Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Theus Hossmann

Analysis of Sobig.F and Blaster Worm Characteristics

Semester Thesis, SA-2004.23
April 2004 to September 2004

Tutor: Thomas Dübendorfer
Co-Tutor: Arno Wagner
Supervisor: Prof. Bernhard Plattner

Abstract

Distributed Denial Of Service (DDoS) attacks are a serious threat to the Internet infrastructure. The importance to fight this threat is growing, as more and more people and enterprises rely on working computer networks. Lately there has been a veritable flood of new worms, with the potential to be used for DDoS attacks.

In order to deal with this danger and be able to take countermeasures as early as possible, it is vital to understand how worms have spread in the past. In this semester theses we present a detailed analysis of the outbreaks and spreadings of two worms, *W32.Blaster* and *Sobig.F*. They are representatives of two different types of worms, Sobig.F is an E-Mail worm and Blaster was relying on a former vulnerability in Windows operating systems. The analysis is based on logged Cisco NetFlow data from the border gateway routers of a Swiss medium sized backbone network.

While Blaster shows heavy traffic increase from scanning for vulnerable hosts, we could observe only few successful infections going over the backbone network. Our analysis showed that because of its design, Blaster is much more successful in spreading within local networks than over the borders of networks. This poses a challenge for countermeasures in the backbone network. Before Blaster outbreak we have observerd some scanning traffic which probably originates from tests with the exploit code used by Blaster.

Filtering E-Mail worms as Sobig.F in a backbone network would require payload inspection of the transmitted packets. Because payload inspection requires a lot of resources this is not common practice in todays backbone networks. At the peak of Sobig.F activity we have observed an increase of 200 % of SMTP flows.

Contents

1 Introduction 4
1.1 Problem Description 4
1.1.1 Semester Theses Task 4
1.1.2 Computer Worms 4
1.2 The DDoSVax Project 4
1.2.1 Project Description 4
1.2.2 Cisco NetFlow 5
1.3 Approach 5
1.3.1 Related Work 5
1.3.2 Definition of Worm Patterns 6
1.3.3 Implementation 6

2 W32.Blaster 8
2.1 Introduction 8
2.2 Technical Details 8
2.2.1 Blaster Analysis 9
2.3 Blaster NetFlow Data Characteristics 10
2.3.1 Blaster Traffic 10
2.3.2 Infection Attempt Scenarios 11
2.3.3 Blaster Flows 13
2.4 Implementation 15
2.5 Traffic Analysis 16
2.5.1 Infection Attempts and Successful Infections 18
2.5.2 Successful Infections 19

3 Sobig.F 21
3.1 Introduction 21
3.2 Technical Details 21
3.2.1 Sobig.F Analysis 21
3.3 Implementation 22
3.4 Sobig.F Traffic Analysis 23
3.4.1 The Testbed 23
3.4.2 Observed Worm Transmissions 23
3.4.3 Flow Size Distribution 24
3.4.4 Flow Sizes 24
3.4.5 Number of Worm Transmissions 24

4 Conclusion and Outlook 28
4.1 Conclusion 28
4.1.1 Blaster 28
4.1.2 Sobig.F 28
4.2 Outlook 29
4.3 Acknowledgments 29

List of Figures

1 *SWITCH network and DDoSVax infrastructure* 5
2 *Blaster infection* 9
3 *Blaster infection packet flow (Part 1)* 11
4 *Blaster infection packet flow (Part 2)* 12
5 *Unsuccessful infection* 12
6 *Number of 'case 1' Blaster infection attempts from Aug 10th to Aug 15th* 20
7 *Number of 'case 2' Blaster infection attempts from Aug 10th to Aug 15th* 20
8 *Number of 'case 3' Blaster infection attempts from Aug 10th to Aug 15th* 20
9 *Number of 'case 4' Blaster infection attempts from Aug 10th to Aug 15th* 20
10 *Number of 'case 5' Blaster infection attempts from Aug 10th to Aug 15th* 20
11 *The testbed for Sobig.F* 23
12 *Dump of (part of) Sobig.F transmission* 25
13 *Dump of rejected Sobig.F transmission* 25
14 *Distribution of flow sizes (dest port 25/tcp, 0 - 500 bytes)* 26
15 *Distribution of flow sizes (dest port 25/tcp, 400 - 500 bytes)* 26
16 *Distribution of flow sizes (dest port 25/tcp, 70'000 - 130'000 bytes)* 26
17 *Distribution of flow sizes (src port 25/tcp, 500 - 6000 bytes)* 26
18 *Cumulative number of flow sizes over time* 27
19 *Number of Sobig.F transmissions over time* 27

List of Tables

1 *Subset of NetFlow v5 Fields (source: [8])* 6
2 *Overview of Blaster variants (source: Symantec [16])* 8
3 *Port 135/tcp flow characteristics* 14
4 *TCP Port 4444 flow characteristics* 15
5 *UDP Port 69 flow characteristics* 15
6 *Flows required for infection cases 1 - 5. 'A->V': Flow from attacker to victim, 'A<-V': Flow from victim to attacker.* 17
7 *Sample output file infip_STARTTIME_ENDTIME.out* 18
8 *Sample output file infec_STARTTIME_ENDTIME.out* 18
9 *Sobig.F message subject, body and attachment (source: Symantec [18])* 21
10 *Sample output file distr_STARTTIME_ENDTIME.out* 22

1 Introduction

1.1 Problem Description

1.1.1 Semester Theses Task

The goal of this semester theses, written in the context of the *DDoSVax project* (see Sec. 1.2.1), is to give a detailed analysis of two major outbreaks of worms, Sobig.F and W32.Blaster, by analyzing archived NetFlow data, collected in a medium-sized Internet backbone. In order to achieve this, we had to define in a first phase (specification) exact recognition patterns of the worm traffic. In a second phase (implementation) we extracted the relevant flows from the NetFlow data and in the validation phase we analyzed the plots for anticipated worm patterns.

1.1.2 Computer Worms

A computer worm is defined in [1] as

> ... a self-replicating computer program, similar to a computer virus. A virus attaches itself to, and becomes part of, another executable program; a worm is self-contained and does not need to be part of another program to propagate itself.

In the past few years several worms have became famous because of their large impact. Loveletter, Code Red, Blaster, Sobig and MyDoom are names which not only experts know, because they were widely covered in the press.

A *taxonomy* of worms has been published in [2]. Of the 5 criteria of worm characterization described in this taxonomy, we use 3 to classify the worms analyzed in this theses:

- the way how victims are discovered
- propagation carriers and distribution mechanisms
- the way the worm is activated.

W32.Blaster Blaster discovers its targets by *scanning sequential ranges* of IP addresses. It uses two channels for distribution. The exploit uses *Remote Procedure Call (RPC)* to start a shell for remote command execution. As a *secondary channel* it uses the *Trivial File Transfer Protocol (TFTP)* for worm code transmission. Blaster is *self-activating*, which means that no human interaction is needed to activate the worm, because the command to start Blaster is issued by the attacker to the remote shell.

Sobig.F Sobig.F is an E-Mail worm, detecting its victims from *E-Mail addresses stored on the hard disk drive*. As distribution mechanism it uses its own *Mail Transfer Agent (MTA)* to send itself as an attachment of a mail message to the victims. Sobig.F is *human-activated*, meaning that the user needs to execute the attachment in order to get infected.

1.2 The DDoSVax Project

1.2.1 Project Description

DDoSVax [3] is a joint project of ETH Zürich and SWITCH [4]. The objectives of the project is the detection of infection phases of DDoS attacks while infection is in progress, detection and analysis of massive DDoS attacks when they start in near real-time and the provision of methods and tools that support countermeasures during both, infection and attack, phases.

The SWITCH network and the DDoSVax infrastructure are shown in Fig. 1. The SWITCH network has four border gateway routers, on which Cisco NetFlow (see Sec. 1.2.2) data is collected. The SWITCH network (AS559) connects all Swiss universities, various research labs, federal technical colleges and colleges of higher education, to the Internet. It carries about 5 %

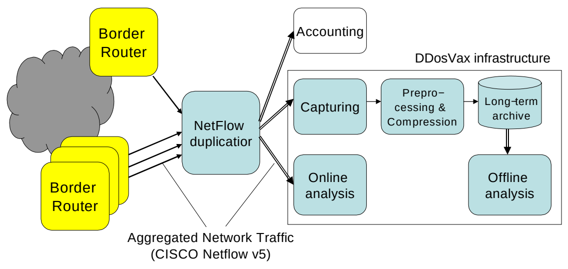


Figure 1: *SWITCH network and DDoSVax infrastructure*

of the Swiss Internet traffic.

For near real-time online analysis, the NetFlow data is forwarded to the DDoSVax data capturer and to the DDoSVax "UPFrame" framework [5]. For the offline analysis of past events, the NetFlow data from the border gateway routers gets logged. Archives range back to April 2003, with only a few missing hours caused by equipment maintenance. For this theses we have used the archived data from August 2003.

For the offline analysis the DDoSVax team operates a Linux cluster called "Scylla" [6], with 22 identical nodes plus one gateway node, providing enough computation power to analyze the large amounts of data.

1.2.2 Cisco NetFlow

The traffic statistics for the DDoSVax project are collected with Cisco NetFlow. A good introduction to NetFlow can be found at [7]. There are different levels of aggregation on which information about the traffic in a network can be gathered. The highest level of aggregation, providing highly concentrated, low volume data is *SNMP*. SNMP can provide for instance information about how much traffic goes through an interface, but it is not possible to see the source or destination addresses of the traffic. The lowest level, providing very detailed, high volume data, is the *packet level*. Packet level can be used to get information on every single transmitted packet. The drawback of collecting information on such a low level is, that it needs a huge amount of storage space, when used for long-term archives. In between these levels, being a trade-off between the level of detail and the data volume, traffic statistics can be collected on a *flow level*.

Flows In Cisco NetFlow, a *flow* is defined as a unidirectional stream of packets from one host to another. A flow, according to [8] is identified by the tuple of source IP address, source port, source interface, destination address, destination port, IP protocol and IP type-of-service. For instance a TCP connection consists of two flows, one in each direction.
There are different conditions, which cause the flow to be terminated: no packets are transmitted for a certain time, the duration exceeds a maximum time, the flow contains a FIN or RST packet or the router cache expires.

Table 1 summarizes the most important fields, which are reported in Cisco NetFlow v5 for each flow.
Relevant for this theses are mainly the pairs of addresses and ports, number of packets, number of octets and the timing information.

1.3 Approach

1.3.1 Related Work

It is easy to find information about worm behavior from anti-virus enterprises. The worm analysis we have used in this theses are mainly from Symantec [9].

Field name	Description
prot	Protocol
srcaddr	Source address
dstaddr	Destination address
input	Input interface
output	Output interface
dPkts	Number of packets
dOctets	Number of octets
First	Start of NetFlow
Last	End of NetFlow
srcport	Source port
dstport	Destination port
tcp_flags	TCP flags
tos	IP type-of-service

Table 1: *Subset of NetFlow v5 Fields (source: [8])*

Analysis of worm generated network traffic are more difficult to find, because data as we use them in the DDoSVax project are rare. The reason for the lack of comprehensive archives of network traffic may be privacy laws, data security concerns and the high costs of maintaining large archives.

For Blaster there are some analysis of traffic, for instance in [10] Jose Nazario from Arbor Networks presents some plots of measured Blaster traffic, including analysis of Blasters effects on routing. In [13] Symantec shows an analysis of the infection rate during the first days of Blaster activity.

1.3.2 Definition of Worm Patterns

For the definition of worm patterns we relied mainly on three sources:

- The worm analysis of anti-virus enterprises to gain an overview of what is sent over the network.
- A testbed we have set up. We have dumped the network traffic with a packet sniffer (Ethereal) and analyzed the worm traffic. As the testbed looks differently for Blaster and Sobig, the according setup is presented in more detail in the particular sections.
- In the archives we have isolated infected hosts and analyzed their traffic for worm patterns.

With these information we gained a clear overview of how the traffic of the worms looks and we were able to define patterns for filtering the NetFlow records.

1.3.3 Implementation

The implementation of our analysis can be divided in three parts.

Worm Traffic Isolation For collecting the worm relevant flows out of the huge amount of logged NetFlow data, we have implemented scanning tools in C language. The scanning tools are based on the `netflow_iterator_template3` tool, written by Arno Wagner for the DDoSVax project. For Blaster traffic, the scanning tool not only extracts the relevant flows, but also matches the flows to connections and one or more connections to infection attempts.
As the scanning of the archive is consuming a lot of time and resources, the statistic files generated by the scanning tools are held as detailed as possible and have to be aggregated further for making plots.

Aggregation The statistic files from the scanning tools are fed to aggregation scripts written in `awk` [11]. These scripts are plot specific and generate statistic files which can be used directly for plotting.

