Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Christoph Göldi, Roman Hiestand

Scan Detection Based Identification of Worm-Infected Hosts

Student Thesis MA-2005-01
18th April 2005 / Winter Term 2004/05

Tutors: Roel Vandewall, Stefan Lampart (Open Systems AG)
ETH-Tutors: Arno Wagner, Thomas Dübendorfer
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

The number of new worms on the Internet increases rapidly. Worm infections cause traffic overloads in office networks and congestions of Internet links which hurt the affected companies and which cost the industry yearly several billion dollars. The goal of this thesis is the implementation of a generic worm detection tool that runs efficiently on Linux-based firewalls and Internet gateways.

This thesis provides a broad survey of all widespread past worms. The analysis and classification of worms has pointed out indicators for a worm detection. An intensive research of already known detection methods has shown that none of the proposed methods can be easily used in office networks.

A specification and implementation of a generic worm detection algorithm has been done based on the analysis of worm scan traffic. The scan detection is based on the open source intrusion detection system Bro and can be installed on a Linux computer with very little effort. Tests have shown that all worms which cause a high network load are detected in short time and with a very low false positive rate. The system runs stably and is economical in resource usage.

The developed detection method enables affected companies to quickly react to worm infections and thus helps preventing major financial losses.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Section 1.1 explains the problem statement, Section 1.2 the task description, Section 1.3 the context and Section 1.5 the chosen approach. Further, the environment is elaborated on in Section 1.4 and related work is summarized in Section 1.6.

1.1 Problem Statement

1.1.1 Computer Viruses

A lot of computer worms and viruses have been rapidly spreading all over the world in the last years. The Code Red worm infected 400'000 hosts within one day, the SQL worm infected 75'000 hosts in 10 minutes and the Sasser worm infected up to 1 million hosts since it was first seen. This is only a small selection of the omnipresent computer worms.
Computer worms and viruses can be destructive in multiple ways. The infected host is affected because:

- Information may be stolen or deleted from the host. Hardware may be used to full capacity or even start to malfunction.
- The host may try to infect other hosts and generate a lot of scan traffic. Due to the network load the host may not be able to do any operations which require network connection and therefore, can not operate appropriately anymore.

Further, the network and other hosts in the network are affected because:

- An infected host which generates lots of scan traffic may overload the network. Consequently, even uninfected hosts may be disrupted from network connection.
- The generated traffic may cause VPN tunnels or gateway machines to malfunction.
- Computer worms may be used to prepare *Distributed Denial of Service Attacks* (DDoS). These attacks may cause network outages.

Computer worms and viruses are an important issue to Internet services. They cost the industry several billion dollars each year. For the year 2003, Trend Micro Inc. has estimated costs of 55 billion dollars caused by viruses and worms [27]. Further, it is expected that the number of new worms and viruses will increase rapidly in the next years. All this shows the requirement for further research in this field.

1.1.2 Drawbacks of Conventional Anti-Virus Measures

Various tools exist which allow to search for worms and viruses in the Internet traffic or in files on servers or hosts. These tools are known as virus scanners and are looking for known patterns of malicious code. The drawback of these systems is that they cannot detect unknown worms and that they have to be regularly updated with the latest virus recognition patterns. Additionally, the time between the publication of new security flaws and the development of worms exploiting these flaws has become shorter in the last years. Consequently, it gets harder to provide security patches within reasonable time and to provide recognition patterns before a worm has significantly spread.

Due to the discussed drawbacks, new virus detection methods which are not based on recognition patterns are required. The demand for generic methods, which analyze the network traffic and detect infected hosts based on their behavior in the network, increases constantly.

1.2 Task Description

The goal of this thesis is to detect worms on the basis of their typical scan traffic. An infected host searches for vulnerable hosts in the network whereas an uninfected host does not show this behavior. Some worms search for random targets in the internal subnet or in the whole IP range by sending thousands of packets. Others try to propagate via emails that are sent to every email address which can be found on the infected host. This process of looking for hosts that are answering on certain ports is known as scanning. The traffic which is generated during such scans is characteristic and therefore, can be detected. An algorithm which detects theses traffic anomalies should be devised and specified. Further, the specification has to be prototypically implemented and tested.

1.3 Context

The master thesis is a ETH research project which has been conducted in a commercial environment. The parties involved are introduced in this section.

1.3.1 DDosVax Project

This thesis has be been carried out in the context of the DDosVax [17] project which is a joint project of ETH Zurich and SWITCH [69]. The objectives of the project are the detection of infections, the detection and analysis of massive DDoS attacks and the provision of methods and tools that support countermea-sures.

1.3.2 Open Systems AG

The thesis has been conducted at Open Systems AG. Open Systems AG is a leading provider of network security in Switzerland and manages security systems in 70 countries. The provided services run on hardware which is set up by Open Systems Engineers and which is sent to the customer locations. The status of the services running on the worldwide installed hardware is controlled centrally from Network Operations Center in Zurich and New York.

1.4 Implementation Environment

The introduction in Section 1.1 has shown the need for detecting network traffic produced by computer worms. Open Systems AG plans to introduce the tool developed during this thesis in a *Virtual Private Network* (VPN) environment which has certain characteristics. In this section we give an explanation of this environment.
Figure 1.1 shows a schema of a typical company network with three sites. Two of them are offices, called *VPN sites*, where employees are working. The third one contains security relevant infrastructure and is called *Security Hub*.
The communication between the VPN gateways is encrypted whereas the communication within the VPN sites or the security hub is not. A network in which physically dislocated networks are securely combined is called a “Virtual Private Network”.
In the following paragraphs the two different kinds of sites are explained.

Security Hub The security hub is often placed close to the headquarter of a company. The critical applications concerning security are running in a security hub.
Mostly, the security hub is connected to the Internet via a firewall. Several *Demilitarized Zones* (DMZ [18]) are behind the firewall. DNS, proxy and mail servers are placed in a DMZ. The DMZ 2 in Figure 1.1 contains a VPN gateway which will be described below.

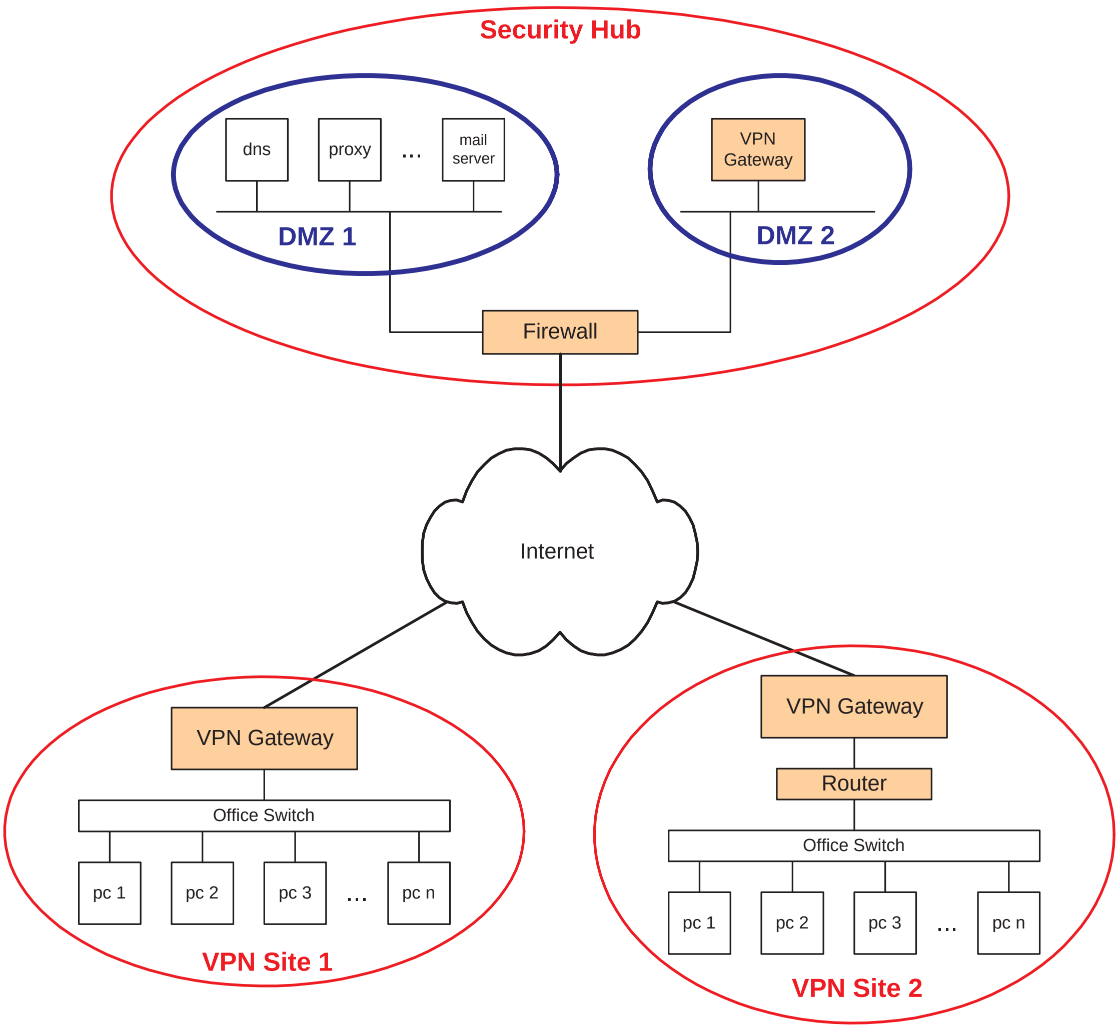

Figure 1.1: Network structure with security hub and two VPN sites

VPN Site VPN sites are small offices which can be situated anywhere in the world. These offices contain about 10 to 200 hosts. The majority are smaller ones with about 10 to 20 hosts. Usually all hosts are in the same subnet. The sites are connected to the Internet over a VPN gateway which is described in the next subsection.
The two VPN sites shown in Figure 1.1 are similar. The difference is the router which is placed on VPN Site 2 between the office switch and the VPN gateway. Both configurations are commonly used. The differences in the network traffic at the VPN gateway are *Address Resolution Protocol* (ARP [1]) packets which can be observed in VPN Site 1 whereas in VPN Site 2 the router passes only packets destined to the VPN gateway. The default route of such a router is the VPN gateway. Therefore, all traffic destined to an unknown IP address is routed to the VPN gateway in both cases.

1.4.1 VPN Gateway

The VPN gateways in Figure 1.1 represent among others the following functionality:

- *Firewall:* The firewall filters the traffic flowing through the VPN gateway. It protects the internal network from unencrypted traffic from the Internet and allows VPN connections from the internal network to all other company internal sites.
- *Routing:* The gateway routes traffic to the desired interface.
- *VPN:* The gateway encrypts traffic which has to be sent over the Internet to another VPN gateway of the company. All packets leaving the gateway are encrypted.

Each VPN gateway can communicate with all other VPN gateways of the same company.
The VPN gateway has the information about the IP ranges of all sites of the company network.
When the VPN gateway receives a packet with an unknown IP address or one which is not allowed, then the packet is rejected or dropped, dependent on the settings. On certain VPN gateways default routes are implemented. These VPN gateways send all packets with unknown destination addresses to its default route address. Default routes lead to gateways in security hubs.

Proxy Server on VPN Gateway Directly accessing the Internet, respectively a company-foreign host, is not possible from the VPN sites. For example all HTTP traffic on port 80 has to run over the proxy server which is placed in a DMZ in the security hub. There exist individual solutions with a proxy server implemented on the VPN gateway which enables direct connections between VPN gateway and external Internet hosts. This offloads the VPN network.

1.5 Approach

The master thesis has been split into several subtasks. The subtasks have been solved in the following sequence.

1.5.1 Analysis of Known Scan Mechanisms of Worms

An overview of general worm characteristics has been written in Chapter 2. Then the scan mechanisms of known worms have been studied and analyzed. The different worms have been classified according to their scan mechanism.
And finally, an existing tool has been adapted to generate worm scan traffic. Attacks have been defined based on the worm classification and have been implemented with this simulation tool. The tool has been documented in Chapter 3.

1.5.2 Analysis of Existing Scan Detection Algorithms

A broad investigation of existing detection mechanisms and algorithms has been done. They have been analyzed and summarized in Chapter 4. The most promising algorithms have been implemented and tested with simulated worm traffic and traffic from various benign programs. The results have been compared and documented.

1.5.3 Specification of a Generic Scan Detection Algorithm

First, the worm indicators have been gathered and the requirements have been summarized. Based on this information and the experiences from the previously implemented detection methods a generic algorithm has been developed and specified in Chapter 5.

1.5.4 Implementation of a Scan Detection Prototype

The specification has been implemented in a prototype. The prototype runs on a Linux based VPN gateway computer that connects an internal LAN with an external network. All traffic to external hosts (other internal sites or the Internet) has to pass the VPN gateway. The prototype monitors and analyzes the traffic of the internal network and reports worm scans.

1.5.5 Testing and Test Setup

The prototype has been extensively tested with simulated worm traffic generated by the previously developed tool. It has also been tested with real worm traffic. Additionally, tests have been done with benign programs which generate traffic similar to real worms. The tests have been done in the office environment of Open Systems AG and with traffic captured at customer sites. The tests have been documented in Chapter 6.

1.5.6 Documentation

The documentation has been written with LaTeX and contains all results. The conclusions and an outlook of further work and research can be found in Chapter 7 and the bibliography and acknowledgments in the appendix.

1.6 Related Work

1.6.1 Worms

Various studies have been done about computer worms. Manufacturers of anti-virus programs document each worm but this information mostly does not contain the required details. [68], [50], [97] and [42] provide comprehensive surveys about worm parameters and characteristics.
At the university of Wisconsin-Madison a framework for malicious workload generation has been developed [29]. This tool has been used in this thesis for worm simulations and is discussed in Chapter 3.

1.6.2 Detection Methods

Several approaches have been documented and published. Various ideas have been developed for backbone systems or for sensitive intrusion detection systems. Most of them have never been implemented and consequently, no clear statements about the functioning of these detection methods can be given. None of the existing detection methods satisfy the requirements of this thesis. The most promising ones are discussed in Chapter 4.
We have found two established intrusion detection frameworks. The first one is Snort [64, 36] which is based on predefined signatures. The second one is Bro [6, 99, 100, 101] which is based on collecting network events. Bro provides its own scripting language and therefore can be used very flexible.

Chapter 2

Worm Characteristics

A large number of different worms have been observed in the past. In this chapter we first discuss computer worms in general, focusing on those aspects that are important for the considerations in this thesis. Section 2.3 gives a summary of the worms with the biggest impact in the past, some of which are discussed in detail. In Section 2.4 we classify the worms on the basis of the traffic they generate.

2.1 Definition

There exist many different definitions of a computer worm. One reason for this dilemma is the large number of worm variants. In this thesis we understand a computer worm as defined in [11]:

> A *computer worm* is a self-replicating computer program, similar to a computer virus. A virus attaches itself to, and becomes part of, another executable program; however, a worm is self-contained and does not need to be part of another program to propagate itself. They are often designed to exploit the file transmission capabilities found on many computers.

The worm spreading can be split into an infection and a propagation phase. During the infection phase the worm scans for vulnerable hosts and exploits the vulnerability to prepare the propagation phase. The propagation phase is used to transmit the worm code to the target hosts. Worms exist for which this segmentation can not be observed (for example the SQL Slammer described in Subsection 2.3.4). This thesis concentrates on detecting worms scanning for vulnerable hosts and therefore focuses on the scanning part of the infection phase.

2.2 General Characteristics

In this section we describe the most important aspects of a worm and explain the mechanisms worms use.

2.2.1 Transport Protocols

In the past, all widespread worms were based on one of the transport protocols UDP or TCP[1]. There is a significant difference between these two protocols.
The *User Datagram Protocol* (UDP, RFC 768 [58]) is a minimal message-oriented transport layer protocol. It provides no guarantees for message delivery and the sender of the packet retains no information about transmitted packets. UDP adds only application multiplexing and data checksumming on top of an IP datagram. Hence a sender does not need to save any information while sending packets. An infected host can send as many packets as the network can handle. In other words a worm that uses UDP is bandwidth limited.
The *Transmission Control Protocol* (TCP, RFC 793 [59]) is a connection-oriented, reliable delivery transport protocol. The handling of each open TCP connection consumes CPU and memory resources on the

[1]Other protocols were used, but only UDP and TCP worms were able to infect a large number of hosts in a short time. For example, NetBIOS over Ethernet, which was often used in LANs with up to 200 hosts, was never an option for a successful worm. In addition NetBIOS over Ethernet is depreciated and will not be supported by future Windows versions. It will be replaced with NetBIOS over TCP/IP [47, 48].

sending host. Therefore, only a limited number of TCP connections from the infected host can be active at any given time. A TCP connection does not release its resources until one of the hosts terminates the connection or a timeout takes place. A typical scanning mechanism often tries to connect to inexistent hosts causing a lot of timeouts. Considering this, an infected host spends most of the time waiting for timeouts and therefore, the scanning speed is limited. The length of these timeouts is a critical factor concerning the spreading speed of a worm. A timeout takes much longer than transmitting even a large worm over a slow connection. Successful worms have used a very well adjusted timeout.
The number of parallel connections and the length of the timeout are the relevant factors with respect to the propagation speed of a worm [97]. Both factors apply when using TCP. UDP is only limited by the available bandwidth . In case of a UDP worm one has to consider *IP fragmentation*, because UDP does not implement a mechanism which preserves the order of the transmitted packets.

2.2.2 Scanning Mechanisms

Past worms have used various scanning algorithms to find their victims. A fast and successful infection of the vulnerable hosts is strongly dependent on the scanning strategy. Below, a list of observed and theoretical possible methods can be found. Comprehensive surveys of these mechanisms are also presented in [68, 50, 97].

Random Scanning *Random Scanning* is the most primitive mechanism when looking for vulnerable IPs. Random Scanning means that the sequence of scanned IPs is random. Nevertheless combined with automated activation of the worm where no user interaction is needed, this mechanism can be very successful in spreading a worm to its targets. Past worms using this technique were remarkably successful and have spread very quickly. The success of a worm depends highly on the quality of the random process. An important factor is also the random seed [2] as Subsection 2.3.5 shows. Past worms have sometimes used a bad random generator which has produced deterministic numbers and has prevented a fast spreading.

Sequential Scanning The *sequential scanning* of IPs is normally a less successful technique than a random scan. However under certain conditions it will be as fast or faster as random scanning but this chance is relatively small. This could be the case, when the worm scans its own subnet and when this subnet contains a lot of vulnerable hosts. The scan should manage to hit one or more networks with many vulnerable hosts early if it wants to spread reasonably fast. As an example the Blaster worm (Subsection 2.3.3) uses this mechanism combined with random IP selection.

Local Preference The *local preference mechanism* is also known as *localized scanning*. The main idea is to first scan the machines in the neighbourhood. This makes sense concerning that hosts in the same subnetworks are often geographically and network-topologically close, resulting in faster and more reliable communication, which will speed up both the probing and the infection itself. Often there is also a firewall which prevents to infect hosts belonging to another subnet. The local preference mechanism regards this aspect and accordingly improves the results in this case. Code Red II (Subsection 2.3.5) and other worms in the past have shown that this mechanism can improve the efficiency of a random scan notably.

Permutation Scanning A limitation to very fast infection is the general inefficiency of random scanning: many addresses are probed multiple times from different infected hosts. *Permutation scanning* solves this problem by assuming that a worm can detect that a particular target is already infected. In a permutation scan, all worms share a common pseudo random permutation of the IP address space. Any infected machines start scanning just after their way through the permutation, looking for vulnerable machines. Whenever the worm sees an already infected machine, it chooses a new, random start point and proceeds from there.

Subdividing In case of the *subdividing mechanism*, each worm creates and spreads children of itself that search only a part of the address space. This method subdivides the address space and scans the resulting spaces in parallel. This seems to be much more efficient.Nevertheless this mechanism seems to offer almost no speed-up like tests in [97] showed.

[2]The *seed* is the starting point of a random number generator. A *static seed* causes a random number generator to output the same sequence of numbers each time the generator is invoked, although the numbers themselves are random in that they have no predictable relationship to each other. A *random seed* uses an unpredictable starting point, so it generates a random sequence of random numbers, rather than a predictable series of random numbers. [7]

Hit-lists Different possibilities are known for creating a *target list* before the spreading of a worm will take place. Having a preferably long list of vulnerable hosts at the launch of a worm turns out as a very big advantage, because the infection of a high number of hosts in a short time is one of the most important parts of a successful and wide spreading worm.
A *pre-generated target list* could be obtained in advance by the attacker. Such a list is also called a *hit-list*. A small hit-list could accelerate a scanning worm, while a complete hit-list creates a *flash worm*. No hit-list based worms have been seen in the wild yet.
Hit-lists can be obtained using different techniques. The most important are the following:

- A *stealthy scan* is easy to achieve. Port scans are so common and widely ignored that even a fast scan of the entire Internet would go unnoticed. However, for attackers wishing to be especially careful, a randomized stealthy scan over several months would be even more clandestine.
- A *distributed scan* would be another possibility to collect the IP addresses of favored victims. A few dozen to a few thousand already-compromised “zombies” are used to scan the Internet.
- *DNS searches* can be used for finding the IP addresses of mail servers or web servers. A list of domains can be assembled by using widely available spam mail lists or something similar.
- *Web searches*, which use search engines like Google, can also lead to a list of vulnerable web servers or email addresses.

Topological Worms There exist worms that use so called *internal target lists*. For example, the Morris worm (Subsection 2.3.7) uses topological techniques including the Network Yellow Pages, /etc/hosts and other sources to find new victims, whereas email worms often search for victims in the internal email address books.
Topological worms are difficult to detect because they know the network very well and do not have to use excessive scanning algorithms which obviously does not result in a lot of abnormal traffic.

Passive Worms A *passive worm* does not seek out victim machines. Instead a passive worm waits until potential victim hosts contact an infected host or relies on user behavior to discover new targets. Although potentially slow, passive worms produce hardly any traffic during target discovery, which makes them highly stealthy.

2.2.3 Propagation Mechanisms

The means by which propagation occurs can also affect the spreading speed and stealthiness of a worm. A worm can either actively spread itself from machine to machine, or it can be carried along as part of normal communication.
A *self-carried* worm actively transmits itself as part of the infection process. This mechanism is commonly employed in self- activating worms.
Some worms, such as Blaster (Subsection 2.3.3) require a *second channel* to complete the infection. For example the exploit uses RPC and the victim machine connects back using TFTP to download the worm body.
An *embedded* worm sends itself along as part of a normal communication channel, either appending to or replacing normal messages. As a result, the propagation does not appear as anomalous when viewed as a pattern of communication [50].

2.2.4 Worm Size

Worm size is an important aspect related to the propagation over the network. The bandwidth used for spreading a worm primarily depends on the size of the worm. A very small worm can be sent in only one UDP packet which has the benefits explained before in Subsection 2.2.1. The Microsoft SQL Server Sapphire Worm described in Subsection 2.3.4 is a good example of this technique.

2.2.5 Vulnerabilities

In this thesis we do not cover the different *vulnerabilities* worms capitalize on. However it is important to know which possible scanning and propagation mechanisms a certain vulnerability do allow. In most cases transport protocol and port number are given by the vulnerability a worm uses. Therefore, a security hole in a often and widely used application represents a very high risk of being exploited by a worm.

In addition such a worm is very difficult to detect because its traffic can almost not be distinguished from normal traffic. Obviously, a vulnerability in popular software leads also to a larger infectable population and makes the propagation of a worm faster as more vulnerable hosts are found more easily.

2.2.6 Multivector Worms

Another strategy to maximize the number of vulnerable target hosts is to exploit several vulnerabilities at the same time. A worm using this technique is called a *multivector* or *hybrid worm*. Nimda, for example, spreads itself using at least five different methods (Subsection 2.3.6).

2.2.7 Application Layer Worms

A trend towards worms which spread using the application layer can be noticed. The use of existent infrastructures and software capabilities is an obvious reason for these topological worms. In addition, these worms will gain importance with the introduction of IPv6 when a random scan of the IP address space will not be a practical possibility any more.

Email Worms Most of today's worms are *email worms*. Most email worms depend on the user executing an attachment of an email, in contrast to Code Red and other worms using vulnerabilities in the operating system or other programs. Consequently, email worms need human interaction and therefore spread slower than comparable worms. The ILOVEYOU worm is a famous email worm and probably the best example of how email worms spread by using a clever email subject and text.
Worms which spread via email have a totally different spreading mechanism than all the other worms and therefore can be seen as a separate class of worms. Most worms do not have any network scanning mechanisms for finding their victims. The targets are mainly discovered by searching for local information on the own host like email addresses in local address books or in the browser cache. The consequence is that there is no remarkable scanning traffic visible on the network. A common anomaly in network traffic is an increase of unusual SMTP connections and a lot of DNS MX requests.

P2P Worms Peer-to-Peer (P2P [53]) worms exploit the special facilities of a P2P network. For example, they can easily find other vulnerable hosts, because each host knows the IP addresses of a significant number of hosts in the P2P network. Even the request for more hosts in the P2P network looks like a standard operation of the P2P system.
A second problem is the visibility of the traffic a worm produces. A P2P file-sharing network generates a lot of traffic itself and therefore, a worm could easily hide in the conventional P2P traffic [3].

Instant Messaging Worms Symantec has reported over 20 different worms that use instant messaging to spread [46]. The worms profit by the topological network which consists of millions of users who daily use their instant messenger clients. Additionally, the clients often include mechanisms to bypass firewalls.

2.3 Survey of Known Worms

This section gives an overview of known worms. Not all past worms have been taken into consideration, but only the most important ones[3]. Table 2.1 shows the twenty worms which have been investigated. The selection criteria for this list were the number of infected hosts, the observed network traffic during the spread of these worms and the ratings of various anti-virus software producers. The detection of worm-infected host should be based on the network traffic the worms generate. Therefore, worms which produce a lot of network traffic while spreading are relevant within this thesis, whereas passive worms and P2P worms (Subsection 2.2.2) will not be considered further.
The list contains twenty worms, whereof 13 are email worms and two are hybrid worms which use email and other methods to spread.
Most worms exist in variants which behave mainly similar. These variants are based on the same original worm code and are called a *worm family*. For our investigation we have chosen the first variant which has been successful in terms of infecting many hosts. For the Netsky worm two variants have been observed to account for the difference in propagation mechanisms.

[3]Symantec has published a total of 68'255 detected viruses (including worms and trojan horses) [70].

