ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Philippe Lovis

Worm Outbreak Detection in the Internet Backbone

Diploma Thesis DA–2006–03
November 2005 to March 2006

Supervisor: Arno Wagner
Professor: Bernhard Plattner

Abstract

The goal of this thesis was to develop new approaches for the detection of worm outbreaks in high–speed Internet backbones. We will present three different methods that use flow–level information exported by backbone border routers. The first method is based on characteristic port sequences generated by worms while trying to exploit vulnerabilities on the target hosts. The second method identifies propagation paths of worms as they spread from host to host. The third approach is based on the increase of the number of ICMP messages due to infected hosts scanning for victim hosts. The approaches are generic in the sense that they do not require any previous knowledge about the exploits used by the worms or their scanning behaviour. Our presented methods have been validated against known minor and major worm outbreaks by analysing the recorded NetFlow data captured in the context of the DDoSVaX project. For this purpose we have designed and implemented appropriate tools for the offline analysis of NetFlow data.

Contents

1 Introduction

Many worm outbreaks occur unnoticed by network operators due to the lack of suitable outbreak detection mechanisms. By the time they get detected a large number of hosts may already be infected causing lots of damage. The goal of this diploma thesis was to develop new approaches for near real–time worm outbreak detection based on the analysis of flow–level data from backbone border routers. The suitability of the different approaches should be validated against known minor and major worm outbreaks in the recorded NetFlow data of the DDoSVaX project. Therefore we had to design and implement tools for the offline analysis that allowed the testing and evaluation of the presented approaches.

The three approaches for worm outbreak detection presented in this thesis will be using:

- Characteristic port sequences
- Propagation triples
- ICMP based analysis

The idea behind the characteristic port sequences is that a self–propagating Internet worm exploits certain vulnerabilities in order to infect new hosts. For each infection attempt the worm will try to connect to the destination ports of vulnerable services according to the worm's attack vector. This causes an infected host to repeatedly connect to the same set of destination ports — the worm's characteristic port sequence — while scanning for vulnerable victim hosts. By observing all port sequences that occur within the observed traffic a worm outbreak will be detectable by a significant increase in the number of occurrences of the worm's characteristic port sequence. By using this approach we were able to reliably identify major as well as minor worm outbreaks.

The second approach is based on discovering the propagation of a worm's traffic pattern as it spreads from host to host. After a host infected another host, the latter will try to connect to other hosts by using the same destination ports as the former did, due to the worm's propagation routine. We will investigate if propagation triples — that is a host connects to a second host by using a given destination port and thereafter the second host connects to a third host by using the same destination port — may be identified in the NetFlow data during worm outbreaks and if their presence reveals an outbreak with respect to periods without ongoing attack. Due to the nature of the border traffic NetFlow data this approach may only identify worm outbreaks that occur within the observed network. We will show that during an internal outbreak a significant amount of propagation triples may be found. But the detection latency of the propagation triple based approach varied a lot between the different outbreaks we have analysed.

A worm causes an infected host to scan randomly generated IP addresses in order to find vulnerable victims. The scanning will trigger ICMP messages (e.g. of type Destination Unreachable) generated for instance by routers of the

corresponding networks or by firewalls. As a third approach we will investigate the suitability of ICMP based analyses of the NetFlow data for worm outbreak detection. We found that large scale worm outbreaks have a significant impact on the number of generated ICMP messages. But the approach is not suited for detecting worm outbreaks of smaller magnitude because they do not trigger sufficiently additional ICMP messages in order to be noticed within the existing ICMP traffic.

1.1 DDoSVaX Project

This thesis is part of the DDoSVaX research project [37] which is a joint project of ETH Zurich and SWITCH [4]. The goals of the project are the development of methods and tools to detect and analyse DDoS attacks and worm outbreaks for supporting appropriate countermeasures. The DDoSVaX project uses the NetFlow data provided by the SWITCH network (Swiss Education and Research Network). The flows are exported by the backbone border routers of the autonomous system AS559, operated by SWITCH, and are collected by the UPFrame [31] framework. UPFrame has previously been developed for this purpose in the context of the DDoSVaX project. In this thesis we also used the flow–level data from the AS559 backbone border routers for our analyses.

1.2 Related Work

In [38] the authors have presented an entropy based analysis method for detecting outbreaks of fast worms in near real–time. The analysis method uses flow–level data from an observed network and calculates the entropy (randomness) of various attributes within the flows (e.g. number of source IP addresses). The idea behind the approach is that worm traffic is more structured than normal traffic in some respects and more random in others. Significant changes in the characteristics of certain attributes in the observed flows result in significant changes in the entropy of the corresponding attribute, hence indicating an important network event like a worm outbreak.

Another method for detecting worm outbreaks in flow–level data has been presented in [11]. The method attributes several behavioural properties to individual hosts, e.g the ratio of outgoing to incoming traffic, the responsiveness and the number of connections. Based on these properties the individual hosts are grouped into distinct classes. As the observed attributes are strongly influenced by a worm outbreak, the number of hosts within a class will significantly change during a worm outbreak and thereby reveal the outbreak.

2 Port Sequences

2.1 Introduction

A worm's propagation routine infects a new victim host by exploiting a given vulnerability. Some worms carry multiple exploits for different vulnerabilities in their attack vectors which are employed one by one to increase the probability for a successful infection. In order to exploit a given vulnerability the worm has to connect to the destination port(s) of the vulnerable service(s). Hence a worm causes an infected host to repeatedly connect to the same set of destination ports of the scanned hosts that the worm tries to infect. As more and more hosts get infected by a worm, its scanning behaviour will significantly increase the amount of traffic to the destination ports determined by the worm's attack vector. The following approach for worm outbreak detection is based on identifying such characteristic traffic patterns, in particular patterns that show a significant increase over a relatively short period of time.

In the following we will describe such a characteristic traffic pattern as a *port sequence* which is a set of destination ports used for the TCP and / or UDP traffic between a source host $srcHost$ and a destination host $dstHost$[1]. Provided that the start times of two consecutive flows are within a given time limit, which in the following will be five seconds, the destination port of each flow between $srcHost$ and $dstHost$ will be added to the corresponding port sequence. If no further flow is observed within the time limit the current port sequence between the two hosts is considered being completed. A later flow will induce a new port sequence, even if its destination port is not contained in the previous sequence. Note that only the unidirectional flows between $srcHost$ and $dstHost$ are considered for a given port sequence — the flows in the opposite direction will create a separate port sequence. Figure 1 summarizes these statements.

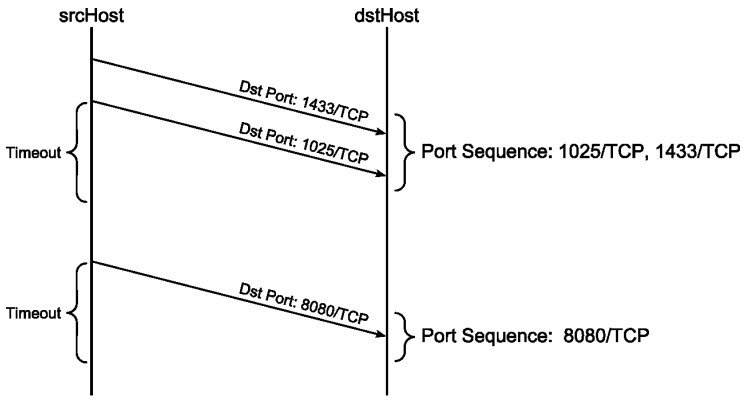

Figure 1: Two port sequences between two hosts.

[1]Note that each port is implicitly associated to its corresponding transport protocol (either TCP or UDP) but for simplicity we will just refer to "port" instead as to "port and protocol".

2.2 Problem Statement

Counting the number of occurrences of the various port sequences within the observed network traffic and aggregating them by interval will result in a set of time series — one per port sequence that has been found during the analysed period. During a worm outbreak the number of occurrences of the corresponding port sequence will significantly increase. Hence the problem of worm outbreak detection may be reduced to the problem of anomaly detection in the time series resulting from the port sequence analysis.

This approach poses to major problems to be solved: First, we have to identify and keep track of a very large amount of different port sequences in the network traffic to create the time series. Secondly, we have to develop an algorithm that analyses these time series in order to detect traffic anomalies due to worm outbreaks.

In Section 2.3 we will present a tool named `netflow_port_sequences` that solves the first problem. The tool reads NetFlow data files and outputs the port sequences found therein. In Section 2.4 we will present a solution for the second problem: An algorithm for detecting anomalies in the time series of port sequences.

2.3 `netflow_port_sequences` Tool

To analyse the port sequences in the archived NetFlow data from the DDoSVaX project we have implemented a tool called `netflow_port_sequences`. The tool reads the NetFlow data files as specified on the command line and outputs the found port sequences. The tool is written in C and has been developed under the Linux operating system. It amounts to about 2'700 lines of code.

See Appendix B.1 for a description of the command line options.

2.3.1 Data Processing

Consider three flows from a source host *srcHost* to a destination host *dstHost* as shown in Figure 1. There has been no prior communication between the two hosts. The first flow to the destination port 1433/TCP generates a new port sequence between the hosts *srcHost* and *dstHost*. The sequence initially only contains the port 1433/TCP. Because the sequence may be extended by subsequent flows from *srcHost* to *dstHost* that occur within the given time limit (being five seconds by default) the newly generated sequence is considered as being an *active* sequence. It will be inserted into the `active_sequences` hash table, containing all active sequences between two arbitrary hosts. The keys of the hash table elements consist of the two IP addresses of the according source and the destination hosts. Figure 2 gives a simplified overview of the used data structures.

When the second flow to port 1025/TCP gets processed the sequence between *srcHost* and *dstHost* is still active. The port 1025/TCP is added to the sequence and the timeout counter of the sequence is reset to its original value such that the sequence is kept active for five more seconds.

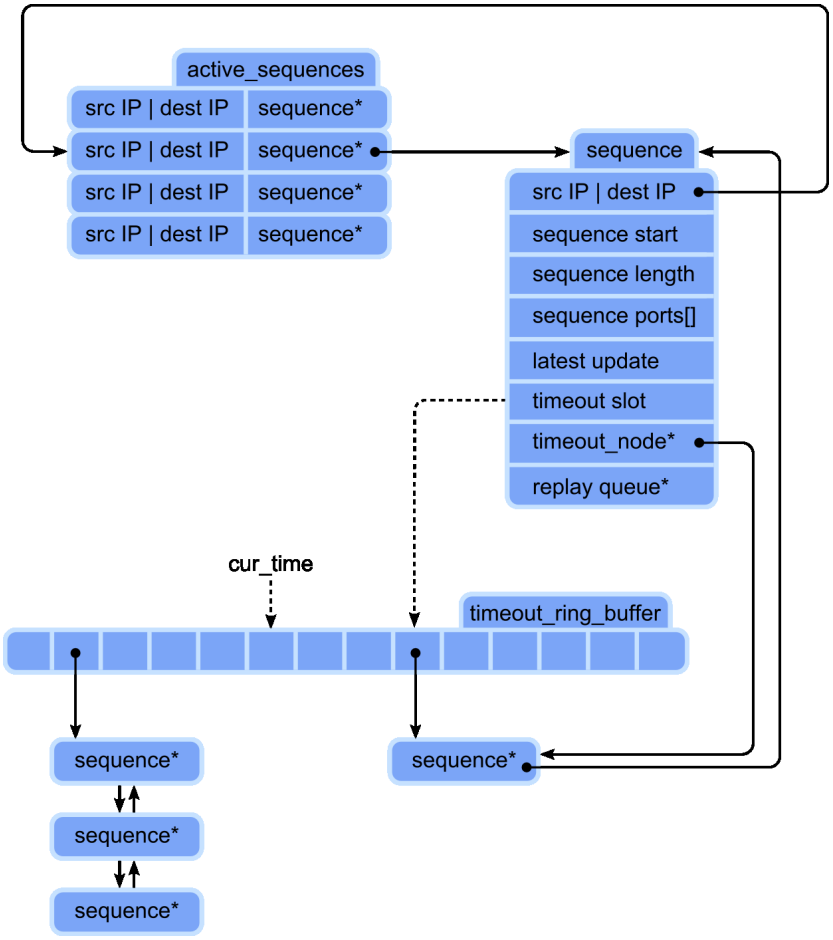

Figure 2: Simplified data structures used by the `netflow_port_sequences` tool.

Eventually the sequence's timeout will expire because no new flow from *srcHost* to *dstHost* has been observed within the given time limit. The sequence is now considered as being completed and the appropriate sequence counter holding the information that will be output by the tool, e.g. number of occurrences and number of unique addresses, of the corresponding interval is updated.

The timeout behaviour of the active sequences is implemented as a ring buffer (`timeout_ring_buffer`). Each time the internal time `cur_time` is incremented by one second all sequences in the timeout list in `timeout_ring_buffer` at the index given by `cur_time` are processed and removed from the `active_sequences` hash table. In order to permit an efficient updating of a sequence's timeout value each sequence contains a pointer to its corresponding timeout entry that belongs to a doubly linked list (see Figure 2). Thereby each entry can be removed from the list in constant time.

The NetFlow v5 records are processed in the same order as they are read from the data files. The internal time `cur_time` used to trigger the timeouts is synchronized by the end times of the processed records. Because the records within the NetFlow data are not ordered by their start times it is possible that a flow of long duration is processed after the appropriate active sequence has timed out (with respect to the internal time), although the current flow would belong to this sequence according to its start time. In order to account flows with long durations to their appropriate port sequence the retention period of an active sequence within the `active_sequences` hash table constitutes of the timeout value plus a maximal allowed flow duration (by default 30 seconds).

Through the extended retention period it is possible that a flow gets processed for which there is an active sequence but the current flow started after the sequence's timeout (such that the flow would belong to a new sequence). As there may still be flows that started within the sequence's timeout, hence extending the timeout such that the current flow would start within the sequence's timeout, the current flow will be postponed by means of the sequence's replay queue. Every time the timeout value of the sequence is updated (e.g. because a new port has been added) the replay queue is inspected for flows that now start within the timeout such that their destination port may be added to the sequence. If the replay queue of a sequence still contains some flows after the retention period has elapsed, the remaining flows will create a new active sequence.

In order to reduce memory usage we have implemented an aging mechanism for sequence counters. If a given sequence has not been observed within the last five minutes since the last update of the corresponding sequence counter, the counter will be removed from the interval. Thereby no memory is wasted for sequences that occur rarely (e.g. port scans), hence being irrelevant for the purpose of worm outbreak detection.

2.3.2 Output

Table 1 shows the attributes collected by the `netflow_port_sequences` tool, together with the according field position within the tool's output. The information provided for each port sequence can be divided into four sections: Timing

Field	Description
1.	Seconds since epoch
2.	Date in human readable representation (UTC)
3.	Time in human readable representation (UTC)
4.	Estimated number of external sources
5.	Estimated number of AS559 destinations
6.	Total number of inbound occurrences
7.	Estimated number of AS559 sources
8.	Estimated number of external destinations
9.	Total number of outbound occurrences
10.	Total number of occurrences (in– and outbound)
11.	Port sequence string, sorted by port number

Table 1: Attributes per port sequence returned by the `netflow_port_sequences` tool. The field number refers to the attribute's position within the tool's output.

information, statistics about inbound port sequences, statistics about outbound port sequences and overall number of occurrences.

Due to the amount of data the results are split into individual intervals (of one hour by default). The timing information of the fields 1. to 3. represent the start time of the corresponding interval.

The number of unique source and destination addresses that were involved in the given port sequence and interval respectively, are *estimates* and do not represent the exact values. Counting the precise amount of unique addresses for every port sequence (e.g. by storing the seen addresses in a hash table) would exceed the available resources by far. Instead we have implemented a multiresolution bitmap counter that uses very little resources per instance but yields just an estimated number of unique addresses instead of the exact number [13]. As we are interested in significant changes of the number of unique addresses over a certain time period, an estimate thereof has proven to be sufficiently accurate for our purpose.

The individual ports of the port sequence string (e.g. `139:TCP_445:TCP_`) in the last field are separated (and terminated) by '`_`' characters such that the strings of variable length may be parsed easily by other tools.