Plots The plotting part of the implementation was done with `gnuplot` [12].

Variant	Program file name	Program size	Discovered	Comments
A	msblast.exe	6176 Bytes	08/11/2003	The most wide spread variant.
B	penis32.exe	7200 Bytes	08/13/2003	A man was arrested in August 2003 suspected of having released the B variant.
C	index.exe root32.exe teekids.exe	32045 Bytes 19798 Bytes 5360 Bytes	08/13/2003	Variant C comes in a package with a backdoor trojan.
D	mspatch.exe	11776 Bytes	09/01/2003	Not observed in the wild.
E	mslaugh.exe	6176 Bytes	08/28/2003	
F	enbiei.exe	11808 Bytes	09/01/2003	
K	mschost.exe	6688 Bytes	02/03/2004	
T	eschlp.exe svchosthlp.exe	28169 Bytes 66048 Bytes	04/21/2004	

Table 2: *Overview of Blaster variants (source: Symantec [16])*

2 W32.Blaster

In this section we describe the W32.Blaster worm in detail. First, in Section 2.1 we put together some general information about Blaster, when and how it first appeared and what was its impact. In Section 2.2 we describe how the worm works technically. As the aim of this work is to analyze Internet traffic data collected with NetFlow, we define the characteristics of traffic generated by Blaster in Section 2.3. The results we achieved with these traffic patterns are presented and discussed in Section 2.5 of this report.

2.1 Introduction

The Microsoft Windows DCOM RPC Vulnerability In July 2003 the "Last Stage of Delirium Research Group" published in [14] a report of a buffer overrun in the Microsoft Windows Remote Procedure Call (RPC) Interface. The RPC protocol defines inter-process communication to allow code execution on remote systems. This vulnerability affected Windows NT4.0, Windows 2000, Windows XP and Windows 2003 operating systems and was rated by Microsoft in a Security Bulletin [15] as critical because it allows the execution of arbitrary code over the Internet. As a consequence Microsoft made a patch available which fixed the overrun.

The Blaster Worm On August, 11th the Blaster worm was first discovered in the wild. It uses the above described RPC vulnerability to spread and infects computers running Windows 2000 and Windows XP operating systems. As no universal rules of how worms and viruses are named exist, different companies give them different names. W32.Blaster.A (Symantec) is also known as W32/Lovesan.worm.a (McAffee), Win32.Poza.A (CA), Lovesan (F-Secure), WORM_MSBLAST.A (Trend), W32/Blaster-A (Sophos), W32/Blaster (Panda) or Worm.Win32.Lovesan (KAV).

The total number of infections since outbreak vary in a wide rang from between 200’000 and 500’000 (Internet Storm Center) and 8’000’000 (Microsoft, April 2004).

Besides the A version of Blaster there has appeared a number of variants and today there are still new ones discovered in the wild. They all use the same exploit code, but the worm code is different. An overview of all variants according to Symantec [16] is shown in Table 2.

2.2 Technical Details

The way Blaster works is described in detail in [17]. This analysis of Blaster is based on the assembly code and concurs with our observation of Blaster infections described in Section 2.3.

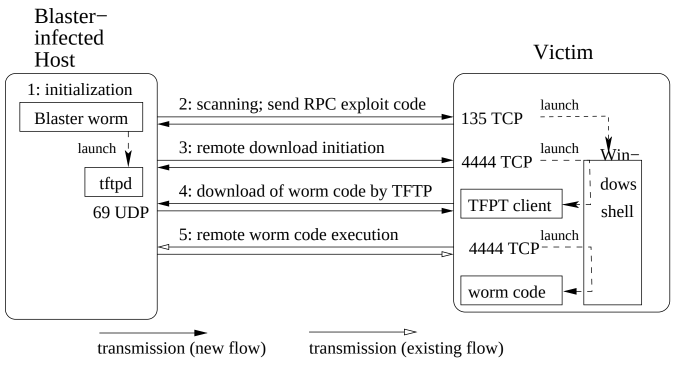


Figure 2: *Blaster infection*

The following description is based on that analysis and holds for *Blaster.A*, though all other variants work very similar, only some of them have some additional features.

2.2.1 Blaster Analysis

The runtime of the Blaster code can be divided in different phases. Phases 1 to 5 are shown in Figure 2.

Phase 1: Initialization The first thing Blaster does when the code is executed is making a registry key in `HKLM\Software\Microsoft\Windows\CurrentVersion\Run` called "windows auto update" with string value "msblast.exe". This registry entry causes Windows to execute the *Blaster code at Boot time*.
Then Blaster makes a mutex called "BILLY" to prevent from multiple infections and tries to initialize Winsock and make a connection to the Internet. If the connection is successfully set up it checks the date. If the current day is the 16th or later or if the current month is from September to December it starts TCP SYN flooding windowsupdate.com with a spoofed source address which consists of the two first bytes of the local address and the two last bytes generated by random.
This attempt of a DDoS attack against windowsupdate was not successful because windowsupdate.com is not the real domain name of the windows update server but only forwarded to windowsupdate.microsoft.com. All Microsoft had to do was to stop that forwarding and the windows update functionality was not affected.
Also in the initialization phase Blaster decides whether it wants to infect Windows 2000 hosts or Windows XP hosts. This decision is randomized, with 80% probability Windows XP is chosen, with 20% Windows 2000. These two possibilities differ by a return address sent with the exploit code described below.
The last step in the initialization phase is to generate an IP address from which scanning will start. This address is with probability of 60% *completely random* (the first three bytes) with the last byte set to zero, with 40% probability it uses the *first two bytes of the local address*, the third byte is also taken from the local address, but if it is greater than 20 a random number from 0 to 19 is subtracted from it. The last byte is set to zero.

Phase 2: Scanning, Send RPC Exploit Code After initialization, Blaster tries to infect other hosts. Phases 2 - 5 are repeated endlessly with *IP addresses growing*, starting with the address created at initialization.

Scanning for vulnerable hosts is done by trying to set up a TCP connection to port 135 (Windows RPC port). Blaster always scans blocks of 20 sequential IP addresses simultaneously.

For the IP addresses where the TCP connection was set up successfully, Blaster tries to send the code exploiting the DCOM RPC vulnerability described in 2.1. If this code is successfully transmitted and the victim is vulnerable, it binds a Windows command shell to port 4444/tcp for remote command execution.

Phase 3: Remote Download Initialization If the Exploit code was transmitted successfully, the attacking host tries to connect to port 4444. If that connection setup is successful, it first starts a TFTP server thread on the local machine and issues a command to download the Blaster code by TFTP (Trivial File Transfer Protocol) to the remote shell on port 4444/tcp.

TFTP servers listen by default to UDP port 69. Windows contains by default a TFTP client, which Blaster uses with the command `tftp -i IP GET msblast.exe` where `IP` is the address of the attacking host.

Phase 4: Download of Worm Code If that command is successful, the victim downloads msblast.exe and after this transmission or after a timeout of 20 seconds, the TFTP server on the attacker is shut down.

Phase 5: Remote Worm Code Execution If the worm code was downloaded successfully, the attacker issues the command to the victim to start msblast.exe. The start command is sent over the same 4444/tcp connection as the download command. If this command is successful, the victim is infected and starts itself to infect other hosts.

2.3 Blaster NetFlow Data Characteristics

2.3.1 Blaster Traffic

In order to define and find exact patterns in the archived NetFlow data, we had to observe the traffic generated by Blaster infections and infection attempts. We did so in a small testbed with two hosts, an attacker and a victim, and captured the IP traffic between them. Both hosts were running Windows XP, first without the security patch, after that with the patch applied in order to see the traffic of an unsuccessful infection attempt. The hosts both had IP addresses in the 192.168.0.0 Class C network, the attacker was 192.168.0.2, the victim was 192.168.0.3

Without Security Patch With the victim being vulnerable we performed 10 infection attempts (msblast.exe was started 10 times on the attacker). After each successful infection we cleaned and rebooted the victim host. With these 10 attempts we observed the following situations:

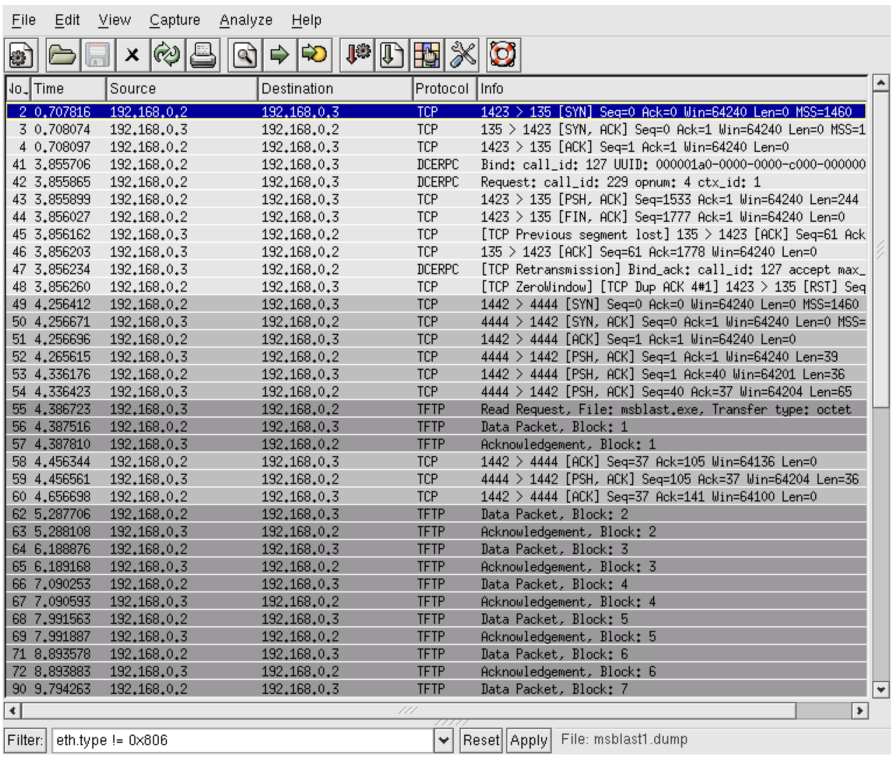
- Scan local IP range, Infection (3/10)
- Scan local IP range, no Infection (1/10)
- Scan random IP range (5/10)
- No scan, DNS Query to windowsupdate.com (1/10)

The dump of captured packets in the first case is shown in Figure 3 and Figure 4. We can see three connections, one for the exploit code, one for the command issue and one for the TFTP transfer. They are marked in different gray tones.

In the second case the exploit code was transmitted, but the connection to port 4444/tcp was refused by the victim. We assume that in this case the Windows 2000 exploit code was sent, to which Windows XP is not vulnerable.

The scan with a random start IP happened in 50% of the attempts, which is close to the 60% indicated by the analysis.

