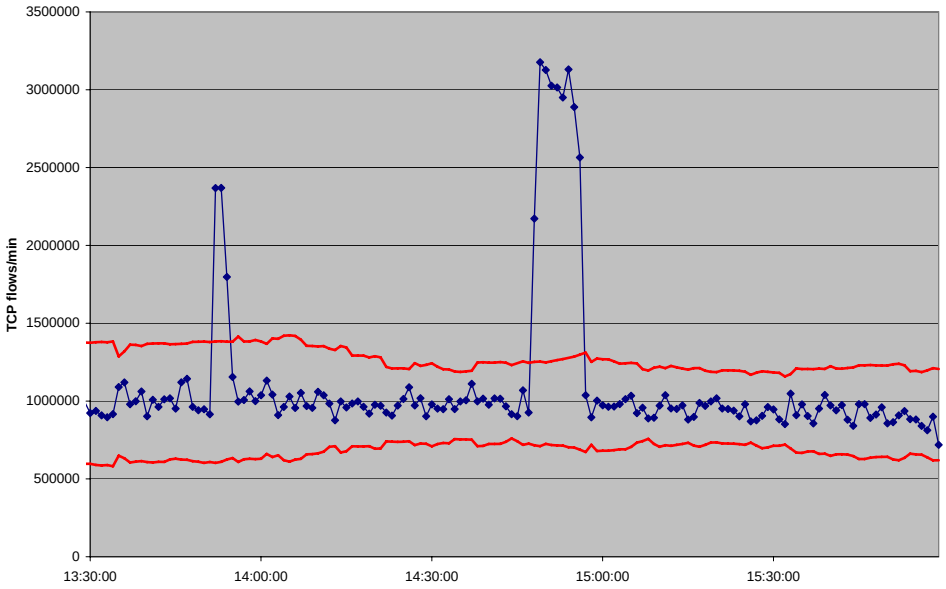
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Niklaus Frey

Localization of Known Anomaly Types in NetFlow Traces



Semester Thesis SA-2007-54
September 2007 to December 2007

Tutor: Daniela Brauckhoff
Co-Tutor: Arno Wagner
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

The goal of this semester thesis is to evaluate the possibilities of attack detection at the network *flow-level*. Using signatures that describe an anomaly is quite common at the *packet-level* but it can also be successfully applied to the *flow-level*, as shown in this work. To achieve this goal a tool-chain has been developed to first store collected NetFlow data in a database and then apply selected flow signatures to locate the network attacks.

Zusammenfassung

Ziel dieser Semesterarbeit ist es, die Möglichkeiten der Erkennung einer Netzwerk-Attacke auf dem ‘Flow-Level’ zu untersuchen. Signaturen, welche eine Anomalie beschreiben, werden häufig auf dem ‘Paket-Level’ verwendet. Dieselbe Technik kann auch auf dem ‘Flow-Level’ erfolgreich angewandt werden, wie hier gezeigt wird. Um die Attacken zu lokalisieren wurde eine Toolkette entwickelt welche die gesammelten NetFlow Daten zuerst in eine Datenbank einfügt und danach darin nach den ausgewählten Flow-Signaturen sucht.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives a short introduction to the semester thesis and states the motivation as well as the goal of the project.

1.1 Motivation

Attacks on the network infrastructure are one of the main threats for today's Internet. An especially harmful kind of attack are the so called denial of service attacks which can potentially cripple large parts of a network. A recent incident occured for example in April 2007 when large parts of the Estonian Internet were taken down by a distributed denial of service attack which most probably originated from Russia [1]. Network security specialists need means to reliably and quickly detect such attacks.
The main goal of the project was thus to gain a deeper understanding of the possibilities to detect attacks. All the conducted studies analyze the traffic at the flow-level instead of the packet-level which is still the common approach for intrusion detection systems. As operations at the packet-level get more and more expensive with larger data streams, it is very promising to check for anomalies at the flow-level since there the data is aggregated and therefore easier to handle. Such a flow based approach can possibly be used to monitor backbone networks more efficiently.

