupervised learning method, less consideration needs to be given to generating a comprehensive similarity measure. That is, the similarity measure will not need to define what characteristics in the data are important, so that the Euclidean distance or some similar measure can be used. However, it should be noted that this simply shifts the burden from generating the similarity measure to the data preprocessing stage. As SOMs are sensitive to relative scales of the input features, care will need to be given to ensure that this does not negatively impact the result.

Additionally, as self organizing maps use unsupervised learning, labeled data is not needed. Rather, SOMs will cluster based on similarities of the features in the data set. This provides the ability for constant learning, rather than learning on labeled data, then being used on test and validation data without being able to continue learning from that data. Thus it is anticipated that the result will be that a SOM learns a network, and is able to indicate that a port scan has happened by the number of related input data records. Once the port scan has been completed, this information should fade over time as other port scans occur.

SOMs also have the advantage of having previously been used for the data exploration phase, so might be more easily adapted to an implementation. However, there are some possible problems in adapting SOMs to the detection of distributed port scans. For example, SOMs tend to learn quickest when they first start learning, but this slows down as the network stabilizes. How well SOMs continue to recognize and cluster distributed port scans will need to be investigated. This can be started using the simulated data sets that were also used for the data exploration phase.

An algorithm that correctly identifies distributed port scans in a controlled environment will not necessarily correctly perform as well in a live environment. In particular, false negatives and false positives will be of concern. Therefore it is proposed that these algorithms be tested on a quasi-live environment as an intermediate stage. Such an environment might be provided by the Lincoln Labs data set, which does not contain any distributed port scans, but which could have them injected into the data set. While detecting the injected port scans will only show that the algorithm detects that which it is programmed to detect (given that the person scanning the network will have designed the algorithm, and so there will be nothing unusual in the scan that the program designer will not have already considered!), this environment will give an indication of the number of false positives the algorithm detects.

Perhaps the most significant result of using self organizing maps to perform these tests will be the generation of rules. Extracting rules from self organizing maps is a new and promising research area[40] and it is anticipated that these techniques will be applicable in this particular experiment. Thus rules will be generated that can be applied to the CERT/CC data set during model verification.

3.3.2 Set Covering

An alternative approach to self organizing maps is the use of heuristics that solve the set covering problem. One possible way to look at the detection of co-operating distributed port scans is to graph the results, and then employ a set covering approach to solving the problem. That is, assume that a motivated attacker will want to map some (large) minimum percentage of the attacked network, and is using a distributed port scan to do so. Assuming that there is not a one-to-one mapping between source IP/port and destination IP/port, the scanning activities could represent sets (by source IP, for example).

If using set covering, the question becomes can the anomaly sets be grouped, hopefully based on some relation between the data, such that some large percentage of the network is covered? Here, a set needs to be defined, and can possibly be defined based on feature similarity, such as using the same source IP. This is a reasonable first assumption given you are assuming a large (e.g. class A) target network to be scanned. Given this, to map a significant portion of the target network where each target system is scanned from a different source would require having previously compromised as many source machines as there are target machines. Assuming a large network, while this is theoretically possible, it is unlikely. A reasonable upper bound to the number of hosts that would be employed in a distributed port scan needs to be determined, and it may be possible that we will get some guidance here from the literature on the dissection of the distributed denial of service attacks against Yahoo, E-Bay, etc., that occurred in February 2000. For example, it is known that an attack against the University of Minnesota in 1999 employed 2200 systems[13].

The set covering approach needs to be modified to be used in this context. Traditionally, the set covering problem consists of finding the minimum number of sets such that all points are covered. In this context the points could be, for example, each machine in the target network. However, a distributed port scan may only cover a portion of the network, not all of the network. Therefore the problem needs to be redefined as finding a subset of sets that cover a certain target percentage of the network, rather than the entire network.

How large will my information space be? That is, how large is the anomalous data in a class A network? Given this, and that set covering is NP-hard, how well will current heuristics scale to the size required? Also, has set covering been performed for more than two dimensions? If not, will I need more than two dimensions? What dimensions will I have?

As an initial approach, it is recognized that set covering as it is currently implemented is likely not the best approach for this problem, but rather it will need to be augmented with weighted calculations, perhaps similar to those used by Spice. That is, the greedy approach, where the largest set is chosen, followed by the next largest set, etc., until some percentage of the network is covered, will likely not work. This is because the largest set is not necessarily part of a distributed port scan. Thus to include sets as part of the covering will involve weighting their relationship to other sets in the covering.

Once an implementation, either self organizing maps or set covering, has been developed, further information can be gathered using the local data set and simulated data set. For example, the time required to detect a distributed port scan and generate an alarm, can be derived. The amount of time required for initial learning will be determined, as well as system constraints such as the amount of memory and disk space required.