The last case where no scanning was performed at all, only periodic attempts to resolve the name of windowsupdate.com is somehow strange, because according the analysis of the assembly code even if the DDoS attack is performed Blaster should still try to infect other hosts.



```
No. Time     Source       Destination  Protocol Info
 2 0.707816  192.168.0.2  192.168.0.3  TCP      1423 > 135 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
 3 0.708074  192.168.0.3  192.168.0.2  TCP      135 > 1423 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1
 4 0.708097  192.168.0.2  192.168.0.3  TCP      1423 > 135 [ACK] Seq=1 Ack=1 Win=64240 Len=0
41 3.855706  192.168.0.2  192.168.0.3  DCERPC   Bind: call_id: 127 UUID: 000001a0-0000-0000-c000-000000
42 3.855865  192.168.0.2  192.168.0.3  DCERPC   Request: call_id: 229 opnum: 4 ctx_id: 1
43 3.855899  192.168.0.2  192.168.0.3  TCP      1423 > 135 [PSH, ACK] Seq=1533 Ack=1 Win=64240 Len=244
44 3.856027  192.168.0.2  192.168.0.3  TCP      1423 > 135 [FIN, ACK] Seq=1777 Ack=1 Win=64240 Len=0
45 3.856162  192.168.0.3  192.168.0.2  TCP      [TCP Previous segment lost] 135 > 1423 [ACK] Seq=61 Ack
46 3.856203  192.168.0.3  192.168.0.2  TCP      135 > 1423 [ACK] Seq=61 Ack=1778 Win=64240 Len=0
47 3.856234  192.168.0.3  192.168.0.2  DCERPC   [TCP Retransmission] Bind_ack: call_id: 127 accept max_
48 3.856260  192.168.0.2  192.168.0.3  TCP      [TCP ZeroWindow] [TCP Dup ACK 4#1] 1423 > 135 [RST] Seq
49 4.256412  192.168.0.2  192.168.0.3  TCP      1442 > 4444 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
50 4.256671  192.168.0.3  192.168.0.2  TCP      4444 > 1442 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=
51 4.256696  192.168.0.2  192.168.0.3  TCP      1442 > 4444 [ACK] Seq=1 Ack=1 Win=64240 Len=0
52 4.265615  192.168.0.3  192.168.0.2  TCP      4444 > 1442 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=39
53 4.336176  192.168.0.2  192.168.0.3  TCP      1442 > 4444 [PSH, ACK] Seq=1 Ack=40 Win=64201 Len=36
54 4.336423  192.168.0.3  192.168.0.2  TCP      4444 > 1442 [PSH, ACK] Seq=40 Ack=37 Win=64204 Len=65
55 4.386723  192.168.0.3  192.168.0.2  TFTP     Read Request, File: msblast.exe, Transfer type: octet
56 4.387516  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 1
57 4.387810  192.168.0.3  192.168.0.2  TFTP     Acknowledgement, Block: 1
58 4.456344  192.168.0.2  192.168.0.3  TCP      1442 > 4444 [ACK] Seq=37 Ack=105 Win=64136 Len=0
59 4.456561  192.168.0.3  192.168.0.2  TCP      4444 > 1442 [PSH, ACK] Seq=105 Ack=37 Win=64204 Len=36
60 4.656698  192.168.0.2  192.168.0.3  TCP      1442 > 4444 [ACK] Seq=37 Ack=141 Win=64100 Len=0
62 5.287706  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 2
63 5.288108  192.168.0.3  192.168.0.2  TFTP     Acknowledgement, Block: 2
64 6.188876  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 3
65 6.189168  192.168.0.3  192.168.0.2  TFTP     Acknowledgement, Block: 3
66 7.090253  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 4
67 7.090593  192.168.0.3  192.168.0.2  TFTP     Acknowledgement, Block: 4
68 7.991563  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 5
69 7.991887  192.168.0.3  192.168.0.2  TFTP     Acknowledgement, Block: 5
71 8.893578  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 6
72 8.893883  192.168.0.3  192.168.0.2  TFTP     Acknowledgement, Block: 6
90 9.794263  192.168.0.2  192.168.0.3  TFTP     Data Packet, Block: 7
```

Filter: eth.type != 0x806 File: msblast1.dump

Figure 3: *Blaster infection packet flow (Part 1)*

The date when we carried out these tests was the 28th of April and lies in the range where DDoS attack should take place.

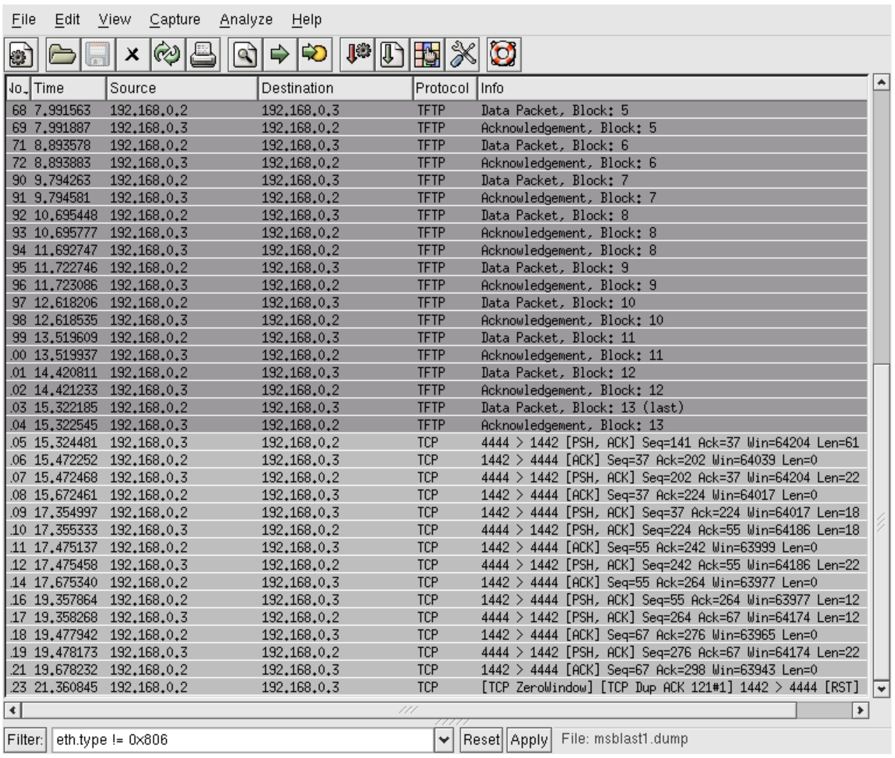
With Security Patch In the case where the victim had the security patch applied, the infection attempt was not successful. The exploit code was transmitted, but as the patch fixed the RPC vulnerability and prevented from the buffer overrun, no shell was bound to port 4444 and the connection attempt to port 4444 was refused by the victim. The dump of the packet flow is shown in Figure 5.

2.3.2 Infection Attempt Scenarios

In order to classify Blaster infection attempts we defined 5 different cases, which are distinguishable in the NetFlow records:

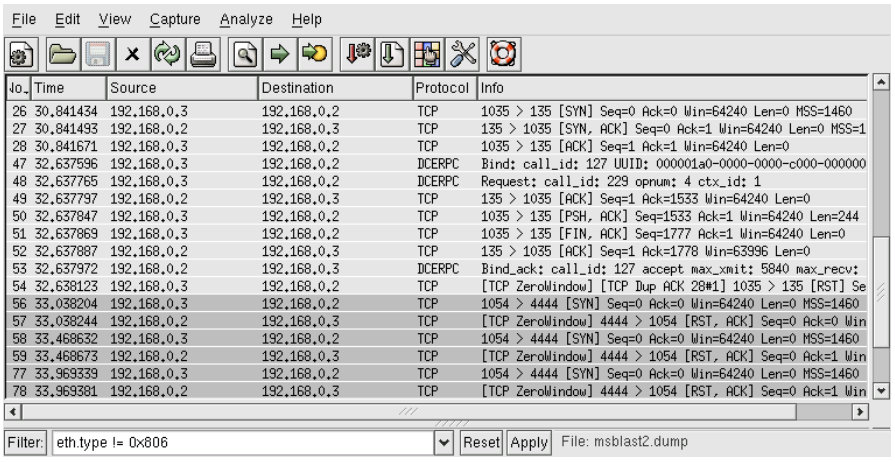
1. Victim does not exist or does not respond
2. Victim responds but is not vulnerable (port 135/tcp is not open)
3. Victim is vulnerable but the code for the wrong operating system is transmitted or the security patch is applied[1]
4. Victim is vulnerable and exploit code is successfully transmitted to port 135/tcp but TFTP server does not respond[2]
5. Infection is successful

[1]These two cases are not distinguishable in the NetFlow records.
[2]If TFTP traffic is blocked by a firewall or TFTP server was shut down or is congested.


File Edit View Capture Analyze Help
Filter: eth.type != 0x806
Reset
Apply
File: msblast1.dump

No.	Time	Source	Destination	Protocol	Info
68	7.991563	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 5
69	7.991887	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 5
71	8.893578	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 6
72	8.893883	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 6
90	9.794263	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 7
91	9.794581	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 7
92	10.695448	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 8
93	10.695777	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 8
94	11.692747	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 8
95	11.722746	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 9
96	11.723086	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 9
97	12.618206	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 10
98	12.618535	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 10
99	13.519609	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 11
.00	13.519937	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 11
.01	14.420811	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 12
.02	14.421233	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 12
.03	15.322185	192.168.0.2	192.168.0.3	TFTP	Data Packet, Block: 13 (last)
.04	15.322545	192.168.0.3	192.168.0.2	TFTP	Acknowledgement, Block: 13
.05	15.324481	192.168.0.3	192.168.0.2	TCP	4444 > 1442 [PSH, ACK] Seq=141 Ack=37 Win=64204 Len=61
.06	15.472252	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [ACK] Seq=37 Ack=202 Win=64039 Len=0
.07	15.472468	192.168.0.3	192.168.0.2	TCP	4444 > 1442 [PSH, ACK] Seq=202 Ack=37 Win=64204 Len=22
.08	15.672461	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [ACK] Seq=37 Ack=224 Win=64017 Len=0
.09	17.354997	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [PSH, ACK] Seq=37 Ack=224 Win=64017 Len=18
.10	17.355333	192.168.0.3	192.168.0.2	TCP	4444 > 1442 [PSH, ACK] Seq=224 Ack=55 Win=64186 Len=18
.11	17.475137	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [ACK] Seq=55 Ack=242 Win=63999 Len=0
.12	17.475458	192.168.0.3	192.168.0.2	TCP	4444 > 1442 [PSH, ACK] Seq=242 Ack=55 Win=64186 Len=22
.14	17.675340	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [ACK] Seq=55 Ack=264 Win=63977 Len=0
.16	19.357864	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [PSH, ACK] Seq=55 Ack=264 Win=63977 Len=12
.17	19.358268	192.168.0.3	192.168.0.2	TCP	4444 > 1442 [PSH, ACK] Seq=264 Ack=67 Win=64174 Len=12
.18	19.477942	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [ACK] Seq=67 Ack=276 Win=63965 Len=0
.19	19.478173	192.168.0.3	192.168.0.2	TCP	4444 > 1442 [PSH, ACK] Seq=276 Ack=67 Win=64174 Len=22
.21	19.678232	192.168.0.2	192.168.0.3	TCP	1442 > 4444 [ACK] Seq=67 Ack=298 Win=63943 Len=0
.23	21.360845	192.168.0.2	192.168.0.3	TCP	[TCP ZeroWindow] [TCP Dup ACK 121#1] 1442 > 4444 [RST]

Figure 4: *Blaster infection packet flow (Part 2)*


File Edit View Capture Analyze Help
Filter: eth.type != 0x806
Reset
Apply
File: msblast2.dump

No.	Time	Source	Destination	Protocol	Info
26	30.841434	192.168.0.3	192.168.0.2	TCP	1035 > 135 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
27	30.841493	192.168.0.2	192.168.0.3	TCP	135 > 1035 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1
28	30.841671	192.168.0.3	192.168.0.2	TCP	1035 > 135 [ACK] Seq=1 Ack=1 Win=64240 Len=0
47	32.637596	192.168.0.3	192.168.0.2	DCERPC	Bind: call_id: 127 UUID: 000001a0-0000-0000-c000-000000
48	32.637765	192.168.0.3	192.168.0.2	DCERPC	Request: call_id: 229 opnum: 4 ctx_id: 1
49	32.637797	192.168.0.2	192.168.0.3	TCP	135 > 1035 [ACK] Seq=1 Ack=1533 Win=64240 Len=0
50	32.637847	192.168.0.3	192.168.0.2	TCP	1035 > 135 [PSH, ACK] Seq=1533 Ack=1 Win=64240 Len=244
51	32.637869	192.168.0.3	192.168.0.2	TCP	1035 > 135 [FIN, ACK] Seq=1777 Ack=1 Win=64240 Len=0
52	32.637887	192.168.0.2	192.168.0.3	TCP	135 > 1035 [ACK] Seq=1 Ack=1778 Win=63996 Len=0
53	32.637972	192.168.0.2	192.168.0.3	DCERPC	Bind_ack: call_id: 127 accept max_xmit: 5840 max_recv:
54	32.638123	192.168.0.3	192.168.0.2	TCP	[TCP ZeroWindow] [TCP Dup ACK 28#1] 1035 > 135 [RST] Se
56	33.038204	192.168.0.3	192.168.0.2	TCP	1054 > 4444 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
57	33.038244	192.168.0.2	192.168.0.3	TCP	[TCP ZeroWindow] 4444 > 1054 [RST, ACK] Seq=0 Ack=0 Win
58	33.468632	192.168.0.3	192.168.0.2	TCP	1054 > 4444 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
59	33.468673	192.168.0.2	192.168.0.3	TCP	[TCP ZeroWindow] 4444 > 1054 [RST, ACK] Seq=0 Ack=1 Win
77	33.969339	192.168.0.3	192.168.0.2	TCP	1054 > 4444 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
78	33.969381	192.168.0.2	192.168.0.3	TCP	[TCP ZeroWindow] 4444 > 1054 [RST, ACK] Seq=0 Ack=1 Win

Figure 5: *Unsuccessful infection*

2.3.3 Blaster Flows

According to the packet flow analysis in Section 2.3.1 we can make some statements about how the NetFlow records of infected hosts look. The number of packets being sent from the attacker to the victim and vice versa and their size are important characteristics to identify the NetFlow records.

Additionally to the information taken from the packet flow analysis we identified a host generating a lot of Blaster traffic in the NetFlow data of 12th August 2003, the date where Blaster distribution reached its climax. The records of this host gave us some more information of how Blaster flows look.