The following line is a sample output from the `netflow_port_sequences` tool for the port sequence to port 4672/TCP. For better readability the field numbers are included, too.

```
1.         2.         3.         4. 5. 6.   7.  8.  9.    10.   11.
1106038800 18.01.2005 09:00:00   41 39 51   120 185 555   606   4672:TCP_
```

Note that the `netflow_port_sequences` tool outputs by default only the sequences that occurred at least 500 times during an interval in order to reduce the number of port sequence time series that have to be analysed afterwards.

2.3.3 Performance and Memory Usage

The performance and the memory usage of the `netflow_port_sequences` tool has been tested by analysing the archived NetFlow data of the Sasser and Zotob outbreaks, processing ten hours of data captured by three out of four SWITCH border routers. The tests have been done on a computer with an AMD Athlon XP 2800+ CPU and 1GB of memory.

Using the tool's default parameters the analysis of one hour of NetFlow data took five to eight minutes with a memory usage of about 250MB. There is a wide scope for analysing more traffic without exceeding any memory limitations. Furthermore the computing is fast enough to be done in real–time.

2.4 Adaptive Anomaly Detection Algorithm

We have developed an algorithm, called *Adaptive Anomaly Detection Algorithm*, that identifies significant deviations within the time series of a port sequence. The found deviations will eventually trigger an *event* that indicates an anomaly in the traffic pattern of the corresponding port sequence. Based on the event's properties it may be classified as being due to a worm outbreak or not.

For each port sequence the algorithm performs three steps in order to find anomalies in the observed traffic pattern:

Determine significant deviations in the individual time series of the port sequence attributes. The found deviations will be regarded as potential events.

Identify relevant events by combining the potential events of several attributes.

Calculate event statistics, e.g. duration of the event or the mean number of sequence occurrences during the event.

These steps will now be discussed in greater detail.

2.4.1 Step 1: Determine Significant Deviations

The first step of our Adaptive Anomaly Detection Algorithm consists in determining significant deviations in the attributes' time series of a port sequence. A value at a given interval i is considered to be a potential event if this value surpasses a certain threshold. Instead of being fixed the threshold is adapted for each interval based on the windowed mean μ_w and the windowed standard deviation σ_w of the attribute values that occurred in previous intervals. The number of intervals that are taken into account for the calculations is specified by the window size w. The threshold value $t_w(i)$ for the interval i and a given window size w is defined as

$$t_w(i) := \mu_w(i-1) + d_w\sigma_w(i-1)$$

with $\mu_w(i-1)$ and $\sigma_w(i-1)$ being the windowed mean and windowed standard deviation respectively over the interval $[i-w, \ldots, i-1]$. A value $x(i)$ is considered as a potential event if it surpasses the threshold value $t_w(i)$:

$$x(i) > t_w(i)$$

In other words a potential event is triggered if the deviation $x(i) - \mu_w(i-1)$ is larger than $d_w\sigma_w(i-1)$. The constant *deviation factor* d_w controls the sensitivity of the algorithm — the larger d_w the more a value $x(i)$ may be off of its expected domain without triggering a potential event.

2.4.2 Step 2: Identify Relevant Events

Compared to periods without ongoing attack a worm outbreak entails a significantly increased amount of connection attempts as well as a significantly increased number of unique destination addresses within the traffic pattern used by the worm's propagation routine. Our algorithm uses this behaviour to generate relevant events that may indicate a worm outbreak and for discarding irrelevant events from the potential events that were found in the first step.

A significant deviation in the number of port sequence occurrences during a given interval has to be accompanied by a significant deviation in the number of unique destination IPs during the same interval in order to trigger an event. A singly appearing potential event is discarded.

The algorithm distinguishes between inbound and outbound events. With respect to Table 1 it tries to match the attributes "estimated number of AS559 destinations" and "total number of inbound occurrences" for inbound events and "estimated number of external destinations" and "total number of outbound occurrences" respectively for outbound events.

2.4.3 Step 3: Calculate Event Statistics

The last step consists in calculating and assigning various properties to each event which will help classifying an event in whether it was due to a worm outbreak or not. Among others our implementation of the algorithm (see Section 2.4.4) determines the following properties:

- Duration of the event
- Mean number of sequence occurrences...
  - during the event
  - before the event
  - after the event

An accurate estimation of an event's duration is an important factor for discerning its cause. An event lasting several days is more likely to be due to a worm outbreak than an event that lasts only a few hours. In our algorithm an event is supposed to be over if the value of at least one involved attribute is back to normal, that is its value is smaller than the corresponding threshold value $t_w(i_{start})$ during the interval when the event started. To smoothen possible fluctuations the values of an attribute have to remain below the trigger threshold during a given amount of time in order to terminate an event (which has been four hours in our analyses).

2.4.4 Algorithm Implementation

We have written a tool named `analyse_port_sequences` that implements our Adaptive Anomaly Detection Algorithm. The tool offers an interactive analysis and plotting environment that processes the output of the `netflow_port_sequences` tool. It uses the aforementioned algorithm do detect traffic anomalies in the port sequences returned by `netflow_port_sequences`. The analysis environment has been implemented in Python — a programming language that is well suited for the rapid prototyping approach that we used for the tool's development. The tool consists of about 1300 lines of code.

In the following analyses the window size w used by the algorithm was set to 108 intervals — with an interval duration of one hour each, the window size corresponds to 4.5 days — whereas the deviation factor d_w was set to three. These values have proven to give reliable results during our tests.

2.4.5 Using Multiple Window Sizes Simultaneously

Initially we designed the previously described algorithm to use multiple windows of different sizes in order to find significant deviations. The threshold values of small windows would adapt more quickly to the data (hence being closer to the actual data) while the thresholds of large windows would be smoother, reflecting the variance of the data over a longer period. A potential event would only be created if a value $x(i)$ exceeded at least half of the windows' thresholds.

The implementation of the algorithm in the `analyse_port_sequences` tool supports the usage of multiple windows. But we found that for detecting large scale anomalies, as they result during worm outbreaks, the usage of multiple windows did not perform better than the usage of a single window. It is important though that the size of the single window has to be large enough to avoid overfitting the data. As mentioned previously a window size covering at least 4.5 days performed very well in our analyses.

2.5 Analysis of Known Worm Outbreaks

To validate the presented port sequences based approach and the Adaptive Anomaly Detection Algorithm we have analysed the archived NetFlow data from the DDoSVaX project that has been captured during the four worm outbreaks

that have been reported during 2005[2]. Further details about the individual worms are given in Appendix A.

We first used the `netflow_port_sequences` tool to get the statistics of all port sequences that occurred during the corresponding time periods. The NetFlow data files were processed by using the tool's default settings (see Appendix B.1). Thereafter we analysed all the found port sequences with the `analyse_port_sequences` tool in order to automatically find all network events in the analysed NetFlow data.

As stated in Section 2.4.4 the algorithm's window size parameter w was set to 108 hours (4.5 days) and the deviation factor d_w to three. Note that the values of the first 4.5 days of the analysed data are used to initialize the windowed mean as well as the windowed standard deviation in order to calculate the threshold value. Hence the algorithm will not report any events within this first period.

2.5.1 Decision Criterion for Significant Events

The Adaptive Anomaly Detection Algorithm reports any anomaly that matches the requirements given in Section 2.4 as an event. An event does not necessarily has to be due to a worm outbreak. Hence a decision criterion is needed to distinguish significant events from irrelevant events with respect to worm outbreak detection. In the analyses of periods with known worm outbreaks given below we will only present such significant events that were found by the algorithm.

For this purpose we used a very simple rule for the classification of events. In order for being classified as significant an event has to last more than ten hours and the mean number of sequence occurrences during the event has to be larger than 500'000 occurrences per hour for inbound events and larger than 100'000 occurrences per hour for outbound events. Events that do not match this rule will not be considered as potential worm related events because they are too small. This simple decision criterion is based on experience while analysing the events from the worm outbreaks during 2005.

2.5.2 MySQL UDF Worm

The outbreak of the MySQL UDF worm occurred on Wednesday, January 26th, 2005. We analysed the time period from January 19th to January 30th, 2005. During that period the `netflow_port_sequences` tool found 4116 different port sequences that occurred at least 500 times in any interval.

The outbreak of the MySQL UDF worm was not perceivable within the AS559 border traffic. Neither the number of inbound connection attempts to the vulnerable MySQL port 3306/TCP nor the number of unique AS559 destination addresses exhibited any evidence for a worm outbreak, as shown in Figure 3[3]. Too few hosts were infected on a global scale (see Appendix A.1) in order to have a significant impact on the observed traffic. Accordingly the Adaptive

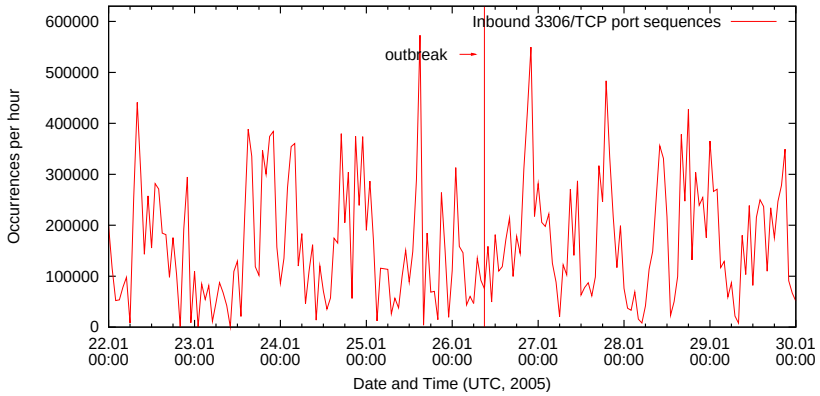

Figure 3: Number of inbound 3306/TCP port sequences during the MySQL worm outbreak.

Anomaly Detection Algorithm did not report any significant event regarding the 3306/TCP port sequence. Note that there was no internal outbreak neither.

Nevertheless the algorithm reported two significant events that occurred during the analysed time period. The first event regarded port 6129/TCP. This port is used by DameWare, a remote control administration software for Microsoft Windows operating systems. According to [24] this tool was also installed by some viruses to allow the remote administration of the infected systems.

Figures 4 and 5 show the number of inbound 6129/TCP port sequence occurrences and the number of unique AS559 destination addresses respectively. Both plots contain the threshold values for the corresponding attributes used by the algorithm. Remember that the algorithm only triggers an event if the values of *both* attributes exceed the threshold during the same interval. Also note that an event is considered to be over if the values of *one* attribute drop below the trigger threshold value for more than four intervals in a row.

Regarding the port sequence 6129/TCP the algorithm found a first event on Tuesday, January 25th, 2005, at 21:00 UTC. Because the number of unique destination addresses was again below the threshold value during the next intervals the event lasted only one hour. A second, significant event has been found on Wednesday, January 26th, at 11:00 UTC. Its beginning is indicated by the first vertical line in the plots. The plots show a significant increase of traffic to port 6129/TCP, accompanied by a significant increase of AS559 destinations. The algorithm determined a duration of 33 hours for the event. Its ending is indicated by the second vertical line in the plots.

During the analysed time period no worm outbreak has been reported that exploited a vulnerability on port 6129/TCP. However, several worm variants known to exploit DameWare vulnerabilities were active during that time — e.g. Agobot (Gaobot) [36], Mockbot [32] or Kobot [32].

[2] We analysed the traffic data captured by three out of four SWITCH border routers.

[3] Because the plot for the unique number of AS559 destination addresses is very similar to the shown plot, it has been omitted for the sake of the plot's clarity.

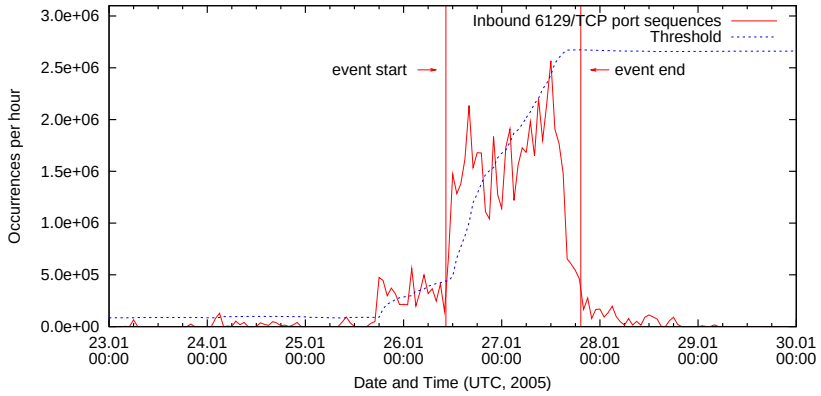

Figure 4: Number of inbound 6129/TCP port sequences.

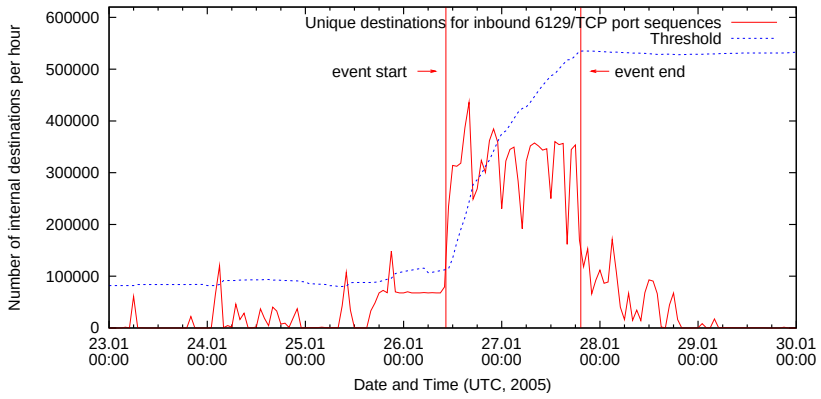

Figure 5: Estimated number of unique AS559 destination addresses for inbound 6129/TCP port sequences.

Note that the plots of the number of sequence occurrences and the number of unique destination addresses in Figures 4 and 5 show a similar curve progression. The major difference is the scale between the two plots — the number of sequence occurrences is about five times larger than the number of destinations. In the following we will omit the plot of the number of destinations if there is such a similarity between the plots. But keep in mind that the algorithm only triggers an event if there are significant deviations within *both* port sequence attributes.

The algorithm reported another significant event in the analysed data on Sunday, January 23rd, 2005, at 21:00 UTC. The event was due to a considerable increase of inbound traffic to port 1433/TCP as shown in Figure 6. During the peak period there were about 30 million inbound connection attempts per hour. On average there were about 2.8 million connection attempts per hour.

Port 1433/TCP is used by the Microsoft SQL Server and the Microsoft Desktop Engine (MSDE). Systems may be compromised by exploiting null or weak default passwords of the database administrator user ([6], [23]), which has been exploited by several worms. For the analysed time period no new worm outbreak was reported that would exploit the vulnerability. Still, according to Symantec port 1433/TCP was the fourth most attacked port between January 1st to June 30th, 2005 [34], which reflects our findings.

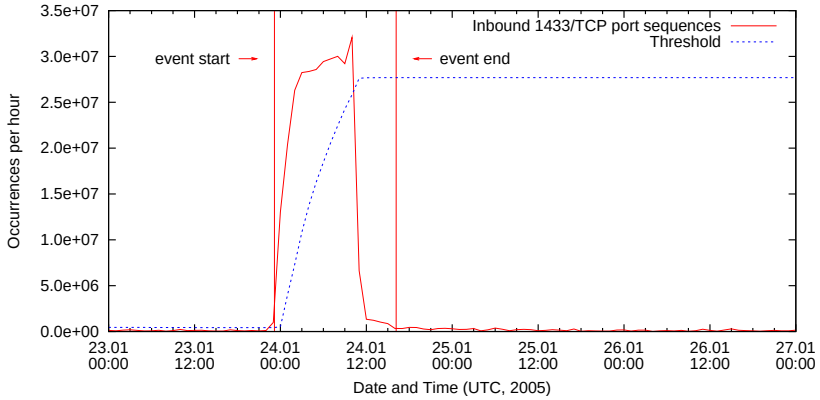

Figure 6: Number of inbound 1433/TCP port sequences.