3.3.3 Validation and Extension

Validation of the implementation will be performed using the data set from CERT/CC, once satisfactory performance is achieved using the local, simulated and Lincoln Lab environments. It is known that this data set contains at least one distributed port scan (discovered accidentally by the researchers at CERT/CC). However, this and other distributed port scans may not be recognized by the implementation. Unfortunately, other than detecting the one known occurrence, there is no method that would indicate that the implementation is effective at detecting distributed port scans. Nor can it be verified that the distributed port scans detected by the implementation are indeed distributed port scans, other than through the opinion of domain experts.

One possible extension to this approach is that of having the model produce a probability associated with the distributed port scan indicating its confidence that the scan is indeed a distributed port scan. If the CERT/CC data includes a percentage likelihood that each record is a scan, then these values can be used to determine the confidence that a distributed port scan occured.

Another possible extension is to modify the algorithm to work from the raw network data, rather than from a database containing suspected port scans only. This would allow the algorithm to determine the presence of newbies and worms, for example, as they will appear as a port scan when the port is not available at the target IP, however there will be a payload associated with any existing port, so previous scans from that source IP could then be removed from the scan database.

3.4 Expected Results and Contributions

The contributions of this research will be primarily in the area of computer security. Attacks against systems are often preceded by port scans aimed at determining which systems are best to attack. Further, the less likely the port scan is to be detected, the more likely the attacker is both knowledgeable about computer systems and motivated to successfully compromise the target systems. Towards this end, the attacker will likely use distributed port scans in an effort to avoid detection. Thus the detection of distributed port scans can serve as an early warning system that a sophisticated attack might be about to occur. The contributions in this area will be the development of a model that represents the behaviour of a port scan, both distributed and not, as well as the development of an implementation that detects the presence of a distributed port scan given a database of single-source port scans.

As the implementation will likely use either clustering or set covering, this research may contribute to either the machine learning community in the area of self organizing maps or to the set theory community in the area of set covering. It is expected that any implementation will require modification to the typical algorithms used in these areas.

To date, self organizing maps (SOMs) have been applied to static data sets on which it trains, creating a map of related clusters. After this map has been created, it can be used for prediction by having the clusters labeled manually. Then new inputs can be provided, the cluster which they map to providing the label for the input, thus predicting what that input represents. However, in the detection of distributed port scans, the use of a static data set may not be appropriate. It is anticipated that the clusters will represent port scans, however these will change over time. SOMs, however, tend to initially learn quickly, but then settle into a particular patterns, with changes occuring much more gradually. But port scans, distributed and otherwise, will be constantly changing over time. If each node represents a distinct port scan, then the information in the nodes will also need to change over time. The modifications to the self organizing map algorithm that it is expected will be required will likely contribute to the machine learning community.

If the set covering approach is used, then modifications will also be required to apply this approach to the detection of distributed port scans. The goal of set covering is to choose the minimum number of sets such that the entire area is covered. However, if a port scan is represented as a set, then the goal is not to choose the minimum number of sets that cover the entire area (network). Rather, the goal is to choose those sets that fit together to cover some large portion of the area, and where the fit is determined both by the area covered and by other features of the data (such as the values of various flags). Solving this may introduce a new twist to set covering in the literature, as well as provide new heuristics for solving the problem.

Finally, the program that generates the simulated data set will likely make a small contribution to the research community. At least it will provide one more tool that researches can use.

3.5 Research Planning

3.5.1 Thesis Details

The final PhD dissertation should include the following chapters:

1. Introduction
2. Literature Survey
3. Problem Definition
4. General Model
5. Tool Development
6. Results and Discussion
7. Conclusions and Future Directions

The introduction will be similar to the introduction in the proposal, making the reader aware of the particular problem and its importance. The literature survey should concentrate on port scans, port scan detection, self organizing maps, and, if used, set covering. The problem definition will detail the difficulties in detecting distributed port scans and in applying current clustering methods to the problem. The problem definition section will contain information on research methods, which will be similar to Chapter 3 (Approach to the Research Problem) in this proposal. The fourth chapter will introduce the general model that has been developed, presenting a model for both generic port scans and the more specific case of distributed port scans. The tool development chapter will detail information about the actual implementation, including information from the data exploration phase, as well as any issues encountered and any research contributions in self organizing maps or set covering based on the approach used. The chapter on results will discuss the results from the experiment, while the conclusions chapter will outline the research contribution and future directions that should be pursued.

3.5.2 Time Planning

Date	Deliverable
13 December 2002	Proposal Defence
1 January 2003	Corrections to proposal made
1 February 2003	Analyzed dps, Rivat and dscan
1 May 2003	Completed data exploration of real data
1 June 2003	Created model
1 September 2003	Completed time at CAS (if required)
1 January 2004	Developed tool(s) based on model
1 March 2004	Validated model against real data
April 2004	First draft of thesis completed
July 2004	Defend thesis

Table 3.1: Suggested Milestones

The aim is to graduate in October 2004. It is hoped that I can spend four to eight months with the Computer Emergency Response Team, and perhaps a term with Jacob at the Technion Institute. Also, I will likely need to spend another three months at CAS. See Table 3.1 for a suggested timeline.