Name	Email Worm	Propagation Mechanisms	Size [Byte]	Protocol and Port	Outbreak Date	# Infected Hosts
W32.Sasser.B.Worm [83]	No	Second channel	15’872	MS-DS 445 FTP 5554 listening on 9996	01.05.2004	500’000 - 1’000’000 [61]
W32.Welchia.A.Worm [87]	No	Second channel	10’240	RPC DCOM 135 WebDav 80	18.08.2003	> 1000
W32.Blaster.A.Worm [75]	No	Second channel	6’176	RPC DCOM 135 TFTP 69 listening on 4444	11.08.2003	200’000 - 8’000’000 [95]
W32.SQLExp.Worm/ SQL Slammer Worm [86]	No	Self-carried	376	MS-SQL 1434 (UDP)	26.01.2003	75’000 [19]
Code Red Worm [71]	No	Self-carried	3’569	HTTP 80	19.07.2001	359’000 [7]
W32.Nimda.A@mm [82]	Both	Own SMTP engine/ Second channel	57’344	DNS 53 (UDP) SMTP 25, TFTP 69 HTTP 80, MS-DS 445 NETBIOS 137-139	18.09.2001	> 1000
Morris [22]	Both	Second channel	n/a	rsh 514, fingerd 79 sendmail 25	02.11.1988	> 6’000 [43]
W32.Netsky.D@mm [80]	Yes	Own SMTP engine	17’424	DNS 53 (UDP) SMTP 25	01.04.2004	> 1000
W32.Netsky.P@mm [81]	Yes	Own SMTP engine	29’568	DNS 53 (UDP) SMTP 25	21.03.2004	50-999
W32.Mydoom.A@mm [79]	Yes	Own SMTP engine	22’528	DNS 53 (UDP) SMTP 25 listening 3127-3198	26.01.2004	>1000
W32.Beagle.A@mm [74]	Yes	Own SMTP engine	15’872	DNS 53 (UDP) SMTP 25 listening on 6777	19.01.2004	> 1000
W32.Sobig.F@mm [85]	Yes	Own SMTP engine	72’000	NTP 123 (UDP) UDP 8998	19.08.2003	> 1000
W32.Bugbear@mm [76]	Yes	Own SMTP engine	50’688	DNS 53 (UDP) SMTP 25 listening on 36794	30.09.2002	> 1000
W32.Gibe@mm [77]	Yes	Own SMTP engine	122’880	DNS 53 (UDP) SMTP 25	04.05.2002	> 1000
W32.Badtrans.B@mm [73]	Yes	MAPI commands to send emails	29’020	DNS 53 (UDP) SMTP 25	24.11.2001	> 1000
W32.Sircam.Worm@mm [84]	Yes	Own SMTP engine	134’000	DNS 53 (UDP) SMTP 25	17.07.2001	> 1000
W32.Klez.E@mm [78]	Yes	Own SMTP engine	60’000	DNS 53 (UDP) SMTP 25	17.01.2001	> 1000
VBS.Loveletter (> 80 variants) [72]	Yes	MAPI commands to send emails	10’307	DNS 53 (UDP) SMTP 25, mIRC 194	05.05.2000	> 500,000 [9]
Wscript.KakWorm [89]	Yes	Embedded	4’116	SMTP 25	30.12.1999	> 1000
W97.Melissa.A [88]	Yes	MAPI commands to send emails	n/a	DNS 53 (UDP) SMTP 25	29.03.1999	> 1000

Table 2.1: Survey of the most important worms

The table contains name and type of the worm, propagation mechanism, size and protocol, outbreak date and the approximate number of infected hosts. Most of these parameters are explained in Section 2.2. The list is first sorted by worm type and secondly by outbreak date with the latest incident first.

In the subsequent paragraphs these worms are described in more detail. The focus will be on the facts which are relevant for this thesis, for example, the scanning and spreading mechanisms.

2.3.1 Sasser.B

The Sasser worm has first been discovered on 1st May, 2004. It exploits the LSASS vulnerability [41] which offers the possibility to execute arbitrary code on the target host over the Internet. The Sasser worm has infected about 500'000 to 1'000'000 hosts [61].

Selection of IP Addresses To define the next IP addresses which will be scanned the worm reads the IP address of its host. The worm ignores the following addresses [83]:

- 127.0.0.1
- 172.16.x.x - 172.31.x.x (inclusive)
- 10.x.x.x
- 192.168.x.x
- 169.254.x.x

This address is used as basis for the new IP address. With a probability of 25% the last two octets of the IP address are changed to random numbers. With a probability of 23% the last three octets or the IP address are changed to random numbers and with a probability of 52% the IP address is completely random[4].

Scanning Mechanisms The worm first connects to the chosen IP addresses on TCP port 445 to check if the remote computer is online. If a connection can be established, it sends a sequence of packets in order to retrieve the host's SMB banner which gives a hint of the Windows system version which is installed [55].
If the worm could establish this TCP connection, the worm will send shell code to the target machine, which may cause it to open a remote shell on TCP port 9996. Using the shell the victim will open a FTP connection to the attacking machine on TCP port 5554 to download the worm.
The worm tries to infect 128 hosts in parallel which results in a heavy decrease of the performance of the infected hosts.

2.3.2 Welchia.A

The Welchia worm has first been seen in the wild on 18th August, 2003. The worm uses a vulnerability of the RPC protocol (Windows RPC port 135, [60]) to spread and infect computers which is described in [40]. This vulnerability offers the possibility to execute arbitrary code on the target host over the Internet. It only infects hosts running Windows XP.
But Welchia also spreads using the WebDav vulnerability infecting hosts running Microsoft IIS 5.0. The worm removes the Blaster worm, downloads and installs windows patches, restarts the computer and installs a TFTP (Trivial File Transfer Protocol) server on all infected hosts.

Selection of IP Addresses This worm selects the victim IP addresses in two different ways. The worm uses either A.B.0.0 from the infected machine's IP (A.B.C.D) and counts up, or it will choose a random IP address. After selecting the start address, the worm counts up through a range of class B networks [10]; for example, if the worm starts at A.B.0.0, it will count up to at least A.B.255.255 [87].

Scanning Mechanisms After creating the IP addresses, the worm sends an ICMP echo request to check if the constructed IP address corresponds to an active machine on the network.
When the worm identifies an active machine, it has two possibilities. The first one is to exploit the RPC vulnerability and therefore sends the data to port 135. The second one is to send the data to TCP port 80 to use the WebDav vulnerability.
If a successful connection could be established the target host starts a remote shell and reconnects to the attacking host on a random TCP port between 666 and 765 to receive instructions. Then the worm starts the TFTP server on the attacking machine and instructs the victim machine to connect and download several files from the attacking machine. Finally the newly infected host downloads the RPC vulnerability patch.

[4]The IP addresses ignored before can also be chosen in this case.

2.3.3 Blaster.A

The Blaster worm has first been discovered on 11th August, 2003. It has infected between 200'000 and 8'000'000 hosts [95], running Windows 2000 or Windows XP operating systems. It uses the same RPC vulnerability as Welchia does to infect machines.

Selection of IP Addresses The target IP addresses are generated in the following manner. With a probability of 60% the first three bytes of the address will be chosen completely randomly, the fourth byte will be set to zero. With 40% probability it choses the first two bytes of the local address, the third byte is also taken from the local address, but if it is greater then 20, a random number from 0 to 19 is subtracted from it. Again the last byte is set to zero.
The worm will then increment the last byte of the IP address by one until it reaches 254 and will try to infect all the hosts located at these addresses [75].

Scanning Mechanisms During the initialization phase the worm writes registry entries and creates the IP addresses as described above. Afterward, it tries to set up a TCP connection on port 135 (Windows RPC port). Blaster scans blocks of 20 sequential IP addresses by sending a connection attempt to each one simultaneously [54].
After two seconds, Blaster tries to send the code exploiting the RPC vulnerability to the hosts, where a TCP connection successfully could be established.
If the code is successfully transmitted and the victim is vulnerable, it causes a command shell to be bound to port 4444 on the infected target. This shell is used to send commands to the victim machine e.g. to start a TFTP client. The transmission of the worm code is finally done using TFTP running on UDP port 69.

2.3.4 Microsoft SQL Server Sapphire

On 26th January, 2003 the SQL worm infected 75'000 hosts in ten minutes. At the beginning the size of the population of infected hosts doubled every 7.5 seconds [19]. The worm uses a vulnerability on Microsoft's SQL Server or MSDE 2000 (Microsoft SQL Server Desktop Engine) [39]. This vulnerability offers the possibility to execute arbitrary code with SYSTEM privileges. The worm consists of one single UDP packet.

Selection of IP Addresses The IP addresses are generated randomly with a random seed. The worm uses a pseudo random number generation algorithm [19]. There is no local subnet preference in the IP selection process. Consequently, large amounts of traffic result from the spreading of this worm.

Scanning Mechanisms The worm opens a socket on the attacking computer and attempts to repeatedly send itself to UDP port 1434 on the IP addresses it has generated. This worm is not limited by the number of parallel connections, as it only needs UDP to spread. Therefore, the worm observed in the wild has sent out up to 26'000 scans per second. This amount depends on the processor speed of the infected host and the bandwidth of this host. The average was around 4'000 scans per second [86].

2.3.5 Code Red Iv2

The first version of the Code Red worm (Iv1) began to infect hosts running unpatched versions of Microsoft's IIS web servers on 12th July, 2001. The worm uses a vulnerability in the Idq.dll file [38]. The first version of the worm uses a static seed for its random number generator. On 19th July, 2001 a random seed variant of the Code Red worm (Iv2) appeared and spread. This second version infected over 359'000 machines within 14 hours [7].

Selection of IP Addresses The worm creates new IP addresses randomly. Code Red version 2 uses a random seed, in contrast to Code Red version 1, which uses a static seed.

Scanning Mechanisms The worm sends a HTTP GET request on TCP port 80 to exploit the buffer overflow mentioned before. This is done by sending itself to any IP addresses on which it can find an open port 80 to connect to. It uses multiple SENDs. On successful completion of SEND it closes the socket and starts new infection processes. The worm generates 99 threads to infect hosts in parallel [56].

2.3.6 Nimda.A

On 18th September, 2001 the Nimda worm began to spread very rapidly through the Internet and persisted for months after it started. Nimda uses at least five different ways to spread [68]:

- By infecting web servers from infected client machines via active probing for a Microsoft IIS vulnerability.
- By bulk emailing itself as an attachment (based on email addresses determined from the infected machine).
- By copying itself across open network shares.
- By adding exploit code to web pages on compromised servers in order to infect clients which browse the page.
- By scanning for the back doors left open by CodeRed II and Sadmin/IIS worms.

Further the mass mailing functionality is similar to the one of the email worm Sobig.F which is described in Subsection 2.3.9.

Selection of IP Addresses Two of the before mentioned spreading mechanisms use an IP address to send themselves to other hosts. The IP addresses are generated randomly.

Scanning Mechanisms The scanning mechanisms depend on the spreading mechanisms mentioned before. In [82] the various steps are documented. It would go to fare to elaborate them in this thesis.

2.3.7 Morris

The Morris worm was one of the first distributed computer worms spreading via the Internet. It is considered the first worm virus and was certainly the first to gain significant mainstream media attention. It was launched on 2nd November, 1988 from MIT [43]. This worm was intended to be spread and did not contain any malicious code, but slows down an infected host by every new infection. The Morris worm exploited known vulnerabilities in Unix sendmail, fingerd, rsh/rexec and weak passwords. It could only infect DEC VAX machines running 4 BSD and Sun 3 systems.

Selection of IP Addresses The worm gathers information about the network interfaces and hosts to which the infected machine is connected. Based on this information, it builds lists of addresses and selects then one of them randomly.

Scanning Mechanisms As mentioned before the worm uses three techniques to propagate. They are explained in [22] in detail.

2.3.8 Netsky.D

Netsky is a mass mailing worm which has been observed in over 30 variants. The variant D was first discovered on 1st April, 2004 and is the first variant of Netsky which has infected more then 1000 hosts. This worm uses its own SMTP engine to propagate. It does not use any vulnerability. The subject, body, and attachment names vary, the attachment always has a .pif file extension.

Selection of IP Addresses The worm sends itself to email recipients and therefore does not need any IP addresses. To collect the email addresses, the worm scans files with certain file extensions[5] on drives C to Z.

Scanning Mechanisms Netsky.D sends itself (using its own SMTP engine) to each email address found. The worm uses the DNS server configured locally, if available, to perform an MX lookup for the recipient address. If the local DNS fails, it will perform the lookup from a list of hard-coded servers [80]. Therefore, firstly a DNS lookup can be observed and then the SMTP connection based on TCP.

2.3.9 Sobig.F

Sobig.F is also a mass mailing worm that was first discovered on 19th August, 2003. The behavior of this worm is similar to Netsky.D described in Subsection 2.3.8. It infects hosts with Windows operating

[5]Files with the following extensions are scanned: .dhtm, .cgi, .shtm, .msg, .oft, .sht, .dbx, .tbb, .adb, .doc, .wab, .asp, .uin, .rtf, .vbs, .html, .htm, .pl, .php, .txt, .eml

systems (all versions from Windows 95 to Windows XP).
The variant F was the most successful variant of Sobig. The first variant was discovered on 9th January, 2003. Sobig.F uses email spoofing[6].

Selection of IP Addresses Equally to the mass mailing worm Netsky, Sobig.F does not need any IP addresses. The email addresses are found in files on the hard disk with certain extensions.

Scanning Mechanisms After some initialization processing (e.g. writing registry entries on the infected hosts), the first step the worm performs is a time request to one of the 19 hard coded NTP (Network Time Protocol) servers[7] using NTP on UDP port 123. Under certain conditions the the worm tries to contact one of the 20 hard coded master server IP addresses on port 8998/UDP to start a download. Note that all these servers had been taken from the net before Sobig.F could use them.
Finally the worm starts spreading. It has its own SMTP engine and sends itself by email with varying subject, body text and attachment file to all the email addresses found.
On the network first an NTP packet, then the DNS query to get the address of the email server and finally typical SMTP traffic can be observed. In [95] the traffic of Sobig.F has is elaborated in detail.

2.4 Worm Classification

With regard to the simulation and detection a thorough analysis of characteristic worm traffic is necessary. This section gives a classification of worms based on their traffic. There are two main areas which can be distinguished and each area can be subdivided into further criteria.

- The selection of potential targets (Subsection 2.4.1)
  - Random or deterministic scanning
  - Preference for the local subnet
- The generation of scanning traffic (Subsection 2.4.2)
  - Protocol on the transport or network layer
  - Port number
  - Number of parallel connections respectively sending rate

After a worm establishes a connection to a victim host, it tries to propagate over the network by sending its code. This worm propagation phase is not considered by this classification, since, as described in Section 1.2, this thesis only covers the scanning traffic. In case of UDP, the scanning traffic includes the propagation of the worm. The following subsections describe the mentioned criteria in detail. Table 2.2 and 2.3 show to which class each worm belongs.

2.4.1 Selection of Potential Infection Victims

Worms differ in their selection of possible targets (Subsection 2.2.2). Table 2.2 gives an overview of the worms and the nature of their IP selection mechanism [42]. As mentioned in Subsection 2.2.2, there are more techniques a worm could use. However, for further considerations permutation scanning can be assumed as some sort of random scanning. A hit-list indeed has an influence on the impact of a worm, but the traffic of such a worm does not differ very much from a worm without hit-list. Therefore, permutation scanning and hit-lists are not discussed in detail any further.
Email worms behave, as mentioned above (Subsection 2.2.7), in a completely different manner. They can be seen as a separate class of worms except for the Nimda and the Morris worm which only have parts behaving like an email worm. Email worms do not need to chose any IP addresses and therefore, they are not mentioned in this subsection.

[6] *Email spoofing* is a technique where the worm randomly selects an email address it finds on the infected host. The worm uses this address as the “From” address when it performs its mass mailing routine [85].
[7] The first server it tries to contact is the NTP server of ETH Zurich. [95]

Random The column "random" in Table 2.2 tells if a random process is involved in the generation of IP addresses. The selection of IP addresses is often only partially random. Only SQL Slammer and Code Red I create completely random IP addresses. Mostly the first 8 to 16 bits are taken from the own IP address and the remaining bits are generated at random.
Some worms do a sequential scanning in combination with random scanning. They count up starting at a randomly calculated IP address.

Local Subnet As mentioned above an IP address is often created by taking the upper bits from the actual local address of the infected host. For example, if only the last 8 bits are chosen at random, this means that the scanned machines are located in the same subnet with the subnet mask 255.255.255.0. A scan of an IP in the local subnet does not pass a router at the boarder of this subnet. Therefore, only a scan of an external IP address can be observed at the outgoing link of a local subnet.

	random	local subnet
Sasser.B	x	x
Welchia.A	x	x
Blaster.A	x	x
SQL Slammer	x	
Code Red Iv2	x	
Nimda.A	x	x
Morris	x	x

Table 2.2: Selection of potential infection victims

Most worms cover multiple scanning mechanisms. Often they use some kind of probabilistic function to decide between the implemented possibilities. Not only the scanning mechanism can depend on a probabilistic function, sometimes the choice between several vulnerabilities is also done in this way.

2.4.2 Generation of Scanning Traffic

This subsection shows a classification based on the scanning traffic of worms. As mentioned in Section 1.2, the scanning traffic consists of the first packets sent by a worm when trying to establish a connection to a potential victim.
Table 2.3 gives an overview of the first packets sent by the various worms. The following paragraphs describe the different columns in detail. The Morris worm has not been considered in this overview, because its mechanisms are not relevant for today's Internet. As a representative for a typical email worm the Netsky worm has been chosen.

Protocol The network protocol used by a worm is often given by the exploited vulnerability. The observed worms use TCP and UDP over IP, except for the Welchia worm which first sends an ICMP request. Subsection 2.2.1 explains the differences between UDP and TCP.

Port The port number respectively the application layer protocol used is an important factor concerning the applied rules in the firewall. If a connection can be established or not, depends on these firewall rules. In other words a port can be open and traffic on this port can be allowed or not. Some worms like Nimda or Welchia make use of multiple vulnerabilities and therefore try to connect to different ports.

Connection With regard to the quantity of packets, it is important how a worm tries to establish connections. In TCP a clever worm would use multiple threads to open many connections and wait for many answers at a time.
UDP is not connection-oriented and therefore, a UDP worm can send as many packets as possible. The precise quantity can be given as *packets sent per second*.

	Protocol	Port	Connection
Sasser.B	TCP	445	128 in parallel
Welchia.A	ICMP	-	as many packets as possible
	TCP	80 & 135	n/a
Blaster.A	TCP	135	20 in parallel
SQL Slammer	UDP	1434	as many packets as possible
Code Red Iv2	TCP	80	99 in parallel
Nimda.A	TCP	80 & 137-139/445	n/a
Netsky.D	UDP (DNS MX)	53	n/a
	TCP (SMTP)	25	n/a

Table 2.3: Scan traffic

Chapter 3

Worm Simulation

The goal of this thesis is to detect worm traffic in company networks. To develop the detection tool the environment is required to be as realistic as possible. Therefore, the detection tool will be tested in the office network of Open Systems AG. Due to the risk that real worms could infect hosts, we will use a simulation tool to generate the scanning traffic normally caused by worms.
In Section 3.1 the tool which has been used to simulate worm traffic is explained. Section 3.2 shows which simulations have been done. In Section 3.3 the simulation environment is documented. Section 3.4 summarizes the experiences gathered with the simulation tool.

3.1 Implementation

At the University of Winsconsin-Madison a tool has been developed to generate malicious workload. This tool is called *MACE (**M**alicious tr**a**ffic **c**omposition **e**nvironment)* and is explained in Subsection 3.1.1 based on [29]. Subsection 3.1.2 shows the functionality of the tool and in Subsection 3.1.3 an example is given.

3.1.1 Malicious Traffic Composition Environment (MACE)

The objective of the researchers was to create a performance benchmarking tool and to recreate attack traffic scenarios in a laboratory setting for testing of software and hardware systems. MACE allows to generate various attacks in a high-level language.
MACE is written in the programming language Python [57]. It is implemented as an extensible library, which consists of two classes and several functions. For each attack a script can be defined as a separate Python file.

3.1.2 MACE Functionality

The functionality provided by MACE does not satisfy all needs of this thesis. Therefore, the tool has been extended with the missing functions. In this subsection both the existing and the new features are described in more detail.

Target IP Selection The IP address and port number of source and target hosts can be defined[1].
A worm chooses the targets according to certain mechanisms, as described in Subsection 2.2.2. MACE provides the following mechanisms for selecting target IP addresses:

- **Random**
- **Sequential**
- **Local preference**
- **Partly sequential:** First an IP address is chosen randomly, then a defined amount of addresses are chosen sequentially.
- **Combination:** It can be defined what percentage of the IPs are chosen totally randomly and what percentage partially randomly in the local subnet.

[1]IP spoofing is possible.

Protocols An attack can utilize the following packet types:

- **UDP:** a UDP packet.
- **TCP:** a TCP SYN packet.
- **DNS:** a DNS query for the host part of a *Uniform Resource Locators* (URL [98]).
- **Ping:** an ICMP Echo Request.

Connection As described in Subsection 2.4.2 worms often try to infect many hosts in parallel. Therefore, the MACE tool can start multiple attack threads.
Further, the number of scans per second can be chosen. This allows simulating hosts with different characteristics.

3.1.3 Example

In this subsection we give an example of an attack. In the following script file all parameters which are needed to generate this attack are defined:

```
# Example attack (script file for MACE tool, simplified)
from macelib import *

#definition of source and destination IP addresses and ports
dstap = AddressPool(AddressPool.Random, ’129.132.0.0/16:100-1000’)
srcap = AddressPool(AddressPool.Random, ’129.132.228.75/32:1024-65536’)

#definition of attack vector
payload = ’\0’ * 376
attack_vector = [ [ rawudp, { ’ip_ttl’:128, ’frag_offset’:0,
                     ’frag_flag’:0, ’payload’:payload} ] ]

#sends every millisecond the attack_vector during 5 seconds
send_periodic(srcap,dstap, attack_vector, 1, 5)
sys.exit(0)
```

First the source and destination IP addresses have to be defined. In this example the IP addresses are chosen randomly from the subnet 129.132.0.0/16 and the port numbers are selected from the range of 100 to 1000. Then the attack vector defines the order of the packets that have to be sent. In this example one simple UDP packet is sent with a payload of 376 Bytes. Finally the frequency and possibly the number of parallel threads are chosen, here during 5 seconds one scan per millisecond is sent.

3.2 Simulation of Scan Traffic

In Section 2.4 the worms have been classified based on their scanning traffic. A simulation should cover all classes of worms and therefore, a set of scanning traffic examples are introduced in this section. Table 3.1 describes the characteristics of typical scanning traffic. It is not feasible to simulate all possible combinations of target selection, protocols, port numbers and speed. Therefore, a detailed selection of meaningful combinations is listed in Table 3.2. The following paragraphs explain this selection.

Target selection	Protocol/Port	Speed
Totally random	ICMP	Slow
Random in local subnet	TCP fixed	Medium
Sequentially with random start	TCP random	Fast
Sequentially in local subnet	UDP fixed	As fast as possible
Combination	UDP random	

Table 3.1: Scan traffic characteristics

Target Selection Column one in Table 3.2 describes different variants of target selection. To simulate various levels of local preference 8, 16, 24 or all 32 bits of an IP address are chosen at random. Some worms are based on a probabilistic function which makes it possible to switch between two address spaces with a certain probability. Therefore, this combination of different techniques was added to Table 3.2. The 60 respectively 40 percent are taken from the Blaster worm. Important is the fact that only a part of the traffic will find its way through the switch where the detector is installed. We also added an example of a worm which chooses IP addresses at random and then scans blocks of 20 IP addresses sequentially.

Protocol and Port Number In column two of Table 3.2 a selection of protocols and port numbers is listed. The selection reflects the typical scanning traffic of a worm. The considered worms use TCP or UDP except for the situation when an ICMP request is sent to check if a host is reachable. The examples use the port numbers 80 (HTTP/Webdav), 135 (RPC DCOM) and 1434 (MS-SQL) which are the port numbers used by past worms. A simulation with randomly chosen port numbers was added, because future worms may use entirely different ports.

Speed The third column in Table 3.2 describes the sending speed of a worm. The number of parallel connections respectively the sending rate has a big influence on the amount of traffic a worm produces. Therefore, the simulations cover different connection types. TCP worms open 10 to 1000 parallel connections whereas UDP worms send packets with different rates.

Target selection	Protocol/Port	Speed
totally random	ICMP	10 connections in parallel
last 8 bits at random	TCP 80, 135, 1-65535 randomly	50 connections in parallel
last 16 bits at random		100 connections in parallel
last 24 bits at random		1000 connections in parallel
60% totally random + 40% last 16 bits at random	UDP 1434, 1-65535 randomly	1 packets per second
each 20th at random and then sequentially		10 packets per second
		100 packets per second
		1000 packets per second
		as many packets as possible

Table 3.2: Scan traffic examples

With the help of the simulation tool mentioned above scripts have been written which produce the traffic described in Table 3.2. The tested simulations and the corresponding scripts are listed in Table 6.1.

3.3 Simulation Environment

This section explains the environment where the scanning traffic simulation of this thesis takes place. The simulation of scanning traffic has been tested in the office network of Open Systems AG.
The network of Open Systems AG is different from the VPN site described in Section 1.4. Figure 3.1 shows a simplified structure of the Open Systems network. The office switch is the same as in the VPN site explained in Section 1.4. The firewall *Liar* represents the VPN gateway explained before. It also contains the same functionalities as the VPN gateway. The router is replaced by a switch and a second firewall, called *Spud*. The firewall Spud does divide the different subnets and also acts as a router. It does not block any traffic destined to the switch and therefore does not influence traffic observations at the switch. The local area network is based on Ethernet (IEEE-Standard 802.3 [52]).
During this thesis outgoing traffic has to be observed and worm attacks have to be simulated. Therefore, two hosts have been installed within the network.
One of this host is called *SIM*, which stands for simulator[2]. This host is to simulate worm traffic in the office network. The other host is called *DET* for detector[3]. The detector is connected to the office switch

[2]The simulator is running on a standard PC with SuSE Linux 9.1 (Kernel 2.6), Intel x86 Pentium 3 800 MHz processor, 256 MB Ram and a 100 Mbit/sec network interface.

[3]The detector host runs on the typical hardware used for VPN gateways which corresponds to the final solution. It contains an Intel x86 Pentium 4 2.4 GHz processor, 1 GB Ram, two 100 Mbit/sec and two 1 GBit/sec network interfaces. The VPN gateway is running on a highly customized Linux (Kernel 2.4).

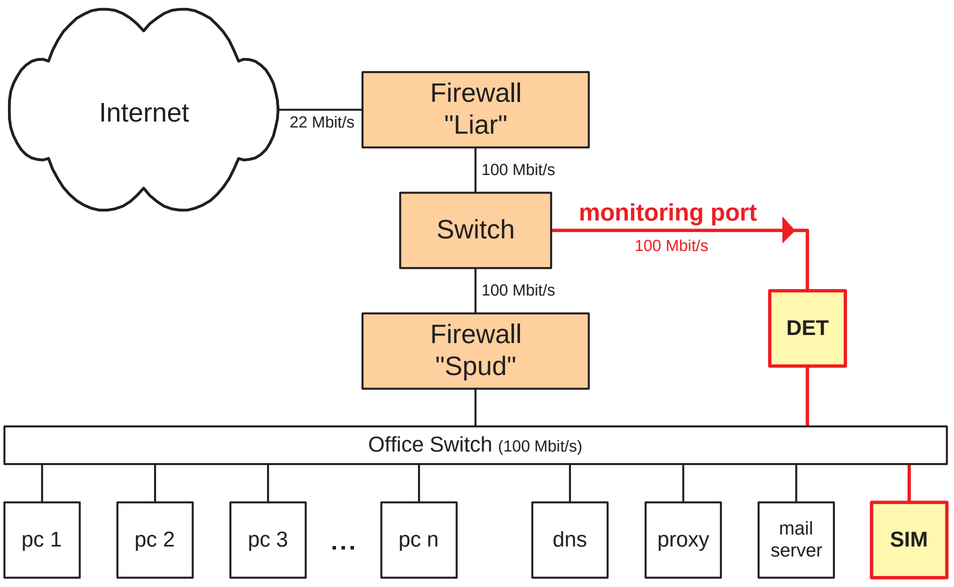

Figure 3.1: Simulation environment at Open Systems

to have access from other hosts within the network. The other interface of the detector is the monitoring interface. It reads all traffic on the switch which flows from Spud to Liar. Therefore, on the detector only outgoing and incoming traffic is visible, whereas the internal office network traffic can not be monitored on the detector.