Besides the events regarding the port sequences 6129/TCP and 1433/TCP no other event reported by the algorithm was significant with respect to the decision criterion given in Section 2.5.1. For all other events there were either too few connection attempts and / or the events lasted too short in order for being considered as potential worm related events.

2.5.3 Zotob

The Zotob outbreak took place on Saturday, August 13th, 2005. We analysed the archived NetFlow data from August 1st to August 21st, 2005. Therein 3701 different port sequences have been found, in which the Adaptive Anomaly Detection Algorithm revealed two significant events — both being due to the Zotob outbreak.

The first event has been found on Saturday, August 13th, 2005, at 3:00 UTC, where the number of inbound connection attempts to port 445/TCP as well as the number of unique AS559 destinations exceeded the corresponding threshold values. Figure 7 illustrates the course of the outbreak. It shows a significant increase of the inbound traffic to port 445/TCP with an average of over one million connection attempts per hour during the event. The start time of the event determined by the algorithm coincides with reports about the Zotob outbreak [29]. Hence we can state that the algorithm detected an event that was due to the Zotob outbreak.

Note that the algorithm identified a preceding event on Friday, August 12th, at 17:00 UTC. As the reported event duration is only two hours the event is not considered as being significant.

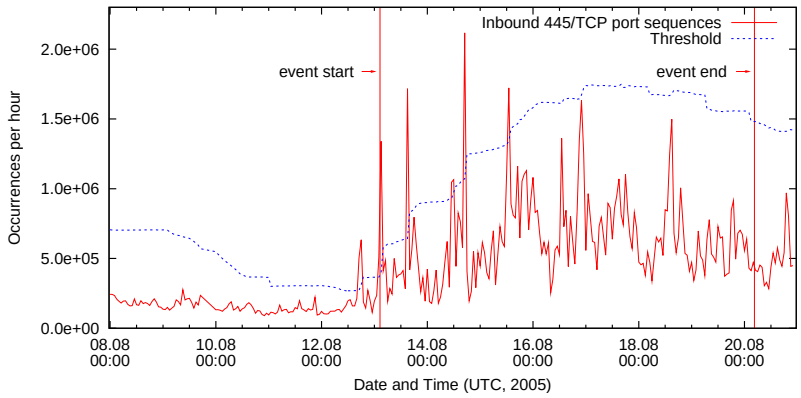

Figure 7: Number of inbound 445/TCP port sequences. The begin of the Zotob outbreak as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

The second significant event reported by the algorithm occurred on Friday, August 19th, at 16:00 UTC and concerned the outbound traffic. It lasted 25 hours with an average of over 200'000 outbound connection attempts per hour. Comparing with Figure 8 there was indeed a significant increase of outbound traffic to port 445/TCP. This second event represents the AS559 internal outbreak of the Zotob worm.

Note that the peak of inbound connection attempts that occurred during the afternoon on Sunday, August 14th, does not represent a significant event as defined in Section 2.5.1 because it lasts less than ten hours.

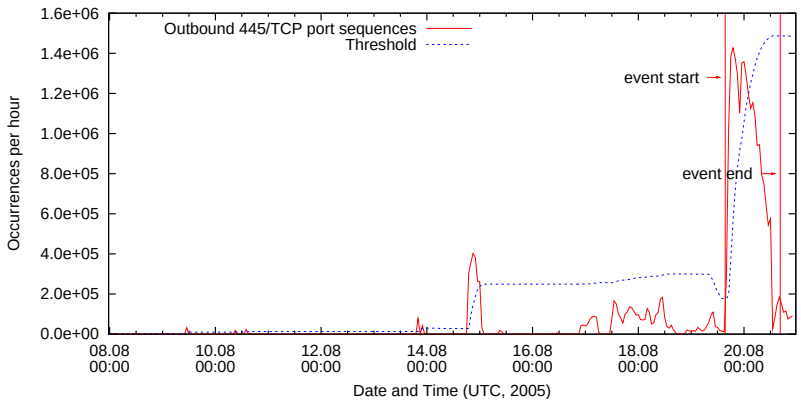

Figure 8: Number of outbound 445/TCP port sequences. The begin of the AS559 internal Zotob outbreak as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

No other significant event was reported by the algorithm within the analysed 13 days.

2.5.4 Lupper

The Lupper worm was discovered on the Internet on Tuesday, November 8th, 2005. The archived NetFlow data we analysed ranged from November 1st to November 14th, 2005. A total of 3705 different port sequences were found during that period.

Figure 9 shows the inbound connection attempts to port 80/TCP — the port that was exploited by Lupper worm. Apparently the outbreak did not have a very large impact on the inbound network traffic. Our algorithm reported an inbound event for port 80/TCP that started on Tuesday, November 8th, at 11:00 UTC and lasted 54 hours. With 230'000 connection attempts per hour the event may not be considered as being significant. Nevertheless it was by far the most prominent event found during the analysed period, both with respect to its duration and to the number of sequence occurrences per hour.

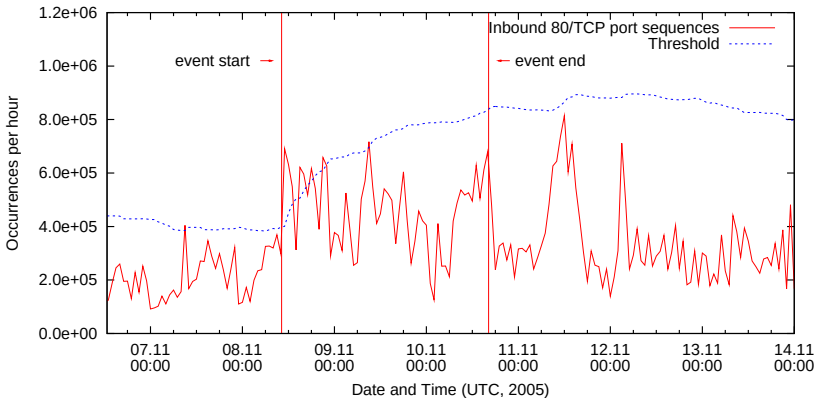

Figure 9: Number of inbound 80/TCP port sequences. The begin of the Lupper outbreak as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

Figures 10 and 11 show the number of outbound connection attempts and number of external destinations respectively destined to port 80/TCP. Especially the number of external destinations contacted by AS559 hosts shows a significant increase on Tuesday, November 8th at 13:00 UTC, giving strong evidence that the Lupper worm propagated into AS559. The number of outbound connection attempts in Figure 10 was also affected by the internal outbreak, which is clearly visible in the elevated amount of outbound traffic during the night from Tuesday, November 8th. Prior and after the outbreak there is only little traffic during the nights — most people browse the web during the day and not at night. This changed during the Lupper outbreak that infected web servers. As the infected servers kept running during the night they generated traffic to port 80/TCP while scanning for further victims, resulting in the increased nightly traffic.

As indicated by the vertical lines within Figures 10 and 11 the beginning of the outbreak has been detected by our algorithm, but its duration has been

estimated to just two hours. The reason for this wrong estimation was that the number of outbound connection attempts to port 80/TCP soon dropped below the threshold value and that the algorithm did not take into account the anomaly in the periodic nature of the data.

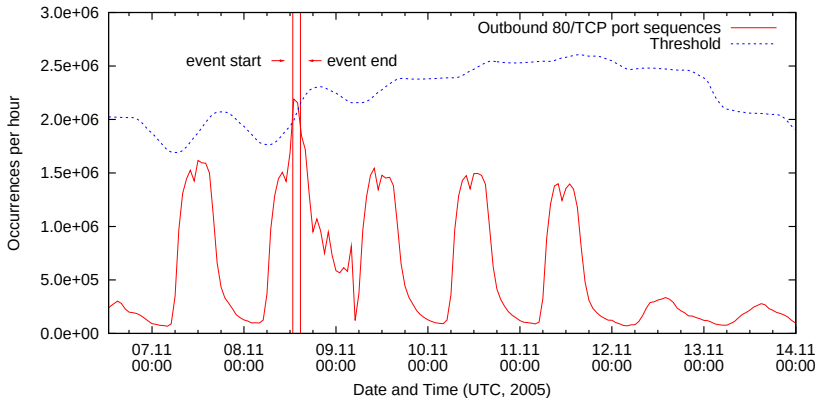

Figure 10: Number of outbound 80/TCP port sequences. The begin of the internal Lupper outbreak as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

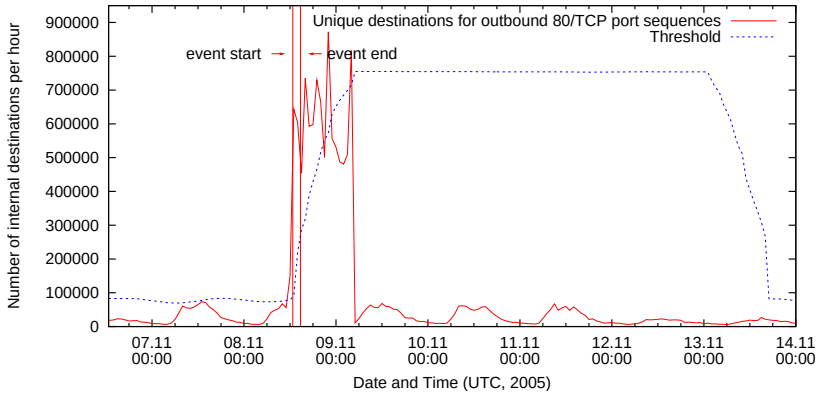

Figure 11: Estimated number of unique AS559 destination addresses for outbound 80/TCP port sequences. The begin of the internal Lupper outbreak as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

2.5.5 Dasher

We analysed the archived NetFlow data ranging from December 3rd to December 31st, 2005. According to antivirus vendors the Dasher outbreak took place on Thursday, December 15th, 2005. During the analysed period the `netflow_port_sequences` tool found 6871 different port sequences in which one significant event has been found.

This event has been reported by the Adaptive Anomaly Detection Algorithm on Thursday, December 8th, at 13:00 UTC, regarding port 1025/TCP — the same port that has been exploited by Dasher's propagation routine. Figure 12 shows the number of inbound connection attempts to port 1025/TCP. The strong and fast increase from virtually no connection attempts prior the event to over two million attempts shortly after the outbreak and to over three million attempts about 24 hours later suggests that this significant event (with an average of two million connection attempts per hour) has been due to a worm outbreak. Puzzling is the fact that antivirus vendors dated the Dasher outbreak a week later, namely on Thursday, December 15th [35], [14]. However, the Information Technology Security Center of Japan [19] made the same observations about the 1025/TCP traffic and name the Dasher worm as its cause.

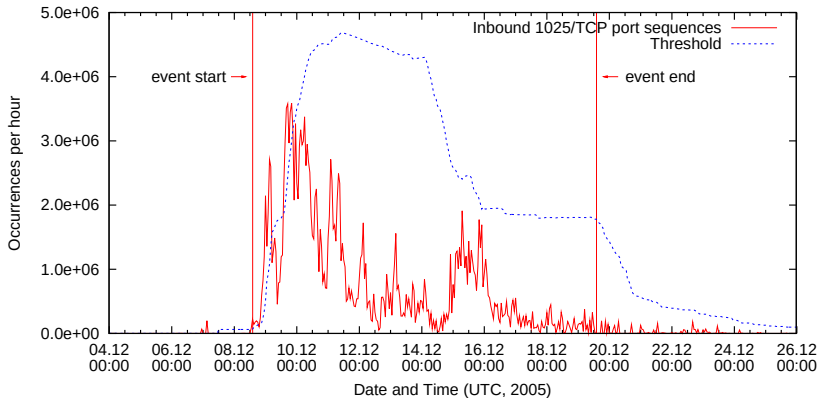

Figure 12: Number of inbound 1025/TCP port sequences. The begin of the Dasher outbreak as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

A second event has been found for the port sequence 1025/TCP, 1433/TCP on Friday, December 16th, at 0:00 UTC (Figure 13). With an average of only 4000 sequence occurrences per hour the event would certainly not be considered as being significant. But as the port sequence shares a port (1025/TCP) with a previously found event that has been classified as being significant, we have evidence that the new event was due to a variant of the worm that triggered the last event and thus should also be considered as being significant.

Four hours later, on Friday, December 16th, at 4:00 UTC a third event has been reported for the port sequence 42/TCP, 1025/TCP, 1433/TCP as shown in Figure 14. With an average of 40'000 sequence occurrences per hour the event

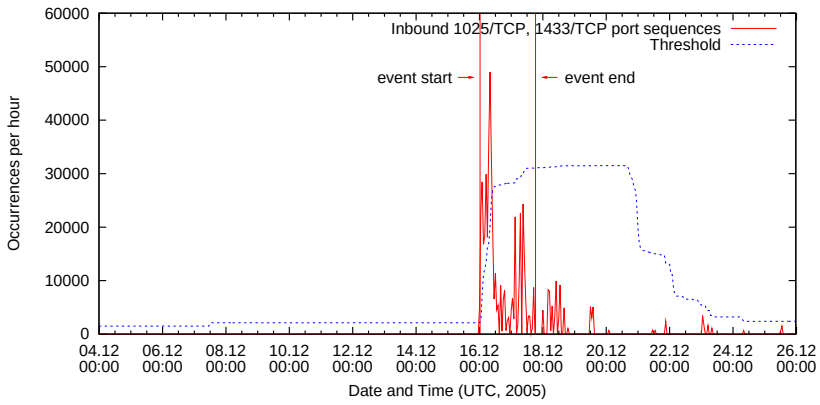

Figure 13: Number of inbound 1025/TCP, 1433/TCP port sequences. The begin of the outbreak of the Dasher variant as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

alone was also not significant. But now there were two other previous events sharing common ports (1025/TCP and 1433/TCP) with this event.

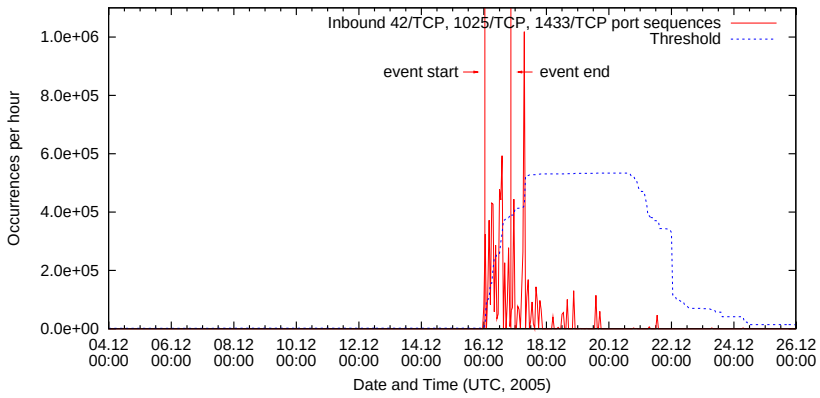

Figure 14: Number of inbound 42/TCP, 1025/TCP, 1433/TCP port sequences. The begin of the outbreak of the Dasher variant as it has been identified by the Adaptive Anomaly Detection Algorithm is indicated by the first vertical line.

In fact later variants of the Dasher worm additionally tried to exploit, besides the MSDTC vulnerability through port 1025/TCP, older vulnerabilities in services like WINS (42/TCP), LSASS (445/TCP) and MSSQL (1433/TCP), depending on the variant (see Appendix A.4). Hence the two events after the first Dasher event were due to Dasher variants. This demonstrates an advantage of the port sequence based anomaly detection compared to just analysing the occurrence frequency of single ports. By matching an event with previous events that share common ports, we may identify worm related events that otherwise would be considered as being nonsignificant due to their lower amount of occurrences.

Additionally to the Dasher related events the algorithm found an event concerning port 8080/TCP. As shown in Figure 15 the event started on Thursday, December 8th, at 19:00 UTC. Although it started the same day as the Dasher outbreak there is no evidence for a correlation of the events. During the day of the event an exploit has been released for a buffer overflow vulnerability in the Oracle 9i XML Database (XDB), exploitable through port 8080/TCP [15]. As there was virtually no inbound traffic to port 8080/TCP prior the release we expect that the event was due to scans for vulnerable system. This event was not significant with respect to worm outbreaks but it shows that the algorithm may also be used to detect the releases of new exploits.