TCP Port 135 The typical flow properties (number of transmitted packets and their size from attacker to victim and vice versa) for all of the above defined infection attempt cases are listed in Table 3. There are two important observations in this table. First, for each case there are different possibilities of the number of packets. The reason is that some packets are sent several times because of packet loss or too short timeouts. Most flows with a difference of one packet differ by 40 or 48 Bytes in length, which is an indication for a repetition of the TCP SYN packet[3]. Another reason could be that packets are fragmented. The second observation is that there are always several flow sizes for the same packet number, which differ by 8 bytes in length (separated with a "/" in the table). It seems that there are two different TCP SYN packets in use. Normally a TCP packet without data and the SYN flag set has a size of 40 bytes (20 Bytes IP Header + 20 Bytes TCP Header). In some TCP SYN packets there is a TCP option field of 8 bytes added with an indication of the maximum segment size accepted. In the table the package number and size of the flows we observed in Section 2.3.1 are printed in bold. The flow properties not printed in bold appeared in the NetFlow data of the host identified as Blaster infected.

Note that cases 1 and 2 can not be uniquely identified as Blaster flows as they look the same as any other program unsuccessfully trying to access port 135/tcp of another host.

With the knowledge about size and number of packets of the 135/tcp flows we have introduced the following two filters for the NetFlow fields (see Table 1 for the meaning of the field names):

```
prot == TCP && dstport == 135 &&
((dOctets == 40 && dPkts == 1)  || (dOctets == 48 && dPkts == 1)   ||
(dOctets == 80 && dPkts == 2)   || (dOctets == 88 && dPkts ==2)    ||
(dOctets == 96 && dPkts == 2)   || (dOctets == 120 && dPkts == 3)  ||
(dOctets == 144 && dPkts == 3)  || (dOctets == 2056 && dPkts == 7) ||
(dOctets == 2064 && dPkts == 7) || (dOctets == 2096 && dPkts == 8) ||
(dOctets == 2104 && dPkts == 8) || (dOctets == 2136 && dPkts == 9) ||
(dOctets == 2144 && dPkts == 9) || (dOctets == 2176 && dPkts == 10) ||
(dOctets == 2184 && dPkts == 10)|| (dOctets == 2216 && dPkts == 11) ||
(dOctets == 2224 && dPkts == 11)|| (dOctets == 2256 && dPkts == 12) ||
(dOctets == 2264 && dPkts == 12))

prot == TCP && srcport == 135 &&
((dOctets == 40 && dPkts == 1) || (dOctets == 48 && dPkts == 1)  ||
(dOctets == 80 && dPkts == 2)  || (dOctets == 88 && dPkts == 2)  ||
(dOctets == 96 && dPkts == 2)  || (dOctets == 120 && dPkts == 3) ||
(dOctets == 144 && dPkts == 3) ||
(dOctets >= 148 && dOctets <= 448 && dPkts >=3 && dPkts <=6))
```

TCP Port 4444 The identification of TCP port 4444 flow characteristics is more difficult than with TCP port 135 flows. This is on one hand due to the fact that 4444 is not a well known port,

[3]The TCP SYN packet is a minimal sized TCP packet composed of only 20 bytes IP header and 20 bytes TCP header

Case	Attacker -> Victim		Victim -> Attacker	
	# Packets	Flowsize (Bytes)	# Packets	Flowsize (Bytes)
1	1 2 3	40/**48** 80/88/96 120/144	- - -	- - -
2	1 2 3	40/48 80/88/96 120/144	1 2 3	40/48 80/88/96 120/144
3	**7** 8 9 10 11 12	2056/**2064** 2096/2104 2136/2144 2176/2184 2216/2224 2256/2264	**3** - 6	148 - 448 **(188)**
4	**7** 8 9 10 11 12	2056/**2064** 2096/2104 2136/2144 2176/2184 2216/2224 2256/2264	**3** - 6	148 - 448 **(188)**
5	same as case 4			

Table 3: *Port 135/tcp flow characteristics*

that means it can be used by any application as source port for outgoing connections. On the other hand the behavior of the shell bound to port 4444 can differ for the same situation. This means that for instance for a successful command the shell sends some status messages over the network which are either sent as one message or split up in several messages. Sometimes the TCP port 4444 connection is split in 2 flows (per direction) because there can be a gap of 20 seconds between the issue of the download command and the execution command. This splitting is controlled by the `inactive_timeout` variable [4], which determines when a flow is considered as completed.

Anyway for *case 3* (wrong exploit code sent or system is patched) characteristic numbers of packets and flow lengths can be defined. This is shown in *Table 4*. For the case where the commands are issued successfully we could only define boundaries of the number of packets and Bytes in the flow.

For the 4444/tcp flows we have applied the following filters:

```
prot == TCP && dstport == 4444 &&
((dOctets >=40 && dOctets <= 188 && dPkts >= 1 && dPkts <= 3) ||
(dOctets >= 200 && dOctets <= 700 && dPkts >= 3 && dPkts <= 15))

prot == TCP && srcport == 4444 &&
((dOctets == 40 && dPkts == 1) || (dOctets == 80 && dPkts == 2) ||
(dOctets ==120 && dPkts == 3) ||
(dOctets >= 200 && dOctets <= 800 && dPkts >= 3 && dPkts <= 12))
```

UDP Port 69 If a UDP flow with source or destination port 69 (TFTP) has the same source and destination IP addresses as the TCP flows on port 135 and 4444, it is with high probability generated by Blaster. We distinguished two cases: either there is only one flow from the victim to the attacker, unsuccessfully trying to download Blaster, or there are flows in both directions

[4]This variable defines the maximum inactivity time of a flow at the router before it expires. (It is currently set to 30s at the SWITCH routers)

Case	Attacker -> Victim		Victim -> Attacker	
	# Packets	Flowsize (Bytes)	# Packets	Flowsize (Bytes)
1	no flow		no flow	
2	no flow		no flow	
3	1 - 3	40 - 188	1 2 3	40 80 120
4	3 - 15	200 - 700	3 - 12	200 - 800
5	same as case 4			

Table 4: *TCP Port 4444 flow characteristics*

Case	Attacker -> Victim		Victim -> Attacker	
	# Packets	Flowsize (Bytes)	# Packets	Flowsize (Bytes)
1	no flows			
2	no flows			
3	no flows			
4	no flow		1 - 10	40 - 450
5	13	6592	14	464

Table 5: *UDP Port 69 flow characteristics*

with specific number of packets and flow lengths as indicated in Table 5.

For 69/udp flows we have applied the following filters:

```
prot == UDP && srcport == 69 &&
(dOctets == 6592 && dPkts == 13)

prot == UDP && dstport == 69 &&
((dOctets >= 40 && dOctets <= 450 && dPkts >= 1 && dPkts <= 10) ||
(dOctets == 464 && dPkts == 14))
```

2.4 Implementation

For the scanning and aggregation of the NetFlow data we have written a tool called `blaster_analysis` in C language.

Parameters The programme takes a list of log files as parameter. As the DDoSVax infrastructure generates two log files for every hour, one file with the log of the router with the majority of the traffic, the other file with an aggregation of the logs of the three other routers, both files have to be in the list of the input files and are processed in parallel. The list of input files has to be in chronological order, first the list of the files from the first router and then the list of the files from the other routers.

Scanning The scanning part of the programme traverses the records in the input files sequentially and writes all records matching the filters described in Section 2.3.3 into hashed tables. We use separate tables for each connection (135/tcp, 4444/tcp and 69/udp) and direction (in total 6 hashed tables).

Because sometimes the packets are routed through two routers, duplicate records with a small time difference appear in the logs. For each new flow matching the filters, the according hashed table is searched for a record with the same address and port pair. If the search is successful and the difference of the start times of the flows is smaller than 50 ms, the new packet is considered to be such a duplicate packet and is not inserted into the hashed table.

After 5 minutes of scanned traffic, `blaster_analysis` tries to match the records in the hashed tables to infection cases 1 - 5 (see Section 2.3.2). This is done by matching the IP addresses and ports for each connection. Furthermore we check that the start time of every flow lies within a time interval of 10 seconds starting with the first 135/tcp flow. This time restriction should reduce the probability of counting a sequence of flows not generated by Blaster but showing the same port numbers as Blaster traffic.

Because the NetFlow infrastructure sometimes loses records, in some cases not all flows have to be recorded to match an infection attempt case. For instance, if all flows are present for a successful infection except the 135/tcp flow from victim to attacker, the infection is still counted as successful. Table 6 shows a list of the flows which have to be present for each of the infection cases.

Output `blaster_analysis` generates two output files. The first is called `infip_STARTTIME_ENDTIME.out`, where STARTTIME is the Unix timestamp[5] of the first input file and ENDTIME is the Unix timestamp of the last input file. In this statistics file, `blaster_analysis` writes for every 5 minutes of analyzed traffic the actual timestamp, followed by a list of active IP addresses with the number of case 1 to 5 infection attempts (see list in Sec. 2.3.2) for this address. A sample of a stat file is given in Tab. 7[6]. The second file contains a list of all observed successful infections in more detail. This file is called `infec_STARTTIME_ENDTIME.out`. It contains the attacker IP, the victim IP and the exact timestamp of the infection. A sample is given in Tab. 8[7].

Plots In order to draw the plots presented in Figure 6 to Figure 10 we had to further aggregate the statistic files. We wrote an awk script which sums the infection attempts for a variable number of 5 minutes intervals and makes the distinction between infection attempts from inside or outside AS559. The script is executed by
```
cat infip_STARTTIME_ENDTIME.out | awk -f awk_sumcases i=12 >plot_file.out
```
where i is the number of 5 minutes intervals to aggregate and plot_file.out is the output file. The output file can then be used to make the plots.

Limitations in the Observations There are several potential sources of errors in our observations. One is that a small part of traffic is not recorded in NetFlow at the routers. We tried to deal with this loss of records by being not too strict in the matching of flows to infection cases. Table 6 shows which flows have to be present for each infection case. This should minimize the effect of record loss.

An other source of error is that for cases 1 and 2, which only require one resp. two flows, we can not uniquely say that they originate from Blaster. If there were some large scale scannings for port 135/tcp during the analyzed interval they are included in our statistics. We believe that the distortions from other events are small, as no larger network events other than Blaster were reported publicly during the analyzed time interval.

2.5 Traffic Analysis

Our analysis focuses on a time interval starting shortly before Blaster outbreak, 10th of August 2003, ending on 16th of August.

[5]Seconds since standard epoch of 1/1/1970

[6]Because of privacy concerns, the IP addresses have been replaced with x.x.x.x. Every line represents a different IP address.

[7]x.x.x.x is a placeholder for the attacker IP address, y.y.y.y for the victim IP address. Different lines may have the same attacker address, as one attacker can infect several victims.

Case	135/tcp		4444/tcp		69/udp	
	A -> V	A <- V	A -> V	A <- V	A -> V	A <- V
1	x	-	-	-	-	-
2	x	x	-	-	-	-
	-	x	-	-	-	-
3	x	x	x	x	-	-
	x	x	x	-	-	-
	x	x	-	x	-	-
	-	x	x	x	-	-
	-	x	x	-	-	-
	-	x	-	x	-	-
	x	-	x	x	-	-
	x	-	x	-	-	-
	x	-	-	x	-	-
4	x	x	x	x	-	x
	x	x	x	-	-	x
	x	x	-	x	-	x
	-	x	x	x	-	x
	-	x	x	-	-	x
	-	x	-	x	-	x
	x	-	x	x	-	x
	x	-	x	-	-	x
	x	-	-	x	-	x
5	x	x	x	x	x	x
	x	x	x	-	x	x
	x	x	-	x	x	x
	-	x	x	x	x	x
	-	x	x	-	x	x
	-	x	-	x	x	x
	x	-	x	x	x	x
	x	-	x	-	x	x
	x	-	-	x	x	x
	x	x	x	x	x	-
	x	x	x	-	x	-
	x	x	-	x	x	-
	-	x	x	x	x	-
	-	x	x	-	x	-
	-	x	-	x	x	-
	x	-	x	x	x	-
	x	-	x	-	x	-
	x	-	-	x	x	-
	x	x	x	x	-	x
	x	x	x	-	-	x
	x	x	-	x	-	x
	-	x	x	x	-	x
	-	x	x	-	-	x
	-	x	-	x	-	x
	x	-	x	x	-	x
	x	-	x	-	-	x
	x	-	-	x	-	x

Table 6: *Flows required for infection cases 1 - 5. 'A->V': Flow from attacker to victim, 'A<-V': Flow from victim to attacker.*

```
*********************************************************************
1060388093
*********************************************************************
x.x.x.x   0       1       0       0       0
x.x.x.x   22      0       0       0       0
x.x.x.x   0       1       0       0       0
x.x.x.x   146     3       0       0       0
x.x.x.x   0       3       0       0       0
x.x.x.x   8       0       0       0       0
x.x.x.x   888     0       0       0       0
```

Table 7: *Sample output file infip_STARTTIME_ENDTIME.out*

```
x.x.x.x     y.y.y.y     1060623734
x.x.x.x     y.y.y.y     1060624838
x.x.x.x     y.y.y.y     1060624967
x.x.x.x     y.y.y.y     1060625896
x.x.x.x     y.y.y.y     1060627748
x.x.x.x     y.y.y.y     1060628838
x.x.x.x     y.y.y.y     1060629790
x.x.x.x     y.y.y.y     1060629958
```

Table 8: *Sample output file infec_STARTTIME_ENDTIME.out*

2.5.1 Infection Attempts and Successful Infections

Figures 6 to 10 show the number of infection attempts for each of the 5 cases defined in Section 2.3.2. In Figure 6 we can see that the number of unsuccessful connection attempts to port 135/tcp (case 1) drastically increases from around 1 mill. to about 13 mill. flows per hour on August 11, at about 17:00. This can be regarded as the *outbreak* of W32.Blaster. At the same time the number of case 2 (victim responding but not being vulnerable) grows from about 50’000 to 1 mill connection attempts per hour. While the number of case 3 (Figure 8) and case 4 (Figure 9) show no heavy increase at that time we can see a sharp peak of successful infections (Figure 10) of 4 infections between 17:20 and 18:20 and even 9 infections between 20:20 and 21:20. The lack of heavy increase of cases 3 and 4 can be explained by the surprise effect of Blaster, as anti-virus software did not yet recognize it. Once the exploit code was transmitted successfully chances were high that the whole infection would be successful.