3.4 Conclusions

The simulation tool has been extensively tested with various combinations of scan traffic according to Table 3.2 and the traffic has been observed with Tcpdump[4]. In general, these observations confirmed that our implementation correctly generates scanning traffic. However, the following two issues where encountered during the tests:

- Simulating a UDP worm with random target address showed that the maximal sending rate was about 750 packets per second. The CPU of the sending host operated under full load. A deeper analysis showed that 60 percent of the total time consumption was used to choose the new target address, to open the socket and to assemble the packet header. Therefore, a trimmed-down version of the spreading tool has been written, which is able to send UDP packets with a sending rate of 4500 packets per second[5]. This version assembles the packet header only once and implements a faster way to generate the target addresses.

- The transport protocol TCP waits for a SYN ACK packet after sending a SYN packet. The firewall rules either drop or reject an unauthorized connection attempt to an external host. The difference is that a dropped packet is not reported to the sending host whereas rejecting a packet implies that the firewall sends a TCP RST packet back. When the sending host receives a RST packet it can immediately try to establish a new connection while in the other situation the host has to wait until the socket times out. The advantage using rejects to have fast responses turns to the drawback that a host which is infected with a TCP worm, scans with a significant higher rate than in an environment with the firewall rule to drop packets. Secondly, the network traffic is approximately doubled using reject rules and consequently the network availability during a worm attack decreases with the double speed.

Based on the tests described above we conclude that the tool adequately simulates the worm classes described in Section 2.4.

[4]*Tcpdump* is a tool to intercept and display packets being transmitted or received over a network to which the computer is attached. The official website can be found at [92].

[5]This shows that our elderly simulator host reaches the requirement to simulate an average infected host.

Chapter 4

Known Detection Methods

After the extensive analysis of known and possible worms in the previous chapters, we are now interested in a detection algorithm which recognizes these worms.
Therefore, a research of published detection methods has been done and is summarized in Section 4.1. A selection of these methods will be elaborated further in Section 4.2. Section 4.3 shows the test results of the implemented algorithms.

4.1 Survey

This section explains the most promising scan detection methods published at the time of this writing [65, 2, 8].
The final detection method will be implemented on a VPN gateway, which routes traffic from one site to others as described in Section 1.4. Certain detection methods cannot be applied in this environment and are consequently not examined in this survey. Concretely, detection methods based on the following methods have been omitted:

Traffic Amount Many methods exist which are based on statistical analysis of huge traffic amounts[1]. Due to their traffic amount requirement they are mainly installed on *Internet Service Providers* (ISP [23]) routers in backbone systems.

Backbone Protocol Analysis Many detection methods have been developed for backbone systems and are based on the analysis of backbone protocols. One possibility to make an assumption of a potential worm spread can be done by analyzing the information gained from *Simple Network Management Protocol* (SNMP [62]) packets. No SNMP messages or other backbone protocols exist in the target environment.

Worm propagation The thesis concentrates on scanning traffic and does not take methods into consideration which are based on the propagation phase of the worm spreading (Section 2.1).

Static pattern matching The final detection method should use a generic approach. Methods which are exclusively based on static pattern matching of packet payloads are not considered.

The detection methods which fulfill the requirements given by the target environment are listed in Table 4.1.
The selected methods have been classified based on their detection techniques. Figure 4.1 gives an overview of this classification.

The methods can be split into two different types which are packet- and connection-oriented worm detection methods. This paragraph explains the categories in more detail.

- *Packet-oriented:* The detection method checks every packet and does the attack analysis on the basis of packets. Packets are not understood as a sequence of packets corresponding to a connection, but rather as individual ones.
    - *Pattern matching:* Arriving packets are compared to packets stored in a database. This database can be static or dynamic. Similar packets indicate a worm attack.

[1]On average, the amount of traffic flowing trough the VPN gateway used in this thesis is expected not to exceed 10 MBit/second.

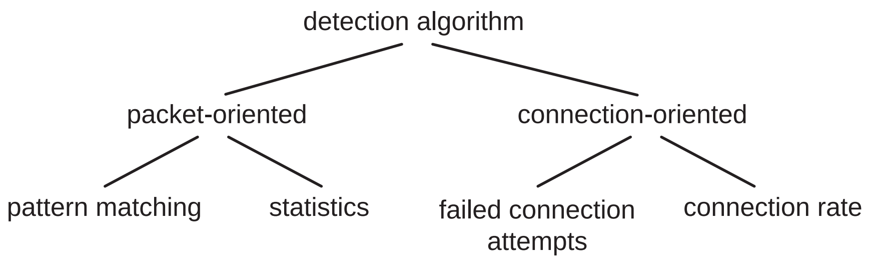

Figure 4.1: Detection algorithms classes

  - *Statistics:* The packet flow is statistically analyzed. This analysis can be based on all packets or only on a selection.

- *Connection-oriented:* Connection-oriented detection methods interpret packets as part of a connection and base their analysis on the connections as a whole.
  - *Failed attempts:* The number of failed connection attempts is counted and compared to a fixed or dynamic threshold. Exceeding this threshold indicates an attack.
  - *Connection rate:* The number of connection attempts (successful and unsuccessful ones) is counted and compared to a threshold. Exceeding this threshold indicates an attack.

		packet-oriented		connection-oriented	
	distributed detection	pattern matching	statistics	failed attempts	connection rate
Bayesian Estimation [37]	x		x		
Connection counter [35, 64, 45, 25]					x
Connection-history based anomaly detection [96]	(x)	x		x	
Entropy [4]					x
Failed connection counter [6, 66]				x	x
FPGA system [5]		x			
GrIDS [91, 90]	x	x		x	
PAYL [31]	(x)		x		
Probabilistic approach [13]					x
Sequential hypothesis test [24, 66]				x	
Spice/Spade [67]	(x)		x	x	
Victim number based algorithm [28]	x		x	x	

Table 4.1: Known scan detection methods

A detection method can belong to several classes. Table 4.1 shows the classification of the observed detection methods in column three to six. The second column describes if the method is based on a distributed concept, where information from several monitoring hosts are gathered for the detection.
A cross in parenthesis is used to indicate that this detection method can be used both with a distributed installation and with an installation on one single host.
In Subsections 4.1.1 - 4.1.12 these algorithms are elaborated on but not tested. All the information is gathered and interpreted from existing papers, which are referenced in the following paragraphs.

4.1.1 Bayesian Estimation

The worm detection system using bayesian estimation has been developed at Mitsubishi Research Institute in 2004 [5].

Method This system detects traffic anomalies in the Internet by analyzing the frequencies of scans to specified IP addresses and applying Bayesian estimation. The frequency trend is regularly updated toward the frequency change.

Advantages

- Algorithm is adapted dynamically based on previous traffic [5]
- Detection is based on information gathered from the whole network (distributed) [5]

Drawbacks

- Does not scale with varying throughput (based on high traffic volumes) [expected property]
- Resource intensive (high CPU usage) [expected property]
- Installation and maintenance of several monitoring systems in the network needed [5]
- Bandwidth overhead by monitoring information exchange [5]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation We expect that the solution is resource intensive and is based on high traffic volumes. Therefore, it will not be considered further.

4.1.2 Connection Counter

The connection counter is a very simple algorithm and has been used in various tools. The Network Security Monitor (NSM) was one of the first detection tools documented in the literature which implemented this algorithm. It has been developed at the University of California, Davis in 1990 [35]. The Virus Throttling, a detection method invented at the HP Laboratories in Bristol in 2002 [45, 25], is based on the same algorithm. The open source network intrusion detection system Snort [99] offers the possibility to use this algorithm, too.

Method The connection counter algorithm is based on the observation that infected hosts try to connect to as many different hosts as possible, whereas uninfected hosts normally open connections at a lower rate and only to a few different hosts. This algorithm counts the connection attempts from each host in a certain time period. If a given host's counter exceeds a defined threshold the corresponding host is declared infected.

Advantages

- Conceptually simple
- Economical use of system resources (low CPU/Memory usage) [expected property]
- Resistant to network outages [expected property]

Drawbacks

- Does not scale with varying throughput [expected property]
- Does not scale with network size [expected property]
- High false positive rate (fixed thresholds) [expected property]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation Worms which cause a lot of network traffic scan many hosts in a short time interval. The idea to count the number of scans is therefore promising but can cause a high false positive rate. The idea will be considered further, because it could be a part of the final solution.

4.1.3 Connection-history based Anomaly Detection

The anomaly detection approach from Toth and Kruegel is connection-history based. It has been developed at the Technical University of Vienna in June, 2002 [96]. The documented solution automatically introduces firewall rules to stop the spreading of a worm.

Method This algorithm is based on GrIDS which is described in Subsection 4.1.7. Each connection is analyzed and the probability that the connection is part of a worm attack is evaluated and weighted. Several detection methods are combined to gain the anomaly weight of a connection. The sum of the anomaly based weightings is compared to a given threshold.

Advantages

- Scalability to varying throughput [expected property]
- Fast detection of infected hosts [96]
- Resistant to network outages [expected property]
- Algorithm is adapted dynamically based on previous traffic [96]

Drawbacks

- Does not scale with network size [expected property]
- Resource intensive (high CPU and Memory usage) [expected property]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation We expect that the relatively complex algorithm is resource intensive. The algorithm has been designed and optimized to detect worms very quickly. The expected downside is a high false positive rate, therefore it will not be analyzed in more detail.

4.1.4 Entropy Based Worm Detection

The entropy based worm detection is an algorithm developed at the Swiss Federal Institute of Technology in Switzerland in 2004 [4].

Method The generic detection approach is based on the idea that the entropy of the source and destination IP addresses and port numbers increases or respectively decreases during an attack.
All destination IP addresses from the whole traffic within a certain time interval are compressed as a bit-sequence. The same is done with the other address fields in the IP header. The inverse compression ratios are compared to their historical values. This ratio is defined as:

$$\text{Inverse compression ratio} = \frac{\text{Size compressed}}{\text{Size uncompressed}}$$

Significant changes in the inverse compression ratio can be observed during a worm attack and are reported.

Advantages

- Conceptually simple
- Scalability to various network sizes [expected property]
- Economical use of system resources (low CPU/Memory usage) [4]

Drawbacks

- Does not scale with varying throughput (based on high traffic volumes) [expected property]
- Inexperience of method (unknown behavior in smaller networks)
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation The algorithm is based on high traffic volumes and has only been tested in backbone networks. The applicability to smaller networks can therefore not be forecast. The algorithm will be considered further to gain experience in small networks.

4.1.5 Failed Connection Counter

The initial proposal for the algorithm was published in [99] in January 1998. Several tools have been developed which are based on this algorithm. Bro is a Unix-based Network Intrusion Detection System *(IDS [49])* developed at Lawrence Berkeley National Laboratory [6] and is one of the tools which offer the possibility to use this algorithm. Credit-based Connection Rate Limiting (CBCRL) was developed in cooperation by experts from the MIT and Harvard University in Cambridge in September 2004 [66] and is based on the same algorithm.

Method This method is based on the observation that infected hosts will try to connect to many unreachable hosts. The number of failed connection attempts during a certain time period is counted and compared to a threshold.
The CBCRL describes a solution in which each host has its own contingent of available connections (credits). A successful connection attempt will extend the host's credits whereas a failed attempt will decrease the number of allowed connections for the concerning host. In addition, it is necessary to prevent a host from acquiring a too large number of credits by a successive inflation.

Advantages

- Conceptually simple
- Economical use of system resources (low CPU/Memory usage) [expected property]
- Fast detection of infected hosts [expected property]

Drawbacks

- Does not scale with network size [expected property]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation Counting the number of failed connection attempts is a very simple way to detect worm spreadings and quite effective. Thus this method will be considered further.

4.1.6 FPGA System

The FPGA System has been developed at the Washington University in August 2004 [5].
The system uses Field Programmable Gate Arrays (FPGA) to scan packets for patterns of similar content.

Method Hashes are calculated over a 10 byte window of streaming data and stored in a database. A count vector counts the occurrence of similar hashes. If the amount of similar hashes exceeds a threshold an infection alert is generated.
The system can be taught to ignore certain hashes. For example HTTP GET requests are commonly occurring strings but are not themselves a worm.

Advantages

- Scalability to varying throughput [5]
- Scalability to various network sizes [expected property]
- Fast detection of infected hosts [5]
- Low false positive rate [expected property]
- Algorithm is adapted dynamically based on previous traffic [5]
- Resistant to network outages [expected property]

Drawbacks

- Causes additional latency or requires a separate Interface on the VPN gateway [expected property]
- Requires special hardware [5]

Evaluation The FPGA solution implements a generic method to detect worms effectively. The biggest drawback in the current implementation is the requirement of separate hardware. The concept is quite complex and it would take time to adopt it to a software solution. In addition, we expect that it can not be used for scan detection of TCP worms because the TCP SYN packets used for establishing a connection do not contain any payload. This method will therefore not be considered further in this thesis. Because of its generic idea it could be interesting to observe it in a further work.

4.1.7 GrIDS

A graph based intrusion detection system (GrIDS) has been developed at the University of California in 1996 [91, 90, 96]. It was the first system on a large scale using hierarchical processing [67].

Method GrIDS builds so called activity graphs that represent network connections between hosts. The graphs are evaluated and possible infections (based on static pattern matching) are reported. The distributed and hierarchical architecture of GrIDS allows it to scale to large networks.

Advantages

- Conceptually simple
- Scalability to varying throughput [91]
- Scalability to various network sizes [90]
- Economical use of system resources (low CPU/Memory usage) [91]
- Fast detection of infected hosts [expected property]
- Detection is based on information gathered from the whole network (distributed) [91]
- Resistant to network outages [expected property]

Drawbacks

- High false positive rate (simplicity of policies) [expected property]
- Installation and maintenance have to be done on several computers in the network
- Bandwidth overhead by monitoring information exchange [96]
- Only detects known attacks (static pattern matching) [expected property]

Evaluation The algorithm represents a distributed graph based approach which uses static pattern matching. Therefore, it is not observed further.

4.1.8 PAYL

The payload-based anomaly detector called PAYL has been developed at the Columbia University in 2004 [31]. The inventors tried to serve high bandwidth environments like firewalls or proxies.

Method The idea is to first compute a profile byte frequency distribution and their standard deviation of the application payload flowing to a single host and port during a training phase. The authors then use the Mahalanobis distance to calculate the similarity of new data against the pre-computed profile. If the measure exceeds a defined threshold an alert will be generated. The model is host- and port-specific and conditioned on the payload length.

Advantages

- Economical use of system resources (low CPU/Memory usage) [31]
- Fast detection of infected hosts [expected property]
- Low false positive rate [31]
- Algorithm is adapted dynamically based on previous traffic [31]

Drawbacks

- Does not scale with varying throughput (based on high traffic volumes) [expected property]
- Does not scale with network size [expected property]
- A lot of false negatives [31]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation We expect that PAYL does not scale with varying traffic volumes, because it is based on high traffic volumes. Consequently, we will not analyze it in more detail.

4.1.9 Probabilistic Approach

Leckie and Kotagiri from the University of Melbourne proposed a probabilistic approach in 2002 [13].

Method This method observes the number of destinations or ports accessed by a source, as well as how unusual these accesses are. For each connection attempt the triple *source address, destination address and port number* is saved. The method then computes each sources destination and port distribution and compares it to the distribution based on the whole population of all sources.

Advantages

- Economical use of system resources (low CPU/Memory usage) [expected property]
- Algorithm is adapted dynamically based on previous traffic [expected property]
- Resistant to network outages [expected property]

Drawbacks

- Does not scale with varying throughput (based on high traffic volumes) [expected property]
- Does not scale with network size [expected property]
- High false positive rate [13]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation The probabilistic approach of Leckie and Kotagiri is based on high traffic volumes and comparable to the approach described in Subsection 4.1.4. It will not be observed in more detail, since the Entropy based approach will be considered further.

4.1.10 Sequential Hypothesis Test

Scan detection with the aid of sequential hypothesis testing was first published in May 2004 by a group of scientists from the MIT, the International Computer Science Institute (ICSI) and the Lawrence Berkeley National Laboratory [24]. Continuative work of this project can be found in [66].

Method This method bases on the computation of conditional probabilities. The method calculates the probability of the hypothesis that a host is infected considering the just observed connection attempts (conditions). An iterative computation of this probability leads to a curve called random threshold walk. If a connection attempt fails, the probability of an infection and the corresponding curve decrease. In case of a connection success the curve increases. The curve “walks” between two thresholds and depending where it passes the hypothesis that the host is benign or infected will become true. A more detailed explanation is given in Subsection 4.2.4.

Advantages

- Conceptually simple
- Scalability to varying throughput [expected property]
- Economical use of system resources (low CPU/Memory usage) [66]
- Fast detection of infected hosts [66]
- Low false positive rate [66]
- High recognition rate (low false negative rate)[2] [66]

Drawbacks

- Recognizes P2P file-sharing clients as worms [66]

Evaluation We think that the sequential hypothesis testing has the potential to become a solid scan detector. Enhanced and combined with other techniques it could be well implemented on a VPN gateway and will therefore be considered further.

4.1.11 Spice/Spade

A conceptual design for a port scan detector called Spice (Stealthy Probing and Intrusion Correlation Engine) is presented in [67] in 2002. The corresponding implementation has been done as preprocessor Spade (Statistical Packet Anomaly Detection Engine) for the Snort tool [64, 36].

[2]The recognition rate is defined as: $\frac{\text{Number of recognized attacks}}{\text{Total number of attacks}}$

Method The algorithms involves maintaining a probability model for the total activity on the defended network. Nested self-balancing binary trees can be used to encode the joint probability tables. This allows to estimate how anomalous a given packet is. Previous states are stored for anomalous packets, keeping them longer the more anomalous they are.

Advantages

- Algorithm is adapted dynamically based on previous traffic [67]

Drawbacks

- Resource intensive (high CPU usage) [expected property]
- Recognizes P2P file-sharing clients as worms [expected property]

Evaluation Spade is an interesting algorithm but badly documented. The potential can therefore not be estimated easily. The algorithm is rule based and thus not entirely generic. Consequently, Spade will not be considered further.

4.1.12 Victim Number Based Algorithm

The victim number based algorithm has been developed at the University of Massachusetts in 2003 [28].

Method Several detecting components are installed in a large enterprise network (or in the Internet). Each of them counts the number of packets a host sends to inactive addresses. A host is named a victim, if it scans at least two inactive addresses, this is called the Two Scan Decision Rule (TSDR). The detecting components report observed victims to the monitoring and detection center. This center evaluates the number of victims statistically. If the number of victims increases rapidly during a short time period a worm infection is reported.

Advantages

- Algorithm is adapted dynamically based on previous traffic [28]
- Detection is based on information gathered from the whole network (distributed) [28]
- Resistant to network outages [expected property]

Drawbacks

- Does not scale with varying throughput (based on high traffic volumes) [expected property]
- Does not scale with network size [expected property]
- Resource intensive (high CPU usage) [expected property]
- Installation and Maintenance have to be done on several hosts in the network [28]
- Bandwidth overhead by monitoring information exchange [expected property]

Evaluation The victim number based algorithm is based on a distributed system and operates well in bigger networks. The test done in [28] shows that worms in a /16 subnet are detected after having infected 4% of the vulnerable machines. The smaller the network is, the more infections are needed to detect the worm. Consequently, this algorithm will not be analyzed further.

4.1.13 Conclusion

The advantages and drawbacks of the discussed algorithms have been summarized in Table 4.2. Only the most important criteria from our point of view are given in this table.
The table helps to select those algorithms which will be considered further in the next section. This selection is based on the evaluations given in Subsections 4.1.1 to 4.1.12 and only on the information which we have found and documented in the previous subsections.
As described in Chapter 1 the final algorithms should run on one machine located at a central point of the observed network. The algorithms which are based on a distributed concept and the FPGA system which uses separate hardware will therefore not be observed further. We expect that the connection-history based anomaly detection and Spade would have high resource consumption and consequently, we do not consider them further. Three of the remaining algorithms are expected neither to adapt to varying network size nor with throughput; these are the connection counter, PAYL and the probabilistic approach.

Detection method	Scalability in throughput	Scalability in network size	Low resource consump-tion	No distributed components	No special HW required
Bayesian Estimation	-	n.a.	-	-	+
Connection counter	-	-	+	+	+
Connection-history based anomaly detection	+	-	-	+	+
Entropy	+	-/+	+	+	+
Failed connection counter	-/+	-	+	+	+
FPGA system	+	+	-/+	+	-
GrIDS	+	+	+	-	+
PAYL	-	-	+	+	+
Probabilistic approach	-	-	+	+	+
Sequential hypothesis test	+	-/+	+	+	+
Spice/Spade	n.a.	n.a.	-	+	+
Victim number based algorithm	-	-	-	-	+

Table 4.2: Overview of expected drawbacks and advantages from surveyed scan detection methods

The latter two are designed for high traffic amounts and therefore probably do not give adequate results in small networks. The connection counter does not scale with high traffic volumes and does probably require a lot of memory in large networks. Nonetheless, we think that the connection counter could provide interesting information. Therefore, we will consider it further, whereas the other two algorithms are not observed in more detail.
Consequently, the four following algorithms remain because they could be useful for detecting viruses in our target environment:

- Connection counter
- Entropy
- Failed connection counter
- Sequential hypothesis test

They will be observed in more detail in the following sections.

4.2 Algorithm Implementations

Based on the conclusions from the survey of published detection methods in the preceding section we decided to further consider four algorithms. This section describes the prototypical implementation of these algorithms. All of them have been developed for use on the detector in the network environment described in Section 3.3. The implementations of the algorithms are not optimized for CPU usage and memory consumption and are based on the two transport protocols TCP and UDP.

4.2.1 Connection Counter

Subsection 4.1.2 describes the simplest form of a scan detection algorithm. The algorithm counts all new connections of each host (successful and failed ones) and compares this number to a fixed threshold. If a host exceeds this limit within a certain time period it is declared infected.
For TCP, a connection is defined as a TCP SYN packet and all the packets following according to the handshake of TCP. A single TCP SYN packet is counted as one connection. For UDP, a connection is defined as a UDP packet sent from host A to host B. If host B sends an answer back to the host

A, the two packets are interpreted as one connection. New UDP packets from host A are counted as new connections. This definition is chosen according to the definition given in the Bro Reference Manual [101].
The Bro IDS offers this simple kind of scan detection as a standard function like most intrusion detection systems and rises an alarm by default when the host contacts 20, 100, 1'000 and 10'000 distinct destinations within one minute[3].
The tested implementation does not distinguish different port numbers and only observes TCP and UDP traffic. Nevertheless, it would make sense to set the threshold for each pair of transport protocol and port number because the connection rate is very specific to each application protocol. A low threshold results in a high detection rate but also brings a lot of false positives. In contrast, with a high threshold not all infected hosts will be recognized but at least the false positives rate is smaller.

4.2.2 Entropy

An entropy based worm detection algorithm has been proposed in Subsection 4.1.4. We have implemented it in the scripting language Perl [94]. External programs, for example *tcpdump* [92], are called via shell commands out of Perl.
With tcpdump all packet headers are recorded and stored in a log file. This log file is split into intervals of equal length and every interval is interpreted separately. Only UDP or TCP packets in one flow direction are stored. The algorithm presented in [4] is based on traffic flows, which are split into two directions which are inbound and outbound. The implementation in this thesis observes only the outbound flow containing the outgoing packets. The source and destination IP addresses of the observed packets are written to two address log files, one for each transport protocol. Consequently, four address log files for TCP and UDP and for source and destination IP addresses are kept. Afterwards, these four files are compressed using *gzip* [26]. The inverse compression ratios of the destination and source log files are compared. If the proportion between the two ratios exceeds a fixed threshold an alarm is raised.

4.2.3 Failed Connection Counter

The failed connection counter is similar to the simple connection counter in Subsection 4.2.1. Instead of counting every connection attempt of a host only the failed connections are counted and taken as indication of a possible infection. Failed connections are those connections which can not be established. For TCP two cases of a failed connection attempt can occur which are (1) no answer within a predefined timeout and (2) a TCP RST packet as answer on the TCP SYN packet. UDP is not connection oriented and the receiving host does not have to send an answer. Therefore, only ICMP unreachable packets which are sent depending on the firewall rules can be interpreted as unsuccessful UDP connection attempts.
The failed connection counter can also be achieved with Bro. We have written a script which rises an alarm when a host has failed to contact 5, 10, 20, 30, 100, 1000 and 10000 distinct hosts within a minute. In contrast to the simple connection counter this method does not provoke a false positive when a host produces benign scan traffic like web crawling. In return it can cause a false positive in case of a network outage.

4.2.4 Sequential Hypothesis Test

Sequential hypothesis test is a more sophisticated way to interpret connection events like failed or successful connection attempts (Subsection 4.1.10). The decision if a host is infected is computed via the probability of two hypotheses: The hypothesis if a host is engaged in scanning (H_1) or not (H_0).
This computation is done considering the observed sequence of events in the near past ($\mathbf{Y_n} \equiv (Y_1,\ ...,\ Y_n)$). The probability that a host is infected and involved in scanning is given as $Pr[\mathbf{Y_n}|H_1]$ whereas the probability of a hosts cleanness is $Pr[\mathbf{Y_n}|H_0]$. To decide which hypothesis is true the likelihoods are compared to each other:

$$\Lambda(\mathbf{Y_n}) \equiv \frac{Pr[\mathbf{Y_n}|H_1]}{Pr[\mathbf{Y_n}|H_0]}$$

[3]The specified functionality is partially covered by the script "scan.bro" of the Bro installation. We wrote an own connection counter script which is especially adapted for our environment. A manual of Bro is available in the Internet [100, 101]. Our tests were made with Bro version 0.9a8.17.

This ratio can be iteratively calculated and compared to an upper and lower threshold. Exceeding the upper threshold (η_1) means that we accept the hypothesis that a host is infected (H_1), whereas a ratio smaller than the lower threshold (η_0) confirms the hypothesis that a host is benign.
Sequential hypothesis testing is integrated in the freely available Bro IDS[4]. Four parameters control the behavior of the algorithm: The probabilities that a host's connection attempt succeeds assumed it is a scanner ($Pr[Y_i = 0|H_1] = \theta_1$) respectively a benign host ($Pr[Y_i = 0|H_0] = \theta_0$) and the two thresholds η_1 and η_0. The latter can be bounded in terms of the probabilities of a true positive (P_D) and a false positive (P_F):

$$\eta_1 \leq \frac{P_D}{P_F} \qquad \eta_0 \geq \frac{1 - P_D}{1 - P_F}$$

The Bro script uses per default $P_D = 0.99$, $P_F = 0.01$, $\theta_1 = 0.8$ and $\theta_0 = 0.2$ but [66] recommends $P_D = 0.99$, $P_F = 0.00005$, $\theta_1 = 0.7$ and $\theta_0 = 0.1$.

4.3 Detection Methods Tests

The tests of the implementations explained in the preceding section are elaborated on here. Subsection 4.3.1 describes the test setup for testing the methods with typical worm traffic. The observations made with each method are given in Subsection 4.3.2 and are summarized in Subsection 4.3.3.
Subsection 4.3.4 discusses various benign programs that resembles worm traffic, such as P2P clients and web crawler. Subsection 4.3.5 finishes this section with the conclusions received from the tests.
The tests described in the following subsections are neither extensive nor generally applicable. The observations hold for the observed office network at Open Systems but can not be generalized without constraints. Our intention was to get a first impression of the methods and their behavior in various situations. Extensive tests will certainly be necessary later on when testing our own detection method.