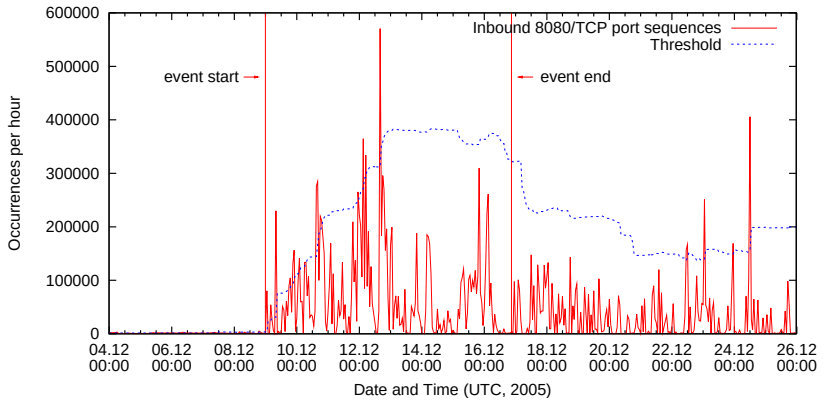

Figure 15: Number of inbound 8080/TCP port sequences.

2.6 Encountered Problems

The biggest challenge during the development of the `netflow_port_sequences` tool was performance and memory usage. Our initial intent was to use very few and simple data structures with few connections between them. The aim was to keep the code as clean and simple as possible. The optimizations should be done by the compiler, not the programmer. Due to the large amount of flow records that have to be processed and the large number of port sequences that have to be tracked, this approach yielded a very poor performance. We first optimized the hash table code where the program spent the most computing time (e.g. inlining often called procedures, avoiding the shrinkage of the table while deleting elements) to find out later that for the particular task we optimized the hash tables for (handling of timeouts), using hash tables was the wrong solution and that doubly linked lists with pointers to their elements was a lot faster. This resulted in interconnected data structures but also in much better performance.

The need for counting the number of unique addresses time and space efficiently posed another very important problem. For every port sequence four address counters are needed (source and destination addresses for inbound and outbound flows). Due to the large amount of different sequences that have to be tracked during a single interval (up to 100'000) we could not use hash tables or bitvector tables covering the whole address space because they would exceed the available memory[4]. We then implemented a bitvector based address counter for a limited number of addresses that belong to a configurable set of address ranges. For the address ranges belonging to AS559 — allowing to count the number of unique destinations for inbound traffic and the number of unique sources for outbound traffic but no external addresses — one counter needed about 300KB of memory. This was still too much for the number of instances we needed. We finally implemented a multiresolution bitmap counter as presented in [13]. One instance requires only little more than 200 Bytes of memory but provides just an estimate value of the number of unique addresses. For detecting significant deviations the accuracy is high enough and the low amount of memory needed allows us to count the number of internal as well as the number of external addresses for inbound and outbound sequences.

Few days before the submission date of the thesis we discovered that our script (`analyse_port_sequences`) that implements the Adaptive Anomaly Detection Algorithm did not report an outbound event during the afternoon on August 14th, 2005, during the Zotob outbreak (see Section 2.5.3), although the number of outbound connection attempts to port 445/TCP as well as the number of unique destination addresses were larger than their thresholds (Figure 8). Unfortunately we never noticed this error before — whether in this particular port sequence nor in other sequences of the analysed time periods. During our test the script did always reliably report the events as expected. Regrettably we had no time left to trace the error in the `analyse_port_sequences` script. Note that this error does not affect the algorithm itself. The values of the attributes are larger than the threshold values and thus represent an event as defined by the algorithm.

[4] E.g. the `ip_table.c` implementation from the DDoSVaX project has an overhead of 64MB per instance.

2.7 Future Work

Proposals for further improvements of the presented approach are:

- We expect that the presented Adaptive Anomaly Detection Algorithm may also be used for detecting (D)DoS attacks. Therefore the algorithm would have to match significant deviations in the number of sequence occurrences and unique source addresses (instead of unique destination addresses as presented in this thesis).
- We have used the decision criterion for the offline analysis of archived NetFlow data for which it performed well. Because the duration of an event is not known when the Adaptive Anomaly Detection Algorithm reports an event during an online analysis, the decision criterion has to be adapted to meet the requirements of the online analysis.

2.8 Conclusion

The implementation of the `netflow_port_sequences` tool shows that it is possible to monitor efficiently all the port sequences that occur within the border traffic of an autonomous system like AS559 simultaneously. Our performance tests have shown that real–time port sequence analysis is feasible in terms of computing time as well in terms of memory requirements.

We have presented an algorithm that identifies significant events within the border network traffic based on the results of the port sequence analysis. We validated the algorithm by analysing known worm outbreaks that occurred during 2005. The algorithm detected reliably events due to worm outbreaks and other significant network anomalies. No false positives were reported. The events were reported as soon as there was any significant deviation in the observed traffic with a detection latency of about one hour. The detection latency is primarily given by the interval duration used for the port sequence analysis.

We have shown that the port sequence approach is able to identify worm outbreaks that are due to variants of previous worms. If one or multiple ports of an event's port sequence coincide with ports of the port sequence of a previous worm outbreak event there is evidence that the current event is due to a variant of the former worm.

3 Propagation Triples

The idea behind the following approach is to detect propagation paths of a worm as it spreads from host to host. A worm's propagation routine causes an infected host X to show a specific traffic pattern. If a second host Y shows the same traffic pattern while communicating with a third host Z *after* having been contacted by the first host X, the pattern may be due to the worm that propagated from host X to host Y, trying now to infect host Z. In the following we will investigate if propagating traffic patterns may reveal worm outbreaks.

3.1 Definition

Assume we observe in a network two communication flows as shown in Figure 16. First host X contacts host Y on port 445/TCP. Thereafter host Y shows the same traffic pattern as host X while contacting host Z. Thus we have observed a propagation path from X to Y and from Y to Z for the traffic pattern "contact the destination host on destination port 445/TCP". The Zotob worm showed the same pattern while exploiting a vulnerability of Microsoft Windows 2000 systems through port 445/TCP (see Appendix A.2). Hence the observed propagation path could have been the result from host X infecting host Y with the Zotob worm, which in turn tried to infect host Z.

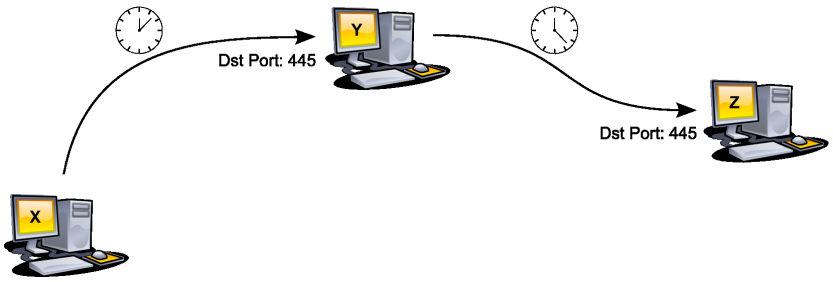

Figure 16: Propagation triple of the Zotob worm between the hosts X, Y and Z.

In order to describe a propagation path of a traffic pattern formally, we introduce the concept of *propagation triples*. A propagation triple consists of two distinct flows between three hosts. One of the hosts has to act as server in the first flow and as client in the second flow.

A *flow* is a tuple describing a unidirectional communication between two hosts:

$$Flow := (srcHost, dstHost, flags, flowstart).$$

The $srcHost$ and $dstHost$ components identify the two communication endpoints of the flow which started at the time $flowstart$. The $flags$ component is an arbitrary sized set containing information about the communication (e.g. TCP fields like source and destination ports).

The two predicates

$$P_1(flow_1, flow_2) := dstHost_{flow_1} = srcHost_{flow_2}$$
$$P_2(flow_1, flow_2) := flowstart_{flow_1} \leq flowstart_{flow_2}.$$

represent the basic traffic pattern

$$pattern(flow_1, flow_2) := \bigwedge_i P_i(flow_1, flow_2)$$

that has to be met by two arbitrary flows in order to constitute a general propagation triple

$$PropagationTriple := (flow_1, flow_2, pattern)$$

with $pattern(flow_1, flow_2)$. In order to define a certain class of propagation triples that show a specific traffic pattern, further predicates may be added to *pattern* according to the given application.

3.2 Algorithm

As a worm propagates from one host to another, the latter will show a similar traffic pattern as the first one due to the worm attempting to infect further hosts. On a global view every successful infection will induce a new propagation triple matching the traffic pattern used by the worm's propagation routine. Therefore the presence (or a significant increase) of propagation triples belonging to a given class may reveal a worm outbreak.

In the following we will analyse the presence of propagation triples within the NetFlow data captured by the SWITCH backbone border routers. Due to the fact that the available NetFlow data contains only traffic flows that enter or leave the autonomous system AS559, we may only detect propagation triples that cross the border of AS559. Therefore we will restrict our analysis in finding propagation triples where the intermediate endpoint belongs to AS559 while the two other endpoints are located outside AS559.

The actual algorithm for finding propagation triples within the border router NetFlow data is as follows: After an AS559 host got contacted by an external host, all outbound flows of the AS559 host will be observed, yielding eventually a propagation triple with the observed AS559 host as intermediate host.

In order to narrow the domain of possible propagation triple classes, we will only consider IP flows using either TCP or UDP as transport protocol. Furthermore we will restrict our analysis to propagation triples having a constant destination port, e.g. where both flows have the same destination port (and protocol). Finally we require the second flow of a propagation triple to occur within ten minutes after the first flow has been observed.

These restrictions yield the following additional, specialised predicates for the *pattern* component of the propagation triples being considered:

$$
\begin{aligned}
P_{Direction}(flow_1, flow_2) &:= dst_{flow_1} \in \text{AS559} \wedge src_{flow_1}, dst_{flow_2} \notin \text{AS559} \\
P_{Protocol}(flow_1, flow_2) &:= protocol_{flow_1} = protocol_{flow_2} = (\text{TCP} \vee \text{UDP}) \\
P_{ConstPort}(flow_1, flow_2) &:= dstPort_{flow_1} = dstPort_{flow_2} \\
P_{TimeDelta}(flow_1, flow_2) &:= flowstart_{flow_2} - flowstart_{flow_1} \leq 10\text{min}
\end{aligned}
$$

Figure 17 summarizes the statements above. Only the traffic between the hosts A, B and C represents a propagation triple matching the required traffic pattern. The traffic between the hosts A, D and E forms a non–constant destination port propagation triple and the traffic between the hosts F, G and H won't be considered because the second flow did not occur within the requested time limit.

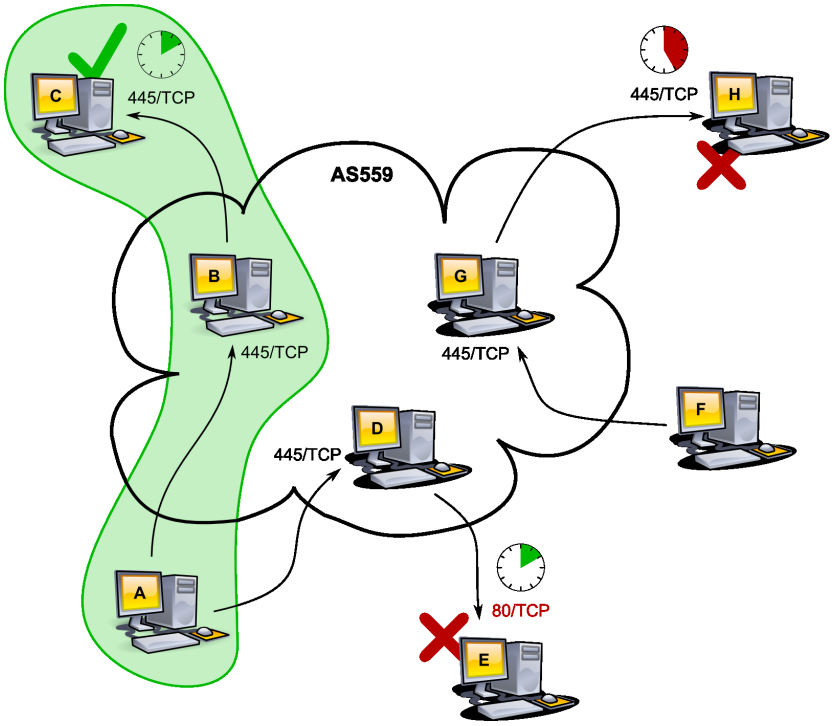

Figure 17: In our analysis we only consider propagation triples like the one between the hosts A, B and C. The other two propagation triples do not match the required traffic pattern.

3.2.1 Algorithmic Limitations

As our view in the context of the DDoSVaX project is limited to traffic entering or leaving the autonomous system AS559, we may only identify propagation triples during a worm outbreak if hosts within AS559 get infected. As long as no internal infections take place the algorithm will not be able to detect the outbreak, because no internal host will show the worm's traffic pattern while contacting external hosts.

Also note that due to the limited amount of time a contacted host is observed and due to the lack of a global view, we may not associate a new propagation triple to a new infection[5]. The corresponding host may as well have been infected earlier but its observation timeout expired before the host began scanning any external hosts. A subsequent scan by an external host then made the host in question being observed again, when it eventually scanned an external host to trigger the propagation triple. The internal host may also have been infected by another internal host and the propagation triple emerged only after the host was contacted by an external host.

3.3 Propagation Triples Without Ongoing Worm Outbreak

In order to validate the concept of propagation triples with respect to worm outbreak detection we have analysed the NetFlow data from the DDoSVaX archive. For this purpose the `netflow_triples` tool was developed, whose implementation details will be discussed in Section 3.5.

Before we are going to analyse the propagation triples emerging during known worm outbreaks we will have a look at two periods of time without any major ongoing worm outbreak as a reference. Therefore we analysed intervals of 48 hours of the traffic data captured by three out of four SWITCH border routers prior the Blaster and Zotob outbreaks.

3.3.1 Prior the Blaster Outbreak

The analysed data prior the Blaster outbreak ranges from Wednesday, August 6th, 2003, 0:00 UTC to Thursday, August 7th, 2003, 24:00 UTC. The Blaster outbreak took place four days later on Monday, August 11th, 2003.

During the analysed 48 hours we found 1685 propagation triples with constant destination ports. Out of 728 different traffic patterns only 18 traffic patterns generated five or more propagation triples during the 48 hours. 899 triples belonged to one of these 18 classes, hence over 53% of all triples were due to only 2.4% of the classes.

Table 2 lists these 18 propagation triple classes that generated at least five triples during the analysed 48 hours. The table also specifies the total number of occurrences as well as the maximum number of occurrences during a one hour interval. The last column contains the common usage of the triples' destination ports.

[5] Assuming we analyse the propagation triples that emerge during a worm outbreak.

Port	Protocol	Occurrences during 48h	Max occ. in 1h	Usage
80	TCP	490	113	HTTP
53	UDP	115	16	DNS
25	TCP	97	12	SMTP
4662	TCP	50	10	eDonkey P2P
1214	TCP	25	3	Kazaa P2P
3531	UDP	24	2	Joltid PeerEnabler P2P[a]
3531	TCP	14	6	Joltid PeerEnabler P2P
6346	TCP	12	2	Gnutella P2P
113	TCP	10	3	Ident Service
4672	UDP	10	2	eDonkey P2P
2234	TCP	9	2	Soulseek P2P
137	UDP	8	2	MS NetBios Name Service
6881	TCP	7	3	Bittorrent P2P
6882	TCP	6	2	Bittorrent P2P
6883	TCP	6	1	Bittorrent P2P
500	UDP	6	2	VPN Key Exchange
6699	TCP	5	2	winMX P2P
6257	UDP	5	1	winMX P2P

Table 2: Most frequent propagation triples found during an interval of 48 hours prior the Blaster outbreak.

[a]Bundled with Kazaa

As shown in the table two thirds of the propagation triple classes (12 out of 18) were due to P2P traffic. These propagation triples are due to the functioning of P2P networks, where a client often acts as server as well, hence representing a triple's intermediate endpoint. The second and third most propagation triple classes were generated by DNS and SMTP traffic — both being widely used protocols which are also used for inter–server communication. The majority (13 out of 18) of the found triples occurred at most 3 times during a one hour interval.