3.5.3 Publications Planning

My goal is to publish one journal article, and two reputable conference articles. Appropriate journals to publish in include the *ACM Transactions on Information and System Security*, Springer-Verlags's *International Journal of Information Security*, and SRI's *Journal of Computer Security*, although preference is given the ACM Transactions.

The three best conferences in which to present are Recent Advances in Intrusion Detection (generally held in mid-October, with submissions due end of March), ACM Conference on Computer and Communications Security (generally held in mid-November, with submissions due in mid-May), and the IEEE Symposium on Security and Privacy (generally held in mid-May, with submissions due early November). Of these, Recent Advances in Intrusion Detection would be the best forum. It may also be interesting to publish a paper in the USENIX Security Symposium (generally held in early August, with submissions due end of January), and appropriate to publish results in either a journal or conference on machine learning or set theory.

Chapter 4

General Model

While port scanning is a common occurrence, there is very little literature surrounding the subject, and most of that is not academic in nature. As a result, while port scanning is a known issue, articles have concentrated on how to perform a port scan, the different techniques for preventing detection of a port scan, and how various software is configured to detect port scans. There are no known publications that present a model of a port scan. This chapter presents the first step in creating a model — a review of the information as it is currently known. This will later be followed by an analysis of this information, from which a formal model will be developed.

The purpose of a port scan is to gather information about a target network. This information could be gathered for legitimate purposes. For example, a network administrator might scan his network to determine if there are any services running of which he was unaware. A user might be having problems connecting to a server and so scan that machine to determine if the desired service is currently running. The information obtained through a port scan can also be used by an adversary. In this case, the information about the target network and its services can be used to pinpoint possible vulnerabilities.

Indeed, one of the essential characteristics by which a port scan can be defined is the purpose of the scan. That is, a port scan occurs when a user, either friendly or adversarial, connects to a port for the purposes of determining if a particular service is running on a particular computer. Thus a port scan can consist of a connection to a single port on a single destination IP. An additional characteristic is that no data transfer occurs between the source and destination. A connection can be established (as in a connect() scan), however once it has been determined that a service is available, the connection is immediately torn down through the sending of a proper close connection sequence (sending a FIN packet, followed by a FIN-ACK from the target, followed by sending an ACK). From this, the following essential characteristics of a port scan can be established:

- The purpose of the port scan is to determine if a particular service or set of services is available on a particular machine or set of machines.
- The minimum number of destination IP/port pairs is one. There is no upper-bound.

- The maximum number of packets exchanged is four packets from source to destination and two packets from destination to source, so no data has been exchanged.

Separate from, but related to, port scanning is the mapping of a network space, which consists of scanning multiple destination IP/port pairs. This can again be performed for legitimate purposes, such as by a network administrator, however it is more often performed by adversaries in an effort to pinpoint potential vulnerabilities in a target network. Network mapping can be thought of as a superset of port scans, with the following essential characteristics:

- The purpose of a network map is to determine if a particular service or set of services is available on a target network or a subset of that network
- Multiple destination IP/port pairs are required for a network mapping. These pairs must represent a logical progression. That is, they should represent a horizontal scan (one port across multiple IPs) or block scan (multiple ports across multiple IPs). If the destination IPs and ports have been chosen randomly, then the purpose of a network map is not met.
- The minimum number of destination IPs is a function of the size of the target network.
- The minimum number of destination ports is one (the scanner could be performing a horizontal scan of just web servers on the target network).
- The maximum number of packets exchanged between the source and any one destination IP/port pair is six (four from source to destination, and two from destination to source). Otherwise data has been exchanged, and this is not the purpose of network mapping.

In this thesis, we will address port mapping, however, with the exception of the current chapter, we will use the terms port scanning and port mapping interchangeably, and assign the same meaning to both terms.

Techniques have been developed to hide that a port scan is occurring, although these techniques are not used by all adversaries. For example, there are some adversaries who do not care about being stealthy, such as those adversaries logged by the HoneyNet Project[66]. This project has noticed a change in behaviour, where adversaries will simply deploy their exploit against a system without first checking to see if the target service is available. However, the group of adversaries studied by this project are primarily newbie/toolkit level[73]. However, as proper reconnaissance improves the chances of success[51][70], it is suspected that more skillful adversaries will employ techniques to hide their scanning and mapping activities.

Therefore, one of the requirements of port scan software is the ability to hide that a port scan is occurring. Software has been developed that allows a large amount of control over the packets being generated by the port scanner. The majority of port scanners employ

TCP, as although it is possible to use UDP and ICMP, not as much information can be gathered from them and the results are not as reliable[19]. Assuming TCP port scanning, the following fields can be (and often are) modified:

- source IP
- source port
- sequence number
- acknowledgment number
- control bits — urgent, acknowledgment, push

Additionally, the destination IP and port can be modified, however this is to gather information about other targets and services, not to hide that a scan is occurring. Appendix A describes how port scans are typically hidden from logging applications.