1.2 Problem Statement

The project is structured in four distinct parts. First I had to get familiar with the NetFlow v5 format of the collected data and the computing cluster used for this research. The next step was to conduct a theoretical assessment of different attack patterns with the main focus on (distributed) denial of service attacks. After having read several research papers on the topic, the next step was to design and implement a tool to build a MySQL trace database. The advantages of using a database are, that once the data is inserted into the database, it is possible to efficiently search the data for specific signatures in a specialized environment. This makes it much easier to evaluate different approaches of attack detection than to operate directly on raw NetFlow data.
The third part consisted of applying the theoretical knowledge gained from the preceding studies. Several Perl scripts to localize selected attacks had to be designed.
The last part of the task consisted of a detailed evaluation of the results from running the scripts over different data sets. Since these sets included well-known attacks, the achieved localization accuracy could be analyzed by comparing the output from the scripts with the actual attack.

1.3 Outline

The rest of the report is structured as follows: In Chapter 2, the results of the theoretical assessment of different attacks is presented along with the proposed methods of detecting them in the flow data.
Chapter 3 describes the chosen approach for designing the NetFlow trace database as well as the scripts which have been used to implement the attack localization on the imported data.
Chapter 4 specifies some details about the implementation of both the import tool for the database and the proposed Perl scripts.
Chapter 5 discusses the obtained results from the evaluation of the performance as well as the localization accuracy achieved by the proposed approach.
Chapter 6 includes a section about problems encountered during the development, the final conclusion of the project and an outlook on possible future work.

Chapter 2

Background

This chapter will first give an overview over some well-known attacks with the main focus on denial of service attacks. In the second section, selected attacks are discussed in more detail and detection methods at the flow-level are proposed.

2.1 Known Attacks

2.1.1 Overview

There are many different kinds of well-known attacks at the network level. The most prominent example is the denial of service (DoS) attack. These direct attacks aim at preventing the legitimate use of a service by either exploiting a network protocol weakness or simply by consuming the whole bandwidth of a victim.
Another more indirect kind of attack are port and host scans with which the attacker is trying to find a weakness in the victim's system. If he is successful in finding such a weak spot he can exploit a probably already known but not yet patched vulnerability and possibly take control of the whole system.
This is often done by attack groups in order to build up a botnet of multiple machines which can then be used to expand the botnet by looking for (and taking over) further victim computers.
One of the main purposes of these botnets is to launch a distributed denial of service attack (DDoS) which is much more powerful than an attack from a single host. Distributed attacks are more harmful because of the great number of systems and associated bandwidth which are under the control of one attacker.
Since these denial of service and distributed denial of service attacks are two of the main attack types on the Internet nowadays they are selected in this thesis for further examination and also for the following attack detection.

2.1.2 Special Case: DDoS Attacks

There exist various different tools to coordinate and launch a distributed denial of service attack. Some of them are discussed in *Analyzing Distributed Denial of Service Tools: The Shaft Case* [4] together with a description of how these tools become more and more advanced over time. The more sophisticated a tool is, the easier it is to control the whole botnet and the better the real attacker is hidden.
A rather detailed overview and taxonomy of different DDoS attacks and defense mechanisms can be found in *A Taxonomy of DDoS Attack and DDoS Defense Mechanisms* [7]. The various attack patterns are classified by their commonalities and important features, and thus provide a good understanding of the field.
In *A Framework for Classifying Denial of Service Attacks* [5], a system to distinguish between DoS attacks coming from a single source and those coming from multiple sources is proposed. This is achieved by examining the header content, ramp-up behaviour as well as a spectral analysis of the attack stream. The paper *Statistical Approaches to DDoS Attack Detection and Response* [6] takes a more statistical approach by studying the entropy of different header fields

to detect an ongoing DDoS attack. Those presented systems work like most current intrusion detection systems at the packet-level.
The goal of this thesis is to implement a detection method based on signatures at the flow-level which is a field that is not yet as thoroughly studied. There are different papers like *Router-based detection of DoS and DDoS attacks* [8] which vaguely describe an approach to detect DDoS attacks with NetFlow data but its effectiveness is not actually proven. Most of the work on flow-level detection defines an anomaly as a spike in one specific metric which is then used to detect an attack.
In the diploma thesis *Analysis and Detection of DDoS Attacks in the Internet Backbone using Netflow Logs* [9] a near real-time UPFrame plug-in has been developed to detect massive flooding attacks as well as TCP SYN flooding attacks. This is done by checking the ratio of incoming and outgoing packets of a host for exceedingly high values. It is demonstrated that this approach works well for the given attacks at the flow-level if the whole traffic between an observed host and its potential attacker can be observed. This was the case in the study which used the same source of data as this thesis, namely the captured NetFlow data from the SWITCH (Swiss Education Backbone Network) [3] border routers. An introduction to the UPFrame framework used here is given in *A Framework for Real-Time Worm Attack Detection and Backbone Monitoring* [10].
Compared to the previous work, the intention of this semester thesis is to try a more general approach of the problem which does not need to run in real-time. It should instead be more flexible and, thanks to using a database approach, also easily extendable to new ideas for suitable signatures to detect different attacks. In the following subsection the studied attacks and their proposed detection methods are discussed.