By far the most frequently observed traffic pattern was port 80/TCP, commonly used for HTTP, amounting to 29% of all found triples. We don't know the reason for this large amount of propagation triples compared to the other classes.

One possibility for this large amount of triples could be port scans. HTTP is one of the most used protocols in the Internet[6]. Hence the probability that a host initiates a connection to a web server within ten minutes after having been scanned on port 80/TCP is significantly higher compared to other, less used protocols. Traffic to port 80/TCP may also be due to P2P clients as users configure their P2P clients to use port 80/TCP instead of their default ports in order to circumvent firewalls.

Analysing the data revealed that only twelve different hosts were the initiating endpoint for 262 out of 490 HTTP propagation triples — hence these twelve

[6]Over 42% of SWITCH's transatlantic traffic in 2001 was due to HTTP [22]

hosts triggered over 50% of all HTTP propagation triples. The temporal succession of the corresponding triples — for each source they occurred within a few minutes — does suggest that these triples were initiated by a scan to port 80/TCP.

As the traffic data to port 80/TCP does not show any anomalies during the analysed period, this analysis endorses the assumption that a large amount of the HTTP propagation triples was due to (human) scanning activities and not, for instance, to worm related communication patterns.

3.3.2 Prior the Zotob Outbreak

The second propagation triple analysis during a period without ongoing worm outbreaks has been done on the archived data prior the Zotob outbreak, more precisely from Wednesday, August 10th, 2005, 0:00 UTC to Thursday, August 11th, 2005, 24:00 UTC. The Zotob outbreak took place two days later on Saturday, August 13th, 2005.

In total we found 2399 propagation triples belonging 1028 different classes during the analysed 48 hours. 29 traffic patterns generated at least five propagation triples — they represent 2.8% of all classes and generated 1357 triples, being over 56% of all found triples.

These propagation triple classes that generated at least five triples during the analysed 48 hours are listed in Table 3. The table also specifies the total number of occurrences as well as the maximum number of occurrences during a one hour interval. The last column contains the common usage of the triples' destination ports.

As in the analysis prior the Blaster outbreak the top three propagation triple classes belong to the HTTP, SMTP and DNS protocols and there is still a fair amount of P2P propagation triple classes. The maximum number of triples per hour is also comparable to the one prior the Blaster outbreak and averages (without the top three triple classes) at less than three triples per hour.

However, the three top classes are followed by a new type of propagation triple classes, destined to ports 1026/UDP and 1027/UDP. These ports are used by spammers to display anonymously pop–up messages on Microsoft Windows systems running the messenger service [25].

Note that much of the messenger spam propagation triples are due to a limitation of the `netflow_triples` analysis tool. In 65% of all the 1026/UDP and 1027/UDP triples the source port of the second flow was 53/UDP. The corresponding propagation triples emerged while a spam server sent a UDP spam message to port 1026/UDP or 1027/UDP of a DNS server. If this server then replied to a legitimate DNS request that had a source port of 1026/UDP or 1027/UDP (being the destination port in the reply), a new propagation triple has been created. This happens if the reply flow is processed before the actual request flow (see Section 3.6.1 for further details).

Port	Protocol	Occurrences during 48h	Max occ. in 1h	Usage
80	TCP	524	194	HTTP
25	TCP	199	40	SMTP
53	UDP	171	26	DNS
1027	UDP	89	7	Windows Messenger Spam
1026	UDP	87	9	Windows Messenger Spam
123	UDP	57	7	NTP
4662	TCP	29	6	eDonkey P2P
6346	TCP	23	3	Gnutella P2P
6346	UDP	14	2	Gnutella P2P
23127	UDP	14	2	Unknown
22	TCP	13	4	SSH
6881	TCP	11	1	Bittorrent P2P
4672	UDP	11	1	eDonkey P2P
23126	UDP	10	2	Unkown
4661	TCP	9	2	eDonkey P2P
4000	TCP	9	1	Terabase Search Engine
3531	UDP	9	2	Joltid PeerEnabler P2P[a]
1863	TCP	8	2	MSN Messenger Protocol
500	UDP	8	2	VPN Key Exchange
0	UDP	8	2	Unknown
3124	TCP	7	2	Beacon Port
2492	TCP	7	3	Groove Virtual Office P2P
10000	UDP	7	1	CISCO VPN-Client
6882	TCP	6	1	Bittorrent P2P
21	TCP	6	4	FTP
5850	UDP	6	2	Open DHT[b]
6348	TCP	5	1	Gnutella P2P
119	TCP	5	2	NTP
10000	TCP	5	2	CISCO VPN-Client

Table 3: Most frequent propagation triples found during an interval of 48 hours prior the Zotob outbreak.

[a] Bundled with Kazaa
[b] Distributed hash table for P2P networks

3.3.3 Conclusion

Except for the propagation triple classes of the HTTP, SMTP and DNS protocols, only few propagation triples can be found for any class. The vast majority of the found classes appeared less than five times per hour — most of them even appeared less than five times over the analysed 48 hours.

3.4 Analyses of Known Worm Outbreaks

In the following we will present the results of the propagation triple analyses of three known worm outbreaks. Each one of the worms was able to propagate into AS559, triggering an internal outbreak. The data is split into one hour intervals. The time given in the plots is the start time of such an interval. All analyses have been conducted on the captured NetFlow data of three out of four SWITCH border routers.

3.4.1 Blaster

The Blaster worm [7] was first observed in the Internet on Monday, August 11th, 2003. Its propagation routine exploited a buffer overflow vulnerability in the DCOM RPC service of Microsoft Windows 2000 and Windows XP operating systems. A detailed description of Blaster and its propagation technique can be found in [12].

Blaster's propagation routine exploited the RPC vulnerability through port 135/TCP. Figure 18 shows the result of the propagation triple analysis conducted on archive data of the Blaster outbreak. The plot covers the range from Monday, August 11th, to Thursday, August 14th, 2003. In order to show the development of the internal outbreak, the plot does also show the number of outbound connection attempts to port 135/TCP initiated by AS559 hosts.

As shown in the plot the internal outbreak of the Blaster worm took place in the early morning working hours of Tuesday, August 12th, namely between 7:00 UTC and 10:00 UTC. The internal outbreak is accompanied by a significant amount of propagation triples — during the interval from 9:00 UTC to 10:00 UTC over 100 propagation triples have been found, followed by another 60 triples within the next two hours. Compared to the number of propagation triples found during periods without any outbreak (see Section 3.3) this is a significant number. The propagation triples emerged two hours after the beginning of the internal outbreak.

3.4.2 Sasser

The Sasser worm affected Microsoft Windows 2000 and Windows XP operating systems [33]. It was first observed on Saturday, April 30th, 2004. The Sasser worm exploited a buffer overflow in the LSASS (Local Security Authority Subsystem Service) component of the affected Windows systems.

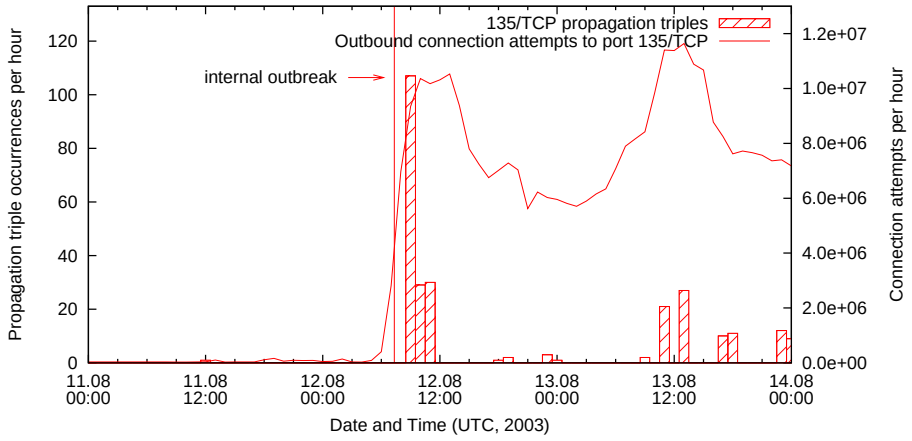

Figure 18: Number of propagation triples and number of outbound connection attempts to destination port 135/TCP during the Blaster outbreak.

The vulnerability was exploitable through port 445/TCP. Figure 19 shows the number of corresponding propagation triples per interval from Thursday, April 29th, to Wednesday, May 5th, 2004. The number of outbound connection attempts to port 445/TCP initiated by AS559 hosts is also shown in the plot.

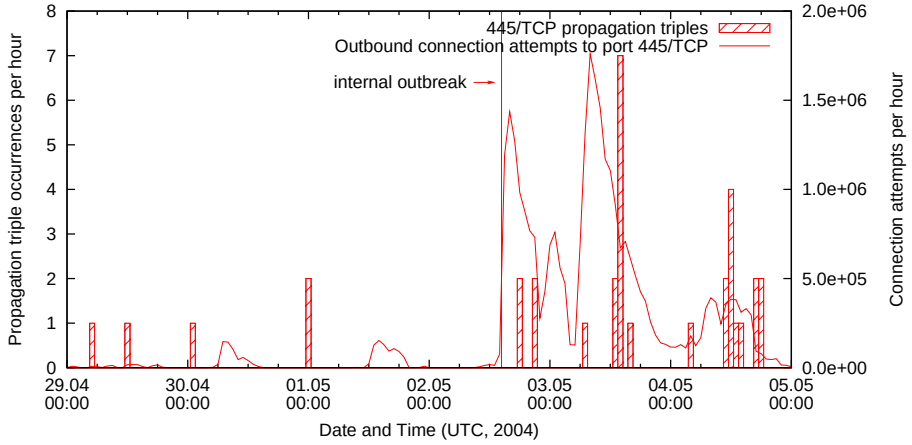

Figure 19: Number of propagation triples and number of outbound connection attempts to destination port 445/TCP during the Sasser outbreak.

The plot shows a first significant increase of outgoing connection attempts to port 445/TCP about noon on Sunday, May 2nd, 2004, followed by a second increase on Monday, May 3rd, during the early morning working hours. Compared to the Blaster outbreak the internal outbreak of the Sasser worm was quite small, peaking at about 1.5 million outbound connection attempts (compared to over one billion outbound connection attempts during the Blaster outbreak).

As a result only few propagation triples emerged, reaching a peak of seven found triples on Monday, May 3rd. Compared with the maximum number of propagation triples without any worm outbreak from Section 3.3, the numbers of triples during the Sasser outbreak do not represent a significant signal.

By cumulating the number of found propagation triples over several intervals the signal becomes stronger. Figure 20 shows the cumulated number of found propagation triples. Between Monday, May 2nd, 18:00 UTC and Tuesday, May 3rd, 16:00 UTC the number of propagation triples increased from five triples so far to 20 triples, representing an increase of 15 triples within less than 24 hours. A second increase by 13 triples can be seen during Tuesday, May 4th.

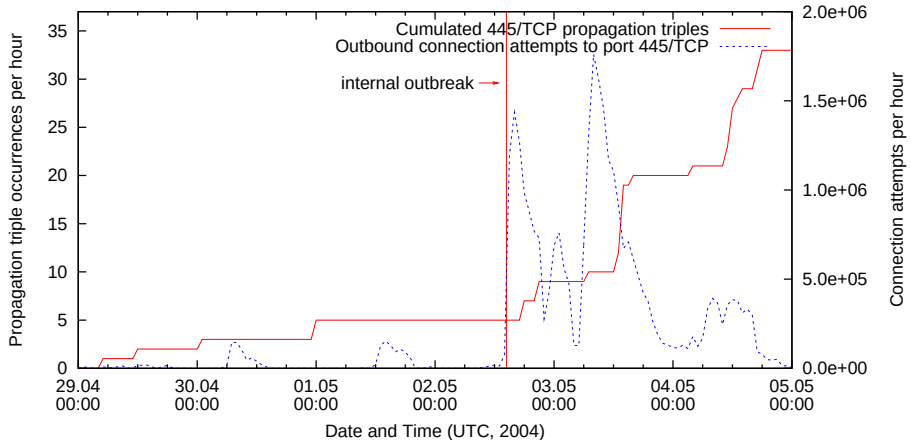

Figure 20: Cumulated number of propagation triples and number of outbound connection attempts to destination port 445/TCP during the Sasser outbreak.

In contrast to the Blaster outbreak the internal Sasser outbreak has not been strong enough to yield a significant amount of propagation triples within a single interval. But by considering the number of propagation triples over several intervals, the signal becomes much more significant.

3.4.3 Zotob

On Saturday, August 13th, 2005, the Zotob worm has been discovered exploiting a vulnerability in the Plug and Play service of Microsoft Windows 2000 systems through port 445/TCP. The plot in Figure 21 shows the number of 445/TCP propagation triples from Monday, August 15th, to Sunday, August 21st, 2005. The plot does also show the number of outbound connection attempts to port 445/TCP initiated by AS559 hosts during that period of time.

The internal outbreak started on Friday, August 19th around 16:00 UTC, apparent in the strong increase of outbound connection attempts to port 445/TCP. Two hours after the internal outbreak 31 propagation triples were found, providing a clear signal for an ongoing outbreak. The outbreak was proceeded by a total of another eleven propagation triples around noon of the same day.

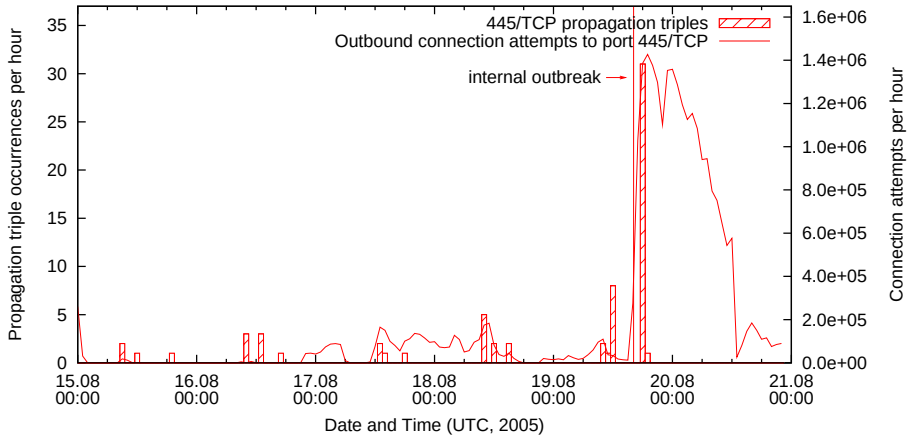

Figure 21: Number of propagation triples and number of outbound connection attempts to destination port 445/TCP during the Zotob outbreak.

3.5 Implementation

The algorithm described in Section 3.2 has been implemented in a tool called `netflow_triples`. The tool reads the NetFlow data files as specified on the command line and outputs all propagation triples matching the traffic patterns as described in Section 3.2. The tool is written in C and has been developed under the Linux operating system. It amounts to about 1'200 lines of code.

See Appendix B.2 for a description of the command line options.

3.5.1 Data Processing

The `netflow_triples` tool processes the NetFlow v5 records in the order as they are read from the data file(s). Only the records that match the specified traffic pattern and that are either in– or outbound get processed.

For an inbound flow we first check the hash table `processed_as559_triple_hosts` if a propagation triple has already been found for the corresponding AS559 destination host and destination port[7]. If so we discard the current flow because it cannot generate a new propagation triple we do not know yet. Thus the tool will only output the first occurrence of a propagation triple for a given traffic pattern and intermediate AS559 host combination, preventing triple duplicates.

If there was no propagation triple yet for the AS559 destination host and its destination port, the current flow may eventually represent the first flow of a new propagation triple. Therefore the AS559 destination host and destination port combination will be inserted into the `as559_destinations` hash table as

[7]In this context a port is implicitly associated to either the TCP or UDP protocol. For simplicity we will not refer every time to "port and protocol" but just to "port".