In addition to modifying the fields in the TCP headers, software such as nmap allows a user to randomize its output. For example, if the goal was to scan a range of IPs, then nmap provides the capability to randomize the order in which the IPs are scanned, as well as randomizing the order of the ports scanned. Additionally, a time delay can be inserted between scans (up to five minutes between each scan, using nmap's "paranoid" mode) in order to prevent detection. The time delay between scans can also be randomized, so that any logging software will not detect that scans are occurring at regular intervals. Finally, nmap provides the capability to specify four IPs, other than the current source and destinations, that will be used as source IPs. Thus the target will receive five packets for each target IP/port combination, yet only one packet will contain the actual source. This can be used to obfuscate the actual source of the scan.

Another technique for hiding a port scan that is employed when multiple destination IP/port pairs are to be scanned is to modify the time distribution of the scans. If all pairs are scanned within a short amount of time, they will likely be detected by any logging occurring on the target site. However, most intrusion detection systems only log port scans if some minimum number of scans were detected within a particular time period. As a result, adversaries can avoid being detected by inserting a long enough time delay between each individual scan. Software such as nmap provides six different timing policies, with the most conservative allowing five minutes between each scan.

What is actually logged when a port scan is performed depends entirely on the capabilities of the intrusion detection system being used and how it is configured. One of the most popular freeware systems is Snort, which is a rule-based system. This system can be configured to log all port scans. It detects that a port scan has occurred by performing a stateful inspection of the packets received. By default, Snort logs all "stealth" TCP packets (e.g. those with malformed headers, such as FIN scans). It also logs a SYN scan as having occurred if SYN packets were sent to four different ports in less than three seconds, although it can be configured to log any SYN scan based on a stateful inspection of the traffic flow.

A more recent development hides the true source of a port scan by distributing the scans among multiple sources. While some software is available in the wild (e.g. dps, Rivat, dscan), it is not high quality software, nor does it allow a user a large amount of control. It is therefore expected that highly skilled adversaries have written their own versions of distributed port scanners, and have not released the source code. It is assumed that any such custom software would have the same capabilities as nmap at a minimum.

While not a requirement, it is anticipated that any distributed port scanner will employ the same client on each source for ease of control and correlation of results. This confers a certain amount of ease of use, so that the adversary does not need to manually collate the information gathered from different software. Given this assumption, the following variables may end up having the same values in each source:

- sequence number
- acknowledgment number
- source port
- control bits — urgent, acknowledgment, push

Other patterns might also emerge, such as the scans from each source having the same time delay between scans, or each client might use the same sequence of source ports.

An adversary who is performing a port scan, and desires not to raise any alarms of his presence, can do so, however it requires that an effort be made to ensure complete randomness in his approach. One possibility is to use a different source for each target IP/port combination. Another possibility is to use a very large number of sources and ensure that the sources when combined do not provide a contiguous set of IPs but that there is some randomly-sized gap between each set of targets.

In order to recognize a distributed port scan, given a database of information that are already suspected to represent single-source scans, it is anticipated that the following properties will exist:

- Each source IP will scan more than one target IP/port combination, and will likely scan multiple ports on multiple IPs.
- The destination ports/IPs, if put together from the multiple sources, will form a roughly contiguous space with little or no overlap between sources.
- Each source will not necessarily scan a contiguous space, but may scan apparently random IPs/ports, where the pattern will not emerge until multiple sources are combined. However, it may still be the case that each source will scan a contiguous space. Also, single source port scans usually scan a contiguous space, even if the IPs/ports are scanned in a random order. So, if a source scans a random set of IPs/ports, this may be an indication that a distributed port scan is occurring, which each source has been assigned a random set of IPs/ports to scan.

- The source ports used by each source for a scan may be the same across sources or may follow the same pattern. For example, nmap chooses a random source port from which to start scanning, and each subsequent scan either uses the same source port (depending on volume of scanning), or increments the source port by one. As each source will be using the same client, they will each employ the same method for choosing the source port. However, it is still possible that the method the client code uses for choosing a source port is to generate one randomly for each scan.

- Each source will scan the target within the same time window, as one of the benefits of performing a distributed port scan is that the same information can be collected that would be generated by a single source scan, but in less time. Thus it is anticipated that a single master will contact all clients, so that the clients all perform their scan starting at approximately the same time. It should be noted that the master may have a time delay between each client contacted, and that this delay may even be randomly determined, however all scans will occur within the same time frame.

- Each packet may have some header fields with the same values. For example, the TTL may contain the same value, or the sequence numbers may be the same (as the client may always craft a packet using the same initial sequence numbers). Other possibilities include all packets having the same options and type of service. However, comparing a scan of four ports on one target IP run from two different sources, both using nmap, shows that nmap randomizes most fields. The fields that were different between the two scans include source port, TTL, ID, sequence number and window size. The fields that were the same include TOS, IpLen, DgmLen, Ack and TcpLen, however these fields provide little information.