Before August 12 the vast majority of Blaster traffic came from outside the SWITCH network.This changes at bout 06:00 and can be considered as the *internal outbreak*. Before that only few hosts within AS559 have been infected. The 7 successful infections coming from internal addresses between 20:20 and 21:20 on August 11, all originate from the same host, which just happened to scan a range of IP addresses with many vulnerable hosts. The reason for the delay of the internal outbreak may be that the external outbreak happened not during working time and most internal hosts are probably switched off during the night.

In the plots of cases 1 and 2 we can observe a *drop of connections* from external hosts from 08:20 to 09:20 on August 12. This was probably caused by some inbound port 135/tcp filter installed somewhere near AS559. We can observe another, smaller, drop of infection attempts coming from external hosts, with its lowest point about 05:00 on August 13. This also might be the effect of filtering.

About the same time there is a peak of cases 3 and 4 infection attempts coming from internal hosts. The first peak of case 3 has its summit between 10:20 and 11:20 on August 12, with 11’681 infection attempts. Our analysis showed that 73 % of the case 3 infection attempts in that interval came from one single class B network. The number of infected machines in that subnet was at that time only 6 % of the total number of infected machines. The vast majority of the victims of these infection attempts are in the next higher class B net lying outside of

AS559. These connections were probably generated by Blaster scanning the local subnet, but the scanned address range growing out of the local subnet and therefore the infection attempts got routed over the SWITCH border gateways. At the same time interval the infected hosts of that subnet generated only 4 case 4 and not a single case 5 infection attempt. The reason for this lack of successful infections may be that in the destination subnet the hosts were already patched.
The same picture but even clearer shows the second case 3 summit from 19:20 to 20:20 on August 14. There, 90 % of the case 3 attacks originate in one single /16 subnet and destinations are almost uniquely in the next higher class B subnet lying outside of AS559. In that subnet too, the hosts were apparently already patched as there were only 8 case 4 and no successful infections.
The reason why these scans show up as peaks in the plot is that probably most of the hosts were infected in a small time range internally and therefore started their scanning about the same time. Consequently, they also reach the next network at the same time and when they have passed the address space of that subnet, they came probably to a network less populated or with some filtering, which caused a drop of case 3 infection attempts. The scanning then appears as case 1 or case 2.

About 82 % of the peak of case 4 infection attempts with its summit from 12:20 to 13:20 on August 12, has its origin also in the two above-mentioned class B networks.

The plot of case 3 infection attempts shows a small peak on August 10, between 19:20 and 20:20, before the outbreak of Blaster. Our analysis showed that this peak originates from a single source address. At that time the exploit code used by Blaster was already published and we assume that someone was testing it. From that specific IP address we observed a scanning of port 135/tcp and for the addresses the scanning was successful the exploit code was sent. It is possible that this was some testing in the development phase of Blaster, but more likely someone just tried out the exploit code for fun or for some other kind of abuse.

The plot of successful infections shows a peak at the right end with 40 infections within 3 hours, from 21:20, August 15, to 0:20 August 16. 36 of these infections originate from one host and have their victims in one class B network. This host obviously scanned by chance a network with plenty of vulnerable hosts and no filtering.

Surprising in Figure 10 is, that, despite the high number of infected hosts, we can only observe very few successful infections going over SWITCH routers. Over the analyzed time period, from outbreak, on August 11, to August 16, 00:20, there is only a total of 302 registered infections. The reason for this low number may be that the vast majority of infections happen locally. Another reason is inherent in the design of Blaster, relying on 3 different connections. Because not widely used services (as TFTP) are filtered in many networks, this is a source of errors. Blaster was very successful in spreading within local networks but for being successful in spreading over the border of networks, there were too many connections and ports involved.

2.5.2 Successful Infections

In order to get a better overview of the observed successful Blaster infections, we have made some statistics with all infections in the interval from the outbreak to August 16, 00:20. We have observed a total number of 302 infections in that time interval. 80 % of the infections originate within AS599 and 20 % are from external hosts.

In total 88 hosts have successfully infected others. The 10 most successful hosts have performed 186 (62 %) of all observed infections. The hosts in the top 10 list happened to coincidentally scan networks with high numbers of vulnerable computers. The 46 infections of the "winner" all go to addresses in a range of 21 adjacent class B networks. The fact that 6 of the 10 hosts in the list are in the same /16 subnet is an evidence that this network suffers from slow patching procedures.

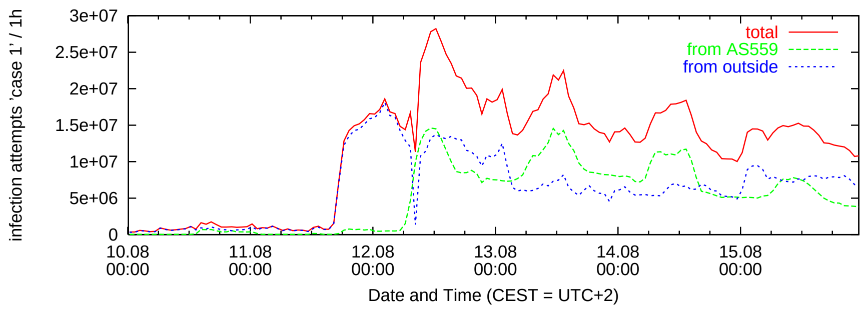


Figure 6: *Number of 'case 1' Blaster infection attempts from Aug 10th to Aug 15th*

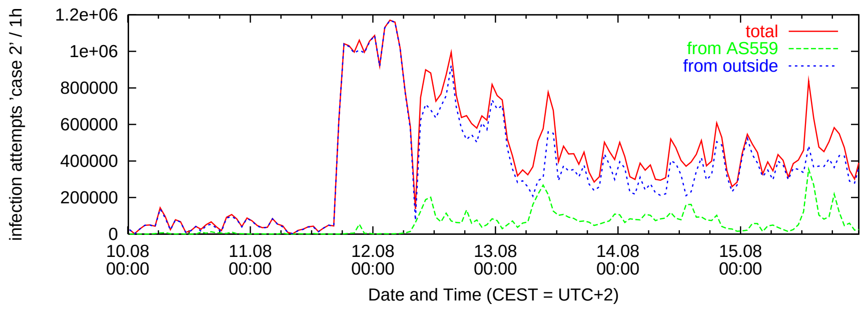


Figure 7: *Number of 'case 2' Blaster infection attempts from Aug 10th to Aug 15th*

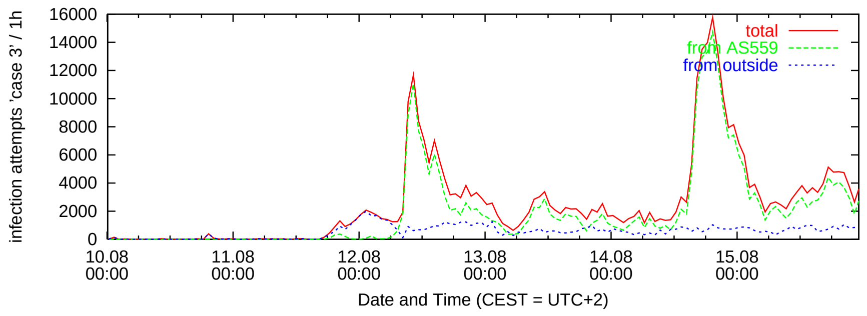


Figure 8: *Number of 'case 3' Blaster infection attempts from Aug 10th to Aug 15th*

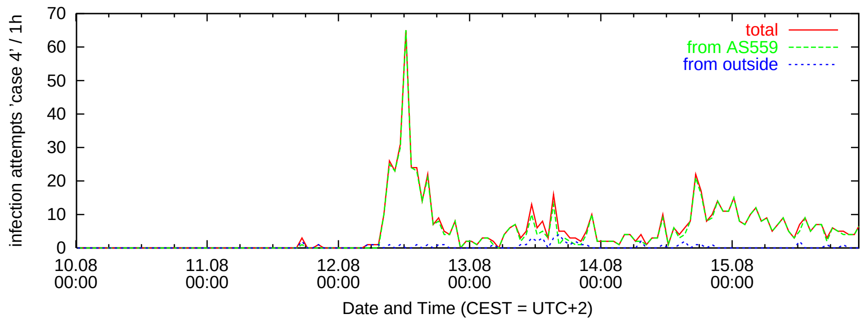


Figure 9: *Number of 'case 4' Blaster infection attempts from Aug 10th to Aug 15th*

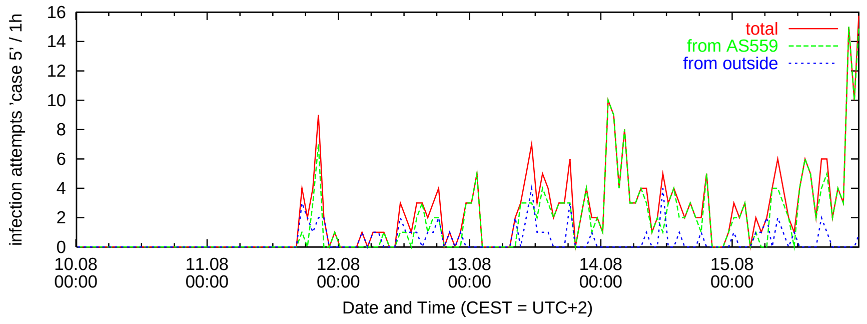


Figure 10: *Number of 'case 5' Blaster infection attempts from Aug 10th to Aug 15th*

Subject	Body text	Attachment filename
Re: Details	See the attached file for details	your_document.pif
Re: Approved	Please see the attached file for details.	document_all.pif
Re: Re: My details		thank_you.pif
Re: Thank you!		your_details.pif
Re: That movie		details.pif
Re: Wicked screensaver		document_9446.pif
Re: Your application		application.pif
Thank you!		wicked_scr.scr
Your details		movie0045.pif

Table 9: *Sobig.F message subject, body and attachment (source: Symantec [18])*

3 Sobig.F

3.1 Introduction

In this section we present our analysis of the Sobig.F worm. After giving a general overview of the worm in this section, Sobig.F is described from a technical point of view in Section 3.2. Our analysis of Sobig.F traffic is presented in Section 3.4.

E-Mail Worms As opposed to Blaster and other worms and viruses using vulnerabilities in the operating system or other programs, most E-Mail worms depend on the user executing an attachment of an E-Mail. The first famous E-Mail worm, and probably the best example of how E-Mail worms spread by using a clever E-Mail subject and text, is the famous *ILOVEYOU* worm first discovered in May 2000. Since then a lot of E-Mail worms with large impact have appeared, some famous names are MyDoom, Bagle and Sobig.