2.2 Selected DDoS attacks and Proposed Detection Methods

2.2.1 ICMP Flooding

With an ICMP flooding attack the attacker tries to consume the whole available bandwidth of the victim by sending a huge amount of echo requests. This attack is often used in a distributed fashion with multiple machines sending echo requests.
Another interesting attack based on the idea of ICMP flooding is a reflector attack called ‘smurf’. It uses the possibility of sending broadcast ICMP messages to a whole subnet so that all (or at least most) of the machines in that subnet then send a reply to the victim whose IP address has been used as a spoofed source address in the sent echo request packet. This attack is a very efficient type of reflector attack since the amplification factor is usually quite high. The reason why this attack is not very widespread anymore today is that, when properly configured, a router will not forward any ICMP broadcast message from the outside to the local net. This way that the attack is already blocked at the edge of the network. Since it is important that every major router in the Internet is set up this way, there are registries like the Smurf Amplifier Registry (SAR) [11] where all the vulnerable networks are listed. This forces the administrators of the affected networks to fix their router settings because they otherwise will be suffering from attacks launched at their infrastructures.
To detect ICMP flooding at the flow-level it suffices to set a threshold for the maximum amount of ICMP traffic to or from one host. This threshold can be determined by analyzing the ICMP traffic under normal conditions. Unusually high traffic indicates an anomaly which could be caused by an ongoing attack in the network. Another check is to look for ICMP flows which are a lot bigger than about 2kB since regular single ICMP messages usually are around 60-80 Bytes and ICMP flows usually contain only a few messages (about 1-20). False positives can possibly occur whenever a legitimate application is extensively using the ICMP protocol. This generates big ICMP flows which are incorrectly detected as flooding attacks.

2.2.2 TCP SYN Flooding

To establish a TCP connection a three-way handshake protocol is used. The host requesting a connection is first sending a TCP packet with the SYN flag set. The responding host is then sending back a packet with the SYN+ACK flag set to acknowledge the request. To complete

the handshake the requesting side is sending back a final ACK packet. After this packet the connection is established and can be used to exchange data.
The TCP SYN flooding attack is exploiting this part of the TCP protocol by sending many connection request packets to one host. This prompts the victim to send a lot of SYN+ACK packets and keep the information for all the half-open connections. The victim is then waiting for the final ACK of the three-way handshake which will never be sent by the attacker. After a while a timeout occurs for these half-open connections but if there are enough SYN packets sent in a short timeframe a lot of resources are bound. This can cause a denial of service since the victim cannot accept any legitimate connection anymore.
TCP SYN attacks are very effective and still in use since their first publication in 1996. The main problem is that they misuse a central part of the TCP protocol. One countermeasure is for example the use of so called *SYN cookies* which ease the problem of large queues by coding the connection information into the sequence number of the SYN+ACK packet. This sequence number will be increased by one at the requesting side and is sent back again in the following ACK packet. Like this the victim does not have to store all the connection information and the denial of service can be mitigated. Another solution to the problem is for every major Internet provider to check his outgoing and incoming traffic for packets with spoofed IP addresses. This has amongst others been proposed in the Computer Emergency Response Team (CERT) advisory [12] on the TCP SYN flooding problem.
To detect TCP SYN flooding at the flow-level several flow signatures can be used, including:

- The number of TCP flows per minute
- The average number of packets in each TCP flow per minute
- The average number of bytes in each TCP flow per minute
- The number of unique IP addresses seen per minute.

Because the flows of the TCP SYN flooding attack usually consist of only one packet per flow, the average number of packets and bytes per flow dips down. At the same time the absolute number of TCP flows per minute increases as the attack generates a lot of distinct flows. The source addresses of these attack packets are often spoofed and set randomly. This leads to a noticeable increase in unique IP addresses.
All these anomalies can easily be detected if an adequate threshold based on the 'normal' traffic is set. False positives can occur whenever there is a massive increase in demand for a ressource hosted within the observed network. This is also known as the 'Slashdot effect' [13] which can be caused by a very popular website linking to a small site. This results in a huge influx of web traffic for this smaller site which could be detected as a flooding attack.