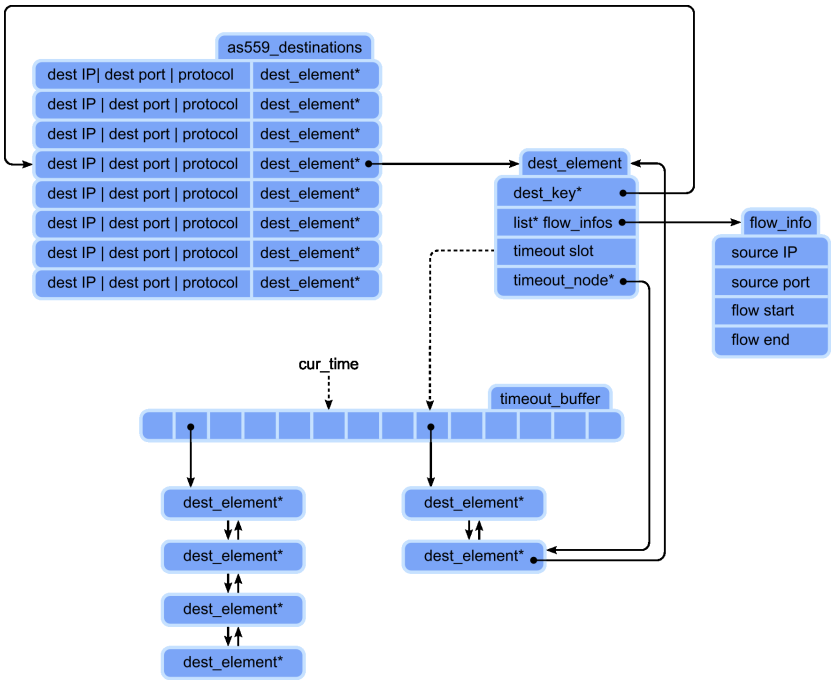

Figure 22: Data structures used by the `netflow_triples` tool.

key together with additional information about the source host and time stamps as element in a `dest_element` struct (see Figure 22).

For outbound flows the `processed_as559_triple_hosts` hash table is also checked in order to determine if a propagation triple has been found yet for the AS559 source host and the corresponding port. Again the flow gets discarded if this is the case. Otherwise the `as559_destinations` hash table is checked if the current outbound flow represents a response flow to a previously processed inbound request flow. Therefore the `flow_infos` list of the corresponding `dest_element` is searched for the appropriate endpoint. If the current flow is a response flow the processing will be aborted because it may not be part of a propagation triple (it does not initiate a new connection).

If there is no corresponding request flow the current flow it is assumed to initiate a new connection. If the AS559 source host has been registered to `as559_destinations` together with the current flow's destination port a new propagation triple has been found. The AS559 host and the destination port will be added to the `processed_as559_triple_hosts` hash table and the found propagation triple is written to `stdout`.

If a previously contacted AS559 host does not initiate an outbound connection with the same destination port within a certain timeout value (which is ten minutes by default), the corresponding entry in the `as559_destinations` hash table has to be removed. Therefore every hash table entry has to be added to the `timeout_buffer` on creation or if being updated. `timeout_buffer` is a ring buffer, implemented as an array, containing a doubly linked list at every

index (each index represents a second), which in turn contains pointers to the `dest_element` entries of the `as559_destinations` hash table that will timeout at the given time.

The `main()` loop keeps track of the internal time `cur_time` (in seconds since epoch) which is used as index pointer into the `timeout_buffer` ring buffer. Every time `cur_time` is increased by one second, all hash table entries of the corresponding `timeout_buffer` list will be removed.

Note that if an AS559 host that is already in `as559_destinations` is contacted by another external host on the same port, the timeout will be reset for the corresponding hash table entry, e.g. the entry will again be valid for the whole timeout period. Therefore each `dest_element` holds a pointer to its timeout node in the `timeout_buffer` such that the node can be removed instantly from the corresponding list.

3.5.2 Output

The output of the `netflow_triples` tool for a found propagation triple has the following form (line wrapped for readability):

```
1124477639  19.08.2005  18:53:59     6  445
    10.235.165.110  1039 --> 445  192.168.102.116  1579 --> 445  172.17.146.64
```

The first two numbers after the timing information (start time of the second flow) specify the protocol number and the destination port number of the propagation triple. The three IP addresses are the involved endpoints of the triple. If multiple hosts contacted the intermediate endpoint only the first one will be output. The two arrows denote the two flows of the propagation triple. The numbers to their left specify the source port of the flow and the numbers to their right the destination port of the corresponding flows.

3.5.3 Performance and Memory Usage

Table 4 shows the results of the performance and memory usage measurements of the `netflow_triples` tool. The tests have been conducted on archived data of the Sasser and Zotob outbreaks, analysing ten hours of data captured by three out of four SWITCH border routers. For both outbreaks the first analysis has been limited to the traffic pattern of the corresponding worm. In the second analysis the propagation triples of any traffic pattern were analysed. The timeout value (`-t` parameter) was set to ten minutes. The tests have been done on a computer with an AMD Athlon XP 2800+ CPU and 1GB of memory.

Analysing the propagation triples of all traffic patterns requires lots of resources, as shown in Table 4. Note that the memory usage differs strongly for the two analysed periods. The `netflow_triples` tool is not suitable for the simultaneous analysis of all types of propagation triples in real–time (analysing one hour of data needed more than one hour of computing time). But note that the tool has not been optimized yet. As the primary objective was determining the potential of the propagation triples based approach the tool's performance was sufficient in order to analyse archived NetFlow data.

	Zotob (445/TCP)	Zotob (all patterns)	Sasser (445/TCP)	Sasser (all patterns)
Total number of records	286'122'401	286'122'401	373'397'202	373'397'202
Total Duration	24m 40s	607m 0s	76m 58s	1040m 30s
Duration per 1h interval	2m 28s	60m 42s	7m 42s	104m 3s
Memory usage	75MB	360MB	240MB	670MB

Table 4: Performance and memory usage measurements of the `netflow_triples` tool.

3.6 Limitations and Encountered Problems

3.6.1 Propagation Triples Containing Reply Flows

The `netflow_triples` tool processes the flows in sequential order as they are being read from the NetFlow data files. Hence it is possible that a request flow (e.g. the flow initiating a connection) will be processed after the corresponding reply flow although it started earlier in time. This can, under certain circumstances, result in propagation triples not expected by the user.

Consider the sample scenario as shown in Figure 23, where host Y is an arbitrary DNS server replying to a legitimate DNS request from host Z and host X being a spam server sending Windows pop–up spam messages to random hosts. As shown in the figure host Z is using the non–privileged port 1026 as source port for its DNS request (flow **b**). As a result the DNS server Y will use this port as destination port for replying to the client's request (flow **c**).

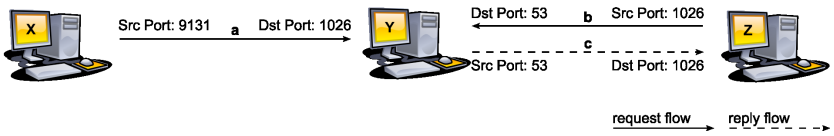

Figure 23: Depending on the processing order of the three flows, the flows **a** and **c** may be recognized as a propagation triple.

If the three flows happen to be processed in the order **a**, **c**, **b**, the flows **a** and **c** will be considered as propagation triple by the `netflow_triples` tool, because both flows have the same destination port, albeit flow **c** is merely a reply flow.

Figure 24 shows a sample scenario where two reply flows may be recognized as a propagation triple. Host Y requests a HTML page from the web server X. At the same time host Y is running, for instance, a P2P client accepting inbound

connections to port 4662. If it happens that a third host Z connects to the P2P client of host Y, using the same source port as host Y, the two flows **b** and **d** are considered as propagation triple with destination port 5300 if flow **d** is processed prior to flow **c**. Otherwise flow **d** would be recognized as reply flow to flow **c** and would not generate a propagation triple[8].

Figure 24: Depending on the processing order of the four flows, the two reply flows **b** and **d** may be recognized as a propagation triple.

Note that the propagation triples from these two examples do not violate the formal definition given in Section 3.2. But we wanted to make sure that a user of the `netflow_triples` tool is aware of the described behaviour.

In order to inhibit any propagation triples containing reply flows, the processing had to be restricted to request flows in `netflow_triples`. This would require to sort all flows by their start time in order to match the reply flows with the corresponding request flows.

3.6.2 `netflow_triples` Tool Out of Memory

We experienced problems with the `netflow_triples` tool while analysing the propagation triples during the Blaster outbreak. As the tool got out of memory the processing virtually stalled due to swapping. We solved the problem by restricting the analysis to only a few (overlapping) NetFlow files at a time and the subsequent manual removal of duplicate entries.

3.7 Future Work

Proposals for further improvements of the approach are:

- Distinguish between request and reply flows and do only process the former, preventing propagation triples that contain reply flows as described in Section 3.6.1.
- In this diploma thesis we restricted our analyses to propagation triples having a constant destination port. An idea would be to extend this approach

[8] As noted in Section 3.5.1 outbound flows are recognized as reply flows if the are processed after the corresponding inbound request flow. These flows will not be further processed.

to propagation triples with non–constant destination ports (although this approach would require even more resources). This would possibly allow to detect botnets and the spreading of the corresponding bots respectively (i.e. inbound 445/TCP flows and outbound 6667/TCP flows could indicate a new vulnerability exploited by bots which, in turn, connect to their botnet server).

3.8 Conclusion

The analyses of known worm outbreaks confirmed the assumption that an internal outbreak is accompanied by the presence of propagation triples matching the worm's propagation traffic pattern while only few propagation triples may be observed during periods without ongoing worm outbreak. Provided that a worm propagates into the observed network the internal outbreak will induce the presence of propagation triples.

We have seen in the analysed outbreaks that the detection latency is very variable between the individual outbreaks. While we observed a significant amount of propagation triples two hours after the internal outbreaks of the Blaster and the Zotob worm, it took 23 hours during the Sasser outbreak until a significant number of propagation triples has been found. The detection latency is not dependent from the number of infected hosts or from the scan activities, because the Sasser and the Zotob worms infected about the same number of internal hosts, resulting in similar scan activities during the outbreaks. Nevertheless the detection latency differed significantly between the two outbreaks. We do not know the reason of this observation.

It is important to note that with the presented propagation triple based approach a worm outbreak cannot be detected as long as the outbreak is limited to one side of the observed network, be it internal or external. It is essential that a considerable number of hosts is infected on both side of the network border. Otherwise the worm's connection attempts to the corresponding destination port will only be observable in one direction. But as long as there are no connection attempts by a previously contacted host into the opposite direction no propagation triple will be generated. For instance it would not have been possible to detect the outbreak of the Witty worm because there were no infections within AS559.

If the performance issues may be solved (see Section 3.5.3) we expect that the propagation triple based approach is suitable for detecting worm outbreaks *within* an observed network. But this is also a major drawback of this approach: As long as a worm has not propagated into the network it is by no means possible to detect an ongoing outbreak, making this approach unsuitable for networks with typically few internal infections during worm outbreaks (with respect to the number of worldwide infections) as it is the case for the SWITCH network.

4 ICMP Based Analysis

4.1 Introduction

With IP based communication the Internet Control Message Protocol (ICMP) [20] is typically used for reporting errors, for diagnostics or for routing purposes. A host that is infected by a self–propagating worm will trigger a lot of ICMP messages while trying to infect other hosts. This is due to the worm's propagation routine that scans randomly generated IP addresses in order to find new vulnerable victims. During this process the worm will also scan unassigned IP addresses. This will trigger ICMP messages (e.g. Destination Unreachable messages) generated by routers of the corresponding networks. Firewalls may also generate appropriate ICMP error messages for prohibited traffic. In the following we will investigate if worm outbreaks significantly increase the amount of ICMP traffic that would indicate an ongoing outbreak.

4.2 Analyses of Known Worm Outbreaks

Below we will present the ICMP based analyses of three known worms outbreaks, namely of the Blaster, Witty and Zotob worms. All analyses are based on archived NetFlow data collected in the context of the DDoSVaX project. The data is split into one hour intervals. Note that in our analyses we only took in– and outbound flows into account. Flows with both source and destination addresses belonging to the same address range (either within AS559 or outside AS559) were ignored. Flow duplicates of traffic routed through several border routers were not eliminated, but they are few and are expected not to influence the results significantly.

4.2.1 Blaster

The outbreak of the Blaster worm took place on Monday, August 11th, 2003. The AS559 internal outbreak happened in the early morning working hours of Tuesday, August 12th. A detailed analysis of the Blaster outbreak has been presented in [12].

Figure 25 shows the number of ICMP messages sent by hosts within AS559 to external hosts and the number of inbound connection attempts to port 135/TCP. The increase of inbound connection attempts to port 135/TCP due to the outbreak is accompanied by a significant increase of outbound ICMP messages. The scanning behaviour of Blaster infected hosts had a clear impact on the number of ICMP messages sent by AS559 hosts. Besides an increased number of outbound ICMP messages the number of unique AS559 source IP addresses seen in outbound ICMP messages showed also a significant increase during the outbreak.

The number of outbound connection attempts to port 135/TCP in Figure 26 shows the internal outbreak of the Blaster worm in the morning working hours of Tuesday, August 12th. As opposed to the global outbreak there has been no significant change in the number of inbound ICMP messages. The scanning

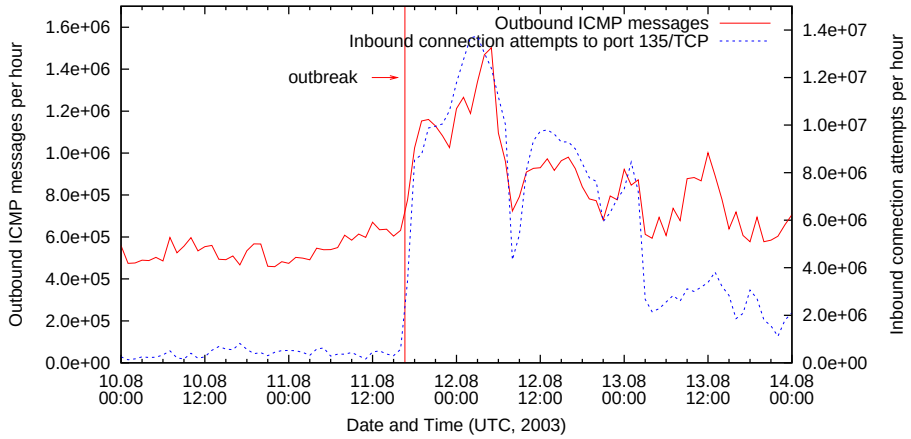

Figure 25: Number of outbound ICMP messages and number of inbound connection attempts to destination port 135/TCP during the Blaster outbreak.

activities of the relatively few infected hosts within AS559 were not able to trigger enough additional ICMP messages in order for being perceivable with respect to the number of inbound ICMP messages prior the outbreak.

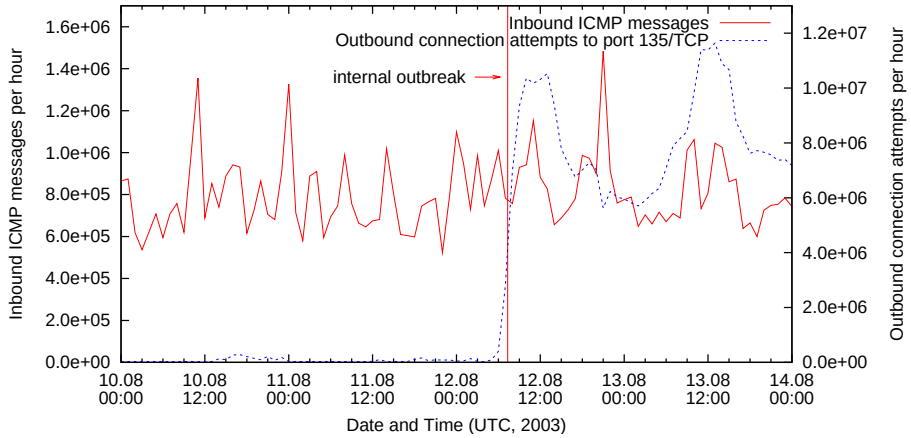

Figure 26: Number of inbound ICMP messages and number of outbound connection attempts to destination port 135/TCP during the Blaster outbreak.