4.3.1 Worm Traffic Test Setup

The test environment described in Section 3.3 is used for these simulations. The simulation of the attacks is done using MACE (Section 3.1) on the simulator (SIM). All traffic on the monitoring port of the detector (DET) is recorded and stored with tcpdump[5] during a period of 60 minutes.
The simulator is used to simulate six worm attacks during the record period. Every ten minutes a new attack is started which lasts five minutes. The sequence of these attacks is given in Figure 4.2.

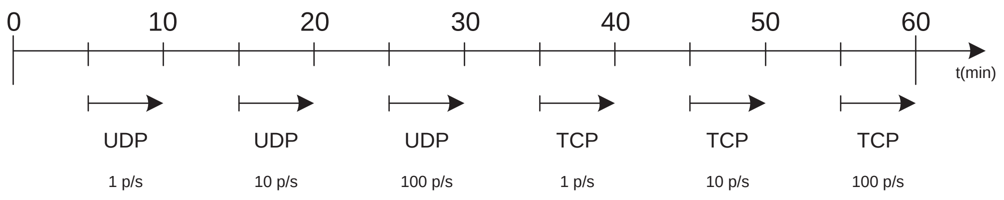

Figure 4.2: One hour test series (injected traffic)

Each double-headed arrow represents one worm attack. The first three arrows (marked as *UDP*) are worms which send UDP packets to random IP addresses. The number of packets sent per second are specified for each attack: 1, 10 and 100 packets per second. The arrows marked as *TCP* stand for simulations of worms which try to establish a TCP connection to random IP addresses. Here the numbers indicate the TCP SYN packets sent per second which are 1, 10 and 100 packets per second. The sending host does not spoof the source IP address.
Past worms mostly sent packets to a single port. However, it can not be guaranteed that they will do so in the future and therefore, the implementations do not distinguish between port numbers. Thus, the destination port number of the attack does not matter at all. We have chosen number 79, whereas the sending port is randomly chosen from 1024 to 65536 for each sent packet.

[4]The corresponding script is "trw.bro".
[5]The following command starts a tcpdump file record: `tcpdump -i eth3 -w tcpdump.log`

The payload for UDP packet has a length of 376 Byte and is filled with zeros. The TCP SYN packets do not contain a payload and the TCP header does not have any option fields.
Tcpdump records both the benign traffic from the office network and the simulated worm traffic from the simulator. The same record file is used as input to all detection methods to provide equal conditions.

4.3.2 Observed Methods

This subsection shows the results and explains the most important aspects of each implemented detection method. All methods have been tested with four different record files. In this Subsection only the results from the dump file from 5th January, 2005 from 10.49 am to 11.49 am are given. The other dump files have shown similar results.
The dump contains a total of 297'069 packets. Therefrom 277'530 packets are IP packets. They can be split into the following groups:

- TCP: 206'659 packets (74% of all IP packets)
- ICMP: 35'379 packets (13% of all IP packets)
- UDP: 33'454 packets (12% of all IP packets)
- OSPF: 2'038 packets (1% of all IP packets)

The attacks have generated a total of 33'300 UDP packets and 33'300 SYN TCP packets. Additionally, the firewall responded to each TCP SYN packets with a TCP RST packet. Consequently, the total amount of generated TCP packets is 66'600 packets which is 32% of all TCP packets. Further, the firewall is configured to send ICMP host unreachable packets as answers on UDP packets which are not allowed to pass the firewall. Consequently, the ICMP packet amount is at least the same as the UDP packet amount. The amount of UDP packets not related to the simulated attacks is 154 which is 0.5% of all UDP packets.

Connection Counter Figure 4.3 shows the connection rate of each host during the observed hour. Connection rate means the number of connections to distinct hosts at the same time. Several connections to the same destination host are counted as one. Due to clarity only hosts which tried to connect to at least 10 distinct hosts per minute are plotted.

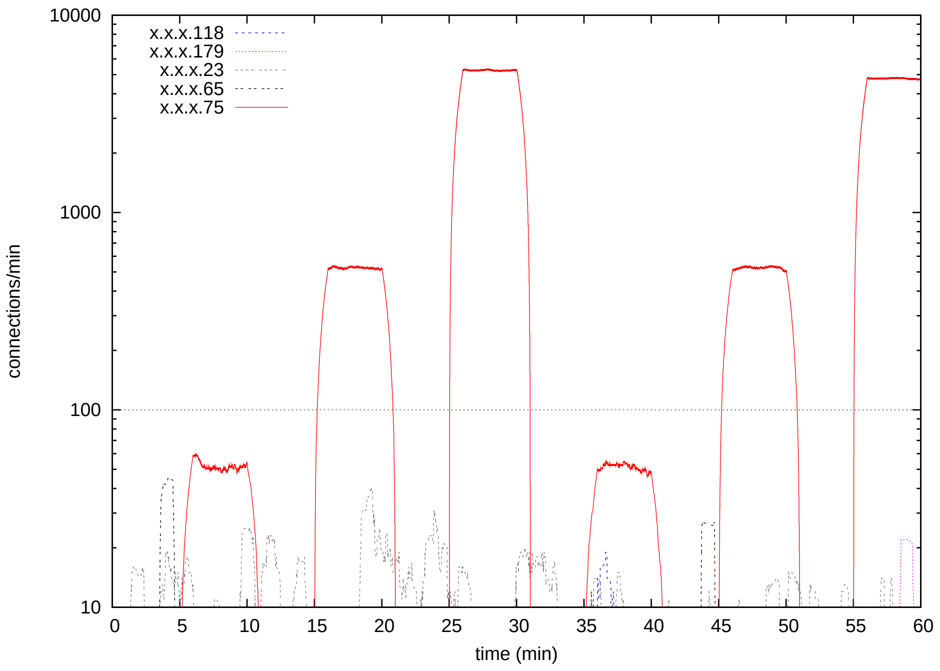

Figure 4.3: Connection rate of TCP and UDP traffic generated with the simulation given in Figure 4.2

The attacks from the simulator with IP x.x.x.75 are clearly recognizable. The slow decreases of the curve after the attacks is specific for our implementation. We regularly plot how many connections took place

within the preceding minute. This value decreases slowly when an attack ends abruptly and reaches zero only after a minute.
A threshold of 100 connections per minute produces no false positives in the observed network traffic, but only two out of three attacks are recognized. The connection rate of the hosts can differ very much and lead to false positives. For example, the x.x.x.23 is a HTTP proxy which has obviously a high connection rate.

Entropy According to the implementation given in Subsection 4.2.2 the IP addresses of the sending hosts from the internal network are written every minute into files. These files are compressed with gzip and the inverse compression ratio is stored and plotted. The resulting plots are given in Figure 4.4 and Figure 4.5. Figure 4.4 shows the inverse compression ratio of the IP address files from UDP packets whereas Figure 4.5 shows the inverse compression ratio of the IP address files from TCP packets.

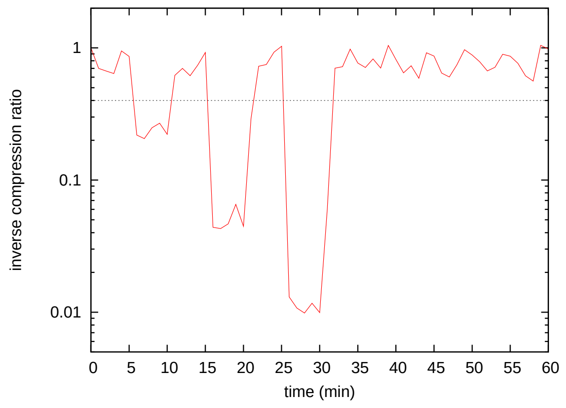

Figure 4.4: Inverse compression ratio of IP address files of UDP connections

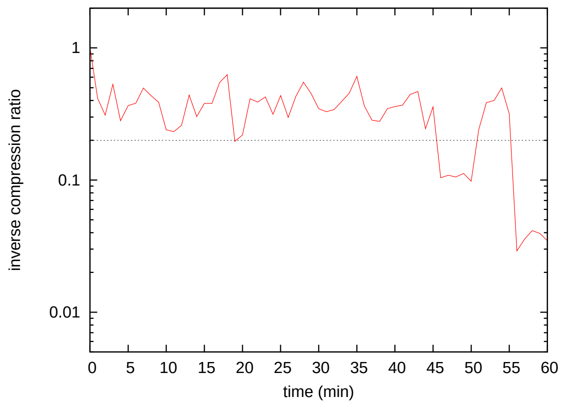

Figure 4.5: Inverse compression ratio of IP address files of TCP connections

According to the test setup the UDP attacks can be identified clearly in the UDP plot. Using a threshold of 0.4 (dotted line in the plot) generates UDP alarms during each UDP attack.
In the TCP plot the TCP attacks can not be identified with the same unambiguousness as in the UDP plot. The second and third TCP attack can be seen clearly. The first TCP attack starting at minute 35 and ending at minute 40 can not be distinguished. Using a threshold of 0.2 generates TCP alarms during the second and third TCP attack. But this threshold also generates an alarm at minute 20 when no attack has occurred. The false positive can be avoided with a lower threshold of 0.1 which results in detecting only the last attack.
Generally speaking, a higher inverse compression ratio can be observed in the TCP plot than in the UDP plot. Due to the fact, that the amount of benign TCP traffic is much higher than the amount of UDP packets it is assumable that this amount has a direct impact on the inverse compression ratios. A higher traffic amount can also cause bigger variations in the inverse compression ratio, as can be observed in Figure 4.5.

Failed Connection Counter Figure 4.6 shows the number of failed connection attempts per minute of all hosts which failed more than 5 times during the observed 60 minutes.
With a threshold of 30 failed attempts per minute the method has recognized all our attacks. No false positive occurred in the plotted hour. However, in the four observed hours the method declared one host as infected which is not.

Sequential Hypothesis Test Sequential hypothesis testing was tested with two different parameter sets: the one which comes with Bro per default and the other one with the values recommended in [66]. The results with the Bro defaults were interestingly disillusioning and ended in 11 false positives. Using the parameters from [66] has reduced the false positives to 5. Although we found a recognition rate of 100%, the parameter settings were definitely not convenient enough for our environment because of the many false positives. The sequential hypothesis test algorithm as it is integrated in Bro does not recognize undeliverable UDP packets and therefore it was not possible to detect UDP attacks.

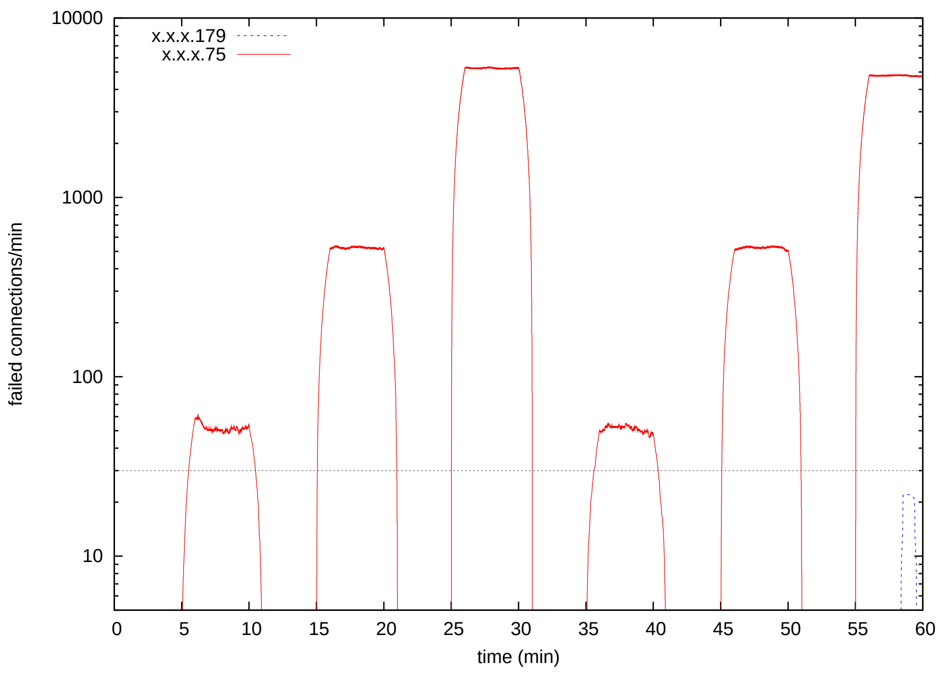

Figure 4.6: Failed connection rate of TCP and UDP traffic generated with the simulation given in Figure 4.2

4.3.3 Test Results

In this subsection the results from the previous subsection are summarized. The recognition rate and the number of false positives of each detection method has been measured. A high recognition rate can result in a high false positive rate. The requirements for the algorithm are to reach a high recognition rate and to keep the false positive rate low. Each false alarm results in a lower attention of the administrator of a system and is cost-intensive. To reach a low false positive rate the thresholds have to be increased. This leads to a lower recognition rate which can result in not detecting slow attacks anymore. This thesis focuses on attacks which generate a lot of traffic. Consequently, in our environment a low false positive rate is more critical than detecting slow attacks.
Table 4.3 shows the recognition rates of the observed algorithms. The values in the table are averages over all four tcpdump files. The recognition rate is distinguished by transport protocols and further by attacks which are defined as 1, 10 and 100 packets per second (p/s). The total column gives the average of the recognition rate over all attacks of one transport protocol.

	UDP				TCP			
	1 p/s	10 p/s	100 p/s	total	1 p/s	10 p/s	100 p/s	total
Connection Counter	0%	100%	100%	66%	0%	100%	100%	66%
Entropy	0%	100%	100%	66%	0%	100%	100%	66%
Failed Connection Counter	100%	100%	100%	100%	100%	100%	100%	100%
Sequential Hypothesis Test	n/a	n/a	n/a	n/a	100%	100%	100%	100%

Table 4.3: Recognition rates in the test environment at Open Systems

The failed connection counter and the sequential hypothesis test show the best results. But also the first two methods detect attacks with at least 10 packets per second, thus all worms that spread rapidly can be easily isolated.
Table 4.4 shows the number of false positives distinguished between protocols. This table shows the sum of all false positives in the four tcpdump files.
The first three methods show few false positives whereas sequential hypothesis test shows 5 false positives during the observed time of four hours. The recognition rate and number of false positives

	UDP	TCP
Connection Counter	0	0
Entropy	0	1
Failed Connection Counter	0	1
Sequential Hypothesis Test	n/a	5

Table 4.4: Number of false positives in the test environment at Open Systems

can be optimized by adjusting the thresholds. For the sequential hypothesis test the adjustment of this parameters is more complex than for the others. For the tests in this section the values from [66] have been taken. Adapting these values might result in lower false positive rates and in a still acceptable recognition rate.

4.3.4 Special Cases of Benign Traffic

This subsection shows the effect of benign programs which are known to scan for hosts and to generate traffic similar to worms. The following tests were made to evaluate the possibility of false positives caused by one of these programs.
We have tested various P2P filesharing clients namely DC++ [15], eMule [20], Freenet [93], Kazaa Lite [33] and LimeWire [34][6]. Kazaa Lite and eMule turned out to be the most interesting and worm alike programs and therefore, an overview of these two clients is given in the next paragraphs.
The algorithms have been tested with tcpdump files which contain 10 minutes of the traffic generated by the mentioned programs. Each program has been tested in two different cases:

- The traffic of the program has been blocked by the firewall and therefore, the program was not able to reach its destinations and run properly.
- The program had direct connection to the Internet and was not blocked by a firewall. The program was able to run properly and to reach its destination hosts.

The second case could not be tested in the Open Systems office network. Therefore, these tests were made in a network with only two running computers. This is acceptable because our solution should also work in office networks with very few hosts. However, the results of the two cases - without and with firewall - can not directly be compared because of the different test environments.

Connection Counter When a user installs Kazaa Lite or eMule on a host located behind a firewall and does not configure the necessary proxy settings, the result is the traffic in Figure 4.7.

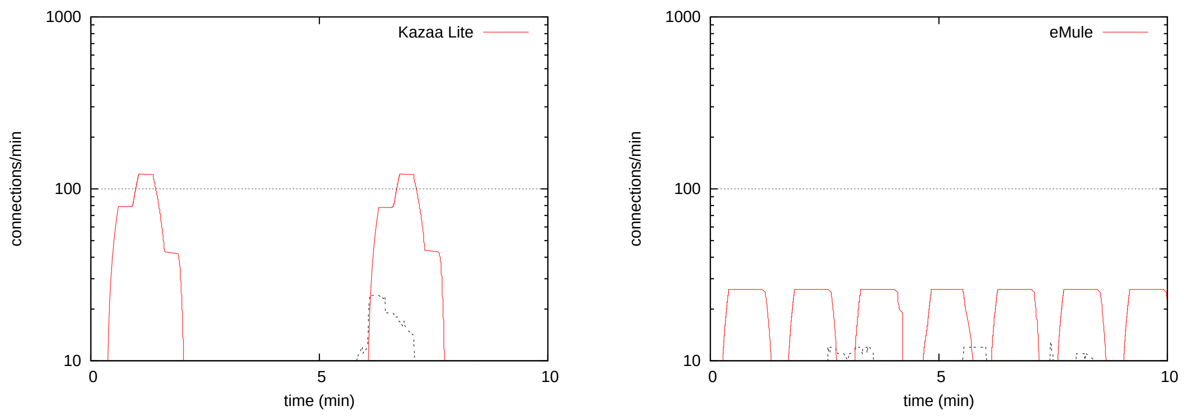

Figure 4.7: Connection rate of UDP and TCP traffic generated with Kazaa Lite and eMule blocked by the firewall

Kazaa Lite tries to connect to over 100 distinct destination hosts per minute and it shows this behavior in intervals of about 6 minutes. In contrast, eMule tries to contact to about 24 distinct hosts but in

[6]DC++ v0.668, eMule v0.44d, Freenet v0.5.2.8, Kazaa Lite v2.61d and LimeWire v4.2.6

shorter intervals of about 1.5 minutes. Using the same threshold of 100 connections per minute as in the previous subsections, Kazaa Lite would causes false positives whereas eMule does not.
Figure 4.8 shows the connection rate of Kazaa Lite and eMule in a network where they are able to connect to the filesharing network. To get an appropriate picture of the filesharing client traffic, we also executed some searches and downloads.

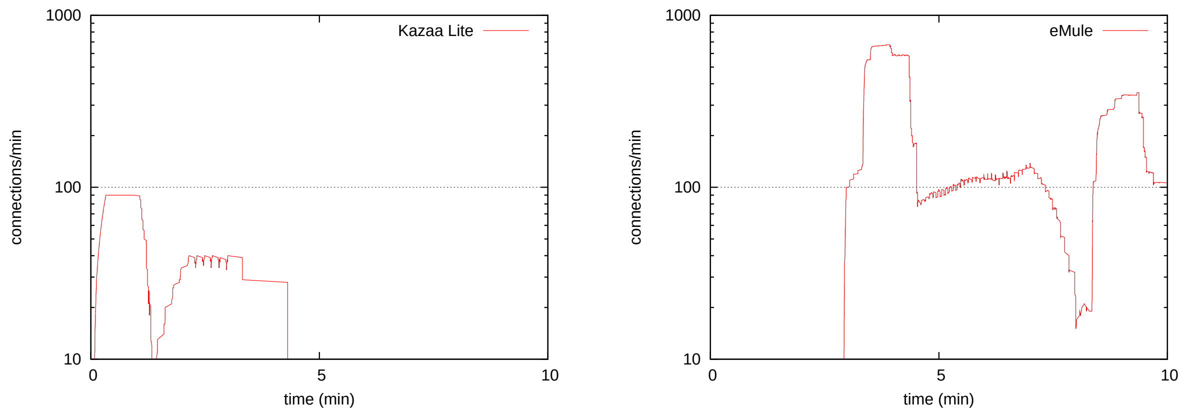

Figure 4.8: Connection rate of UDP and TCP traffic generated with Kazaa Lite and eMule

Kazaa Lite has not reached the threshold and has connected to maximal 90 distinct hosts per minute. eMule instead has connected to 700 distinct hosts per minute and has resulted in several false alarms.

Failed Connection Counter Counting the failed connection attempts to distinct hosts per minute and using the same constellation and traffic dump files like before leads to the following results. Figure 4.9 shows the failed attempt rate when Kazaa Lite or eMule have been installed in a network located behind a firewall and no proxy configuration has been done.

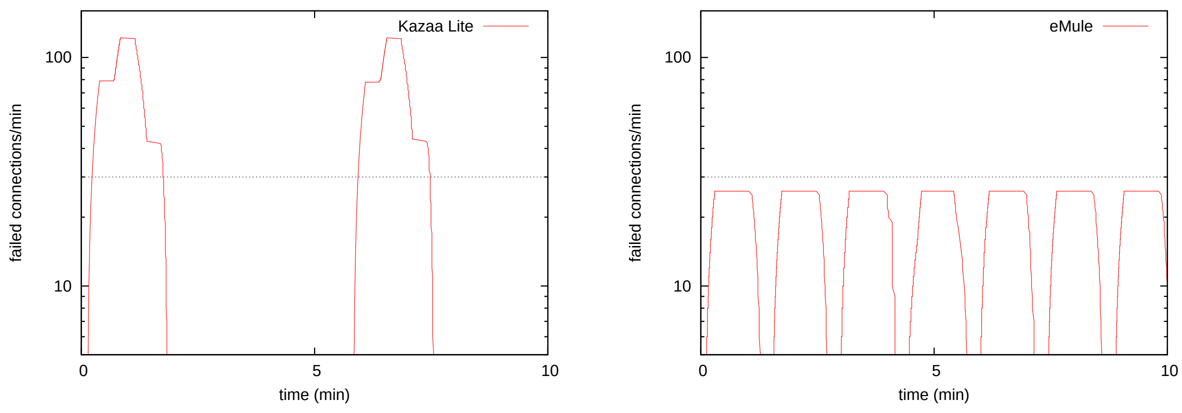

Figure 4.9: Failed connection rate of UDP and TCP traffic generated with Kazaa Lite and eMule blocked by the firewall

The plots are similar to the plots in Figure 4.7, because the firewall blocks all direct traffic and all connection attempts fail. Kazaa Lite produces clear false alarms, while eMule almost reaches the threshold.
Figure 4.10 shows the failed connection rate when using the two P2P clients without a firewall.
Both clients trigger a false alarm because they have a failed attempt rate of over 30. Kazaa Lite even has 60 failed connection attempts per minute over a period of 2.5 minutes.

Entropy The following plots show the results of the entropy algorithm. The curves represent the inverse compression ratio gained from the UDP and TCP traffic of the observed P2P clients. The resolution of the plots is very low because we have only one inverse compression ratio value each minute.
Figure 4.11 and 4.12 show the inverse compression ratios when Kazaa Lite and eMule are installed in a network separated from the Internet by a firewall. The UDP plots are listed on the left and the TCP plots on the right.

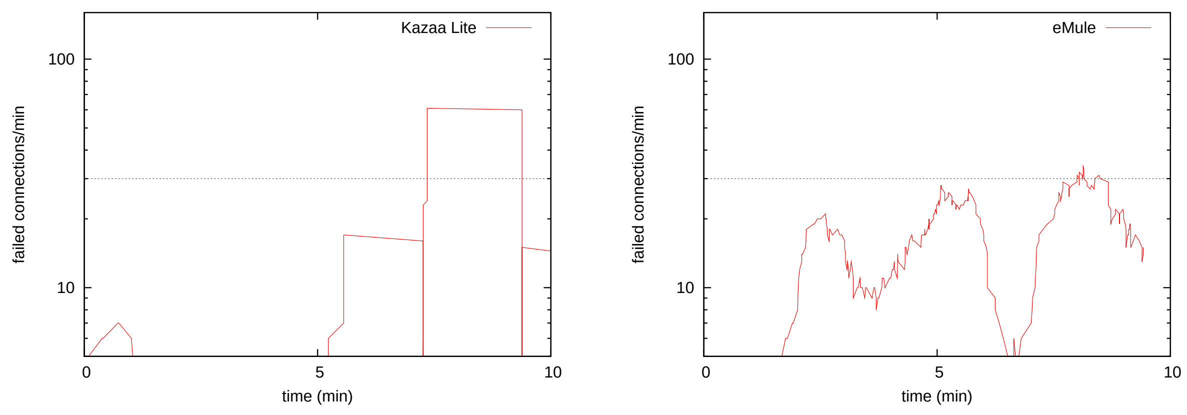

Figure 4.10: Failed connection rate of UDP and TCP traffic generated with Kazaa Lite and eMule

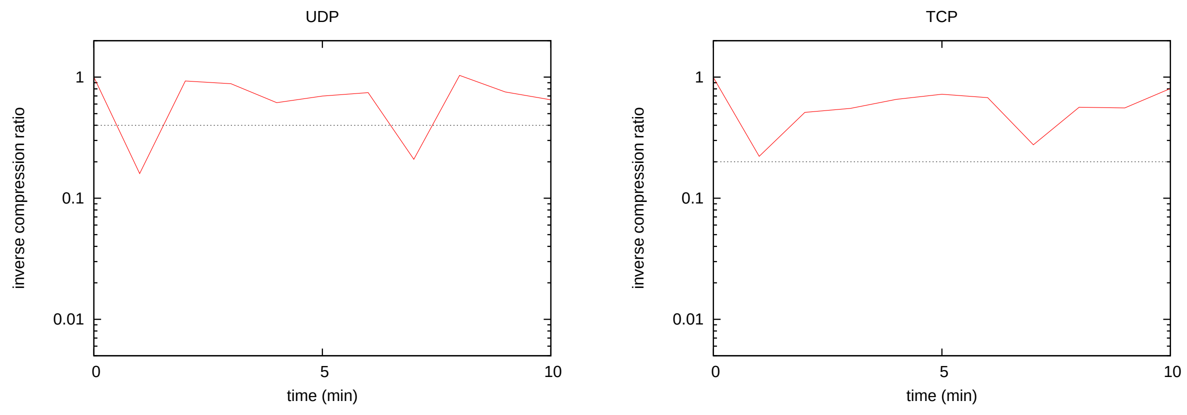

Figure 4.11: Inverse compression ratio of UDP (left) and TCP (right) traffic generated with Kazaa Lite blocked by the firewall

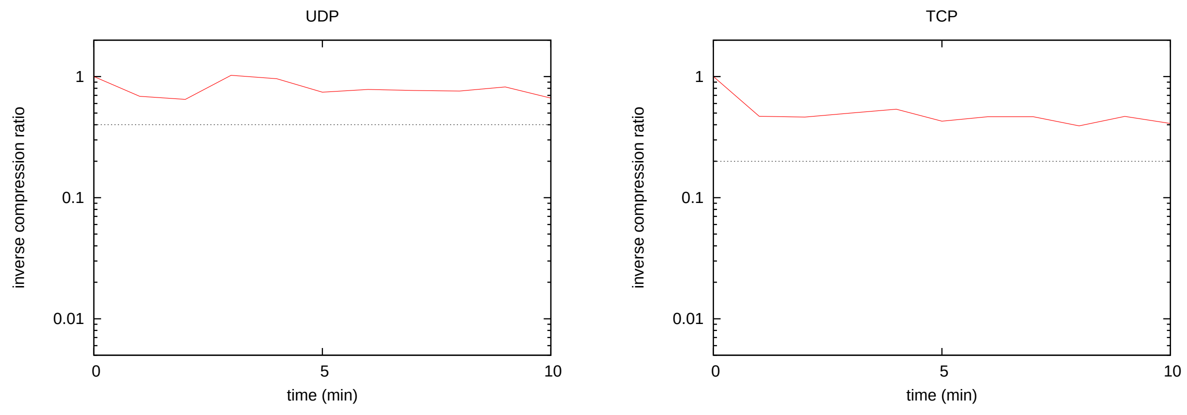

Figure 4.12: Inverse compression ratio of UDP (left) and TCP (right) traffic generated with eMule blocked by the firewall

The scans of the Kazaa Lite client are clearly visible in Figure 4.11. Both, the UDP and the TCP plot show little peaks when Kazaa Lite tries to establish connections. In contrast, the eMule plots show no observable changes.
Figure 4.13 and 4.14 represent the inverse compression ratios gained from traffic when Kazaa Lite and eMule were running and downloads and searches have been done.
The traffic of the clients can be seen clearly in all four plots (Figure 4.13 and Figure 4.14). With the chosen thresholds, entropy rises alarms in all cases: For the Kazaa Lite and the eMule clients, for UDP and TCP traffic.