2.2.3 UDP Flooding

In a UDP flooding attack a very large number of UDP packets is sent to one or several random ports of a victim host. These packets will eventually use up all of the available bandwidth and thus lead to a denial of service. UDP flooding is a relatively simple brute-force attack which can also easily be started from several different machines resulting in a distributed DoS attack.
Countermeasures against UDP flooding are mainly well configured firewalls throughout the network which filter unwanted UDP traffic before any damage is done.
To detect UDP flooding the number of UDP flows per minute is analyzed which will reveal massive flooding. For smaller flooding attacks, which are not visibile in the whole traffic stream, an analysis on a per host basis can be done. If there is a host, which receives an unusually high number of UDP flows on many different ports, it is a clear sign for a possible attack. The number of unique IP addresses seen per minute can be used to detect attacks which are heavily distributed or use random source addresses.

2.2.4 DNS Reflector Attacks

To conduct a DNS reflector attack the attacker sends a flood of DNS requests with a spoofed IP address (the one of the victim) to one or more DNS servers which results in a flood of DNS responses sent to the victim. If enough traffic is generated this can lead to a denial of service.

DNS reflector attacks are quite efficient reflector attacks since the DNS requests are smaller than the responses of the DNS server which leads to the desired amplification effect.
The detection of DNS reflector attacks is either done by checking for a very high rate of DNS request flows from the same (spoofed) IP address to a DNS server inside the network or by filtering hosts which receive an unusually high number of UDP flows with source port 53, which corresponds to the port from which a DNS server is sending his responses. False positives can occur if a user is sending a lot of legitimate DNS requests in a short timeframe. This can be caused by an application checking a long list of domain names for certain peculiarities.

2.2.5 Summary

Table 2.1 lists the studied attacks along with the proposed flow signature(s) as a short summary.

Attack	Flow Signature(s)
ICMP Flooding	ICMP flows per minute ICMP flows > 2kB
TCP SYN Flooding	TCP flows per minute Average number of packets per TCP flow per minute Average number of bytes per TCP flow per minute Number of unique IP addresses seen per minute
UDP Flooding	UDP flows per minute Number of unique IP addresses seen per minute
DNS Reflector Attacks	DNS requests per IP per minute

Table 2.1: Attacks with proposed flow signatures

2.3 Related Work

In the following, a short overview of interesting related work to the topics of this thesis is given.
The paper *Flow-level Anomaly Detection: Blessing or Curse?* [14] discusses ways to evaluate the general advantages and disadvantages of using NetFlow data for anomaly detection. In *Comparison of Anomaly Signal Quality in Common Detection Metrics* [15] a simple and intuitive measure for comparing anomaly detection metrics regarding their capability to expose certain types of anomalies is presented. The proposed detection is based on signal detection theory.
The effect of packet sampling methods on anomaly detection metrics is studied in *Impact of Packet Sampling on Anomaly Detection Metrics* [16]. The research has shown that entropy based summarizations of packet and flow counts are more resilient to sampling than volume metrics.
Another interesting application of flow-based anomaly detection is presented in *Flow-Based Identification of P2P Heavy-Hitters* [17]. An algorithm has been proposed which allows the detection of active P2P clients from NetFlow data in near real-time. This allows a network operator to effectively monitor and limit the available bandwidth for all P2P filesharing tools.
In *Flow-Level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone* [18] two major worm outbreaks are studied and analyzed on the flow-level.

Chapter 3

Approach

3.1 Overview

This chapter describes the chosen two-step approach to localize known anomalies using NetFlow traces. In a first phase the raw NetFlow data is stored in a MySQL database. Details about this trace database are discussed in the next section. After having set up the database, the second phase starts, which consists of a script-based attack localization implemented in Perl. This is described in the third section of this chapter.