- While nmap provides a spoof option, this will not affect the results. The purpose of the spoof option is to hide the actual source IP by generating scan packets with spoofed source IPs, so that the target does not know which source was the true source. In this instance, it is possible to detect that such a spoof is occurring — the target will see the same order of source IPs for each IP/port scanned, each with the same header information (e.g. the same sequence number, TTL, etc), indicating that the spoof option is being deployed. In this case, only one of the sources needs to be employed in determining if a distributed port scan is occurring.

There are a number of programs, both legitimate and not, whose activities could be perceived as scanning. We will address five of the most common such activities, and indicate if they should be detected as scanning.

The behaviour of various worms, such as CodeRedI and CodeRedII, could be interpreted as port scanning. Both worms generate a random list of IP addresses to infect, although CodeRedII has some restrictions on this that bias it to generating IPs primarily in the same class A network[55]. To infect a machine, a TCP SYN packet is first sent to port 80 on the target IP to establish a connection. If a web server is present, then the connection is established and the payload delivered. If a web server is not present, then a TCP RST-ACK

is sent back to the source and the connection is closed. As it is likely that a large portion of the IPs targeted by the worm will not have a web server, this will appear in the router logs as a large number of SYN probes, while those targets running web servers will look more like legitimate traffic. In fact, exactly this behaviour has been identified by Moore et al. in [55], where a graph is presented showing the normal background level of SYN probes (between 100 to 600 hosts probed per two hour period) and the sudden increase with the spread of CodeRedI (an additional 400 to 800 hosts probed per two hour period). Thus worms exhibit characteristics of port scanning (determining if a particular service is available on a particular target). However, they do not exhibit characteristics of port mapping. For example, a payload is delivered if a target has the service available, which is not characteristic of port mapping. Worms will therefore need to be recognized in the data, but will not be considered as contributing to a distributed port mapping effort.

Web robots (also known as crawlers or spiders) are automated programs that index the web, and are used by many search engines. While at first glance it would appear that the activity of web crawlers will appear as a scan, this is not actually the case. Web crawlers do not scan a network looking for web servers to index, but rather start from a seed set of web pages, and only follow links to other pages[44]. As a result, there is always a data transfer from a legitimate connection, and there are no connection attempts to computers that do not have web servers running. Thus there will be no port scan-like activities.

As has been reported by the HoneyNet Project, it has become the habit of newbie/toolkit users (also known as script kiddies) to attempt exploits immediately, without first performing a port scan to determine if the service is even available on the target machine[66]. This behaviour will appear similar to worm behaviour, except that exploit attempts will likely work systematically through a range of IPs, whereas a worm is more likely to probe random IP addresses. If a service is not available, exploit attempts will appear similar to a SYN probe as a SYN packet will have been sent, resulting in a RST-ACK sent back to the source, resulting in a new target being probed. If, however, a service is available, the exploit will be attempted immediately, resulting in a connection complete with payload that will not be picked up as a port scan. While the initial probes are certainly port scans, this is obviously not a port mapping attempt, and will therefore be discarded from being considered as part of a distributed port mapping attempt.

Backscatter is another phenomenon that could be interpreted as a port scan. Backscatter refers to the network packets sent in response to a denial of service (DoS) attack where the source IP has been spoofed[56]. That is, a source with IP X will send packets to a target T, but will put in the source IP field a spoofed source of IP Y. Target T will then respond to IP Y, which discards the packet as it is a reply to a non-existent request. The most common protocols used in DoS attacks are TCP and ICMP, so the responses sent tend to be TCP SYN/ACK, TCP RST, ICMP echo replies, ICMP port unreachable, etc. As the purpose of a port scan is to gather information about a target service, packets with these particular flag settings and characteristics would not be used, as they would not generate a reply that provides service information. Thus backscatter is not a port scan. However, if

only the packet counts are available, with no information provided on the flag values, then a program will likely not be able to distinguish between backscatter and port scans. Even using source port will not provide any distinction, as DoS attacks often choose random ports above 1024 (although ports 80, 23 and 113 are also common).

Another phenomenon that can be interpreted as a port scan is that of operating system (OS) fingerprinting. Traditional methods of determining the operating system on a target were to make a telnet or ftp connection, as the banners on a site would state the operating system and version. However, many system administrators now turn these banners off, so this method is no longer effective. OS fingerprinting is now achieved through examining the packets returned by a target system after it has been sent packets crafted in a specific manner. Due to differences in TCP/IP stack implementations across the different operating systems, it is possible to determine the remote operating system, in many cases down to the version level. Example behaviours that are different across systems include the value of the initial sequence number returned when a connection is being created, the value of the acknowledge number when a connection can not be created, if IP fragmentation is allowed and how it is handled, and the window sizes on returned packets[20]. In order to obtain information from the target system that can be used in determining its operating system, packets must initially be sent to it. This could look very similar to a port scan, as complete connections will not be made and a number of packets, possibly with unusual flag combinations and other information, will be used. Although this form of information gathering does not explicitly determine if various services are available, it does determine basic information about the target site using similar tactics to port scanning, and so is defined in this thesis as being a port scan. When OS fingerprinting is deployed against many machines, it also falls under the realm of port mapping. It should be possible to determine if an adversary is performing OS fingerprinting based on the packets sent, if details of those packets are available.