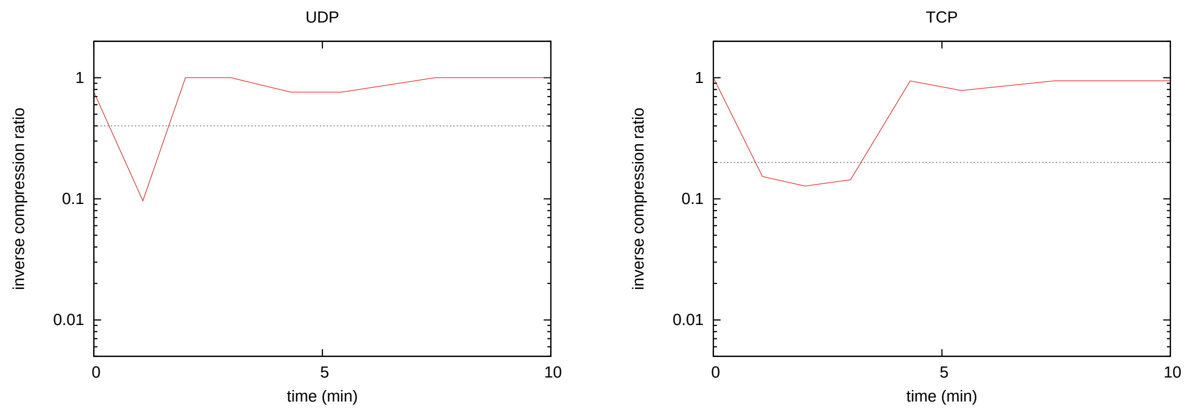

Figure 4.13: Inverse compression ratio of UDP (left) and TCP (right) traffic generated with Kazaa Lite

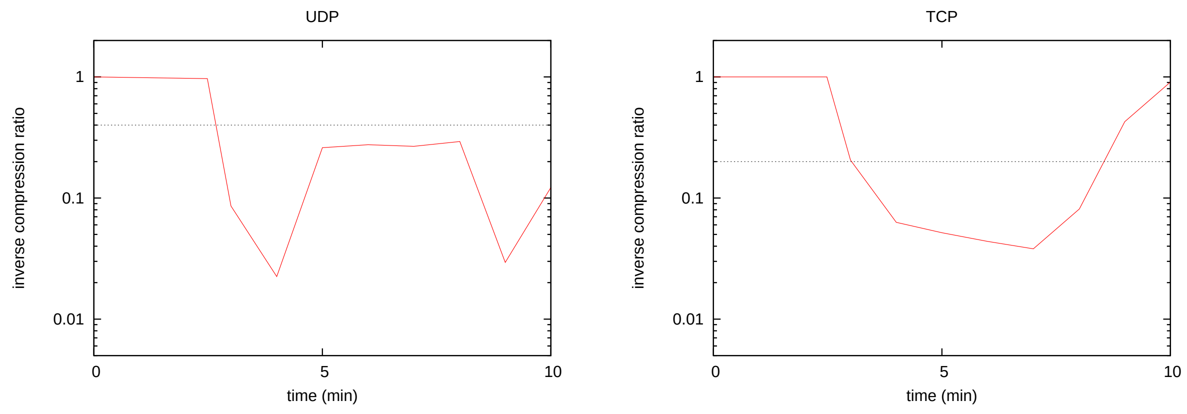

Figure 4.14: Inverse compression ratio of UDP (left) and TCP (right) traffic generated with eMule

Sequential Hypothesis Test The sequential hypothesis test produced false alarms while analyzing the traffic generated with the Kazaa Lite and the eMule client, independent if the client was running behind a firewall or not. The parameters have been chosen as described before according to [66].
In addition to the tests made with P2P clients, we also took tcpdump records of the web crawler[7] Copernic [12]. Different versions of Copernic exist. We only considered the freely available Copernic Agent Basic[8] which collects its results from maximal 13 search engines in parallel. The other commercial Copernic Agent variants allow searches to much more sites in parallel.
Using Copernic without proper proxy configuration leads to a failed connection rate of 13 failed attempts per minute. A small increase of the connection rate of the proxy can be seen when the search request of the Copernic Agent goes over the proxy. We expect that the much more powerful commercial variants of the Copernic Agent would result in much higher failed attempts rate respectively connection rates.
The number of connection attempts is too low to see any results with the entropy algorithm. The sequential hypothesis test results in a false positive when the Copernic Agent is running behind a firewall without proper proxy configuration.

4.3.5 Conclusion and Outlook

The observed four hours in the Open Systems office network do not represent all possible constellations of traffic and environment. Nonetheless, the following important conclusions can be made:

- In our environment most simulated worms can be easily recognized because they produce a large amount of scanning traffic and consequently, infected hosts show a very typical behavior.
- The difficulty is the appearance of false positives. There are a lot of programs which can partly behave like a worm namely web crawlers, P2P filesharing clients and so on.

[7]Here, a web crawler is a program which contacts several search engines and collects the search results to offer a summary to the user.
[8]Copernic Agent Basic v6.11

- The connection counter, the failed connection counter and the entropy based algorithm have shown similar results regarding recognition rate and number of false positives. The sequential hypothesis test has a very high false positive rate and does therefore not fulfill our requirements.
- The entropy based algorithm gives a fast overview of the whole network whereas the connection counter and the failed connection counter provide more detailed information about the scanning rate of each host.

Consequently, it is necessary to develop an algorithm or a combination of algorithms which can distinguish between scans originated from benign programs and scanning traffic produced by worms. The design of this algorithm will be described in the next chapter and will be based on the requirements and the information gathered in this chapter.

Chapter 5

Design and Implementation

This chapter elaborates on the design of the scan detection algorithm and explains the implementation. Section 5.1 gives the requirements and shows the worm indicators. Section 5.2 gives the specification of the algorithm and Section 5.3 first explains the used IDS framework Bro and then examines the implementation.

5.1 Initial Situation

Before starting the implementation of our own algorithm, we will summarize the situation. In particular, we resume the requirements which our algorithm should fulfill (Subsection 5.1.1) and recapitulate the worm traffic characteristics which makes a worm detectable (Subsection 5.1.2).

5.1.1 Requirements

This subsection describes the needed functionality of our algorithm. The following list defines the requirements sorted by importance.

- High recognition rate
- Very low false positive rate
- No influence on benign traffic
- Economical use of system resources
- Scalability to variable network size
- Adequate response time after infection

The following three points are not essential but rather appreciated:

- Low maintenance
- Expandability
- Provide additional information about the infection

High recognition rate As mentioned before, it is our goal to detect all worms which potentially flood the internal network and the network link to the Internet with scanning traffic. Therefore, our solution should detect all worms which exceed a certain amount of scanning traffic. Stealth worms will stay undetected. Our algorithm should recognize the worm traffic described in Section 3.2 and, additionally, email worms (Subsection 2.2.7).

Very low false positive rate Our method should generate no false positives. Each false positive affects the attention of the observing person and it costs time and money every time a false alarm occurs.

No influence on benign traffic Benign traffic should not be influenced in any way. Neither drops of packets nor a slow down of the benign traffic should occur.

Economical use of system resources The final solution will be running on a VPN gateway as mentioned in Subsection 1.4.1. The scan detection tool must not interfere with the other programs installed on the VPN gateway.
The CPU can temporarily be used up to 100% but the scan detection process should run with lower priority and, if required, hand over CPU time to other, more important processes. Further, the detection method should control its memory usage and not claim over 100 MB RAM. Additionally, too much hard disk accesses and high disk space requirements should be avoided.

Scalability to variable network size The networks observed by our detection method vary very much in size. A network could consist of only four hosts or it could contain up to 5000 hosts. Therefore, the throughput at the VPN gateway will be very variable, too. The maximum required throughput is 250 megabit per second.

Adequate response time after infection The time from an infection of a host to the corresponding alarm should not exceed 10 minutes.

Low maintenance The installation on a VPN gateway should be possible without much configuration and take little time. In addition, the detection method should run with low maintenance.

Expandability It should be easy to add features and to integrate other detection mechanisms in the future. The method should be adaptive in a way that totally different worms can be recognized with little additional work in the future.

Provide information about the infection It is highly appreciated that our detection method provides as much information about the infection as possible. Especially, the observed traffic anomalies and the origin of the scans are of evident importance for the person who has to find the problem in case of an infection.

5.1.2 Worm Indicators

The worm characteristics have been discussed in Chapter 2. Further, the worms have been classified based on their target selection and on the generation of the scanning traffic. The worm detection focuses on the scanning traffic occurring in an office network.
The scanning traffic has typical characteristics. The most important aspects are explained in this subsection. The final method is based on these aspects. Worms can use diverse methods to scan for hosts, therefore each scan has its own characteristics. Each of the aspects explained below can be interpreted as an indication of a worm infection.

1) High number of first connection packets First connection packets are those packets which are sent first from one host to another. With TCP, this is typically a TCP SYN packet. With UDP, the first UDP packet between two hosts is interpreted as first connection packet whereas a reply on a UDP packet is assigned to the same connection and is therefore not counted as first connection packet. Furthermore, ICMP echo requests are also considered first connection packets.
A worm which scans for victims can use these first connection packets to see if the target replies and if it is possible to establish a connection to this host. But there also exist other methods to check if a host replies. A list is given in [67].

2) High number of failed connection attempts According to Subsection 4.2.3, a failed connection attempt is defined as a connection which could not be established. With TCP, a timeout or a TCP RST packet indicate a failed connection. With UDP, failed connections are more difficult to detect. Not all application layer protocols require a response. ICMP unreachable packets are one possibility to find out that a UDP packet was not successfully received from the target host.
A worm often scans random targets which do not exist. The sending hosts can therefore not establish TCP connections and these TCP scans result in failed attempts. Consequently, we think that this aspect is a strong indication for a worm.
An Internet Service Provider (ISP) outage causes all connections to fail, which also results in a high number of failed attempts. This case needs to be handled separately to avoid false positives.

3) High number of packets to the same port Past worms have mostly exploited a certain vulnerability in an application on a predefined port. These worms have sent all packets to the same destination port and this aspect is therefore highly typical for this kind of worms.

4) Scanning over a longer period Past worms have often scanned over several hours or days. Further, a worm which scans only during a short time is hard to detect and does not really load the network. This is a very important criterion because worms which do not scan over a certain period are not interesting for us.

5) One host contacts many distinct hosts A worm normally tries to infect as many other hosts as possible and therefore, the scanning traffic is typically flowing from one host to a lot of other hosts.

6) Distribution of the target IPs over the whole IP address space The target addresses of a worm scan are chosen within certain ranges of the whole IP address space. This can be done in various ways, for example they can be randomly chosen or taken from lists.
Past worms often have scanned the whole IP address range. Some of them have scanned local addresses more intensively.

7) Few connection attempts from one host to the same destination IP and destination port The number of connection attempts from one host to the same destination IP and destination port is low or equals one because a worm normally tries to scan different hosts or different ports on the same host.
In the past worms have mostly scanned on the same port and therefore, every scan was destined to another host. Future worms could scan various ports on one host but are not expected to scan the same port on one target many times.

8) Same UDP packet size In the past, only one worm which has used UDP has spread remarkably. This was the Sapphire worm (Subsection 2.3.4). This worm has sent one packet which has done the scanning, the infection and the spreading. The packets always had the same size which was a clear indication for this worm. TCP worms scan for victims without sending the worm code and therefore these packets do not have a typical size.
Future worms could behave similar to the Sapphire worm or they could use UDP only to scan for victims. However, a worm could choose varying packet sizes to stay undetected.

5.2 Algorithm Specifications

This section introduces the specification of our scan detection method. The method has to fulfill the requirements in Subsection 5.1.1 and is based on the worm indicators in Subsection 5.1.2.
At this point we have enough knowhow to define a successful detection algorithm for the target environment. We do not base our solution on one particular algorithm like entropy or sequential hypothesis test. Sequential hypothesis test has produced too many false positives and was therefore not well suited for this environment. Entropy has demonstrated an economical way of a good detection algorithm but is rather intended for high speed links and a bigger effort would be necessary to provide detailed information about an infection. Additionally, we think that we can handle the appearing traffic with several different counters and can therefore collect a lot of information about the infection and the state of the network and single hosts.
Although our approach is based on similar worm indications like other detection methods, it is different in the way of realization. We suggest a multilevel approach which provides differently detailed views of the network and single hosts. Only hosts which appear to be infected using low-cost checks have to be investigated in more detail. This idea allows to save valuable system resources.
The suggested detection method treats each host individually. At first, all of the hosts are observed and tested for a possibly appearing traffic anomaly. If this general low-cost test states a host as suspicious, it will be observed in a second step in more detail. These steps lead to more and more specific tests which analyze the behavior of this host. Additionally, with each step a further sequence of the host traffic will be observed and so it is possible to consider a longer period of the host behavior to decide if a host is infected or not. Our approach has the benefit that totally benign hosts which never generate abnormal traffic will produce only little resource consumption within the scan detection tool.
The following subsections specify our scan detection method in detail. Because of the strong differences between the protocols used by a worm scan it is necessary to discuss TCP, UDP and ICMP separately.

5.2.1 TCP Scan Detection

The transport protocol TCP is a connection-oriented protocol. With benign TCP connections, each packet sent causes the destination host to answer with a packet. In the given network environment

with strict firewall rules a host can not directly connect to external hosts and therefore, the packets are dropped or rejected by the firewall. Consequently, a failed connection attempt can be recognized when no answer or a TCP RST packet follows a TCP SYN packet. A TCP timeout has to be defined, which defines after which time a connection attempt is declared to have failed.
The algorithm concentrates on failed connection attempts (number 2 of the indicators in Subsection 5.1.2) whereas the number of first connection packets (number 1) is not taken into consideration for TCP. The multilevel algorithm can be illustrated as a flowchart diagram per host and is given in Figure 5.1.

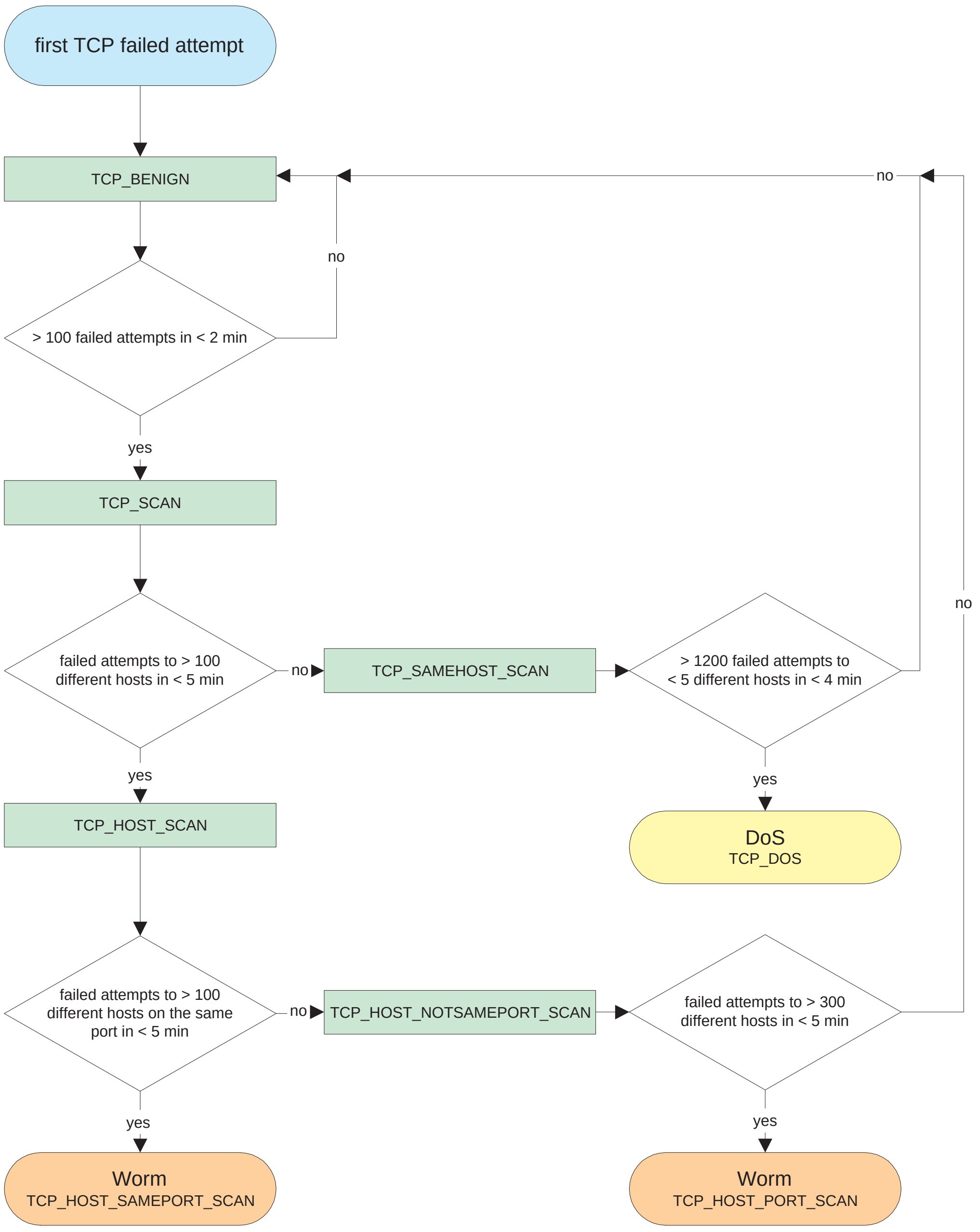
first TCP failed attempt
TCP_BENIGN
no
no
> 100 failed attempts in < 2 min
yes
TCP_SCAN
no
failed attempts to > 100 different hosts in < 5 min
no
TCP_SAMEHOST_SCAN
> 1200 failed attempts to < 5 different hosts in < 4 min
yes
yes
TCP_HOST_SCAN
DoS
TCP_DOS
failed attempts to > 100 different hosts on the same port in < 5 min
no
TCP_HOST_NOTSAMEPORT_SCAN
failed attempts to > 300 different hosts in < 5 min
yes
yes
Worm
TCP_HOST_SAMEPORT_SCAN
Worm
TCP_HOST_PORT_SCAN

Figure 5.1: Finite state diagram for TCP scan detection per host

Each host is treated individually and is in one of the TCP states after the first seen TCP failed attempt. A host without failed connection attempts will not be tracked and does not have a state. Eight TCP states exist.
The rhombuses in the flowchart diagram represent tests. A test checks if a certain number of certain equal events within less than a given time has been tracked. When the number of equal events has been reached the result of this rhombus is *yes* and the host receives a new state. If the limit has not been

reached within the given time, the result of the rhombus is *no* and the host receives the according state.
A host which has caused a failed connection gets first the TCP_BENIGN state. The number of failed connections is counted within the given time. When the limit is reached the hosts gets the new state TCP_SCAN. Now, the number of failed attempts to different hosts is counted (according to number 5 of the indicators). In the state TCP_HOST_SCAN the number of failed connection attempts to different hosts on the same port is counted (number 3). For hosts in the state TCP_HOST_NOTSAMEPORT_SCAN the number of failed attempts to different hosts is counted, which is the same criterion with a different threshold as a host tested in state TCP_SCAN. A host in the state TCP_SAMEHOST_SCAN is tested on a distributed denial of service attack which is discussed below.
A host in the states TCP_HOST_SAMEPORT_SCAN, TCP_HOST_PORT_SCAN is deleted from the list of tracked hosts after a certain time (for example 24 hours). A host in the state TCP_BENIGN is also deleted from the list after not having caused any failed connection attempt in 24 hours.
Number 4 of the indicators shows that it is advisable to observe a worm scan over a longer period. This requirement is fulfilled for slow scanning worms, because in each state a worm scan has to reach a limit to cause a positive state change. If a worm scans with a very high rate, it can be detected within a short time. For example a worm sending 500 packets per second with a firewall rejecting the packets can be detected in less then 1 second. Several possibilities exist to assure a host scans over a longer period to prevent a too fast decision.
The worm indicators number 6 and number 7 are not covered here, but could be used in future work. Number 8 is only an indicator for a worm scanning with UDP packets and is covered here neither.

Other scan methods In the past, TCP SYN scans have been the most common method to scan for hosts and to detect if these ports are open. Future worms could use other TCP flags to check if a target exist. The most important ones are the following ones (based on [32]):

- *TCP FIN scans* (FIN flag set): the target system should reply based on RFC 793 with a TCP RST
- *TCP Xmas scans* (FIN, URG and PUSH flags set): closed ports should reply based on RFC 793 with a TCP RST
- *TCP Null scans* (no flags set): closed ports should reply based on RFC 793 with a TCP RST

Although these scans are not connection attempts they can be treated similar to TCP SYN scans. The implementation in Subsection 5.3.3 explains how these scans are treated.

DoS Attacks A denial-of-service (DoS) attack is defined in [16] as:

> Attack on a computer system or network that causes a loss of service to users, typically the loss of network connectivity and services by consuming the bandwidth of the victim network or overloading the computational resources of the victim system.

This can be done by sending TCP SYN packets at a high rate to a certain host.
A host in the state TCP_SAMEHOST_SCAN is tested for sending all packets to a very small number of targets at a certain rate. This rate is chosen higher then the other limits, to ensure not to declare benign traffic as DoS attack.
DoS attacks are normally destined for a small group of destination hosts. Therefore, the test counts and restricts the number of different destination hosts.
A host can only be in one state per protocol at a time and a host which is in a worm end state is not tested anymore to detect further infections. We do not detect DoS attacks which are started after this host has entered a worm end state. But it would be possible to do these tests after having reached a worm end state and could be implemented in future work.

Email Worms The email worms described in Subsection 2.2.7 are discussed separately from TCP worms. We only consider email worms which try to connect directly to SMTP servers without using an internal SMTP relay or server. This sort of email worms can not establish successful TCP connections because of the firewall. All attempts will fail and we can again count the number of failed connection attempts.
Email worms often search for email addresses on the infected hosts and then try to send themselves to these addresses. We expect that the majority of the address books contain many addresses from the same domain and that the number of different domains is rather small. Some of the email worms try to connect to each mail server once, others will try to each mail server for each found email-address once. Consequently, we have chosen a low limit for this kind of failed attempts and count the number of different hosts. Figure 5.2 shows the flowchart diagram to detect email worms.

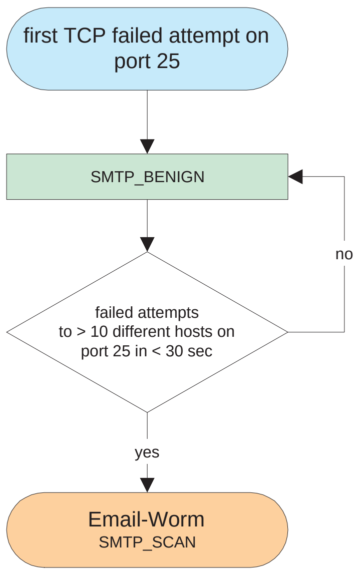
first TCP failed attempt on port 25
SMTP_BENIGN
failed attempts to > 10 different hosts on port 25 in < 30 sec
no
yes
Email-Worm
SMTP_SCAN

Figure 5.2: Finite state diagram for SMTP scan detection per host

5.2.2 UDP Scan Detection

UDP is not connection-oriented and therefore, we can not expect that each transmitted packet causes the receiving host to answer with a packet. Consequently, we can not simply count the number of unanswered packets to get the number of failed connections.
Figure 5.3 shows the flowchart diagram for UDP scan detection per host. A host is tracked when it has sent its first UDP packet. Then the host is declared as UDP_BENIGN. According to the checks, the host is in one of the 16 UDP states.
The first check counts how many UDP packets the observed host sends within a defined period (number 1 of the worm indicators in Subsection 5.1.2). If it exceeds the limit the host changes its state to UDP_SCAN.
If the firewall rejects the UDP scan packet it replies with an ICMP unreachable. Those are counted for hosts in the state UDP_SCAN. If the number of ICMP unreachable packets reaches the given limit the host gets into the state UDP_FAILED. For hosts in this state and all states below (ending on ”_FAILED”) only ICMP unreachable packets are observed. Similar to TCP individual tests such as number of packets to different hosts on the same port and different ports are done according to the flowchart diagram in Figure 5.3.
If the firewall does not send back any ICMP unreachable packets a host in the state UDP_SCAN changes its state after the defined time to UDP_NO_FAILED. For hosts in this state and all states below (gray shaded area), the number of sent UDP packets is counted. If a destination host sends back a UDP packet this connection is ignored. Consequently, only connections without a reply are counted for the checks in the gray shaded area. A timeout has to be defined after which we determine that the destination host has not replied. The most common application layer protocols based on UDP, which are DNS/53, TFTP/69, NTP/123 are bidirectional and therefore packets are sent in both directions. Thus, we think that observing UDP packets without replies brings adequate results.

DoS Attacks Distributed denial of service attacks are treated similar to the solution with TCP.

5.2.3 ICMP Scan Detection

In the past, ICMP echo requests have also been used to find out if a target exists and therefore we have to detect these scans, too. Either the firewall replies on ICMP echo requests with a ICMP unreachable packets or the firewall drops the packets. For ICMP we observe both cases and call them ICMP failed attempts. For the second case a timeout is defined.
Figure 5.4 shows the flowchart diagram for ICMP scan detection per host. The flowchart is similar to TCP but it is simpler due to the fact that ICMP does not have port numbers. Consequently, only five ICMP states exist.