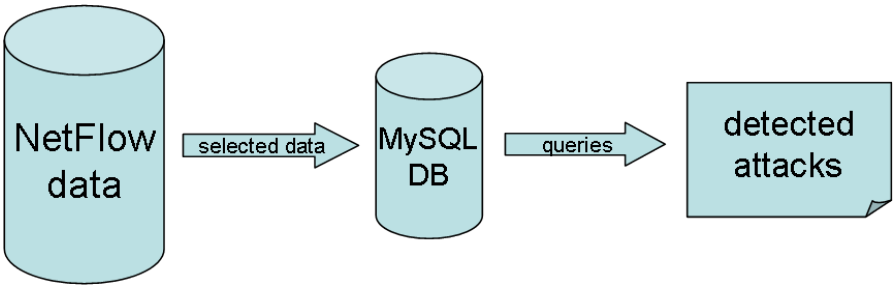


Figure 3.1: Schematic of the approach

3.2 NetFlow Trace Database

3.2.1 Motivation

It is rather difficult to handle the NetFlow data in its raw binary format because any transaction has to operate on the original little structured binary stream. Storing the traces in a database however allows for easy access as well as fast and efficient searching. The advantage in the search efficiency is mainly due to the indexing functionality provided by the database by which the data is stored in a sorted tree structure which allows a very fast access to a specific entry. The database also provides an easy interface in the form of the MySQL command-line tool or via phpMyAdmin (a graphical user interface).
Using a database of course has some disadvantages too. These are mainly the memory overhead which is due to the whole database structure needed around the actual data as well as the quite time consuming insert and index operations. It therefore doesn't make sense to save all the collected data in a database but rather to import interesting bits of the recorded data on which a more detailed analysis is to be performed.

3.2.2 Database Structure

This section describes the general structure of the proposed MySQL database. The study has been carried out on one database with multiple tables for the different evaluated attacks. For all tables the default MyISAM storage engine has been used which provides the needed functionality. Table 3.1 shows (i) all the fields of such a table along with (ii) the used data type, (iii) indication whether the field is indexed and (iv) a short description of the stored value.

Field	Data Type	Index	Description
fid	int		Primary key (only used for the DB)
addr	int	x	Source IP address
dstaddr	int	x	Destination IP address
dPkts	int		Number of packets in flow
dOctets	int	x	Number of Layer 3 bytes in flow
start	bigint	x	Start time of flow in milliseconds since the Unix epoch
end	bigint		End time of flow in milliseconds since the Unix epoch
port	smallint	x	TCP/UDP source port number or equivalent
dstport	smallint	x	TCP/UDP destination port number or equivalent
prot	tinyint		IP protocol number
router	smallint		Router from which the NetFlow data was received

Table 3.1: Structure of NetFlow trace tables

All of the data stored in the database is extracted directly from the NetFlow records, except for the start time, end time and the router value. Start and end time are computed by combining time information from the NetFlow header and record data. The values stored in the router field aren't extracted from the NetFlow data but taken from the name of the exported file. Further details about the import process will be explained in Subsection 3.2.3. The used data types have all been selected with the goal of providing enough space for the respective values and at the same time minimizing the total size of the database. Therefore the IP addresses are stored as integers instead of character arrays which poses no problem because functions to convert the integer values back to the standard notation and vice versa are provided by MySQL.
To identify each flow unambiguously, a unique auto-increment primary key is used. The introduced indexes permit a fast search for specific values and are thus created for all values which are interesting for attack detection queries, namely

- the IP address pair for finding a potential attacker/victim or a specific subnet with detected anomalies
- the number of bytes in a flow for distinguishing attacks with characteristic packet sizes
- the start time to quickly find flows starting in a given timeframe
- the TCP/UDP ports for filtering attacks on specific ports.

For performance reasons, a table contains about 3-4 hours of traces. This keeps the table size small enough so that indexing and searching run in an acceptable timeframe. Still it keeps the number of tables low as one table is sufficient for storing an attack scenario since the evaluated attacks don't spread over more than an hour and last only a few minutes each.

3.2.3 Import Tool

The tool for importing the NetFlow traces into the MySQL database has been developed and is written in C. It uses the C API provided by the *mysqlclient* library included in the standard MySQL distribution. The tool is implemented similarly to the command-line tool *mysql* which is available as open-source code. All of the needed parameters can either be specified as constants in the source code or set by calling arguments of the tool. These arguments include the hostname, port and socket of the MySQL server, the username and password for the database, the name of the table to import to and the location of the NetFlow datafile. To insert all the data corresponding to one attack which consists of several files, each containing one hour worth of data, a shell script was used.

For the extraction of the raw NetFlow data, some of the tools from the DDoSVax project [19] are used. These tools include a function to extract single records and store them in a NetFlow specific struct which can then again be read out to insert the values in the corresponding fields of the table. This is done for all fields except for 'start', 'end' and 'router'. As the start and end time of a NetFlow record is only stored relative to the corresponding NetFlow header packet, an additional function is used here to calculate the absolute time which corresponds to the UNIX timestamp (i.e. the number of milliseconds elapsed since midnight UTC of January 1, 1970). These time values are then stored in the respective fields of the table. The information for the 'router' field is extracted from the filename of the NetFlow trace as this information is not enclosed in the data itself and is only needed to distinguish between data coming from Router A and from Router B.