4.2.2 Witty

The Witty worm began to spread on Saturday, March 20th, 2004, at about 4:45 UTC [27]. It targeted a buffer overflow vulnerability in several firewall and other security applications from Internet Security Systems (ISS). The flaw was located in the ICQ instant messaging protocol parsing routines of the various

products [21]. It could be exploited by sending a manipulated UDP datagram with source port 4000 and an arbitrary destination port, pretending to be an ICQ response datagram from an ICQ server.

Within approximately 45 minutes the worm had infected the majority of the vulnerable population (about 12'000 hosts). There were two main reasons for the dramatic spreading speed of the worm. Firstly, the worm began its spreading from a seed population of about 100 preinfected host. Secondly, the vulnerabilities in the ISS products were exploitable by UDP — there was neither a need for a 3-way handshake as with TCP nor for waiting on timeouts. The worm could just send the UDP datagram containing the payload and forget about it. Refer to [27] for an in–depth analysis of the Witty outbreak. Our analyses of outbound UDP traffic with source port 4000 have shown that the worm did not propagate into AS559.

Figure 27 shows the number of outbound ICMP messages sent by AS559 hosts and the number of inbound potential UDP worm datagrams having source port 4000. The large scale worm outbreak (with respect to the number of inbound datagrams) was again accompanied by a significant increase of outbound ICMP messages — after the outbreak almost 2.5 million sent ICMP messages were registered compared to 1.5 million messages before the outbreak.

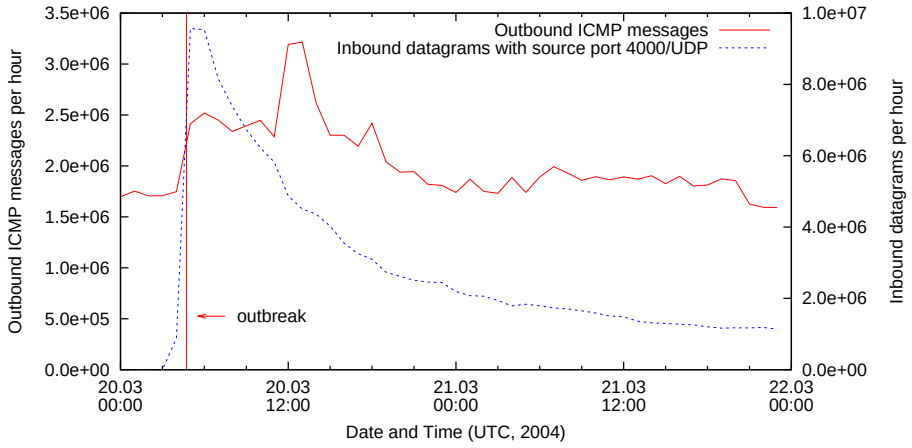

Figure 27: Number of outbound ICMP messages and number of inbound datagrams with source port 4000/UDP during the Witty outbreak.

However there was a second significant peak of outgoing ICMP messages from 12:00 to 14:00 UTC on Saturday, March 20th. The Witty worm infected the majority of the vulnerable hosts within 45 minutes. As shown in the plot the number of inbound Witty datagrams has been declining clearly for several hours when the second ICMP peak occurred. The number of unique source IP addresses of the outbound ICMP messages suggests that the second peak has not been due to random scanning, because the number of unique sources did not increase significantly during the peak as it did during the actual Witty outbreak. We expect that the second peak has not been induced by the scanning activities of Witty infected hosts, but we lack the knowledge for its exact reason.

4.2.3 Zotob

The Zotob outbreak took place on Saturday, August 13th, 2005. Its scale in terms of the number of inbound connection attempts was much smaller than the scale of the Blaster and Witty outbreaks. See Section 2.5.3 and A.2 for further information about the Zotob worm.

Figure 28 shows the number of outbound ICMP messages and the number of inbound connection attempts to port 445/TCP during the Zotob outbreak. Although there was a significant increase of inbound connection attempts to port 445/TCP, the plot shows that the outbreak of the Zotob worm did not affect the number of sent ICMP messages by AS559 hosts. The relatively low amount of inbound scanning of the Zotob worm compared to the Blaster and Witty outbreaks[9] could not trigger enough additional ICMP messages in order to significantly affect the existing amount of ICMP messages, e.g. due to normal traffic, P2P traffic, port scans or traffic generated by other worms.

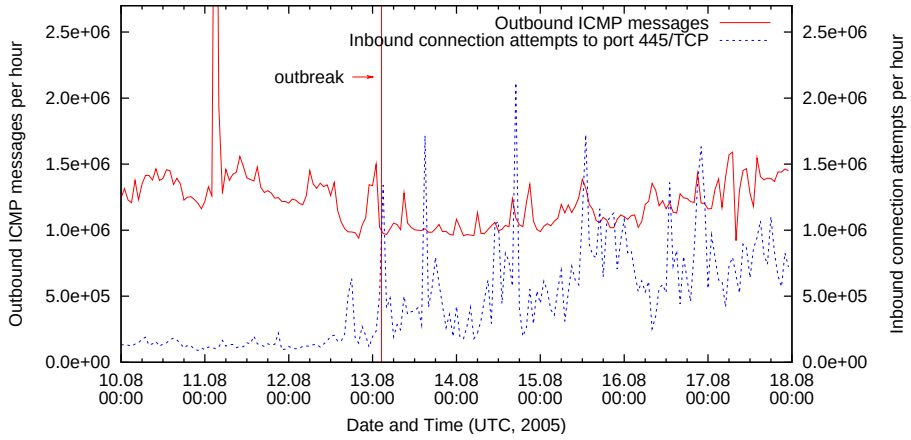

Figure 28: Number of outbound ICMP messages and number of inbound connection attempts to destination port 445/TCP during the Zotob outbreak.

We also have analysed the number of inbound ICMP messages during the internal Zotob outbreak. As during the internal Blaster outbreak the scanning activities of infected AS559 hosts did neither affect the number of inbound ICMP messages nor the number of unique source or destination IP addresses thereof.

4.3 Extended ICMP Based Analysis

According to the NetFlow specifications the type and code of an ICMP message record is provided in the destination port field of the flow record, encoded as `256*type + code`. That is the type is stored in the high–order byte and the corresponding code in the low–order byte of the destination port field. An

[9] During the Zotob outbreak there was about ten times less inbound worm traffic than during the Blaster and Witty outbreaks.

extended analysis of the ICMP messages by their types and codes would allow us to possibly gain additional information. We were especially interested in ICMP messages of type Destination Unreachable that would be triggered by the worms' random scanning procedures.

In the recorded NetFlow data from the Blaster and Witty outbreaks from 2003 and 2004 respectively, the type and code of virtually all ICMP records were set to zero (representing Echo Reply messages). On average only about 0.02% of the records were of another type, which were too few to make statistically sound conclusions.

The ICMP NetFlow data from the Zotob outbreak of 2005 provided more information about the types and codes of the different ICMP flows. About 25% of all flows had an ICMP type and code unequal than zero. We could not determine the reason for these differences between the various outbreaks. We found that during the Zotob outbreak the hardware engines as well as the software engines of the routers created ICMP flows with types and codes unequal than zero. Hence we do not expect the differences being (solely) due to changes in the software engines.

4.3.1 Missing Information

Although the NetFlow data of the Zotob outbreak contains considerably more messages with another type than zero, there were still missing information in the data. In other words there were still many ICMP flows with incorrect types and codes. In order to clarify this claim Figure 29 shows the number of ICMP messages of type Echo Request (type 8) sent from external hosts to hosts within AS559 and the number of ICMP messages of type Echo Reply (type 0) sent in the opposite direction. The outbound Echo Reply messages prevailed by far the inbound Echo Request messages, although the latter are responses to the former. Hence there were still many ICMP flows of other types than zero whose destination fields have not been set correctly.

4.3.2 Observed Peculiarities

Another oddity is the fact that many ICMP flows provide their type and code in the source port field of the flow record (with destination port set to zero) instead of the destination port field as stated in the specifications. Such flows have the additional peculiarity that type and code are stored the other way around as they should be, namely the type in the low–order byte and the corresponding code in the high–order byte. Our tool `netflow_icmp_stats` circumvents this by taking the source port into account if the destination port is equal to zero (if the destination port was set to zero correctly the source port will still be set to zero, yielding the same result).

4.3.3 Results

Being aware of these characteristics we still performed an analysis of the ICMP messages of type Destination Unreachable in order to determine if the Zotob

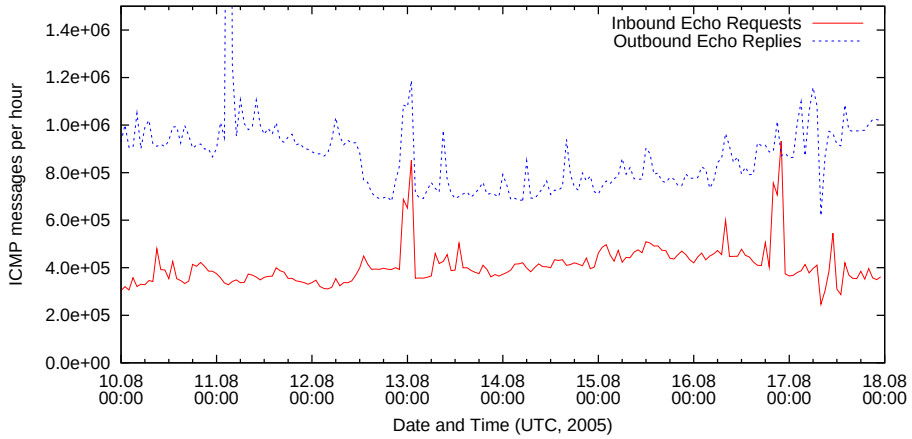

Figure 29: Number of inbound ICMP Echo Requests and outbound ICMP Echo Replies during the Zotob outbreak.

outbreak was perceivable therein. Having 25% of ICMP flows with type other than zero should provide sufficient data to work with. Because we do not know when an ICMP flow is recorded in the NetFlow data with a different type than zero, we analysed the data under the following assumptions in order for being able to draw any conclusions:

1. The ICMP type and code information is written into a flow record (either into the destination port or the source port field) with probability p.
2. With probability $1-p$ a type and code of zero is written into a flow record, regardless of the correct values.
3. The probability p is constant over time (although it does not have to be the same for all border gateway routers).
4. A router never writes the wrong type or code into a flow record.

Except for messages of type zero these assumptions turn the available ICMP data of every interval into random samples drawn uniformly with probability p from all recorded ICMP messages. According to the law of large numbers the individual interval sample distributions will converge to their actual distributions.

In other words: Plotting the number of sampled messages for the different types will result in curves that are — if the number of drawn samples is high enough — very similar to their original ones but only p in size. Hence we analysed the ICMP messages according to their types as if there were no missing information in the NetFlow data.

We analysed the following types of Destination Unreachable messages (ICMP type 3) that occurred during the Zotob outbreak and that could have been influenced by a worm outbreak:

- Net Unreachable
- Host Unreachable
- Port Unreachable
- Communication with Destination Network Administratively Prohibited
- Communication with Destination Host Administratively Prohibited
- Communication Administratively Prohibited
- Cumulated Destination Unreachable messages of all types

For each type of Destination Unreachable messages we analysed the number of messages, the number of unique source addresses and the number of unique destination addresses for inbound as well as for outbound messages. None of the analysed data sets showed a significant change during the Zotob outbreak, neither for the external nor for the internal outbreak. Hence the extended ICMP analysis could not reveal the outbreak of the Zotob worm and gave no further information than the plain ICMP based analysis.

4.4 `netflow_icmp_stats` Tool

To analyse the ICMP messages in the archived NetFlow data from the DDoSVaX project we have implemented a simple tool called `netflow_icmp_stats`. The tool reads the NetFlow data files as specified on the command line and outputs the number of ICMP messages per interval. The tool is written in C and has been developed under the Linux operating system. It amounts to just over 860 lines of code. Due to the simplicity of the code we will not go into further details about its implementation.

4.4.1 Output

Table 5 shows the attributes collected by the `netflow_icmp_stats` tool, together with the according field position within the tool's output. For each interval the tool outputs the statistics about all ICMP messages found in the input data. Additionally the tool outputs the number of occurrences for each ICMP type / code and the aggregated number of messages of type Destination Unreachable (aggregated over all codes).

The following line is a sample output from the `netflow_icmp_stats` tool for ICMP Destination Unreachable messages of type Communication Administratively Prohibited (ICMP type 3, ICMP code 13). For better readability the field numbers are included, too, and the output line is split over two lines.

```
1.          2.          3.        4.    5.    6.     7.     8. 9.
1124193600  16.08.2005  12:00:00  3_13  1207  32040  28590  0  61837

      10. 11. 12.  13.  14.   15.
      54  342 396  184  7310  7494
```

By default the interval length the output gets divided into is one hour. The user may set another duration by using the `-m` command line option.

Field	Description
1.	Seconds since epoch
2.	Date in human readable representation (UTC)
3.	Time in human readable representation (UTC)
4.	ICMP type and code (encoded as `type_code`)
5.	Number of inbound ICMP messages
6.	Number of outbound ICMP messages
7.	Number of ICMP messages with both external source and destination addresses
8.	Number of ICMP messages with both internal source and destination addresses
9.	Total number of ICMP messages
10.	Number of unique AS559 source addresses
11.	Number of unique external source addresses
12.	Total number of unique source addresses
13.	Number of unique AS559 destination addresses
14.	Number of unique external destination addresses
15.	Total number of unique destination addresses

Table 5: Attributes returned by the `netflow_icmp_stats` tool. The field numbers refer to the attribute's position within the output.

4.5 Future Work

A possible improvement of the presented approach would be to correlate ICMP flows to the TCP or UDP flow that triggered the ICMP messages. This would allow to analyse the ICMP messages in the context of the communication channel in which they occurred (e.g. analyse only the ICMP messages that were triggered by flows to destination port 445/TCP during the Zotob outbreak). A small but significant change in the ICMP traffic pattern of a given communication channel having few ICMP messages would then not be superimposed by other channels carrying lots of ICMP messages. It should be investigated if the additional information about the cause of the ICMP messages can improve the ICMP based approach such that it could be used for worm outbreak detection.

4.6 Conclusion

We have investigated the suitability of the ICMP based analysis for worm outbreak detection by analysing the three outbreaks of the Blaster, Witty and Zotob worms. While the large scale outbreaks of the Blaster and Witty worms had a significant impact on the amount of generated ICMP messages by AS559 hosts, the smaller Zotob outbreak did not trigger enough additional ICMP messages in order for being noticeable in the existing ICMP traffic. The number of unique AS559 source addresses that generated ICMP messages did not show the outbreak neither. The same conclusions hold for the extended analysis of the Destination Unreachable messages that occurred during the Zotob outbreak which were neither affected by the outbreak.

Neither the AS559 internal outbreak of the Blaster worm nor the internal outbreak of the Zotob worm resulted in a significant increase of inbound ICMP messages. For both outbreaks the scanning behaviour of the relatively small amount of infected hosts within AS559 was not strong enough in order to have an influence on the number of inbound ICMP messages or on the number of unique IP addresses.

The ICMP based approach is not suited for worm outbreak detection. Our analyses have shown that it is not able to identify smaller scaled outbreaks like the outbreak of the Zotob worm. The same holds for worm outbreaks within an observed network where relatively few internal hosts get infected as it is the case for the SWITCH network. These outbreaks do not generate enough traffic in order to have a significant impact on the number of ICMP messages.

5 Summary

The task of this thesis was to develop new approaches, based on flow–level traffic data, for detecting worm outbreaks in high–speed Internet backbones. In order to validate the proposed methods we had to design and implement appropriate tools for analysing archived NetFlow data from the DDoSVaX project. We have presented three generic methods for the detection of worm outbreaks. They do not require any previous knowledge about the exploits used by the worms or their scanning behaviour.

The first method is based on the characteristic port sequences used by a worm in order to infect a victim host. We have implemented a tool named `netflow_port_sequences` that returns statistics of all port sequences that occur within NetFlow data files. We have presented and implemented an algorithm for detecting significant network events by analysing all the sequences found by the port sequence analysis. The approach proved its suitability for worm outbreak detection by reliably identifying the outbreak events of large scale as well as of small scale worms within the analysed data.