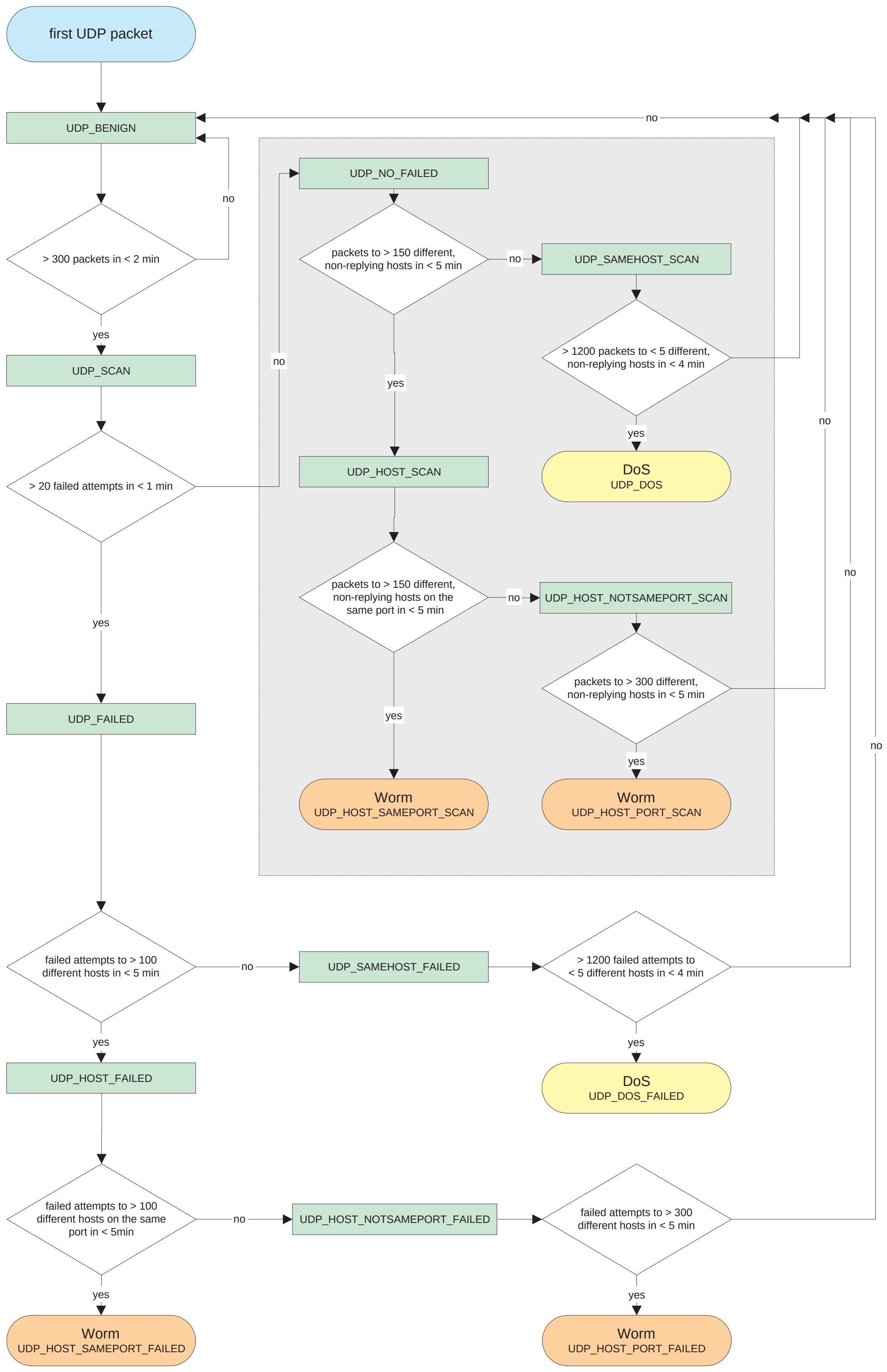
first UDP packet
UDP_BENIGN
> 300 packets in < 2 min
no
yes
UDP_SCAN
> 20 failed attempts in < 1 min
no
yes
UDP_NO_FAILED
packets to > 150 different, non-replying hosts in < 5 min
no
UDP_SAMEHOST_SCAN
> 1200 packets to < 5 different, non-replying hosts in < 4 min
yes
DoS UDP_DOS
yes
UDP_HOST_SCAN
packets to > 150 different, non-replying hosts on the same port in < 5 min
no
UDP_HOST_NOTSAMEPORT_SCAN
packets to > 300 different, non-replying hosts in < 5 min
yes
Worm UDP_HOST_SAMEPORT_SCAN
Worm UDP_HOST_PORT_SCAN
UDP_FAILED
failed attempts to > 100 different hosts in < 5 min
no
UDP_SAMEHOST_FAILED
> 1200 failed attempts to < 5 different hosts in < 4 min
yes
DoS UDP_DOS_FAILED
UDP_HOST_FAILED
failed attempts to > 100 different hosts on the same port in < 5min
no
UDP_HOST_NOTSAMEPORT_FAILED
failed attempts to > 300 different hosts in < 5 min
yes
Worm UDP_HOST_SAMEPORT_FAILED
Worm UDP_HOST_PORT_FAILED

Figure 5.3: Finite state diagram for UDP scan detection per host

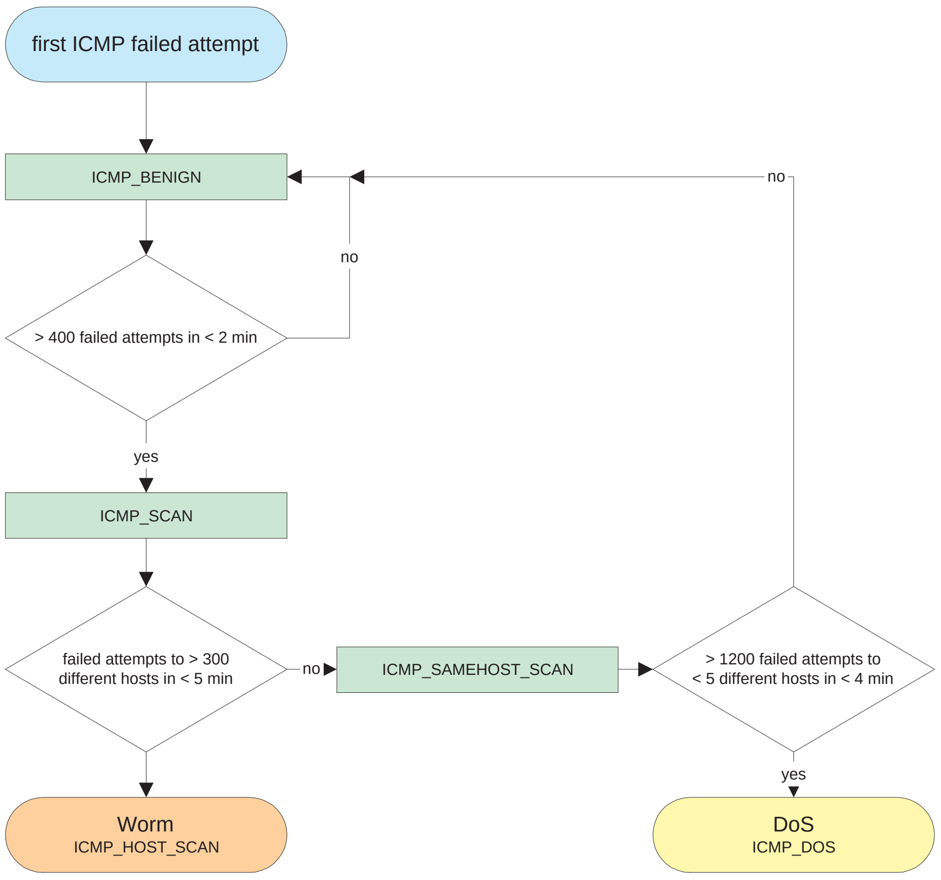
first ICMP failed attempt
ICMP_BENIGN
no
no
> 400 failed attempts in < 2 min
yes
ICMP_SCAN
failed attempts to > 300 different hosts in < 5 min
no
ICMP_SAMEHOST_SCAN
> 1200 failed attempts to < 5 different hosts in < 4 min
yes
Worm ICMP_HOST_SCAN
DoS ICMP_DOS

Figure 5.4: Finite state diagram for ICMP scan detection per host

A host starts in the state ICMP_BENIGN when the first ICMP failed attempt has been observed. The first test counts the number of failed attempts and the further tests count to how many hosts these failed attempts occurred.

DoS Attacks Distributed denial of service attacks are treated similar to TCP and UDP.

5.3 Implementation

The following subsections explain the implementation of our scan detection method. The first subsection gives reasons for the use of the Bro IDS and illustrates the architecture of this intrusion detection system. Subsection 5.3.2 contains an overview of our policy scripts written for Bro IDS and Subsections 5.3.3 to 5.3.7 describe other important aspects of the implementation.

5.3.1 Bro IDS Framework

We decided to use the freely available Bro IDS Framework [6] to implement our scan detection algorithm. Bro is designed for high-speed monitoring of network traffic and real-time notification. The architecture of Bro allows integrating own algorithms utilizing all the functionality which Bro provides.
In contrast to Snort which is another open source intrusion detection system Bro is not primarily designed for analyzing packets and doing pattern matching. Bro provides a more general observation of connections which is important for our connection-oriented solution. Further, we can save much time by using Bro instead of programming our own framework. With Bro we can program on a higher level and do not have to pay attention to things like packet filtering, connection state identification and memory handling. Additionally, it is possible to participate in the development process of Bro and bring in own ideas to this open source project.

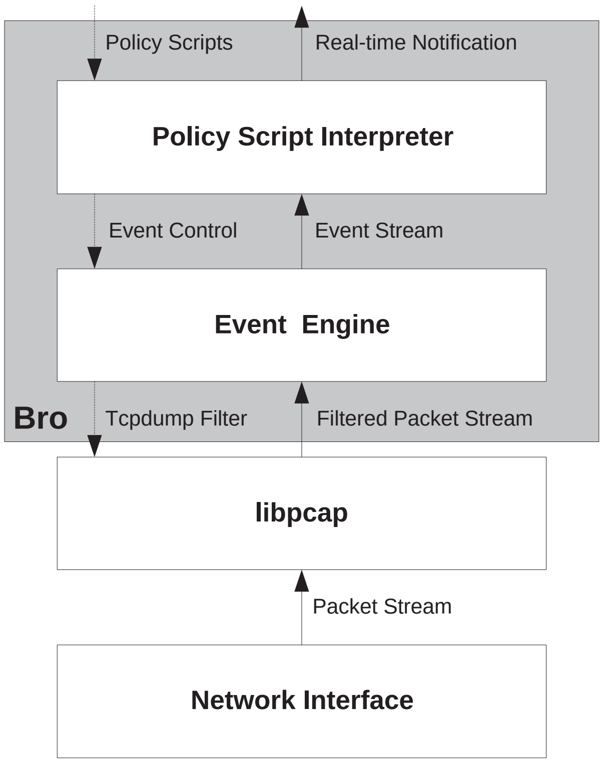

Figure 5.5: Bro Architecture [99]

Figure 5.5 shows the architecture of Bro. Bro is based on the *libpcap* library which makes it highly portable and let's Bro run on recorded tcpdump files instead of monitoring live traffic on a network interface. Additionally, libpcap can be instructed to pass only specific packets to Bro and thereby reduce the traffic load which Bro has to process. If the network load is too high to process all packets on the link Bro will automatically down sample the observed packet stream and process only a feasible amount of packets per second.
The gray box demonstrates how Bro abstracts the observed traffic on the network interface and reduces this traffic to a few meaningful events which can be handled by our own policy scripts. Bro is internally divided into two parts: The first part analyzes the packet stream from libpcap and extracts various occurring events like failed TCP connection attempts, ICMP unreachable packets or spontaneous TCP FIN packets. The second part is called the policy script interpreter and represents the interface to our own policy scripts. This interpreter loads our scripts when Bro is started.
The policy scripts are written in the Bro scripting language which supports special variable types like IP addresses, port numbers, time intervals and offers statements and expressions to react on events generated by the event engine. The Bro language also supports various data structures. For example, a table type is available which can hold entries and delete these in an automatic manner after a defined period.
In the policy files we can implement the behavior of our algorithm: We can count the occurrences of various events, do calculations, implement finite state machines, record information to files, generate notifications via *syslog* [30] and so on.
Bro works with many Unix systems, including Linux and Solaris, but has been primarily tuned for FreeBSD. The developers of Bro recommend using FreeBSD version 4.10. We installed it on the soft- and hardware like described in Section 3.3 and have positive experiences with it.

5.3.2 Scan Detection Policy Scripts

As stated before, the implementation of our algorithm was done by writing policy script files for the Bro IDS framework.
The scan detection architecture and the corresponding bro policy script files which contain the implemen-

tation are shown in Figure 5.6. The main file `osag-sd.bro` adjusts Bro internal settings and contains global variables and tables. Further, it includes all parameters which control the scan detection and loads the files shown in the second line of Figure 5.6.

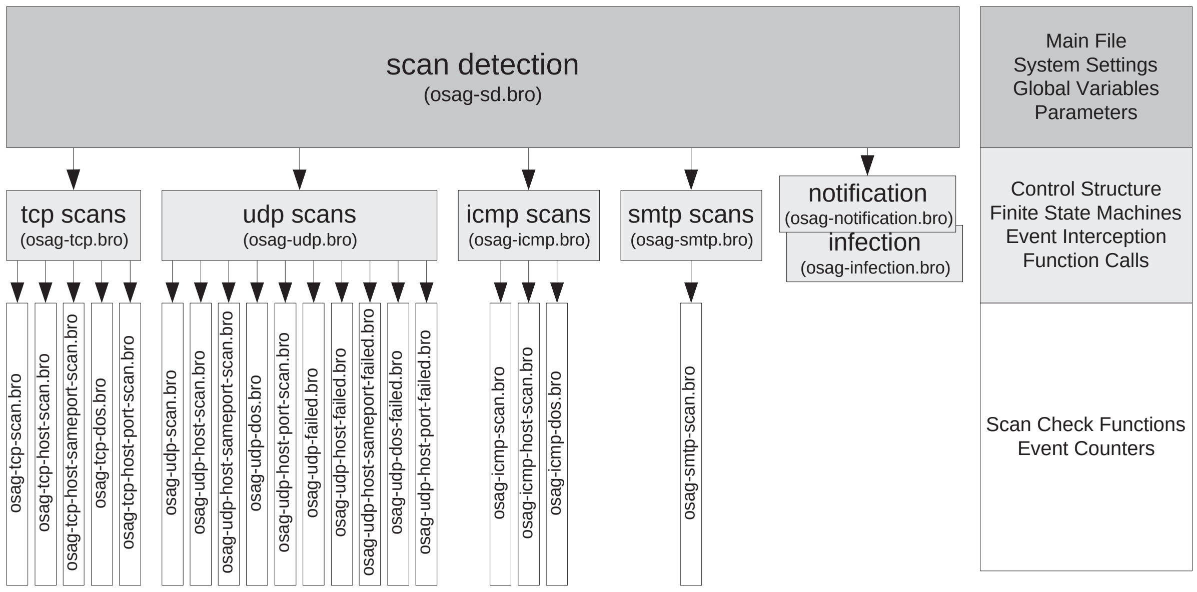

Figure 5.6: Scan detection architecture and corresponding files containing the implementation

In the four files `osag-tcp.bro, osag-smtp.bro, osag-udp.bro` and `osag-icmp.bro` the scan detection for the different protocols is implemented. This includes the interception of events, the subsequent calling of the corresponding functions and the finite state machines which control the state of each source host seen.
The functions for counting events and checking the behavior of the source hosts are implemented in the files showed in the third line of Figure 5.6. The exceedance of thresholds will be recognized here.
The notification when a host changes its state requires additional functions which are implemented in the file `osag-notification.bro`. This file provides functions for saving information about the behavior of a suspicious host and for writing this information to files or pass it to syslog. The script file `osag-infection.bro` provides functions which are called when a host is recognized as infected.

5.3.3 Catching events

The following paragraphs give an overview of the events which are caught and used by our scripts for counting several events like failed connection attempts. Some events are directly covered by Bro and others have to be abstracted and assembled from several different events.

TCP Events To count the failed connection attempts we intercept the Bro event `connection_attempt` which is triggered when a TCP SYN times out and the event `connection_rejected` which announces a TCP RST received from the firewall. Additionally, `conn_weird` is caught and tracked if it is caused by a spontaneous TCP FIN packet which does not belong to an active connection. This allows to detect TCP FIN and Xmas scans like described in Subsection 5.2.1.

UDP Events With UDP two cases have to be distinguished when detecting failed connection attempts: If a UDP packet is answered with an ICMP unreachable packet, it is clear that the connection failed. This case can be covered by intercepting the Bro event `icmp_unreachable` and by testing if it corresponds to a UDP packet. But if the firewall drops UDP packets and no ICMP unreachable packets occur it is more difficult to detect failed connection attempts. We count all UDP packets to hosts which do not send back a UDP packet within the defined UDP timeout. These UDP timeouts are implemented with the aid of the Bro events `udp_request`, `udp_reply` and the scheduler which is provided by the Bro policy script language.

ICMP Events A failed ICMP echo request can be recognized by catching `icmp_unreachable` events or by checking if an ICMP echo request (`icmp_echo_request`) is followed by an ICMP echo reply (`icmp_echo_reply`). The second case can be implemented again with a scheduler and a defined timeout parameter.

SMTP Events For SMTP we intercept the `connection_attempt` events and check if they correspond to a connection attempt on port 25.

5.3.4 Counting and Thresholds

This subsection describes the basic implementation concepts behind the counting of certain events and the calculation of scan rates.

Counting Events For all non-benign states we can simply count the respective events until the defined period is elapsed or the threshold is reached and a new state is assigned to the host. The case is more complicated for all benign states. It is necessary to successively count events over the last few minutes because a host stays in the benign state as long as it does not reach the threshold. Events are kept in a table indexed by source IP address and the time of occurrence. Each entry in this table expires after a defined time. For each host a counter exists which contains the actual number of events saved in the table. This concept allows that a host is recognized as infected whenever it exceeds a certain number of events within the last few minutes.

Calculating Scan Rates The scan rate of a host can be calculated by measuring the time until a host has reached a threshold. This approach causes two difficulties: If a host exceeds a threshold in very little time, the calculation can contain a rounding error which results in too high rates. A second problem concerns the previously described table with expiring entries which is used in benign states and with this concept it is not possible to detect when the scan of a host exactly started. Nonetheless, the calculated scan rates of a host give a meaningful clue as to how fast a host scans.

Exceeding Thresholds When a host exceeds a threshold it changes its state as described before. These changes and the time of occurrence are saved in a table for each protocol. When a host does not change its state for 24 hours, its table entry expires. Additionally, a list is maintained which contains hosts declared as infected. A Bro mechanism allows us to update the libpcap filter and block the packet stream of such an infected host on a lower level. This can save a lot of system resources.

5.3.5 Alerting

This subsection describes the output of our scan detection. Based on the host states in the flowchart messages are generated which can be used for further processing. Bro can write messages directly to syslog. Bro does not support the standard syslog levels and we have therefore introduced our own.
We have defined four types of messages, which are:

- *WARNING:* A warning is a message which tells that a worm or DoS has been detected. Hosts in the red shaded end states with the keyword worm and in the yellow shaded end states with the keyword DoS cause Bro to generate this kind of message.
- *SUSPICION:* If a host enters several times for example the state TCP_SCAN but never reaches an end state this host behaves suspiciously. We therefore generate a message of the type SUSPICION. Because the tests to identify an email worm are very simple, we declare a SMTP worm detection only as SUSPICION and not as WARNING. In future work the algorithm to detect email worms could be improved.
- *LOG:* All log messages are written to Syslog with the flag LOG. At the moment we log all changes of host states. Further logs can be added easily.
- *ERROR:* Internal checks ensure that the host does not enter a not allowed state which cannot been reached due to the flowchart diagram. If a host still reaches a not allowed state a message with the flag ERROR is generated.

The messages should contain all available information. Therefore, we designed a data structure which contains for each host and reached state the following information:

- Time when the state has been entered.
- Scanning rate of the host in this state.
- The number of times the state has been reached by this host.
- The port number in case of scans on the same port.
- A list of destination IP addresses in case of DoS attacks.

A function has been provided which can be used by all scan check functions (lowest third in Figure 5.6). This function changes the state and generates the messages dependent on the state of the current host and previous states of this host.

5.3.6 IP Spoofing

An infected host could send packets with faked source IP addresses. This behavior is called IP spoofing. In our solution we store for each source IP address the state and several table entries. Consequently, a scanning host sending packets which all have different source IP addresses causes a high memory and CPU consumption. Therefore, we have to control this difficulty.
The length of each host state table is tracked and warnings are written to syslog if a table exceeds a predefined length. This limit can be set to the expected size of networks.

5.3.7 Excluding Hosts

The scan detection allows to exclude hosts a priori from the observation. This can make sense if a host plays a special role in the network like a proxy or if a host has an unusual proprietary functionality.

Chapter 6

Tests and Results

The final implementation of scan detection is tested here. The test setups and results are given in this chapter. The tests are split into four parts. Section 6.1 shows the MACE tests which verify the functionality of the scan detection tool. The second part are the tests with real worm traffic. They are documented in Section 6.2. The tests with benign programs are documented in Section 6.3 and finally the Section 6.4 shows results from performance tests.
All tests have been done with the parameters from the specification given in the previous chapter.

6.1 Worm Simulation with MACE

The next four subsections describe test setups which represent infections of hosts in an office LAN. All these tests are realized using MACE.

6.1.1 Typical Worms

Table 6.1 shows a list of typical worm scan traffic. The list contains 23 combinations of elements from Table 3.2 in Subsection 3.2 and is mainly based on past worms.

Setup Each of the 23 scans have been running 10 minutes and each time a new instance of Bro has been started to detect this scan. The tests have taken place in the office LAN of the Open Systems AG (Section 3.3). The attacks have been generated on our simulator (SIM) and our scan detection has been installed on the detector (DET).
The source IP has not been spoofed and the source port number has been randomly chosen for each packet between 1024 and 65535. The TCP scans consisted of TCP SYN packets with no payload and the timeout which the host waits for an answer has been set to 3 seconds. For UDP we have chosen a payload of 376 bytes which corresponds to the SQL Worm (Subsection 2.3.4).
All 23 test scans have been conducted with two different firewall setups:

- Firewall rejects undeliverable packets.
- Firewall silently drops undeliverable packets.

However, two scans could not be tested with both firewall rules:

- ICMP could not be tested with a firewall rule which rejects ICMP echo requests[1].
- Scan number 22 produces too many traffic to test it in the office LAN of Open Systems. Therefore, number 22 has been generated and recorded on a single host on the local interface. The recorded dump has been replayed as described in Subsection 6.2.1 and obviously, it corresponds to a scan with silent drops.

Results Table 6.2 shows the scan rates which are measured by our scan detection and the time which it takes until a scan is detected and after which a host is declared as infected.
The two firewall setups have a major impact on the performance of the scan detection. As described in Subsection 2.2.1 a TCP worm scans noticeably faster when it receives rejects and does not have to await timeouts. In the case of UDP the scan rate is not effected by the setup, however the performance of the

[1]The reason why the firewall did not accept this rule is not known so far.

No.	Target selection	Protocol/Port		Speed
#1	totally random	ICMP		100 in parallel
#2	totally random	TCP	80	100 in parallel
#3	last 8 bits at random	TCP	80	100 in parallel
#4	last 16 bits at random	TCP	80	100 in parallel
#5	last 24 bits at random	TCP	80	100 in parallel
#6	60% totally + 40% last 16 bits at random	TCP	80	100 in parallel
#7	each 20th at random and then sequentially	TCP	80	100 in parallel
#8	totally random	TCP	135	10 in parallel
#9	totally random	TCP	135	50 in parallel
#10	totally random	TCP	135	100 in parallel
#11	totally random	TCP	135	1000 in parallel
#12	totally random	TCP	1-65535 (randomly)	100 in parallel
#13	totally random	UDP	1434	1000 packets/s
#14	last 8 bits at random	UDP	1434	1000 packets/s
#15	last 16 bits at random	UDP	1434	1000 packets/s
#16	last 24 bits at random	UDP	1434	1000 packets/s
#17	60% totally + 40% last 16 bits at random	UDP	1434	1000 packets/s
#18	each 20th at random and then sequentially	UDP	1434	1000 packets/s
#19	totally random	UDP	1434	1 packets/s
#20	totally random	UDP	1434	10 packets/s
#21	totally random	UDP	1434	100 packets/s
#22	totally random	UDP	1434	as many as possible
#23	totally random	UDP	1-65535 (randomly)	1000 packets/s

Table 6.1: Typical scan traffic

scan detection is. When the firewall rejects undeliverable UDP packets the scan detection immediately counts this as a failed connection attempt. Whereas in the dropping setup, the scan detection must wait for a given timeout before suspecting that the connection has failed (Subsection 5.2.2). Therefore, it takes longer to detect an infected host if the firewall silently drops undeliverable packets.
The tests have shown that worms with higher scan rates are detected faster. Scan number 19 sends only 1 packet per second and therefore scans too slow to be detected by the detection tool. Number 3 and 14 only scan internal hosts and absolutely no packet leaves the internal office network which means that we can not detect any traffic at the gateway to the Internet.
The conclusion can be made that we detect all scans which send packets to external hosts except number 19 which scans too slow. But this scan is not in our focus because it does not overload the network with its scan traffic.

6.1.2 Email Worms

Setup To test our email worm detection we have simulated an attack which tries to open SMTP connections to 10 different hosts. These connection attempts have been running sequentially and the sending host has been waiting maximal 30 seconds for an answer from the targeted email server. The tests have also been done in the office network of Open Systems.

Results If the firewall rejects the TCP SYN packets the simulated email worm can be detected in under a second. If the firewall silently drops the packets and the email worm waits 30 seconds until it performs another connection attempt, recognition is not possible because the failed attempts rate is too low.

6.1.3 DoS

Setup We have tested the detection of DoS attacks with two different MACE scripts. The first one simulates a UDP DoS attack on port 1434 and sends 1000 packets per second to four distinct destination hosts. The second one represents a TCP DoS attack on port 80 and tries to open 100 connections in parallel to four hosts.

	Firewall drops packets		Firewall rejects packets	
No.	Scan rate [p/s]	Detection time [s]	Scan rate [p/s]	Detection time [s]
#1	25	24.51	n/a	n/a
#2	34	9.21	485	0.57
#3	0	not detected	0	not detected
#4	32	9.17	168	1.27
#5	33	6.21	537	0.55
#6	34	9.25	310	0.77
#7	32	9.04	268	1.13
#8	3	90.75	114	1.85
#9	16	18.21	431	0.60
#10	32	6.27	481	0.59
#11	344	0.61	454	0.59
#12	26	318.13	462	301.00
#13	998	72.60	970	0.60
#14	0	not detected	0	not detected
#15	987	72.60	909	0.71
#16	993	72.60	990	0.60
#17	987	72.61	926	0.54
#18	996	72.60	993	0.60
#19	1	not detected	1	not detected
#20	10	132.61	10	60.10
#21	99	78.06	94	3.14
#22	8337	72.50	n/a	n/a
#23	714	372.87	977	300.00

Table 6.2: Detected sending rate and recognition time of an infected host with drops respectively rejects of the firewall

Results Table 6.3 shows the results of the DoS attack tests. When the firewall rejects the undeliverable packets, our scan detection recognizes the originator of the two DoS attacks within about 301 seconds. For UDP it takes a minute longer if the firewall silently drops the packets because there is an additional test in the state flow in this case. The scan detection measures a rate of about 1000 with UDP, respectively about 790 packets per second with TCP.

	Firewall drops packets		Firewall rejects packets	
	Scan rate [p/s]	Detection time [s]	Scan rate [p/s]	Detection time [s]
TCP DoS	790	301.65	794	301.64
UDP DoS	1000	361.50	1001	301.52

Table 6.3: Detected sending rate and recognition time of a simulated DoS attack with drops respectively rejects of the firewall

6.1.4 Several Infected Hosts

Setup The simulation of several infected hosts has been done by spoofing the source IP address of the sending simulator. IP spoofing does not have an impact on the scan detection. We simulate the parallel infection of 4, 32, 128 and 256 hosts and the scan scripts send 1000 TCP respectively UDP packets per second.
The infection of several hosts is challenging for our detection method because this causes a lot of resource intensive calculations. The resource consumption will be discussed later in this chapter.