The Sobig.F Worm Sobig.F was first discovered August 19, 2003. It is an E-Mail worm that spreads as attachment of E-Mails with varying subject, message body and attachment filename. All possible variations are listed in Table 9. The F variant is the successor of Sobig.A [8] to Sobig.E but was by far the most successful in terms of impact of the Sobig series.
Affected operating systems are Windows 2000, Windows 95, Windows 98, Windows Me, Windows NT and Windows XP.
Sobig.F is also known as W32/Sobig.f@MM (McAffee), WORM SOBIG.F (Trend), W32/Sobig-F (Sophos), Sobig.F (F-Secure), Win32.Sobig.F (CA) or I-Worm.Sobig.f (KAV).
A noticeable feature of Sobig is that it has an automatic deactivation routine and stopped spreading on September 10, 2003.

3.2 Technical Details

The following is based on the *Symantec Security Response - W32.Sobig.F@mm [18]* and concurs with our observations of Sobig.F infections in our testbed.

3.2.1 Sobig.F Analysis

When the worm binary is executed it performs the following actions: It first installs itself as *winppr32.exe* in the Windows installation directory (usually `C:\Windows`) and adds the value `"TrayX"="%Windir%\winppr32.exe /sinc"` in two registry keys, `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` and `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run` in order to get started automatically at Windows startup.
Sobig.F then enumerates all network shares to which it has write access, but because of a bug in the code it is not able to spread over the shares. After that, it uses NTP[9] to determine the

[8]The A variant was discovered in the wild on January 09, 2003
[9]NTP: Network Time Protocol runs over port 123/udp

```
**********************************************************************
timestamp      0-200  400-500 103k-125k <200     <500    <125k  total
**********************************************************************
1061338845      46036   17626   25751   46036   81401   163698  163986
1061342445      52075   18098   28217   52075   86916   173135  173420
1061346046      59943   13903   25397   59943   92346   175965  176205
1061349646      56309   16501   27254   56309   91055   174396  174682
1061353247      55475   18700   31943   55475   93884   182219  182639
1061356847      70425   22077   50477   70425   115530  233956  234797
1061360447      91477   23388   74384   91477   138567  307008  308350
1061364047      96292   28183   97029   96292   150220  367406  369112
```

Table 10: *Sample output file distr_STARTTIME_ENDTIME.out*

time from one of 19 hardcoded NTP servers[10]. This time fetch operation is repeated every hour. If the date is later than August 22, 2003, weekday is Friday or Sunday and the time is between 1900 and 2200 UTC it tries to contact one of the 20 hardcoded master server IP addresses on port 8998/udp. This feature was probably intended to be an automatic update functionality, with the master servers providing a URL from which Sobig would download an executable. But all the master servers were taken from the net before Sobig could use them.
After this initialization phase it starts the spreading phase. It has its own MTA included for sending the E-Mails. The destination addresses are taken from all files it finds on the harddisk with extensions

- .dbx
- .eml
- .hlp
- .htm
- .html
- .mht
- .wab
- .txt

The sender address of the E-Mails is spoofed and also taken from these files.

3.3 Implementation

For the analysis of Sobig.F traffic we have implemented a tool called `sobig_analysis`.

Parameters `sobig_analysis` takes as parameter a list of log files, which have to be ordered as described for `blaster_analysis` in Sec. 2.4.

Scanning The records in the log files are scanned for port 25/tcp flows with sizes typical for Sobig.F E-Mails (see Sec. 3.4.3). For every typical flow size a counter counts the flows of a time interval of one hour.

Output After every hour of scanned traffic, `sobig_analysis` writes the counters together with a timestamp in a file called `distr_STARTTIME_ENDTIME.out`, where STARTTIME and ENDTIME are the Unix timestamps of the first/last input file. A sample of the output file is shown in Tab. 10.

[10]The server it tries to contact first is the NTP Server of ETH Zurich

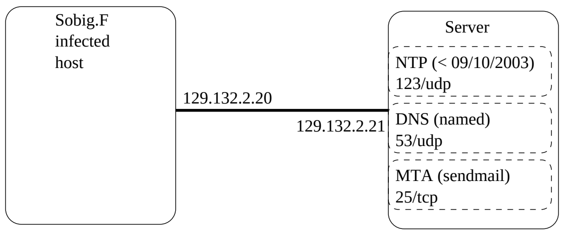


Figure 11: *The testbed for Sobig.F*

Limitations of the Observations As there is some transit traffic in the SWITCH network, some flows are counted twice in our analysis. This is only a small percentage of the total traffic and the distortion from these duplicate records should be small. Furthermore the filtering criteria which are only based on the size of the flows do not distinguish between Sobig.F E-Mails and other messages of the same size. In order to get an idea of how much of the flows really come from Sobig we have added graphs of one day before Sobig.F outbreak.

3.4 Sobig.F Traffic Analysis

3.4.1 The Testbed

In order to observe Sobig.F operation we have set up a testbed with two computers, an attacker and a server (see Fig. 11). The attacker was a Windows XP host on which we ran Sobig.F. On the server machine (Linux Fedora Core1) we set up three services: NTP, MTA and DNS. Because Sobig.F determines the time with NTP from servers with hardcoded addresses and the date has to be earlier than September 10, 2003, we ran an NTP server and gave the server machine the IP address 129.132.2.21, which is the hardcoded address Sobig first tries to contact. The DNS service (bind) was configured to resolve all name queries from the attacker to the server IP address (129.132.2.21) so that the E-Mails from Sobig are sent to the MTA (sendmail) running on the server. For packet capturing we ran Ethereal on the server machine.

3.4.2 Observed Worm Transmissions

On the testbed we captured the packets of several Sobig.F transmission. A dump of the first few packets of a successful transmission is shown in Figure 12. Over several such transmissions we observed an average of about 100 port 25/tcp packets sent from attacker to MTA with a total size (including IP and TCP headers) of about 104’000 Bytes. The flows in the other direction consisted of about 40 packets with a total size of about 2’000 Bytes.
On our testbed we observed that only about 5 % of the Sobig.F transmissions were listed complete and correct in the Ethereal dump files. The rest of the transmissions showed up incomplete in the dumps, most of them with only about 80 packets and 80’000 Bytes from attacker to MTA. As sendmail accepted the incomplete transmissions anyway with a "Message accepted for delivery" status report, we can only speculate that the server machine might have been overloaded and Ethereal could not capture all packets. If the packets would really have been lost on the network, the TCP flow control would have caused retransmissions, which we did not observe.

Because of the increasing number of spam mails many MTAs have installed some kind of filter. There are two important kinds of filters, whitelist and blacklist, which reject E-Mails, depending on the sender address, before they are even transmitted and therefore show special flow characteristics. Whitelist filters only accept mails from MTAs known to be spam free, while blacklist filters only reject mails from MTAs known to send a lot of spam. Both kinds of filters possibly filter Sobig mails. Whitelists do because they do not know the infected host. Blacklists on the other hand filter Sobig traffic only if there is already some other spam relay program installed on the infected host, which is known to send spam. In order to see how the flows of rejected E-Mails look we installed a filter on the MTA and ran the packet sniffer. A dump of such a rejected transmission is shown in Figure 13. The flow from attacker to MTA consists of 8 packets with a total size of about 400 Bytes, while the flow in the opposite direction shows 11 packets with a total size of about 850 Bytes.

3.4.3 Flow Size Distribution

The number of packets in a flow and the flow size of Sobig.F flows vary and therefore we can not define flow characteristics as precise as with Blaster. In order to find some boundaries of flow size we analyzed one hour of NetFlow data, where Sobig transmission had its summit, on August 19, from 12:20 to 13:20. The plots of the flow size distribution are shown in Figures 14 to 17. For comparison we added the same distribution plots of August 18, before Sobig.F outbreak.

Figure 14 to 16 show different intervals of flow sizes with destination port 25/tcp. We see some clear peaks of small flow sizes (Figure 14) which come from requests to servers not responding. The comparison of August 18 and 19 show that the peaks lie at the same sizes but are significantly higher after the outbreak. On Figure 15 we see that there are two wide peaks originating from Sobig.F flows, one from about 400 Bytes to 440 Bytes and the other, less high, from 450 to 490 Bytes with a sharp peak at 478 Bytes. These peaks are probably caused by filtered transmissions. About 70% of the flows of the sharp peak have the same source and destination IP address, the peak is generated by a single Sobig infected host probably trying to send the same E-Mail hundreds of times. The plot of August 18 shows only two sharp peaks at 444 and 491 Bytes. These peaks are not caused by Sobig and we can only speculate about their origin. As they have many different source and destination IP addresses, they might be caused by some other E-Mail worm. While the first peak is still present on August 19 the second has disappeared. Figure 16 shows the wide peak of successfully transmitted Sobig.F E-Mails, starting at about 103’000 Bytes then decreasing at about 109’000 Bytes but still being significant up to about 125’000 Bytes.

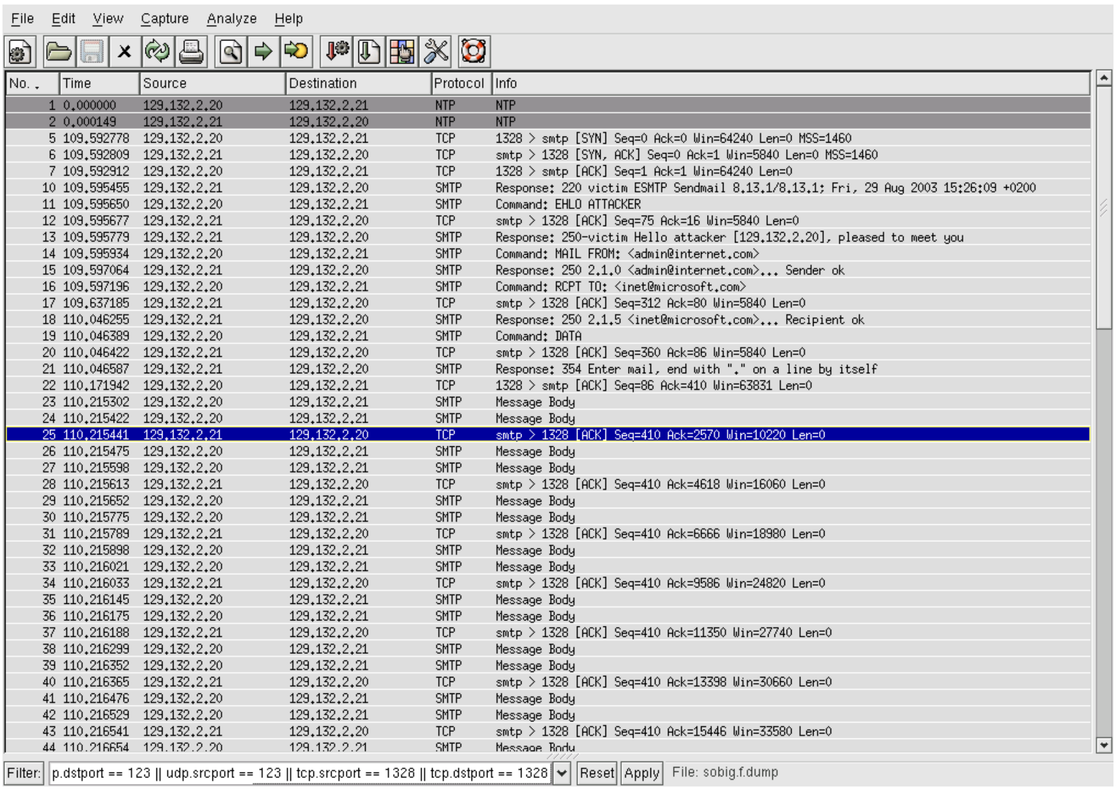
Figure 17 shows the distribution of flow size in the other direction, with source port 25/tcp. The figure shows that on August 19 there are about twice as much flows of size 0 - 1000 Bytes as on August 18. These are probably generated by rejected Sobig.F messages. There is also a significant increase of flows with size ranging from about 1800 Bytes to 5500 Bytes. These flows originate probably from successful transmissions. Besides some peaks between 2500 Bytes and 2800 Bytes there are some noticeable sharp peaks between 4800 Bytes and 5100 Bytes. Our analysis showed that all these peaks originate from flows with only two source addresses, which lie in the same subnet. As Sobig.F infected hosts can be used as open relay mail servers (see [18]) these servers might have been abused for sending spam.

3.4.4 Flow Sizes

In Figure 18 we plotted the development of flow sizes with destination port 25/tcp over the time, in an interval from August 18 to August 21. The figure shows three levels of flow sizes. The first level is the number of flows with size less than 200 Bytes. These flows originate from failed connection attempts to mail servers not existing, or rejecting the connection for some reason. The second level shows all flows smaller than 500 Bytes. These flows originate from either very small or rejected (by some filter) Mail messages. The third level shows the total number of flows, including successful Sobig.F transmissions. The plot clearly shows a daily rhythm in the number of flows with traffic increasing at about 06:00 in the morning, a peak around noon and decreasing until about 18:00. After the outbreak of Sobig.F we see an increased number of flows of all sizes. The biggest increase show the flows larger than 500 Bytes, where the number goes up from around 50’000 on August 18 to about 250’000 on August 19, at noon.