The second presented method is based on the identification of propagation triples, i.e. two flows between three hosts that share the same communication pattern. We have implemented a tool named `netflow_triples` for finding propagation triples within NetFlow data files. During periods without ongoing attack only few propagation triples were present in the border traffic of an observed network. This changed during a worm outbreak were a significant amount of propagation triples could be observed. It is important to note that by using the flow–level data of border traffic only outbreaks within the observed network may be detected. In our analyses of known worm outbreaks the detection latency of this method ranged from two hours to 23 hours..

The third approach is based on the analysis of ICMP traffic. An infected host will trigger ICMP messages while scanning randomly generated IP addresses in order to find vulnerable victims. We have investigated if the number of ICMP messages changes significantly during worm outbreaks. Therefore we have implemented a tool named `netflow_icmp_stats` that generates statistics about the number of ICMP messages within NetFlow data files. We found that only large scale outbreaks like Blaster or Witty trigger enough additional ICMP messages in order for being noticeable within the existing ICMP traffic. Worm outbreaks of smaller magnitude, like the outbreak of the Zotob worm, do not significantly affect the number of ICMP messages, making it impossible to detect such outbreaks by using this method.

A Worm Outbreaks in 2005

In the following we provide a survey of self–propagating Internet worm outbreaks that occurred during 2005. The summarised worm descriptions are in chronological order. The survey has been conducted based on the information available from CERT [1], CERT India [2], CERT Malaysia [3] and Worm Blog [5]. Note that the outbreak dates may vary between the different information sources.

A.1 January – MySQL UDF Worm

On Wednesday, January 26th, 2005, a new worm has been reported that compromised MySQL database servers running on Microsoft Windows with weak or null passwords for the root account. The worm used the User Defined Function (UDF) capability of MySQL in order to install a variant of the Forbot / Spybot worm. Its propagation routine scanned for other vulnerable systems on port 3306/TCP.

According to [10], on January 27th there were over 8000 hosts connected to the IRC channel that controlled the infected systems. More systems could have been infected but may have been prevented from connecting to the IRC server. Symantec classified the distribution of the MySQL UDF worm as “medium”.

References: [30], [41], [9]

A.2 August – Zotob

On August 9th, 2005, Microsoft released six security patches as part of the scheduled release cycle. One of them, patch MS05-039 [26], fixed a vulnerability in the Plug and Play service of Microsoft Windows 2000. On Saturday, August 13th, 2005, a worm has been discovered exploiting the Plug and Play vulnerability and propagating through port 445/TCP. After a successful infection the Zotob worm, as it has been named, tried to connect to random IP addresses within the class B network of the infected system in order to infect other systems.

According to [39] there existed seven Zotob variants three days after its initial outbreak. Additionally, by then eight different bots like Rbot or SDbot had been updated for using the Plug and Play vulnerability as well.

There is no information available about the exact number of infected systems. Symantec classified the distribution of the various Zotob variants from “medium” to “high”.

References: [8], [17], [29], [44], [39]

A.3 November – Lupper

The Lupper worm (also called Plupii) was a Linux worm that exploited a vulnerability in XML–RPC for PHP — a common PHP extension module used by

a large number of web applications like (among others) PostNuke, WordPress, PHPGroupWare and TikiWiki. The worm scanned for vulnerable web servers in its class B network by sending malicious HTTP requests to port 80/TCP. On success the attacked server downloaded a copy of the worm and executed it within the vulnerable PHP environment.

Lupper has been discovered on Tuesday, November 8th, 2005. A variant has been found on Thursday, November 17th, 2005. Both versions had a "medium" distribution according to Symantec.

References: [18], [43]

A.4 December – Dasher

Dasher exploited a vulnerability in Microsoft's Distributed Transaction Coordinator (MSDTC) through port 1025/TCP. According to antivirus vendors like Symantec and F–Secure the outbreak took place on Thursday, December 15th, 2005 ([35], [14]). But our port sequence analysis suggests that the outbreak took place during the evening of Thursday, December 8th, 2005 (UTC), which is supported by Japan's Information Technology Security Center [19].

During the following days several Dasher variants were detected. In addition to the MSDTC exploit the attack vectors of the variants contained exploits for the older WINS (42/TCP), LSASS (445/TCP) and MSSQL (1433/TCP) vulnerabilities (the actual set of included exploits depended on the variant).

Symantec classified the distribution of the various Dasher variants from "medium" to "high".

References: [16], [28], [42], [40]

B User Guide

B.1 `netflow_port_sequences` Tool

The syntax for the `netflow_port_sequences` tool is:

```
netflow_port_sequences <options> <input files(s)>
```

Table 6 shows the available options that will now be discussed briefly.

By default the number of port sequence occurrences is output by intervals of one hour which may adapted by using the `-m` option[10].

Two successive flows between two hosts have to occur within a given timeout in order to be considered as belonging to the same port sequence. If the second flow occurs after the timeout it will be regarded as a separate port sequence. The `-t` option allows to set an arbitrary timeout value.

If the timeout for a given port sequence has exceeded the sequence still remains active for a certain amount of time. This allows flows with long durations (that started within the sequence's timeout but took too long in order to arrive within the timeout) to be accounted to the appropriate port sequence. By default a flow may have a maximum duration of 30 seconds (longer flows will generate a new sequence) which may be changed by using the `-f` option.

The `-i` option defines the minimum number of ports that a port sequence has to consist of in order to be output by `netflow_port_sequences`. By default the value is one. For defining the maximum number of ports use the `-a` option.

By default the tool does only output a sequence if it occurred at least 500 times. This default value may be changed by the `-s` option.

The tool's output is written to `stdout`.

Option	Description
`-m <seconds>`	Set interval length. (3600s, i.e. 1 hour)
`-t <seconds>`	Set timeout value. (5s)
`-f <seconds>`	Set maximum flow duration. (30s)
`-i <integer>`	Discard sequences containing less ports. (1)
`-a <integer>`	Discard sequences containing more ports. (10)
`-s <integer>`	Discard sequences that occurred less than the given value per interval. (500)
`-h`	Print help message.

Table 6: Options accepted by the `netflow_port_sequences` tool. The default values are given in parentheses.

[10] `m` stands for modulo.

B.2 `netflow_triples` Tool

The syntax for the `netflow_triples` tool is:

```
netflow_triples <options> <input files(s)>
```

Table 7 shows the available options that will now be discussed briefly.

By default the tool processes all TCP and UDP flows read from the NetFlow data files and outputs all found propagation triples with constant destination ports. In order to speed up the processing the user may specify a transport protocol and / or destination port as argument such that the tool will restrict the search for propagation triples matching the given traffic pattern.

To restrict the analysis to a single protocol the `-p` option may be used followed by either `TCP` or `UDP`. The `-d` option specifies a fixed destination port if desired.

As mentioned in Section 3.2 the second flow of a propagation triple has to occur within a given time after the first flow. By default the `netflow_triples` tool uses a timeout of ten minutes. The `-t` option allows the user to set a specific timeout value.

The tool's output is written to `stdout`.

Option	Description
`-t <seconds>`	Set timeout value. (600s, i.e. 10 minutes)
`-p <protocol>`	Process only flows matching the given protocol. `<protocol>` is either `TCP` or `UDP`. (Both)
`-d <port>`	Process only flows matching the given destination port. (All ports)
`-h`	Print help message.

Table 7: Options accepted by the `netflow_triples` tool. The default values are given in parentheses.

B.3 netflow_icmp_stats Tool

The syntax for the `netflow_icmp_stats` tool is:

```
netflow_icmp_stats <options> <input files(s)>
```

By default the statistics about the ICMP messages are output by intervals of one hour. This may adapted by using the `-m` option[11].

The tool's output is written to `stdout`.

Option	Description
`-m <seconds>`	Set interval length. (3600s, i.e. 1 hour)
`-h`	Print help message.

Table 8: Options accepted by the `netflow_icmp_stats` tool. The default values are given in parentheses.

[11] `m` stands for modulo.

C Task Description

The following is a copy of the task description of this diploma thesis.

The Problem

Internet attacks such as massive worm spreading events have increased in frequency and impact over the last few years. However, still most worm outbreaks occur unnoticed by network operators. By the time they get detected a large number of hosts may already be infected causing lots of damage.

The Setting

The context of the DDoSVaX project collects and keeps a long–term archive of flow–level Internet backbone traffic data.

The Task

By using current Internet flow–level traffic data and by replaying traffic data of earlier minor and major worm outbreaks from our NetFlow archive, the student will develop algorithms and implement them. The outbreak detection algorithms will be validated against known outbreaks in our recorded data.

One approach will be based on characteristic port sequences generated by worms while trying to exploit several vulnerabilities on the target hosts. Another approach will be based on ICMP messages triggered by propagating worms. A third approach will be the identification of propagation–indicating triples, e.g. host A contacts host B with a specific traffic pattern and later host B contacts host C with the same pattern. The presence of such triples may indicate the propagation of a worm.

This task is split into the following subtasks:

Understand the NetFlow data

Develop algorithms for worm outbreak detection

Algorithms will be developed aiming at a (near) real–time detection of worm outbreaks by successively analyzing NetFlow data. For potential detection of slow worms, near real–time can be in the range of hours or days.

Implement and test the algorithm

A design of the implementation of the detection algorithms will be developed. Then the programs will be written for an offline analysis of NetFlow data. In

the testing phase, the function of the programs will be verified by processing archived NetFlow data of known worm outbreaks of the past.

Deliverables

During the work on this thesis the following deliverables are expected:

- Description of the algorithms to be implemented: Before the algorithms are implemented, they have to be presented along with performance estimations and proposals of the software design.
- Offline analysis tool that allows testing and evaluation of the designed algorithms on stored flow data.
- Evaluation report on algorithm effectiveness and observed characteristics when tested on stored data.
- Documentation: A written report will conclude this thesis.

Dates

This diploma thesis starts on November 7th, 2005 and will be finished on March 6th, 2006.

Supervisors

Arno Wagner, wagner@tik.ee.ethz.ch, +41 44 632 70 04, ETZ G95

References

[1] *CERT Coordination Center*. http://www.cert.org.

[2] *Indian Computer Emergency Response Team*. http://www.cert-in.org.in.

[3] *Malaysian Computer Emergency Response Team*. http://www.mycert.org.

[4] *SWITCH*. http://www.switch.ch.

[5] *Wormblog*. http://www.wormblog.com.

[6] CERT: *Exploitation of Vulnerabilities in Microsoft SQL Server*. http://www.cert.org/incident_notes/IN-2002-04.html, May 2002.

[7] CERT Coordination Center: *Security Advisory: W32/Blaster worm (CA-2003-20)*. http://www.cert.org/advisories/CA-2003-20.html, August 2003.

[8] CERT Coordination Center: *August 15, 2005 — Current Activity*. http://www.cert.org/current/archive/2005/08/15/archive.html, August 2005.

[9] CERT Coordination Center: *January 28, 2005 — Current Activity*. http://www.cert.org/current/archive/2005/01/28/archive.html, January 2005.

[10] Computerworld: *MySQL installations targeted by Forbot worm variant*. http://www.computerworld.com/printthis/2005/0,4814,99282,00.html, January 2005.

[11] Dübendorfer, T. B. Plattner: *Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones*. 2005.

[12] Dübendorfer, T., A. Wagner, T. Hossmann B. Plattner: *Flow–Level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone*, 2005.

[13] Estan, C., G. Varghese M. Fisk: *Bitmap Algorithms for Counting Active Flows on High Speed Links. IMC'03*, 2001.

[14] F–Secure: *Dasher.A*. http://www.f-secure.com/v-descs/dasher_a.shtml, December 2005.

[15] FrSIRT: *Oracle 9i Database XDB HTTP Authentication Remote Stack Overflow Exploit*. http://www.frsirt.com/exploits/20051208.oracle9i_xdb_http.pm.php, December 2005.

[16] Indian Computer Emergency Response Team: *Dasher.C*. http://www.cert-in.org.in/virus/dasher-c.htm, December 2005.

[17] Indian Computer Emergency Response Team: *W32.Zotob.A*. http://www.cert-in.org.in/virus/worm-zotob-a.htm, August 2005.

[18] Indian Computer Emergency Response Team: *Worm Linux/Lupper*. http://www.cert-in.org.in/virus/worm-lupper.htm, November 2005.

[19] Information–Technology Security Center ISEC, Japan: *Observation Status by Internet Monitoring System*. http://www.ipa.go.jp/security/english/virus/press/200512/TALOT200512.html, January 2006.

[20] Internet Engineering Task Force: *RFC 792 — Internet Control Message Protocol*, September 1981.

[21] Internet Security Systems: *Vulnerability in ICQ Parsing in ISS Products*. http://xforce.iss.net/xforce/alerts/id/166, March 2004.

[22] Leinen, S.: *UDP vs. TCP Distribution*. http://www.postel.org/pipermail/end2end-interest/2001-March/000211.html, March 2001.

[23] Link Logger: *TCP Port 1433*. http://www.linklogger.com/TCP1433.htm, February 2004.

[24] Link Logger: *TCP Port 6129*. http://www.linklogger.com/TCP6129.htm, February 2004.

[25] LURHQ Threat Intelligence Group: *Windows Messenger Popup Spam on UDP Port 1026*. http://www.lurhq.com/popup_spam.html, June 2003.

[26] Microsoft: *Microsoft Security Bulletin MS05-039 — Vulnerability in Plug and Play Could Allow Remote Code Execution and Elevation of Privilege*. http://www.microsoft.com/technet/security/bulletin/ms05-039.mspx, August 2005.

[27] Moore, D. C. Shannon: *The Spread of the Witty Worm*. http://www.caida.org/analysis/security/witty/, 2004.

[28] MyCERT: *MyCERT Special Alert — W32.Dasher Worm*. http://www.mycert.org.my/advisory/MA-098.122005.html, December 2005.

[29] MyCERT: *MyCERT Special Alert — W32.Zotob Worm*. http://www.mycert.org.my/advisory/MA-094.082005.html, August 2005.

[30] MySQL AB: *MySQL AB — Security Alert*. http://dev.mysql.com/tech-resources/articles/security_alert.html, January 2005.

[31] Schlegel, C. DDoSVax Team: *UPFrame — An Extendible Framework for the Reception and Processing of UDP Data*. http://www.tik.ee.ethz.ch/~ddosvax/upframe/.

[32] Symantec: *W32.Mockbot.A.Worm*. http://securityresponse.symantec.com/avcenter/venc/data/w32.mockbot.a.worm.html, February 2004.

[33] Symantec: *W32.Sasser.Worm*. http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html, April 2004.

[34] Symantec: *Symantec Internet Security Threat Report — September 2005*. 2005.

[35] Symantec: *W32.Dasher.A*. http://securityresponse.symantec.com/avcenter/venc/data/w32.dasher.a.html, December 2005.

[36] Virus Information Center: *Win32/Agobot Family*. http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?ID=37776, July 2004.

[37] Wagner, A., T. Dübendorfer B. Plattner: *The DDoSVaX Project at ETH Zurich*. http://www.tik.ee.ethz.ch/~ddosvax/, 2005.

[38] Wagner, A. B. Plattner: *Entropy Based Worm and Anomaly Detection in Fast IP Networks*. 2005.

[39] White, D.: *MS05-039 and the Zotob summary*. http://singe.rucus.net/blog/archives/510-MS05-039-and-the-Zotob-summary.html, August 2005.

[40] Worm Blog: *Dasher.C Now In The Wild*. http://www.wormblog.com/2005/12/dasherc_now_in_.html, December 2005.

[41] Worm Blog: *MySQL Worm Activity*. http://www.wormblog.com/2005/02/mysql_worm_acti.html, February 2005.

[42] Worm Blog: *New Worm: Dasher*. http://www.wormblog.com/2005/12/new_worm_dasher.html, December 2005.

[43] Worm Blog: *New Worm: Lupper/Lupii*. http://www.wormblog.com/2005/11/new_worm_lupper.html, November 2005.

[44] Worm Blog: *New Worms: Zotob*. http://www.wormblog.com/2005/08/new_worms_zotob.html, August 2005.