Appendix A

Port Scans

Port scans have been classified by the TCP flag settings they use. The most commonly employed scans are:[19]

- TCP connect() scanning,
- TCP SYN (half open) scanning,
- TCP FIN (stealth) scanning,
- SYN/FIN scanning using IP fragments,
- UDP raw ICMP port unreachable scanning,
- XMas scans, Null scans, etc.

A TCP connect() scan can be performed by anyone on any machine. A standard connection is made to a port which indicates that the port is open by establishing a connection, or closed if a connection can not be established. Thus a complete TCP handshake is performed. The disadvantage to this method is that sites often log the connections made to various services offered. A number of packets are sent in this case — a SYN packet to initiate the connection, a SYN-ACK to to acknowledge the connection, an ACK to establish the connection, some data from the scanned site to the scanner, a FIN to end the connection, and a RST to close it. At least three packets are sent from the scanner, and at least three packets are received by the scanner. However, despite the logging performed, this type of scan is sometimes performed, in particular by newbies and script kiddies, in order to determine the particular version of the service being offered at the port of interest.

A TCP SYN scan begins a connection, but never completes it, and is therefore also known as a half open scan. The scanner sends a SYN packet to initiate a connection. If the port is open, a SYN-ACK is received, else a RST-ACK is received that both acknowledges the connection attempt and closes it as the port is not accepting connections. If a SYN-ACK is received, the scanner sends an RST packet to close the connection without having completed the handshake. The advantage to this method is that the service probed does not log the access as the connection was never completed. Only two packets are sent from the scanner, and only one packet received.

In some cases, firewalls are configured to block SYN packets to closed ports, thus preventing the SYN-ACK from being returned to the scanner. In these cases a TCP FIN scan, also known as a stealth scan, can be used as firewalls are often configured to allow these to pass unconditionally. When a FIN packet is sent to a port that is closed, it will result in a RST packet. However, if the port is open, no response will be made at all. This attack is unreliable in that it is based upon an incorrect implementation of the TCP protocol, which is operating system dependent. In these instances, only one packet is sent from the scanner, and at most one packet is received by the scanner.

Several variations on the FIN scan exist, the most common of which are Xmas scans and Null scans. In the case of Xmas scans, the URG, PSH and FIN flags are set, along with a sequence number of zero. In the case of Null scans, none of the flags are set, and the sequence number is zero. In all these cases, the aim is to create a packet that will not be stopped by a firewall. Again, only one packet is sent from the scanner, and at most one packet is received by the scanner.[7]

It is also possible to by-pass firewalls by using IP fragmentation. In this case, the TCP packet is spread across multiple IP packets, resulting in very small packets, on the order of 24 or 36 bytes. Firewalls are often not configured to reassemble the TCP packets due to performance considerations.

In some situations, it may be preferable to use UDP packets over TCP packets (although this is rare). In these cases, a UDP packet is sent to a destination. If the destination port is closed, a packet will be returned with the RST flag set. If the destination port is open, there will be no response. This is not a reliable scanning method as UDP transport is unreliable and packets can be dropped from the network. In these cases, the result will look like an open port, when in fact the packet never reached the destination. In addition, such a scan can be slow as some kernels (such as some variants of Linux) place limits on the number of responses to UDP packets made within a certain period of time.

Bibliography

[1] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3):186 – 205, 2000.

[2] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy, 2000. Technical Report No 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden.

[3] Donald L. Brinkley and Roger R. Schell. What is there to worry about? an introduction to the computer security problem. *Information Security: An Integrated Collection of Essays*, pages 11 – 39, 1995.

[4] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121 – 167, 1998.

[5] CERT Coordination Center. Denial of service attacks. http://www.cert.org/tech_tips/denial_of_service.html, 2001. Last visited: 28 September 2002.

[6] National Counterintelligence Center. Annual report to congress on foreign economic collection and industrial espionage, 2000. Last visited: 15 August 2002.

[7] Laura Chappell. You're being watched: Cyber-crime scans. *Novell Connection: The Magazine for Networking Professionals*, 2001. Last visited: 18 February 2002.

[8] Fred Cohen. Simulating cyber attacks, defenses, and consequences, 1999. http://www.all.net/journal/ntb/simulate/simulate.html. Last visited: 5 April 2002.

[9] SAINT Corporation. Saint :: Scanning engine. http://www.wwdsi.com/products/saint_engine.html. Last visited: 27 August 2002.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273 – 297, 1995.

[11] D.E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222 – 232, 1987.