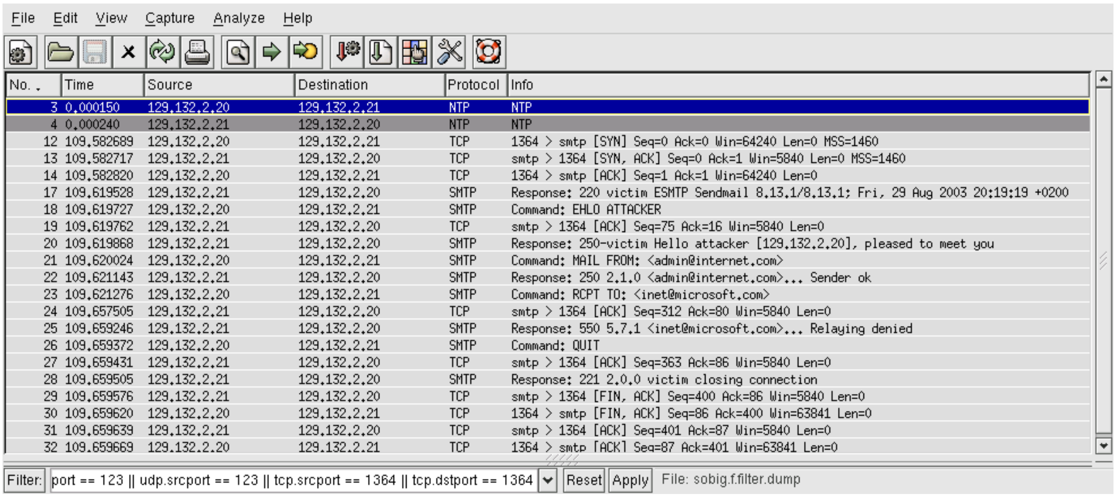
3.4.5 Number of Worm Transmissions

Our analysis of the flow sizes in Section 3.4.3 showed that the number of flows with destination port 25/tcp and size between 103’000 and 125’000 Bytes is about 350 on August 18, 12:20 to 13:20 and drastically increases to about 137’000 on August 19, between 12:20 and 13:20.



```
No. . Time       Source         Destination    Protocol Info
   1 0.000000   129.132.2.20   129.132.2.21   NTP      NTP
   2 0.000149   129.132.2.21   129.132.2.20   NTP      NTP
   5 109.592778 129.132.2.20   129.132.2.21   TCP      1328 > smtp [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
   6 109.592809 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
   7 109.592912 129.132.2.20   129.132.2.21   TCP      1328 > smtp [ACK] Seq=1 Ack=1 Win=64240 Len=0
  10 109.595455 129.132.2.21   129.132.2.20   SMTP     Response: 220 victim ESMTP Sendmail 8.13.1/8.13.1; Fri, 29 Aug 2003 15:26:09 +0200
  11 109.595650 129.132.2.20   129.132.2.21   SMTP     Command: EHLO ATTACKER
  12 109.595677 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=75 Ack=16 Win=5840 Len=0
  13 109.595779 129.132.2.21   129.132.2.20   SMTP     Response: 250-victim Hello attacker [129.132.2.20], pleased to meet you
  14 109.595934 129.132.2.20   129.132.2.21   SMTP     Command: MAIL FROM: <admin@internet.com>
  15 109.597064 129.132.2.21   129.132.2.20   SMTP     Response: 250 2.1.0 <admin@internet.com>... Sender ok
  16 109.597196 129.132.2.20   129.132.2.21   SMTP     Command: RCPT TO: <inet@microsoft.com>
  17 109.637185 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=312 Ack=80 Win=5840 Len=0
  18 110.046255 129.132.2.21   129.132.2.20   SMTP     Response: 250 2.1.5 <inet@microsoft.com>... Recipient ok
  19 110.046389 129.132.2.20   129.132.2.21   SMTP     Command: DATA
  20 110.046422 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=360 Ack=86 Win=5840 Len=0
  21 110.046587 129.132.2.21   129.132.2.20   SMTP     Response: 354 Enter mail, end with "." on a line by itself
  22 110.171942 129.132.2.20   129.132.2.21   TCP      1328 > smtp [ACK] Seq=86 Ack=410 Win=63831 Len=0
  23 110.215302 129.132.2.20   129.132.2.21   SMTP     Message Body
  24 110.215422 129.132.2.20   129.132.2.21   SMTP     Message Body
  25 110.215441 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=2570 Win=10220 Len=0
  26 110.215475 129.132.2.20   129.132.2.21   SMTP     Message Body
  27 110.215598 129.132.2.20   129.132.2.21   SMTP     Message Body
  28 110.215613 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=4618 Win=16060 Len=0
  29 110.215652 129.132.2.20   129.132.2.21   SMTP     Message Body
  30 110.215775 129.132.2.20   129.132.2.21   SMTP     Message Body
  31 110.215789 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=6666 Win=18980 Len=0
  32 110.215898 129.132.2.20   129.132.2.21   SMTP     Message Body
  33 110.216021 129.132.2.20   129.132.2.21   SMTP     Message Body
  34 110.216033 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=9586 Win=24820 Len=0
  35 110.216145 129.132.2.20   129.132.2.21   SMTP     Message Body
  36 110.216175 129.132.2.20   129.132.2.21   SMTP     Message Body
  37 110.216188 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=11350 Win=27740 Len=0
  38 110.216299 129.132.2.20   129.132.2.21   SMTP     Message Body
  39 110.216352 129.132.2.20   129.132.2.21   SMTP     Message Body
  40 110.216365 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=13398 Win=30660 Len=0
  41 110.216476 129.132.2.20   129.132.2.21   SMTP     Message Body
  42 110.216529 129.132.2.20   129.132.2.21   SMTP     Message Body
  43 110.216541 129.132.2.21   129.132.2.20   TCP      smtp > 1328 [ACK] Seq=410 Ack=15446 Win=33580 Len=0
  44 110.216654 129.132.2.20   129.132.2.21   SMTP     Message Body
Filter: p.dstport == 123 || udp.srcport == 123 || tcp.srcport == 1328 || tcp.dstport == 1328   File: sobig.f.dump
```

Figure 12: *Dump of (part of) Sobig.F transmission*



```
No. . Time       Source         Destination    Protocol Info
   3 0.000150   129.132.2.20   129.132.2.21   NTP      NTP
   4 0.000240   129.132.2.21   129.132.2.20   NTP      NTP
  12 109.582689 129.132.2.20   129.132.2.21   TCP      1364 > smtp [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
  13 109.582717 129.132.2.21   129.132.2.20   TCP      smtp > 1364 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
  14 109.582820 129.132.2.20   129.132.2.21   TCP      1364 > smtp [ACK] Seq=1 Ack=1 Win=64240 Len=0
  17 109.619528 129.132.2.21   129.132.2.20   SMTP     Response: 220 victim ESMTP Sendmail 8.13.1/8.13.1; Fri, 29 Aug 2003 20:19:19 +0200
  18 109.619727 129.132.2.20   129.132.2.21   SMTP     Command: EHLO ATTACKER
  19 109.619762 129.132.2.21   129.132.2.20   TCP      smtp > 1364 [ACK] Seq=75 Ack=16 Win=5840 Len=0
  20 109.619868 129.132.2.21   129.132.2.20   SMTP     Response: 250-victim Hello attacker [129.132.2.20], pleased to meet you
  21 109.620024 129.132.2.20   129.132.2.21   SMTP     Command: MAIL FROM: <admin@internet.com>
  22 109.621143 129.132.2.21   129.132.2.20   SMTP     Response: 250 2.1.0 <admin@internet.com>... Sender ok
  23 109.621276 129.132.2.20   129.132.2.21   SMTP     Command: RCPT TO: <inet@microsoft.com>
  24 109.657505 129.132.2.21   129.132.2.20   TCP      smtp > 1364 [ACK] Seq=312 Ack=80 Win=5840 Len=0
  25 109.659246 129.132.2.21   129.132.2.20   SMTP     Response: 550 5.7.1 <inet@microsoft.com>... Relaying denied
  26 109.659372 129.132.2.20   129.132.2.21   SMTP     Command: QUIT
  27 109.659431 129.132.2.21   129.132.2.20   TCP      smtp > 1364 [ACK] Seq=363 Ack=86 Win=5840 Len=0
  28 109.659505 129.132.2.21   129.132.2.20   SMTP     Response: 221 2.0.0 victim closing connection
  29 109.659576 129.132.2.21   129.132.2.20   TCP      smtp > 1364 [FIN, ACK] Seq=400 Ack=86 Win=5840 Len=0
  30 109.659620 129.132.2.20   129.132.2.21   TCP      1364 > smtp [FIN, ACK] Seq=86 Ack=400 Win=63841 Len=0
  31 109.659639 129.132.2.21   129.132.2.20   TCP      smtp > 1364 [ACK] Seq=401 Ack=87 Win=5840 Len=0
  32 109.659669 129.132.2.20   129.132.2.21   TCP      1364 > smtp [ACK] Seq=87 Ack=401 Win=63841 Len=0
Filter: port == 123 || udp.srcport == 123 || tcp.srcport == 1364 || tcp.dstport == 1364   File: sobig.f.filter.dump
```

Figure 13: *Dump of rejected Sobig.F transmission*

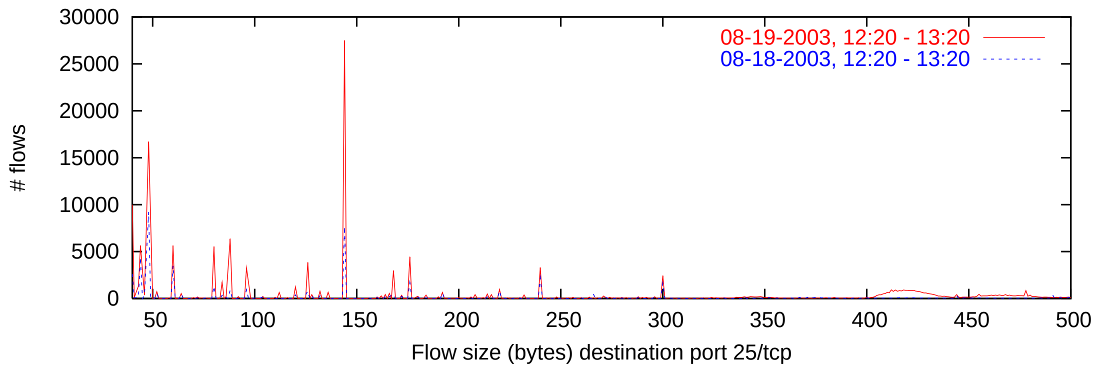


Figure 14: *Distribution of flow sizes (dest port 25/tcp, 0 - 500 bytes)*

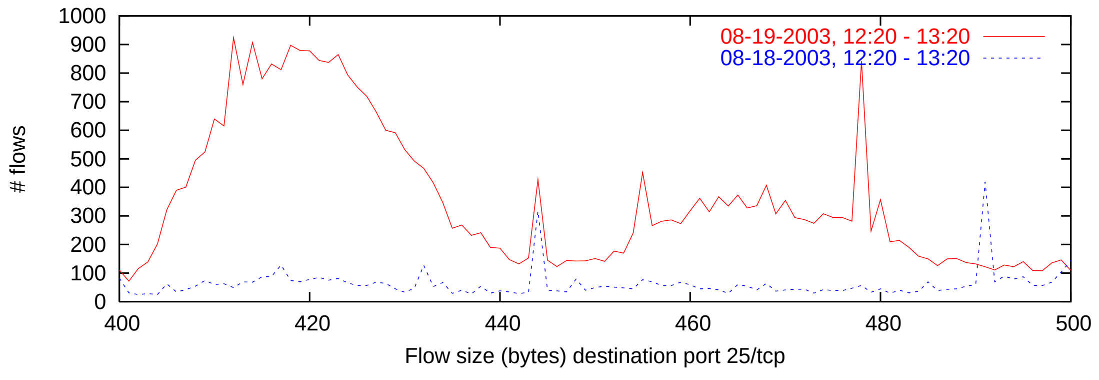


Figure 15: *Distribution of flow sizes (dest port 25/tcp, 400 - 500 bytes)*

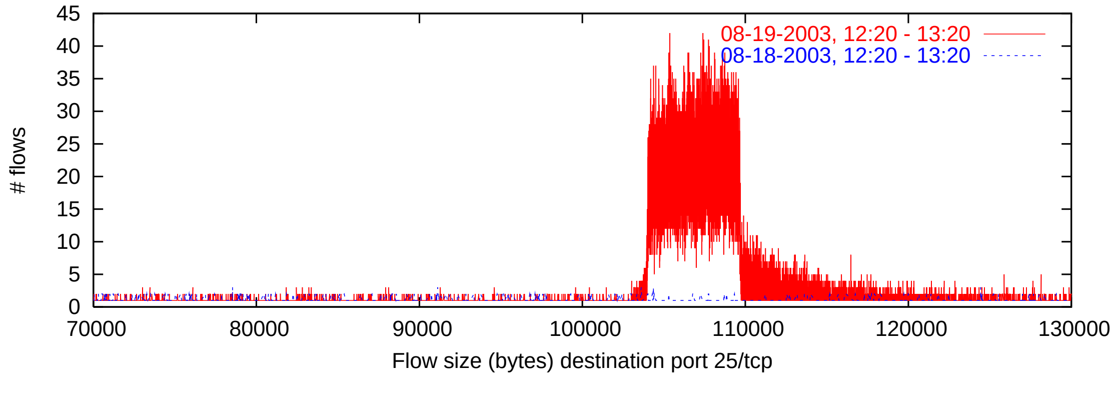


Figure 16: *Distribution of flow sizes (dest port 25/tcp, 70'000 - 130'000 bytes)*