Results Table 6.4 summarizes the results from the tests with several infected hosts. All scan rates and detection times are averages over the detection of all recognized infections. The total number of sent

packets by our simulator is more or less constant over all 10 attacks and therefore, the packet rates which are seen from a single host decreases the more IP addresses are spoofed. If we simulate 252 source host addresses the rate is divided by 252. If we send the packets from totally random source IP addresses the scan rate is under the threshold for TCP and also for UDP and no infection will be detected. Nonetheless, our scan detection generates the warning that too much hosts have been seen in the internal network and that IP spoofing is likely. In all other cases all 4, 32, 128 respectively 252 hosts are recognized and declared as infected.
The more infected source hosts we simulate the bigger become the differences of the detection times. For UDP, the first of 252 source hosts has been detected within about 118 seconds whereas the last one has taken 258 seconds until it has been detected.

		detection time [s]		
	scan rate [p/s]	average	min	max
4 infected hosts (TCP)	210	1.48	1.34	1.57
32 infected hosts (TCP)	29	10.39	9.08	11.30
128 infected hosts (TCP)	7	42.51	35.34	52.87
252 infected hosts (TCP)	3	98.85	77.16	137.32
randomly infected hosts (TCP)	<1	spoofing detected		
4 infected hosts (UDP)	245	2.10	2.06	2.14
32 infected hosts (UDP)	39	16.84	15.69	18.24
128 infected hosts (UDP)	8	68.21	58.09	79.18
252 infected hosts (UDP)	4	142.36	118.15	257.66
randomly infected hosts (UDP)	<1	spoofing detected		

Table 6.4: Detected average scan rates and minimal, maximal and average detection times

6.2 Tests with Real Worms

To test the scan detection tool as realistically as possible, the tool has been tested with real worm traffic. This section describes the setup and the results from these tests.

6.2.1 Setup

Tests have been done with two different worms:

- Blaster Worm (Subsection 2.3.3), scans on 135/TCP with 20 parallel threads
- SQL Worm (Subsection 2.3.4), scans on 1434/UDP as fast as possible

The SQL Worm has been installed in two different ways:

- Initial infector: The so called initial infector is used to initiate the worm spreading.
- Infected SQL server: An infected SQL server represents the typical case of an infected host during the spread of the SQL Worm.

These worms have been installed and captured with tcpdump by Thomas Dübendorfer in the *mobile Security Demo Lab* (mSDL at TIK at ETHZ [44]). None of the target hosts does reply with any packets.
Figure 6.1 shows the environment where the captures have been replayed on the simulator (SIM2). Scan detection has been installed on the detector (DET) which is directly connected to the simulator. The hardware for the detector is the same as in in Section 3.3. For the simulator the same hardware has been used as for the detector, this is different from the one used for the simulator in Section 3.3.

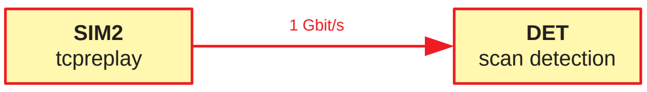

Figure 6.1: Test environment for real worm tests

6.2.2 Results

The Blaster worm has been detected after 57 seconds as stated in the following syslog message:

```
1110806167.815448 WARNING: 10.0.0.103 scanned 11 hosts/sec on port
135/tcp (TCP_HOST_SAMEPORT_SCAN, detection after 57.366312 sec;
TCP_SCAN: 3 packets/sec; TCP_HOST_SCAN: 11 hosts/sec)
```

According to the TCP specification (Subsection 5.2.1) a host has to send 300 packets on the same port until it is detected as infected. Blaster scans with 11 packets per second and therefore, we expect to detect it within approximately 27 seconds. Because the worm has started to scan at a much lower rate (approx. 3 seconds/sec) the host has entered the state TCP_SCAN after 40 seconds.
The initial infector of the SQL Worm has been detected after 83.9 seconds as stated in the following syslog message:

```
1110805912.479616 WARNING: 10.0.0.103 scanned roughly 291 hosts/sec
on port 1434/udp (UDP_HOST_SAMEPORT_SCAN, detection after 83.907368
sec; UDP_SCAN: 34 packets/sec; UDP_HOST_SCAN: 62 hosts/sec)
```

Similar to Blaster the SQL Worm starts slowly and then increases the scan rate. According to the UDP specification (Subsection 5.2.2) the infected host remains 60 seconds in the state UDP_SCAN and then waits another 12 seconds (UDP timeout) before the first scan detection internal timeout is triggered. The total detection time for this type of infection takes at least 72 seconds. The remaining 11.9 seconds are scan rate dependent.
The infected SQL server has scanned with a higher rate and has been detected after 74 seconds as stated in the following syslog message:

```
1111162860.664880 WARNING: 10.0.0.102 scanned roughly inf hosts/sec
on port 1434/udp (UDP_HOST_SAMEPORT_SCAN, detection after 74.865976
sec; UDP_SCAN: 14980 packets/sec; UDP_HOST_SCAN: 53 hosts/sec)
```

Due to rounding errors the rate could not be correctly calculated for all states. The infected host has been a few milliseconds in state UDP_HOST_SCAN, which is measured by Bro with 0 seconds. Consequently, this causes a rate of infinity (inf).

6.3 Benign Programs

In the previous sections the scan detection tool has been tested in various situations where a worm scan has been simulated or taken place. These tests show that the scan detection tool detects worm scans according to the specifications.
This section presents tests with benign programs to get an idea of the resistance to false positives. Tests have been done with customer traffic (Subsection 6.3.1), traffic from P2P programs (Subsection 6.3.2) and traffic from further programs which we think could cause the scan detection to generate false positives.

6.3.1 Customer Traffic

It is planned to use the scan detection tool in customer networks. Therefore, tests have been done with traffic from these networks.

Setup Traffic captures have been recorded with tcpdump in 15 internal networks of Open Systems customers. The test setup has been chosen similar to the one described in Section 6.2. Each dump has been replayed separately and Bro has been restarted after each dump file.
Table 6.5 lists all 15 dumps and gives additional information such as duration and size of each dump. Further, for each file the average used bandwidth is given. All files contain only benign traffic with the exception of dump 14 and dump 15 which contain worm scan traffic.

Name	File Information				Results		
	Duration [h:min:sec]	Packets /sec	Used bandwidth [Mbits/sec]	Worms in traffic	Worms found	Suspicions	False positives
dump 1	01:07:44	8.5	0.02	no	no	0	no
dump 2	01:07:34	21.6	0.02	no	no	0	no
dump 3	01:07:19	54.0	0.18	no	no	1 host 5 times: TCP_HOST_SCAN	no
dump 4	01:08:24	29.8	0.07	no	no	0	no
dump 5	01:07:02	19.8	0.07	no	no	0	no
dump 6	01:02:22	109.2	0.47	no	no	2 hosts 2 times, 4 hosts 3 times: TCP_SCAN	no
dump 7	01:02:11	9.1	0.04	no	no	0	no
dump 8	01:01:57	8.9	0.02	no	no	0	no
dump 9	01:01:45	11.8	0.04	no	no	1 host 5 times: ICMP_SCAN	no
dump 10	01:02:00	11.7	0.02	no	no	0	no
dump 11	01:25:03	200.0	1.03	no	no	0	no
dump 12	04:58:22	44.4	0.04	no	no	0	no
dump 13	04:55:59	7.0	0.01	no	no	0	no
dump 14	00:06:00	1372	1.51	yes	yes	0	no
dump 15	00:15:09	183.4	0.10	yes (3)	yes (3)	0	no
Total	22:28:51						

Table 6.5: Customer traffic tcpdumps with details and scan detection test results

Results The second part of Table 6.5 shows the test results. In dump 14 and dump 15 worms have been detected. These captures are the only ones containing worms. Dump 14 has been recorded in a network where one host was infected. The corresponding message sent to syslog is:

```
1108464484.687052 WARNING: 10.194.16.136 scanned 11 hosts/sec on port
135/tcp (TCP_HOST_SAMEPORT_SCAN, detection after 25.381164 sec;
TCP_SCAN: 14 packets/sec; TCP_HOST_SCAN: 11 hosts/sec)
```

Dump 15 has been recorded in a network with 3 infected hosts. The corresponding syslog messages are:

```
1111002591.658124 WARNING: 10.76.16.33 scanned 24 hosts/sec on port
445/tcp (TCP_HOST_SAMEPORT_SCAN, detection after 11.278947 sec;
TCP_SCAN: 47 packets/sec; TCP_HOST_SCAN: 20 hosts/sec)

1111002591.717333 WARNING: 10.76.16.176 scanned 20 hosts/sec on port
445/tcp (TCP_HOST_SAMEPORT_SCAN, detection after 11.358516 sec;
TCP_SCAN: 46 packets/sec; TCP_HOST_SCAN: 24 hosts/sec)

1111075474.851984 WARNING: 10.76.32.52 scanned 0 hosts/sec on port
445/tcp (TCP_HOST_SAMEPORT_SCAN, detection after 550.758097 sec;
TCP_SCAN: 1 packets/sec; TCP_HOST_SCAN: 0 hosts/sec)
```

The first two hosts have scanned with a significant higher rate than the third one.
The next column in this table shows how many hosts have been how many times in a suspicious state. As described in Subsection 5.3.5, hosts are defined as suspicious when they are several times in certain states. The corresponding syslog message is:

```
1105027141.041689 SUSPICION: host 10.172.16.233 was 5 times in state
TCP_HOST_SCAN within less than 1.0 day
```

The test results have shown that dump three, six and nine have caused suspicion messages and therefore are discussed here in more details.

- **dump 3:** One host in this network has sent TCP SYN packets to random target addresses and on random ports but has never received any answers. The host has reached the state TCP_HOST_SCAN five times. The sending rate is too low to be detected as worm scanning to different ports. Based on the SYN packets no conclusion can be made about the application which has sent these packets.
- **dump 6:** Five hosts have sent TCP SYN packets to a certain host on port 79. Therefore, these five host have reached several times the state TCP_SCAN. The sending rate is much lower then the threshold set for DoS attacks. These packets do certainly not belong to a worm, but no statement can be made about the application sending the packets.
- **dump 9:** One host in this network has sent on average three ICMP echo requests per second to one host. The rate is high enough to reach several times the state ICMP_SCAN, but to low to be detected as DoS attack. Interestingly, the two hosts have been communicating intensively and successfully with each other on TCP. A statement about the sending application can be made here neither.

The mentioned hosts show an abnormal behavior and load the network with their traffic. Therefore, it can be valued as success to have detected these hosts and to have declared them as suspicious. The traffic does not indicate any worm infection and therefore the message level SUSPICION has been correctly chosen instead of the level WARNING.
The scan detection has detected all worms in dump 14 and dump 15 and has not generated any false positives in all these tests with customer captures.

6.3.2 P2P Traffic

P2P networks and their clients are used to share and change files over the Internet. In these networks frequently files are shared which are copyrighted (e.g. songs in mp3 format) and it is therefore illegal to share these files.
However, the use of these clients is not illegal and therefore each company has to decide ifself if these programs are allowed in their networks. We assume them here as benign programs.
As discussed in Subsection 4.3.4, P2P clients scan for other clients using host lists. The clients on these lists hold further lists with other hosts, file lists or shared files. The scan for these clients is similar to a worm scan for targets. Depending on the list length and the scan rate hosts with P2P clients can cause the scan detection tool to generate suspicion messages or warnings.

Setup The tests have been done similar to the ones described in Subsection 4.3.4. All tested P2P clients (Freenet, Kazaa Lite, DC++, LimeWire and eMule) have been installed in the two different networks, once in the Open Systems network without Internet connectivity, once in the smaller network with Internet connectivity. The dumps have been recorded and replayed according to Figure 6.1.

P2P client	No Internet connection		With Internet connection	
	Duration	Results	Duration	Results
Freenet	n/a	n/a	00:04:19	no message
Kazaa Lite	00:10:39	SUSPICION: twice in state TCP_SCAN	01:20:19	no message
DC++	00:10:16	-	00:11:44	no message
LimeWire	00:11:46	-	00:17:26	no message
eMule	00:10:57	SUSPICION: twice in state TCP_SCAN	00:38:49	WARNING: host scanned on port 4662

Table 6.6: P2P traffic tcpdumps with and without Internet connection and corresponding results

The description of the dump files for each P2P client in both networks can be found in Table 6.6. Freenet can not be installed without an Internet connection. Consequently, a dump has only been recorded in the second network environment. When the clients have connected to their networks the user has searched for an mp3 file of Britney Spears and then downloaded the first found file.

Results Table 6.6 shows the test results. Kazaa Lite and eMule have generated suspicion messages in the first network environment. Both P2P clients have sent more than 100 TCP SYN packets several times to less than 100 different hosts. Consequently, they have entered several times the state TCP_SCAN. eMule has generated a warning in the second network environment, which is:

```
1105275155.894968 WARNING: 192.168.123.120 scanned 1 hosts/sec on
port 4662/tcp (TCP_HOST_SAMEPORT_SCAN, detection after 277.882139 sec;
TCP_SCAN: 1 packets/sec; TCP_HOST_SCAN: 1 hosts/sec)
```

When eMule is connected to the P2P network it receives lists with addresses of other clients and tries to connect to them by sending TCP SYN packets. eMule does this with a high rate and tries to contact over 300 different clients on the same port while downloading a file. Consequently, the scan detection tool reports this behavior with the warning printed above.
These P2P tests have shown that our tool detects Kazaa Lite and eMule when they are installed without Internet connectivity and reports them correctly with suspicion messages. The tool generates a warning while observing eMule in a connected P2P network. This case could be reported as a false positive pertaining to worm detection. However, from a network operations perspective, the detection of eMule can be rated as a success because of the large amount of scanning traffic it generates.

6.3.3 Other Traffic

Various programs exist which do not have bad intents but nevertheless generate failed attempts. A selection of them has been tested and is discussed here.

VoIP Voice over IP gains in importance [14] and will probably be used more intensive in the future. We have tested the freely available VoIP client Skype[2].
Both versions have been installed on a PC with Microsoft Windows 2000 (Service Pack 4). This host is situated in the office network of Open Systems AG. Scan detection has been installed on the detector according to Section 3.3. The results are similar for both versions and therefore, no difference is made in the discussion here. When Skype is started it tries to connect to the hosts on an internal list and therefore, sends TCP SYN packets to these hosts. Skype tries to connect to them once or twice on various ports.
In our test environment Skype 1.1 was not able to connect to the Skype network. It has tried to connect to 76 different hosts in 90 seconds each time it has been started. When the user starts Skype several times within a short time the host can reach the state TCP_HOST_SCAN but never will cause a false positive.
Skype 1.2 has connected to the network within 61 seconds and has reached the state TCP_SCAN. The second time Skype is started it connects in only 20 seconds to the Skype network but also reaches the state TCP_SCAN.
The traffic which is generated by Skype when it is able to connect to the network does not show any further failed connections.

Web Crawler The scan detection has been tested with the Copernic Agent Basic. The tcpdumps have been taken from the tests with web crawlers documented in Subsection 4.3.4. Both dump files have been replayed with the setup in Figure 6.1. Due to the low number of different search engines no messages have been generated with scan detection.

Video Conferencing Video conferencing over the Internet is often used for business purposes. These systems can communicate over both transport protocols and frequently use various ports. Wrongly configured systems could cause many failed attempts and could therefore be detected by the scan detection tool. During our tests no video conference traffic has been captured and consequently, no final statement can be given here.

[2]Skype version 1.1.0.79 and Skype version 1.2.0.21 [63].

Nmap Scans Nmap [51] is a utility for network exploration or security auditing. It can be used to scan a host for open ports or to scan an IP range for existing hosts. The tool uses TCP packets with various flags, UDP and ICMP echo requests to scan for hosts and open ports. Dependent on the chosen settings of the Nmap scan the scanning host is detected with the scan detection tool.

6.4 Performance Tests

The resource consumption of our scan detection with Bro is an important issue concerning the future use in the field. The consumption of CPU and memory mainly depend on the amount of scan traffic and the number of infected hosts in the observed network. This section shows the interesting cases when one or more hosts in the internal network are infected and are scanning with a high rate.
The number of infected hosts has a big impact on the resource consumption of the scan detection of Bro and therefore, we simulate different numbers of infected hosts. The attacks are realized as described in Subsection 6.1.4. The performance tests showed nearly the same results for UDP and TCP and therefore, all the following conclusions which are presented for TCP hold also for UDP.
The measurements are made on the detector hardware described in Section 3.3. The command `sar -x $bro_process_id 10 60` successively gives us the average percent of CPU which Bro has used during the last 10 seconds whereas the memory usage of the process is repeatedly read from `/proc/$bro_process_id/status`.
Figure 6.2 shows the CPU and memory usage when 4 hosts are infected. If up to 4 hosts are infected the scan detection needs less than 1% CPU time and less than 8 MB memory.

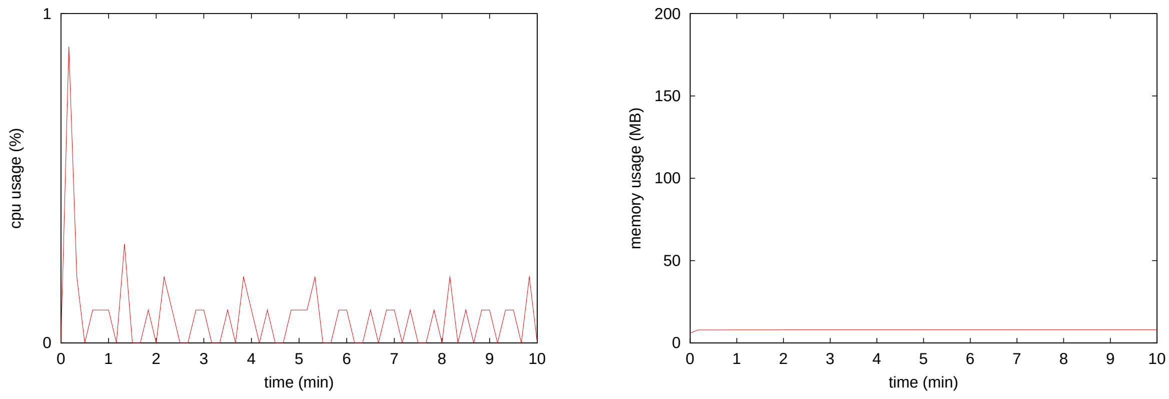

Figure 6.2: CPU and memory usage with 4 infected hosts (TCP worm)

When 32 hosts are infected the scan detection uses about 10% CPU for a short time. The peak appears at the time when the scan detection recognizes all 32 hosts, generates the messages and updates the libpcap filters. Afterwards, the CPU usage falls off to under 1%. The memory usage remains below 10 MB.

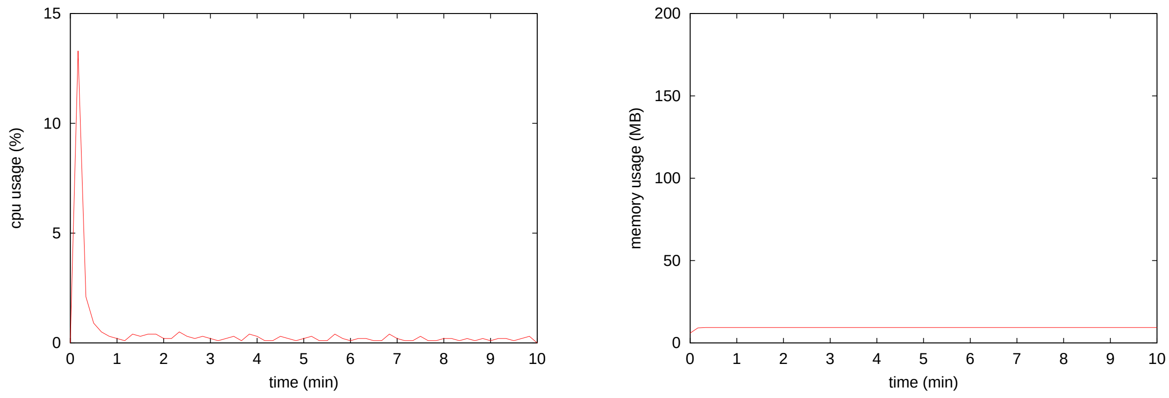

Figure 6.3: CPU and memory usage with 32 infected hosts (TCP worm)

If 128 infected hosts appear in the internal network the CPU usage rises to 37% and then goes back to almost 0% when all hosts are recognized and filtered on libpcap level. The used memory to save all source host states is about 14 MB.

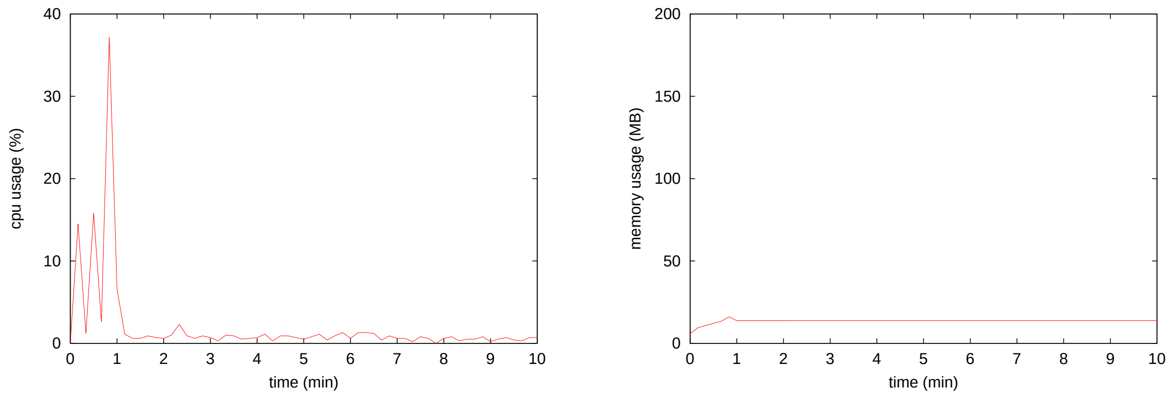

Figure 6.4: CPU and memory usage with 128 infected hosts (TCP worm)

The detection of 252 infected hosts uses up to 75% CPU during a minute. Figure 6.5 shows that after 2.5 min when all 252 infected hosts have been detected the CPU usage falls off. The memory usage does not exceed 20 MB.

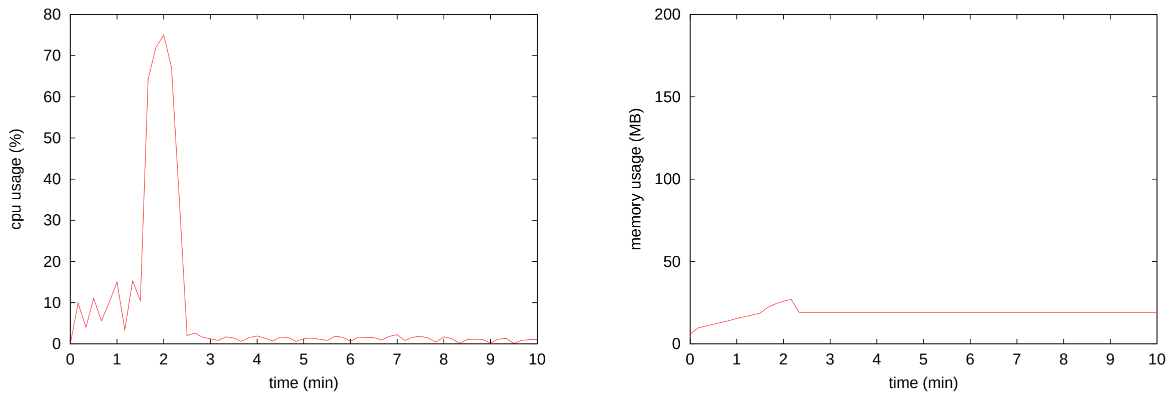

Figure 6.5: CPU and memory usage with 252 infected hosts (TCP worm)

Figure 6.6 shows what happens when the simulator sends 1000 TCP SYN packets per second with random source IP addresses. The scan rate of a single source IP address is too low for the scan detection to detect it. After 10 minutes 200 MB of the memory is exhausted because of the many table entries for all seen source hosts. At the moment, there is no mechanism implemented which prevents this exhaustive memory consumption. But our scan detection detects this kind of attack after 50 seconds as described in Subsection 6.1.4.
The last attack showed a vulnerability of our scan detection method and the need for future work. Various solutions are possible to prevent Bro from allocating all the memory. A memory limit could be externally set for example with the shell command `ulimit`. Further, a dynamic filter could be loaded into libpcap which totally stops the monitoring of the network traffic when the warning “IP spoofing” appears.
If we disregard the last attack, we can conclude that our requirements are fulfilled or topped. Even the infection of 252 hosts does not need more than 20 MB memory and this is clearly below the constraint of 100 MB from Subsection 5.1.1. The CPU usage is also quite low and increases only for short times.

6.5 Best Practice: Firewall Settings

The tests in this chapter have shown that the settings and the constellation of the observed office network have an important impact on the spreading and detection of a worm. We present here certain advices for the system administrators concerning the firewall settings based on the experience we gained during the tests.

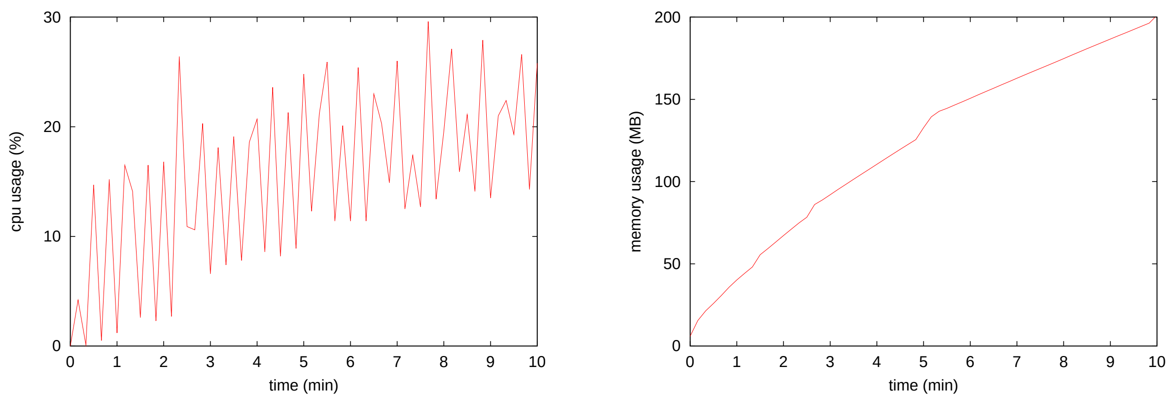

Figure 6.6: CPU and memory usage of a scan with random source IP addresses (TCP worm)

Subsection 2.2.1 describes that the spreading speed of a TCP worm highly depends on the timeout and the number of parallel connections. A worm normally has to await a lot of timeouts while scanning and therefore, has a low rate of connection attempts. When the firewall rejects undeliverable packets, the worm knows within milliseconds if a host is unreachable and can go on with scanning. A reject rule for TCP does significantly accelerate the spreading of a worm.
In contrast, a UDP worm scans without waiting for any incoming packets and the firewall rule does not have a direct impact on the infected host and the spreading of the worm. Nevertheless, the scan detection is based on counting failed UDP deliveries and therefore, rejects of the firewall can simplify this counting.
We can conclude this section with a short summary of the above statements:

- To slow down the spread of a TCP worm, the firewall should silently drop undeliverable packets.
- Rejects of undeliverable UDP packets can significantly increase the performance of the scan detection.