[12] Sven Dietrich, Neil Long, and David Dittrich. Analyzing distributed denial of service tools: The shaft case. In *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, pages 329 – 339. USENIX Association, 2000. December 3-8, 2000. New Orleans, LA.

[13] Dave Dittrich. Ddos – is there really a threat?, 2000.

[14] David Dittrich. The "stacheldraht" distributed denial of service attack tool, 1999. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt. Last visited: 27 September 2002.

[15] S.B. Thrun et al. The monk's problems: A performance comparison of different learning algorithms, 1991. Technical Report. Carnegie Mellon University CMU-CS-91-197.

[16] Stephanie Forrest, Lawrence Allen, Alan S. Perelson, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 202 – 212, 1994.

[17] Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88 – 96, 1997.

[18] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120 – 128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[19] Fyodor. The art of port scanning. *Phrack Magazine*, 7(51), 1997. Article 11 of 17.

[20] Fyodor. Remote os detection via tcp/ip stack fingerprinting. *Phrack Magazine*, 8(54), 1998. Article 9 or 12.

[21] Tal Grossman and Avishai Wool. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101(1):81 – 92, 1997.

[22] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151 – 180, 1998.

[23] Albert J. Hoglund, Kimmo Hatonen, and Antti S. Sorvari. A computer host-based user anomaly detection system using the self-organizing map. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 5, pages 411 – 416, 2000.

[24] horizon. Defeating sniffers and intrusion detection systems. *Phrack Magazine*, 54(10), 1998. http://www.phrack.org/show.php?p=54&a=10. Last visited: 25 March 2002.

[25] Kevin J. Houle and George M. Weaver. Trends in denial of service attack technology, 2001.

[26] http://packetstorm.decepticons.org/distributed/, 2000. Last visited: 7 July 2002.

[27] http://securityresponse.symantec.com/avcenter/threat.severity.html, 2002. Last visited: 19 April 2002.

[28] http://www.nessus.org/. Nessus, 2002. Last visited: 27 August 2002.

[29] http://www.nmap.org/, 2002. Last visited: 20 March 2002.

[30] http://www.silicondefense.com/, 2002. Last visited: 13 December 2002.

[31] http://www.snort.org/, 2002. Last visited: 28 March 2002.

[32] http://www.u-n f.com/dscan.html, 2002. Last visited: 2 July 2002.

[33] Ming-Yuh Huang, Robert J. Jasper, and Thomas M. Wicks. A large scale distributed intrusion detection framework based on attack strategy analysis. *Computer Networks*, 31(23):2465 – 2475, 1999.

[34] Richard O. Hundley and Robert H. Anderson. Emerging challenge: security and safety in cyberspace. *IEEE Technology and Society Magazine*, 14(4):19 – 28, 1995.

[35] hybrid. Distributed information gathering. *Phrack Magazine*, 55(9), 1999.

[36] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264 – 323, 1999.

[37] Erland Jonsson and Tomas Olovsson. An empirical model of the security intrusion process. In *Proceedings of the Eleventh Annual Conference on Computer Assurance, Systems Integrity, and Software Safety.*, pages 176 – 186, 1996.

[38] Erland Jonsson and Tomas Olovsson. A quantitative model of the security intrusion process based on attacker behavior. *IEEE Transactions on Software Engineering*, 23(4):235 – 245, 1997.

[39] Curtis A. Carver Jr., John M.D. Hill, John R. Surdu, and Udo W. Pooch. A methodology for using intelligent agents to provide automated intrusion response. In *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, pages 110 – 116, United States Military Academy, West Point, NY, 2000. 6-7 June.

[40] Samuel Kaski. *Data Exploration Using Self-Organizing Maps.* Helsinki University of Technology, 1997. PhD Thesis.

[41] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464 – 1480, 1990.

[42] Jan Komorowski, Zdzislaw Pawlak, Lech Polkowski, and Andrzej Skowron. Rough sets: A tutorial, 1998.

[43] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422 – 469, 2000.

[44] Martijn Koster. The web robots faq ... http://www.robotstxt.org/wc/faq.html. Last visitied: 15 November 2002.

[45] Bill Landreth. *Out of the Inner Circle: A Hacker's Guide to Computer Security.* Microsoft Press, 1985.

[46] W. Lee, W.J. Stolfo, and K.W. Mok. A data mining framework for adaptive intrusion detection. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1999.

[47] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227 – 261, 2001.

[48] Li Li. Fuzzy support vector machines for multiclass classification. Last visited: 11 September 2002.

[49] Ulf Lindqvist and Erland Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 154 – 163, 1997.

[50] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition*, volume 2, pages 12 – 26, 2000.

[51] Lieutenant Colonel Thomas C. McCarthy. U.s. army heavy brigade reconnaissance during offensive operations, 1994. Monograph.

[52] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by linoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262 – 294, 2000.