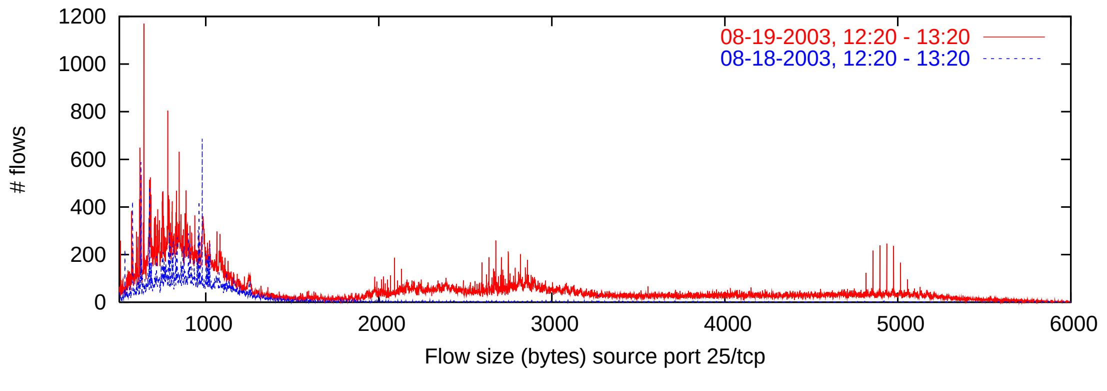


Figure 17: *Distribution of flow sizes (src port 25/tcp, 500 - 6000 bytes)*

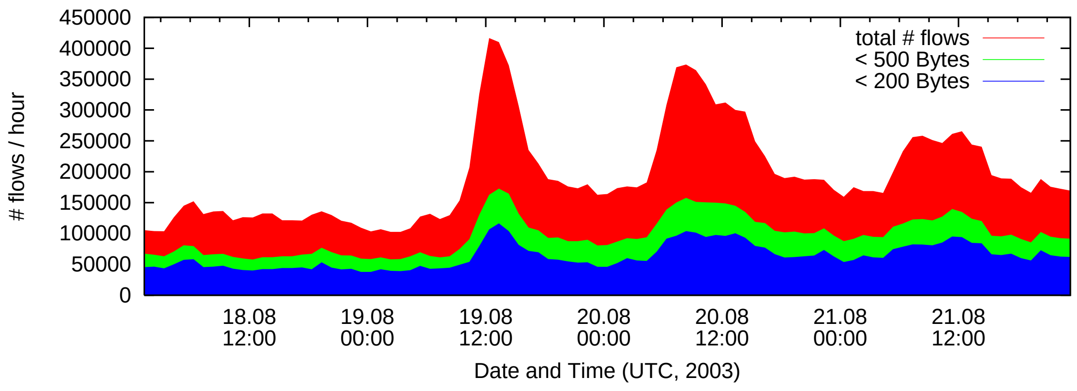


Figure 18: *Cumulative number of flow sizes over time*

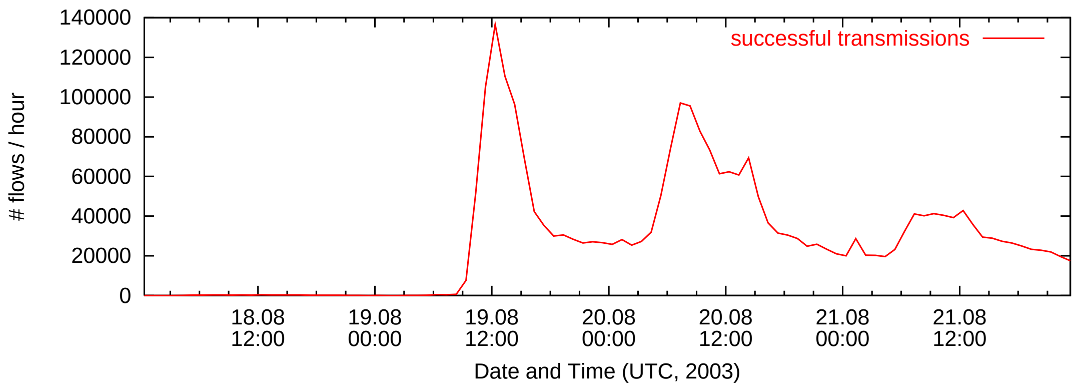


Figure 19: *Number of Sobig.F transmissions over time*

So we can assume that less than one percent of the flows of these size are not Sobig.F generated. The rest originates from successful Sobig.F transmissions. Figure 19 shows a plot of the number of flows with size between 103’000 and 125’000 Bytes, from which we can assume that they originate from successfully transmitted Sobig.F Mails. As the number of E-Mails in that size range starts to increase on August 19 at about 09:00, this can be regarded as the Sobig.F outbreak. The number of successful transmissions raises drastically until about 12:00 and then starts to decrease until the end of the working day at about 18:00. The peak of 137’000 transmissions on August 19 is by far the highest, on August 20 the peak reaches about 100’000 and on August 21 only 50’000 transmissions. This decreasing height of the peaks probably originate by people updating their anti-virus software and cleaning up their machines. During the night the number of transmissions reaches only about

4 Conclusion and Outlook

4.1 Conclusion

In this semester thesis we presented our analysis of Blaster and Sobig generated network traffic. We now draw our conclusion of this analysis with respect to the goal of the DDoSVax project to detect infection and attack phase of DDoS attacks in near real-time and to take countermeasures as early as possible.

4.1.1 Blaster

The goal of the thesis, to define exact worm traffic pattern, implement them and observe the worm traffic in the generated plot was achieved. With the analysis of the plots we gained a good insight on how Blaster spread.

The most obvious observations in the Blaster traffic is, that there is loads of scanning traffic and unsuccessful infection attempts recorded at the gateway routers, but only very few successful infections. This lack of large numbers of infections comes from the design of Blaster. Blaster relies on three connections to be successful, ports 135/tcp, 4444/tcp and 69/udp have to be open in the network. In many networks where some sort of security is implemented, ports 135/tcp and 69/udp are filtered. The vast majority of Blaster infections happen in the local network and can not be seen on the gateway routers.

The scanning traffic can be used to detect the worm outbreak in a very early stage. This can be achieved by scanning the network traffic in real-time for some worm typical patterns. Plug-ins for the UPFrame framework which perform these scans have been implemented in the context of the DDoSVax project, for instance the flow size histogram [19], which plots in near real-time the distribution of flow sizes. The effectiveness of such detection tools has been confirmed with our analysis.

On the other hand, the low number of successful worm transmissions makes it very hard to take countermeasures in an early phase in the backbone network. Once the detection tools have shown a pattern of the worms scanning traffic, filters could automatically be adapted to block the worm traffic. This potential automatic countermeasure holds the risk to filter regular traffic, for instance filtering port 135/tcp or 69/udp would cause the regular services running on these ports to stop working. More promising would be to identify infected hosts and only block their traffic on the relevant ports. This would minimize the negative impact of the filtering, but has to be handled carefully, as for instance the filters have to be opened as soon as a host is cleaned or the IP address is handed over to another host. Furthermore it would require a very powerful filtering infrastructure.

4.1.2 Sobig.F

For Sobig.F we had to rely mainly on the criteria of the flow sizes to isolate the traffic. Patterns as exact as for Blaster could not be defined. However, by finding boundaries for these flow sizes, we could still gain good criteria for successful Sobig.F transmissions. The plots gave us a good picture of the amount of transmitted Sobig.F E-Mails.

Our analysis shows that detecting an E-Mail worms like Sobig.F is possible by analyzing the distribution of the size of SMTP flows over time.

The problem with countermeasures is about the same as with Blaster. Filtering SMTP traffic is not possible in the backbone network without analyzing the payload of the packets. A possibility to reduce worm traffic would be, as with Blaster, to first identify infected hosts and block their SMTP traffic. This would cause a Denial of Service for all E-Mails of infected hosts and therefore can not be regarded practical. The effect would even be worse if a worm does not have its own

MTA, but uses the standard mail gateway of the network. Filtering the mail traffic of this gateway would cause a Denial of Service for E-Mail traffic of the whole network.

4.2 Outlook

In the DDoSVax project a lot of interesting work is still to be done.

Based on the knowledge gained from this and other theses, other worms could be analyzed. At the time this theses was written most of the worm activity was caused by two E-Mail worms, Bagle and MyDoom. But it is no question that also in the future new "interesting" worms will appear.

Besides analysis, automatic detection of new worm activity is an interesting subject to which the insights from this theses can contribute.

As we have seen that simple countermeasures like blocking are difficult to take in the backbone network, automatic identification of infected hosts together with dynamic, IP specific filters could be worth to take a closer look at.

4.3 Acknowledgments

I would like to thank my tutor Thomas Duebendorfer and co-tutor Arno Wagner for their support and constructive input during the work on this theses. It was a great pleasure to work in a highly professional environment with excellent support for technical problems but still in a very humanly atmosphere.

References

[1] Computer worm - Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Computer_worm
2004

[2] A Taxonomy of Computer Worms
http://www.cs.berkeley.edu/˜nweaver/papers/taxonomy.pdf
N. Weaver, V. Paxson, S. Staniford, R. Cunningham, 2003.

[3] The DDoSVax project at ETH Zürich
http://www.tik.ee.ethz.ch/˜ddosvax/
A. Wagner, T. Dübendorfer, B. Plattner, 2004.

[4] SWITCH: Swiss Academic and Research Network
http://www.switch.ch/
2004.

[5] UPFrame - An Extendible Framework for the Reception and Processing of UDP Data
http://www.tik.ee.ethz.ch/˜ddosvax/upframe/
T. Dübendorfer, C. Schlegel, 2004.

[6] ETH TIK Cluster ’Scylla’
http://www.tik.ee.ethz.ch/˜ddosvax/cluster/
T. Dübendorfer, A. Wagner, 2004.

[7] White Paper: NetFlow Services and Applications
ttp://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm
Cisco, 2002.

[8] NetFlow: Information loss or win?
http://www.icir.org/vern/imw-2002/imw2002-papers/163.ps.gz
R. Sommer, A. Feldmann, 2002.

[9] Symantec Worldwide Homepage
http://www.symantec.com
Symantec Corporation, 2004.

[10] The Blaster Worm: The View From 10,000 Feet
http://monkey.org/˜jose/presentations/blaster.d/index.html
J. Nazario, 2003.

[11] Gawk
http://www.gnu.org/software/gawk/gawk.html
Free Software Foundation, 2004.

[12] Gnuplot Homepage
http://www.gnuplot.info
2004.

[13] Microsoft DCOM RPC Worm Alert
https://tms.symantec.com/members/analystreports/030811-alert-dcomworm.pdf
Symantec Corporation, 2003.

[14] Buffer Overrun in Windows RPC Interface
http://lsd-pl.net/special.html
The Last Stage Of Delirium, July 2003.

[15] Microsoft Security Bulletin MS03-026
http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx
Microsoft Corporation, 2003.

[16] Symantec Security Response - W32.Blaster.Worm
http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html
Symantec Corporation, 2003.

[17] Blaster Worm Analysis
http://www.eeye.com/html/Research/Advisories/AL20030811.html
eEye Digital Security, 2003.

[18] Symantec Security Response - W32.Sobig.F@mm
http://securityresponse.symantec.com/avcenter/venc/data/w32.sobig.f@mm.html
Symantec Corporation, 2003.

[19] Plug-Ins for DDoS Attack Detection in Realtime
ftp://www.tik.ee.ethz.ch/pub/students/2004-So/SA-2004-19.pdf
A. Weisskopf, 2004.