Chapter 7

Summary

7.1 Conclusion

Our task to implement a prototype of a scan detection is fulfilled. We worked out a way to detect worms based on their scan traffic. Extensive tests in the Open System VPN environment have shown promising results. Open Systems will follow up on this project and plans to introduce our solution for their customers in the near future.
Our prototype runs very stably and without any unpredictable behavior. We can detect all important worm scans and even more. Our scan detection additionally includes a DoS recognition and a simple email worm detection. In all tests we only got one false positive due to the P2P client eMule which shows a behavior very similar to worms.
The relatively low resource consumption of our scan detection with Bro is also very positive and the resource constraints could be fulfilled. Nevertheless, there are some memory issues concerning Bro which are discussed in Section 7.2.
Even though the scan detection satisfies its purpose very well, there are certain things which could be improved:

- The scan detection can not detect worms yet which spread over UDP and spoof the source IP address. The scan detection only notices an IP spoofing attack but can not find out the originator. The resulting drawback is described in Section 6.4.
- It also does not recognize stealth worms which scan slowly. But these also do not generate much traffic and therefore, were not in the main focus of this thesis.
- If one host is infected with more than one worm, scan detection can only detect one of these infections.

Additionally, we want to point out that our scan detection is not the solution forever. Future worms could use various (stealth) scan techniques which can not easily be detected. They also could use already available infrastructures like P2P or instant messenger networks to distribute themselves. This could mean that worms could spread without scanning at all.
We also notice a trend toward programs which scan with benign intentions. They scan the network to find out firewall settings and alike to support the user in configuring the network settings. The VoIP client Skype is a popular example of this technique. Although, we could not provoke a false positive in the tests, this sort of scanning could nevertheless become a problem in the future if benign programs behave and scan like worms.

7.2 Outlook

This section provides an outlook to describe further research topics and to show where improvements on the existing solution can be done.

7.2.1 System Integration and Graphical Interface

The scan detection has been implemented according to the specifications. Warning, log and error messages are written to syslog. No management tool has been developed to process these messages during

the thesis. The integration in some sort of management portal could be done in the future.
One possibility to show the activity of scan detection would be to show the state of each host on an overview page and actualize this successively.
Graphical tools exist to visualize the connections between hosts in a network. These tools could be used to visualize scan behavior of infected hosts. One of these tools is EtherApe [21].
Several gateways with scan detection could share information about detected worm infections. These details could be used to adapt firewall rules to prevent infections.

7.2.2 Scan Detection Enhancements

The scan detection tool has been developed in a first version according to the specifications. Various things could be improved:

- The detection of other scan attacks like TCP null scans (Subsection 5.2.1) is not covered yet.
- The SMTP algorithm is implemented in a very basic form and has to be improved in future work.
- The decision which is taken in an early step in the flowchart diagram of UDP causes the tool to observe either ICMP unreachable packets or to count UDP packets but not to observe both events at the same time which could yield better results in some firewall environments. In future work the algorithm could be adapted to observe both events in parallel but it would cause major changes for UDP worm detection.
- Worm attacks could be done using proxy settings from installed browsers and consequently, would not cause any failed attempts anymore. These kinds of attacks demand new detection methods.
- In the current implementation the traffic from a host is not observed anymore after an infection on this host has been detected. Additional information could be gathered by observing this host for a longer time. Further, it could be required that a host can be in several states at a time.
- The current implementation concentrates on the traffic of individual hosts and does not observe the total traffic of the whole link. It could be very informative to observe the link with a general detection tool as for example Entropy (Subsection 4.1.4).

7.2.3 Bro Enhancements

Bro has been released in Version 0.9a8 in February 2005. The developers are still working on various improvements. The most important issues which have to be solved concerning scan detection are listed here.

- The memory consumption of Bro is quite high. An entry in a table of booleans indexed by IP requires about 137 Bytes. The developers of Bro are trying to reduce this memory demand for future releases.
- Bro does not provide internal mechanisms to generate events which represent UDP and ICMP timeouts. These events have been implemented with the scripting language in scan detection which is less efficient.
- The timeout event on ICMP echo request is not provided by Bro.
- UDP packets from port 53, 111, 123 are always classified as replies, even if no packets have been sent before.
- The syslog mechanism which is provided by the current version of Bro does not allow to choose the syslog level.
- Bro does not support `for` or `while` loops to iterate a defined number of times or to repeat until a certain expression evaluates to true.

7.2.4 General

The scan detection tool is not resource intensive unless an infected host scans with random spoofed IP addresses. As explained in Section 6.4 there are several ways to observe and limit the memory consumption. One of the solutions should be realized in future work.
Various tests have been done with traffic dumps. Extensive long-term tests have to be done at customer locations to get a complete view of the occurrence of false positives and to be able to make a concluding statement about scan detection.

Appendix A

Acknowledgments

We would like to thank all the people who supported us during the last half year.
First of all, we want to thank Roel Vandewall for the many hours he spent, for his critical and constructive statements, the many helpful discussions and for the proofreading. Special thanks go also to Stefan Lampart which was responsible for the organization of this thesis. He did a good job, everything flowed very smoothly.
We are grateful for the support of Thomas Dübendorfer and Arno Wagner from the ETH. They gave us useful feedbacks and helped that this thesis took a good way. We are also thankful for the consent of Professor Bernhard Plattner which made this external thesis actually possible.
Additionally, we have to laud the whole Open Systems and its staff. It was a pleasant time here at Open Systems. The people here were very kind and helped us whenever needed.
The MACE tool of Joel Sommers from the University of Wisconsin-Madison was a great help and saved us a lot of time. We want to thank him for providing his tool. Big thanks go also to the developers of Bro for making available such a great tool. They always answered our emails and helped us by supporting patches.

Appendix B

CD-ROM Contents

This appendix describes the content on the compact disc which comes with this thesis.

- **Report:** A pdf- and a ps-version of the documentation. Additionally, all raw latex files and pictures of the report.
- **Final Presentation:** The slides of our final presentation at Open Systems AG and the ETH Zürich.
- **Task:** The formulation of our task as latex project, pdf- and ps-file.
- **Scan Detection:** The heart of our work. The scan detection policy scripts which have to be applied with Bro.
- **MACE:** The original MACE code and our extensions.
- **Bro:** The source code of the current Bro version 0.9a8 and some corresponding documentation from [6].
- **Published Detection Methods:** The four prototypical implementations of published detection methods and the results of the corresponding tests.
- **Tests:** Everything concerning the tests of scan detection. MACE simulation scripts and results.

Appendix C

Scan Detector Installation

The scan detection can be installed in very short time. Here is a quick start manual:

- Bro installation on a unix, bsd or linux:
  - Download the Bro tarball from the Bro website [6] and extract it.
  - Change to the new folder bro-x.xx and compile Bro with the standard unix commands (`./configure; make; make install`).
- Installation of the scan detection policy scripts:
  - The folder where Bro has been installed (default `/usr/local/bro/`) contains a folder named `<bro_path>/policy/` with all policy scripts in it. Most files in the folder `<bro_path>/policy/` can be deleted. Only the files `bro.init`, `pcap.bro`, `reduce-memory.bro` and all files ending with `*.bif.bro` are needed.
  - The scan detection policy scripts `osag-*.bro` have to be copied to `<bro_path>/policy/`. They can be found on the CD of this thesis or in the Internet [17].
- Starting scan detection:
  - Change to the bro-folder and start the scan detection with `./bin/bro -i ethX osag-sd`.
  - All warnings, logs and messages will be written to syslog.
  - The file `osag-sd.bro` contains all settings which control the behavior of the scan detection.

Scan detection has been tested on a linux host running with the SuSE 9.1 distribution (kernel 2.6) and Bro has been installed in version 0.9a8.

Appendix D

Schedule

This schedule has been planned at the beginning of this thesis. The numbers in the boxes correspond to the numbers in the task.

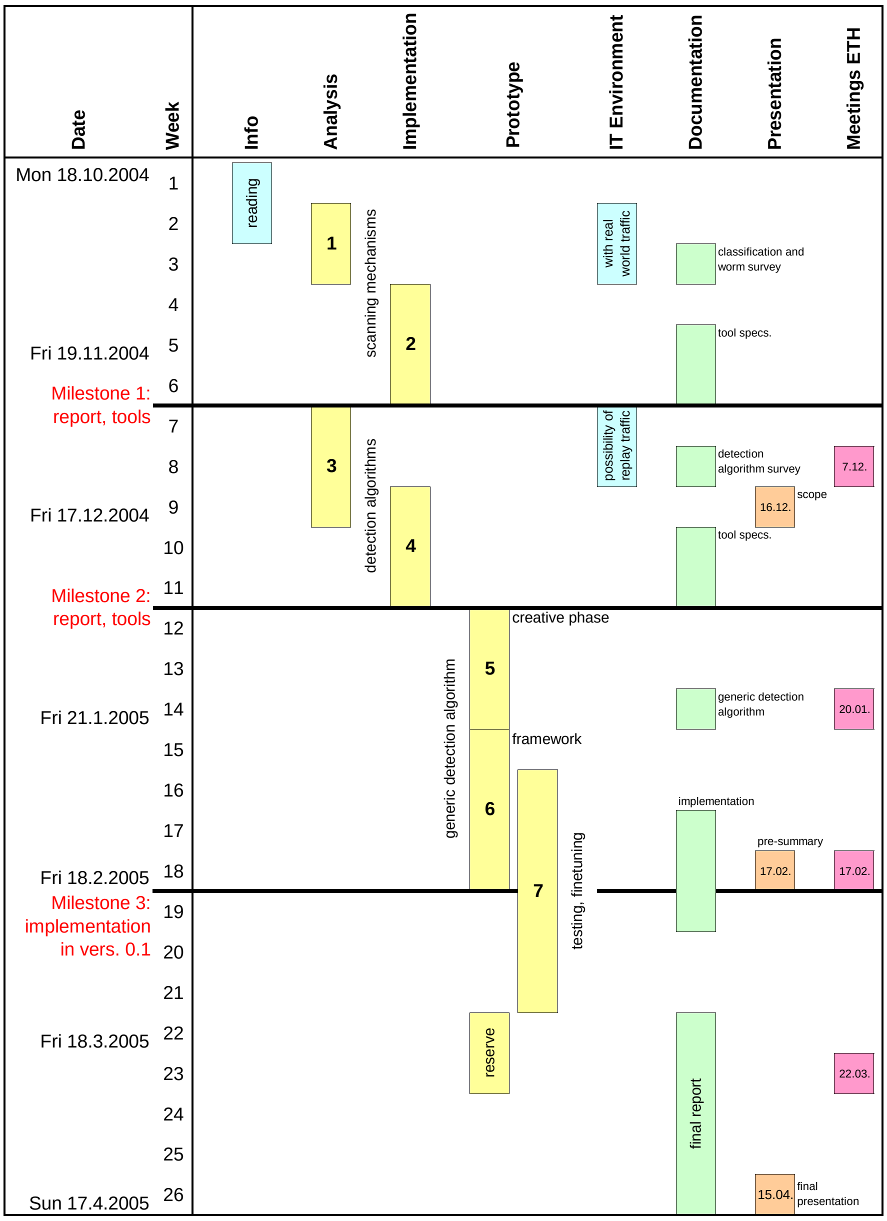

Figure D.1: Schedule

Bibliography

[1] *Address Resolution Protocol - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Address_Resolution_Protocol, Wikipedia - The Free Encyclopedia, 23.11.2004.

[2] Anukool Lakhina, Mark Crovella, Christophe Diot.
Diagnosing Network-Wide Traffic Anomalies
http://www.cs.cornell.edu/People/egs/syslunch-fall04/anomalies.pdf, Boston University, UK, September 2004.

[3] Arno Wagner, Bernhard Plattner.
Peer-to-Peer Systems as Attack Platform for Distributed Denial-of-Service
http://www.tik.ee.ethz.ch/˜wagner/papers/sact2002.ps, In ACM SACT Workshop 2002, Washington D.C., 2002.

[4] Arno Wagner, Bernhard Plattner.
Entropy Based Worm and Anomaly Detection in Fast IP Networks
http://www.tik.ee.ethz.ch/˜ddosvax/publications/, Swiss Federal Institute of Technology, Zurich, Switzerland, 2005.

[5] Bharath Madhusudan, John Lockwood.
Design of a System for Real-Time Worm Detection
http://www.hoti.org/hoti12/program/papers/2004/paper4.2.pdf, Washington University, St. Louis, August 2004.

[6] *Bro Intrusion Detection System*
http://bro-ids.org/, 2.12.2004.

[7] *CAIDA Analysis of Code-Red*
http://www.caida.org/analysis/security/code-red/, Caida.org, 18.04.2003.

[8] Carrie Gates, Michael Collins, Michael Duggan, Andrew Kompanek, Mark Thomas.
More Netflow Tools: For Performance and Security
http://flame.cs.dal.ca/˜gates/papers/lisa04.pdf, Atlanta, GA, November 2004.

[9] *CERT Advisory CA-2000-04 Love Letter Worm*
http://www.cert.org/advisories/CA-2000-04.html, CERT Coordination Center, Pittsburgh PA, 09.05.2000.

[10] *Classful network - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Classful_network, Wikipedia - The Free Encyclopedia, 24.11.2004.

[11] *Computer worm - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Computer_worm, Wikipedia - The Free Encyclopedia, 20.10.2004.

[12] *Copernic: Software to search, find, and manage information*
http://www.copernic.com, 27.01.2005.

[13] C. Leckie, R. Kotagiri.
A Probabilistic Approach to Detecting Network Scans
http://www.ee.mu.oz.au/staff/caleckie/noms2002.pdf, In Proceedings of the Eighth IEEE Network Operations and Management Symposium, April 2002.

[14] *c't 6/2005 magazin für computer technik*
http://www.heise.de, Heise Zeitschriften Verlag GmbH & Co. KG, Hannover, Germany, March 2005.

[15] *DC++ your files, your ways, no limits*
http://dcplusplus.sourceforge.net, 27.01.2005.

[16] *Denial-of-service attack - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Denial-of-service_attack, Wikipedia - The Free Encyclopedia, 03.03.2005.

[17] *DDosVax Project Homepage*
http://www.tik.ee.ethz.ch/˜ddosvax, Zurich, Switzerland, 20.10.2004.

[18] *Demilitarized zone (computing) - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Demilitarized_zone_(computing), Wikipedia - The Free Encyclopedia, 15.11.2004.

[19] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver.
The Spread of the Sapphire/Slammer Worm
http://www.cs.berkeley.edu/˜nweaver/sapphire/, University of California, Berkeley, 14.04.2003.

[20] *eMule-Project.net - Official eMule Site. Downloads, Help, Docu, News, ...*
http://www.emule-project.net, 27.01.2005.

[21] *EtherApe, a graphical network monitor*
http://etherape.sourceforge.net, 11.08.2004.

[22] Eugene H. Spafford.
The Internet Worm Program: An Analysis
http://www.cerias.purdue.edu/homes/spaf/tech-reps/823.pdf, Department of Computer Sciences, Purdue University, West Lafayette, IN, December 1988.

[23] *Internet service provider - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Isp, Wikipedia - The Free Encyclopedia, 30.11.2004.

[24] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan.
Fast Portscan Detection Using Sequential Hypothesis Testing
http://nms.lcs.mit.edu/papers/portscan-oakland04.pdf, In Proceedings of the IEEE Symposium on Security and Privacy, May 2004.

[25] Jamie Twycross, Matthew M. Williamson.
Implementing and testing a virus throttle
http://www.hpl.hp.com/techreports/2003/HPL-2003-103.pdf, HP Laboratories Bristol, May 2003.

[26] Jean-loup Gailly, Mark Adler.
The gzip home page
http://www.gzip.org, 2004.

[27] Jennifer Tan.
INTERVIEW-Trend Micro says 2003 viruses caused $55 bln damage
http://forbes.com/home/newswire/2004/01/16/rtr1214100.html, Forbes.com, January 2004.

[28] Jiang Wu, Sarma Vangala, Lixin Gao.
An Effective Architecture and Algorithm for Detection Worms with Various Scan Techniques
http://www-unix.ecs.umass.edu/˜lgao/ndss04.pdf, University of Massachusetts, 2003.

[29] Joel Sommers, Vinod Yegneswaran, Paul Barford.
A Framework for Malicious Workload Generation
http://www.cs.wisc.edu/˜jsommers/pubs/p82-sommers.pdf, University of Wisconsin-Madison, October 2004.

[30] Juergen Schoenwaelder.
syslog - Write messages to the system logger.
http://www.infodrom.org/projects/sysklogd/, 04.01.2001.

[31] Ke Wang, Salvatore J. Stolfo.
Anomalous Payload-based Network Intrusion Detection
http://www1.cs.columbia.edu/ids/publications/RAID4.PDF, RAID, September 2004.

[32] Krishna.
Scanning Networks
http://www.ebcvg.com/articles.php?id=152, 08.06.2003.

[33] *K++ / KaZaA Lite 2.6.1 Deutsch - MP3 Download Software - [MPeX.net]*
http://www.mpex.net/software/details/kazaalite.html, 27.01.2005.

[34] *LimeWire.org - Open Source P2P File Sharing*
http://www.limewire.org, 27.01.2005.

[35] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J.Wood, and D.Wolber.
A Network Security Monitor
In Proc. IEEE Symposium on Research in Security and Privacy, p. 296-304, 1990.

[36] Martin Roesch.
Snort - Lightweight Intrusion Detection for Networks
http://www.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf, Stanford Telecommunications. Inc., November 1999.

[37] Masaki Ishiguro, Hironobu Suzuki, Ichiro Murase, Hiroyuki.
Internet Threat Detection System Using Bayesian Estimation
http://h2np.net/clscan/archives/FIRST2004-Submit-paper.pdf, Mitsubishi Research Institute, June 2004.

[38] *Microsoft Security Bulletin MS01-033*
http://www.microsoft.com/technet/security/bulletin/MS01-033.mspx,
Microsoft Corporation, 31.01.2003.

[39] *Microsoft Security Bulletin MS02-039*
http://www.microsoft.com/technet/security/bulletin/MS02-039.mspx,
Microsoft Corporation, 31.01.2003.

[40] *Microsoft Security Bulletin MS03-026*
http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx,
Microsoft Corporation, 10.09.2003.

[41] *Microsoft Security Bulletin MS04-011*
http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx,
Microsoft Corporation, 10.08.2004.

[42] Miguel Vargas Martin.
Overview of Worms and Defence Strategies
http://www.scs.carleton.ca/˜mvargas/lecture95.4108-v1.0.pdf, School of Computer Sciences, Carleton University, Ontario, Canada, October 2003.

[43] *Morris worm - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Morris_worm, Wikipedia - The Free Encyclopedia, 06.10.2004.

[44] *mSDL - mobile Security Demo Lab*
http://www.csg.ethz.ch/research/projects/mSDL, 09.03.2005.

[45] M. Williamson.
Throttling Viruses: Restricting propagation to defeat malicious mobile code
http://www.acsac.org/2002/papers/97.pdf, HP Laboratories Bristol, June 2002.

[46] Neal Hindocha.
Threats to Instant Messaging
http://securityresponse.symantec.com/avcenter/reference/threats.to.instant.messaging.pdf, Symantec Security Response - White Paper, October 2002.

[47] *NetBEUI - Wikipedia*
http://de.wikipedia.org/wiki/NetBEUI, Wikipedia - The Free Encyclopedia, 25.11.2004.

[48] *NETBIOS - Wikipedia*
http://de.wikipedia.org/wiki/Netbios, Wikipedia - The Free Encyclopedia, 25.11.2004.

[49] *Network intrusion detection system - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Network_intrusion_detection_system,
Wikipedia - The Free Encyclopedia, 15.10.2004.

[50] Nicholas Weaver, Vern Paxson, Stuart Staniford, Robert Cunningham.
A Taxonomy of Computer Worms
http://www.cs.berkeley.edu/˜nweaver/papers/taxonomy.pdf, Washington, October 2003.

[51] *Nmap - Free Security Scanner For Network Exploration & Security Audits.*
http://www.insecure.org/nmap, 18.03.2005.

[52] Patrick Schnabel.
IEEE 802.3 / Ethernet / Fast Ethernet / Gigabit Ethernet / 10-Gigabit-Ethernet
http://www.elektronik-kompendium.de/sites/net/0603201.htm, Ludwigsburg,
Germany, 25.01.2005.

[53] *Peer-to-peer - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/P2p, Wikipedia - The Free Encyclopedia, 17.01.2005.

[54] *Published Advisories - ANALYSIS: Blaster Worm*
http://www.eeye.com/html/Research/Advisories/AL20030811.html,
eEye Digital Security, 11.08.2003.

[55] *Published Advisories - ANALYSIS: Sasser Worm*
http://www.eeye.com/html/research/advisories/AD20040501.html,
eEye Digital Security, 11.08.2003.

[56] *Published Advisories - ANALYSIS: .ida "Code Red" Worm*
http://www.eeye.com/html/research/advisories/AL20010717.html

[57] *Python Programming Language*
http://www.python.org, Stichting Mathematisch Centrum, Amsterdam, The Netherlands,
October 2004.

[58] *RFC 768 - User Datagram Protocol*
August, 1980.

[59] *RFC 793 - Transmission Control Protocol*
September, 1981.

[60] *RFC 1050 - RPC: Remote Procedure Call Protocol specification*
April, 1988.

[61] *Sasser shows there must be a better way*
http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci963170,00.html?track=NL-358&ad=482644, Searchsecurity.com, 10.05.2004.

[62] *Simple Network Management Protocol - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol,
Wikipedia - The Free Encyclopedia, 5.12.2004.

[63] *Skype - Free Internet telephony that just works*
www.skype.com, 17.03.2005.

[64] *Snort - The Open Source Network Intrusion Detection System*
http://www.snort.org/, 2.12.2004.

[65] Stefan Axelsson.
Intrusion Detection Systems: A Survey and Taxonomy
www.mnlab.cs.depaul.edu/seminar/spr2003/IDSSurvey.pdf, Chalmers University of Technology, Göteborg, Sweden, March 2000.

[66] Stuart Schechter, Jaeyeon Jung, Arthur W. Berger.
Fast Detection of Scanning Worm Infections
http://nms.lcs.mit.edu/papers/scanworm.pdf, 7th International Symposium on Recent Advances in Intrusion Detection (RAID), French Riviera, France, September 2004.

[67] Stuart Staniford, James A. Hoagland, Joseph M. McAlerney.
Practical Automated Detection of Stealthy Portscans
http://packetstormsecurity.nl/papers/IDS/spice-ccs2000.pdf, Eureka, CA, 2002.

[68] Stuart Staniford, Vern Paxson, Nicholas Weaver.
How to 0wn the Internet in Your Spare Time
http://www.icir.org/vern/papers/cdc-usenix-sec02/, In Proc. USENIX Security Symposium, 2002.

[69] *SWITCH Homepage*
http://www.switch.ch/, Zurich, Switzerland, 20.10.2004.

[70] *Symantec Security Response*
http://securityresponse.symantec.com/, Symantec Corp., Califorina, 25.10.2004.

[71] *Symantec Security Response - CodeRed Worm*
http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html,
Symantec Corp., California, 30.07.2004.

[72] *Symantec Security Response - VBS.LoveLetter and variants*
http://securityresponse.symantec.com/avcenter/venc/data/vbs.loveletter.a.html,
Symantec Corp., California, 31.05.2001.

[73] *Symantec Security Response - W32.Badtrans.B@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.badtrans.b@mm.html,
Symantec Corp., California, 30.07.2004.

[74] *Symantec Security Response - W32.Beagle.A@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.beagle.a@mm.html,
Symantec Corp., California, 27.07.2004.

[75] *Symantec Security Response - W32.Blaster.Worm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html,
Symantec Corp., California, 26.02.2004.

[76] *Symantec Security Response - W32.Bugbear@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.html,
Symantec Corp., California, 29.07.2004.

[77] *Symantec Security Response - W32.Gibe@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.gibe@mm.html,
Symantec Corp., California, 29.07.2004.

[78] *Symantec Security Response - W32.Klez.gen@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.klez.gen@mm.html,
Symantec Corp., California, 29.07.2004.

[79] *Symantec Security Response - W32.Mydoom.A@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.mydoom.a@mm.html,
Symantec Corp., California, 27.07.2004.

[80] *Symantec Security Response - W32.Netsky.D@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.netsky.d@mm.html,
Symantec Corp., California, 27.07.2004.

[81] *Symantec Security Response - W32.Netsky.P@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.netsky.p@mm.html,
Symantec Corp., California, 27.07.2004.

[82] *Symantec Security Response - W32.Nimda.A@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html,
Symantec Corp., California, 30.07.2004.

[83] *Symantec Security Response - W32.Sasser.B.Worm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.b.worm.html,
Symantec Corp., California, 27.07.2004.

[84] *Symantec Security Response - W32.Sircam.Worm@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.sircam.worm@mm.html,
Symantec Corp., California, 30.07.2004.

[85] *Symantec Security Response - W32.Sobig.F@mm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.sobig.f@mm.html,
Symantec Corp., California, 28.07.2004.

[86] *Symantec Security Response - W32.SQLExp.Worm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.sqlexp.worm.html,
Symantec Corp., California, 29.07.2004.

[87] *Symantec Security Response - W32.Welchia.Worm*
http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html,
Symantec Corp., California, 18.07.2004.

[88] *Symantec Security Response - W97.Melissa.A*
http://securityresponse.symantec.com/avcenter/venc/data/w97.melissa.a.html,
Symantec Corp., California, 26.10.2004.

[89] *Symantec Security Response - Wscript.KakWorm*
http://securityresponse.symantec.com/avcenter/venc/data/wscript.kakworm.html,
Symantec Corp., California, 26.10.2004.

[90] S. Staniford-Chen et. al.
GrIDS – A Graph Based Intrusion Detection System For Large Networks
http://www.cs.virginia.edu/˜jones/cs551S/slides/grids3.ppt, 1996.

[91] S. Staniford-Chen et al.
GrIDS-A Graph-Based Intrusion Detection System for Large Networks
http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper065/GRIDS.PDF,
University of California, Davis, CA, March 1996.

[92] *TCPDUMP public repository*
http://www.tcpdump.org, 22.06.2004.

[93] *The Freenet Project - index - beginner*
http://www.freenetproject.org, 27.01.2005.

[94] *The Perl Directory - perl.org*
http://www.perl.org, The Perl Foundation, 2004.

[95] Theus Hossmann.
Analysis of Sobig.F and Blaster Worm Characteristics
Semester Thesis, ftp://www.tik.ee.ethz.ch/pub/students/2004-So/SA-2004-23.pdf, Zurich,
September 2004.

[96] Thomas Toth, Christopher Kruegel.
Connection-history based anomaly detection
http://www.infosys.tuwien.ac.at/Staff/tt/publications/Connection_History_Based_Anomaly_Detection.pdf,
Technical University of Vienna, June 2002.

[97] Tom Vogt.
Simulating and optimising worm propagation algorithms
http://www.securityfocus.com/data/library/WormPropagation.pdf, Hamburg, September 2003.

[98] *Uniform Resource Locator - Wikipedia, the free encyclopedia*
http://en.wikipedia.org/wiki/Url, Wikipedia - The Free Encyclopedia, 24.10.2004.

[99] Vern Paxson.
Bro: A System for Detecting Network Intruders in Real-Time
http://www.ece.cmu.edu/˜adrian/731/readings/paxson99-bro.pdf,
Lawrence Berkeley National Laboratory, Berkeley, January 1998.

[100] Vern Paxson, Jim Rothfuss, Brian Tierney.
Bro User Manual
http://www.bro-ids.org/Bro-user-manual.pdf, Lawrence Berkeley National Laboratory, Berkeley, Juni 2004.

[101] Vern Paxson, Brian Tierney.
Bro Reference Manual
http://www.bro-ids.org/Bro-Ref-Manual.pdf, Lawrence Berkeley National Laboratory, Berkeley, December 2004.