[53] Jelena Mirkovic, Janice Martin, and Peter Reiher. A taxonomy of ddos attacks and ddos defense mechanisms, 2002. University of California, Los Angeles, Computer Science Department Technical Report No. 020018.

[54] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Publishing, Boston, Massachusetts, 1997.

[55] David Moore, Colleen Shannon, and k claffy. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd Internet Measurement Workshop*, Marseille, France, 2002. Nov. 6–8, 2002.

[56] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. In *10th USENIX Security Symposium*, 2001.

[57] Code XDC3 Naval Surface Warfare Center, Dahlgren Division. Shadow indications technical analysis: Coordinated attacks and probes, 1998. http://www.nswc.navy.mil/ISSEC/CID/co-ordinated_analysis.txt. Last visited: 23 March 2002.

[58] Office of the National Counterintelligence Executive. Annual report to congress on foreign economic collection and industrial espionage 2001, 2001.

[59] Computer Emergency Response Team/Co ordination Center. Cert/cc statistics 1988-2001. Last visited: 5 February 2002, 2002.

[60] Donn B. Parker. *Fighting Computer Crime*. Wiley Computer Publishing, New York, 1998.

[61] Zdzislaw Pawlak, Jerzy Grzymala-Busse, Roman Slowinski, and Wojciech Ziarko. Rough sets. *Communications of the ACM*, 38(11):89 – 95, 1995.

[62] Richard Power. *Current and Future Danger: A CSI Primer on Computer Crime and Information Warfare*. Computer Security Institute, 2000. 4th Edition.

[63] Richard Power. 2001 csi/fbi computer crime and security survey. *Computer Security: Issues and Trends*, 7(1):1 – 18, 2001.

[64] Richard Power. 2002 csi/fbi computer crime and security survey. *Computer Security Issues and Trends*, 8(1):1 – 22, 2002.

[65] Honeynet Project. Know your enemy: A forensic analysis, 2000. http://project.honeynet.org/papers/forensics/. Last visited: 28 March 2002.

[66] The Honeynet Project. *Know Your Enemy*. Addison-Wesley, 2002.

[67] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection, 1998. http://www.securityfocus.com/data/library/ids.ps. Last visited: 23 March 2002.

[68] Nicholas Puketza, Mandy Chung, Ronald A. Olsson, and Biswanath Mukherjee. A software platform for testing intrusion detection systems. *IEEE Software*, 14(5):43 – 51, 1997.

[69] Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719 – 729, 1996.

[70] RAND. Quantifying the battlefield. http://www.rand.org/publications/RB/RB3014/RB3014.html, 1999. Last visited: 10 July 2002.

[71] Eric Steven Raymond. A brief history of hackerdom. http://www.tuxedo.org/ esr/writings/hacker-history/. Last visited: 11 March 2002., 2000.

[72] Marc Rogers. Modern-day robin hood or moral disengagement: Understanding the justification for criminal computer activity. http://www.escape.ca/~mkr/moral_doc.pdf. Last visited: 12 March 2002, 1999.

[73] Marc Rogers. A new hacker taxonomy. http://www.escape.ca/~mkr/hacker_doc.pdf. Last visited: 12 March 2002., 2000.

[74] Marc Rogers. Psychological theories of crime and hacking. http://www.escape.ca/~mkr/crime_doc.pdf. Last visited: 12 March 2002., 2000.

[75] Ira Sager. First yahoo! then ebay. the net's vulnerability threatens e-commerce–and you, 2000. February 21, 2000.

[76] sasha. Distributed tools. *Phrack Magazine*, 10(56), 2000. Article 12 of 16.

[77] Ed Skoudis. *Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall, Upper Saddle River, NJ, 2002.

[78] Eugene H. Spafford. Are hacker break-ins ethical?, 1997. 2nd edition.

[79] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, Athens, Greece, 2000.

[80] Peter Stanski and Arkady Zaslavsky. Distributed and persistent mobile agents for heterogeneous personal communications systems. In *Proceedings of the Sixth International Conference on Computer Communications and Networks*, pages 256 – 261, 1997.

[81] Andrew S. Tanenbaum and Robbert Van Renesse. Distributed operating systems. *ACM Computing Surveys*, 17(4):419 – 470, 1985.

[82] D. Tsaptsinos and M.G. Bell. Medical knowledge mining using rough set theory. In *International Conference on Neural Networks and Expert Systems in Medicine and Healthcare*, Plymouth,UK, 1994.

[83] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, San Francisco, 2000.

[84] xtremist. http://online.securityfocus.com/tools/1502, 2001. Last visited: 5 January 2003.

[85] Nong Ye, Xianyang Li, Qiang Chen, Syed Masum Emran, and Mingming Xu. Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man, and Cybernetics*, 31(4):266 – 274, 2001.

[86] Dan Zhu, G. Premkumar, Xiaoning Zhang, and Chao-Hsien Chu. Data mining for network intrusion detection: A comparison of alternative methods. *Decision Sciences*, 32(4):635 – 660, 2001.