3.3 Script-based Attack Localization

3.3.1 Overview

After having inserted all the desired NetFlow data into the database it is now time to localize selected attacks. This is done by running database queries driven by Perl scripts. To access the database and execute these queries, the DBI (Database independent interface) Perl module [20] is used. All the necessary parameters for the module are set directly at the head of the script.
The returned results from the queries can either be written as text to an output file or it is possible to insert them into a new temporary table for further processing.

3.3.2 Attack Detection Approach

To detect flooding attacks, the first approach was to simply count the number of flows per minute for one IP protocol (TCP, UDP or ICMP) and check for anomalies in the obtained statistics. The number of flows per minute can be evaluated in a single query on the database. The only problem encountered with this approach was that the tables storing the flow data of one attack were too big (containing up to 350 million entries) to be searched efficiently. This is mainly due to the size of the temporary tables created by the database during the search which cannot be held in the main memory of the system but instead have to be written to disk which is much slower. To resolve this problem the data is handled hour after hour. Like this the temporary tables are smaller and can be kept in the main memory which results in a much better performance.
As this relatively simple approach already delivered promising results for one of the studied TCP flooding attacks, the next step was to extend this kind of search by calculating several other values on a per minute basis. The finally used query calculates the following values for a dataset:

- number of flows per minute
- number of packets per minute
- number of bytes per minute
- average number of packets per flow per minute
- average number of bytes per flow per minute
- number of unique IP addresses seen per minute

The results are stored in a new table of the database which is used to locate the attack.
The following detection of TCP SYN, UDP and ICMP flooding is accomplished by identifying fast rises and unusally high values in the 'flows-per-minute-statistics' of the respective protocol. To locate these peaks, a threshold is applied, which is calculated as the mean value over the last 20 minutes plus/minus five times its standard deviation. All the minutes for which this threshold is exceeded are candidates for a flooding attack.
For TCP SYN flooding attacks the average number of packets or bytes per flow is a good indicator of a possible attack as well: As soon as the attack starts, the number of packets per flow

decreases, since all the flows which are part of the attack contain only one or two packets, unlike regular TCP flows which usually contain several packets. This behaviour cannot be observed for UDP or ICMP flooding as the flows of these protocols usually include only few packets. Thus, there isn't a big discrepancy between attack traffic and normal traffic.
A third indicator used to distinguish between single-source and distributed versions of all three attack types is the number of unique IP addresses seen in one minute. It could be observed that during a massive distributed flooding attack the number of different addresses rises noticeably although not as strikingly as for example the number of flows per minute.
The next step after the localization of a possible attack is then to take a closer look at the incident. Thanks to the versatile database it is possible to study all the details of the attack in a straightforward manner. Interesting aspects are for example which addresses or subnets were involved in the attack, whether the attacker was making use of a botnet, or if it is probable that all the traffic was coming from one source. Other aspects include the magnitude of the attack or observable known patterns in the corresponding flows. All of this further analysis is currently done manually but could also be automated in a more advanced version of the Perl script.

Chapter 4

Implementation

4.1 Database Import Tool

4.1.1 Overview

The database import tool is fully implemented in C and uses the API provided by the MySQL database.
The source code is split in two files: the main application called *mysql_app.c* and *mysql_tools.c* which provides methods to connect and disconnect to a MySQL database.
To store the data in the database, a NetFlow record is first read into a predefined *struct* data-structure with the NetFlow tools provided in the DDoSVax framework. In a next step it is written to the selected table with an `INSERT INTO` query which contains the data from the previously generated struct. This procedure is repeated until all records of the selected file are imported.

4.1.2 Methods

In the following, the main functions which have been written for the tool are shown and briefly explained:

```
MYSQL * do_connect(char *host_name, char *user_name, char *password,
    char *db_name, unsigned int port_num, char *socket_name,
    unsigned int flags)
```

Connect to the MySQL database with the specified parameters.

```
void do_disconnect(MYSQL *conn)
```

Close the connection to the database.

```
void process_query(MYSQL *conn, char *query)
```

Run a predefined `query` *on the open connection* `conn`*. If the query returns values from the database they are printed on the commmand line.*

4.1.3 Syntax

The syntax to call the import tool is as follows:

```
mysql_app -h <hostname> -u <username> -p -P <port number> -S <socket>
          -f <filename> -t <table> -i
```

The *-p* switch is used to set the password via a prompt on the command line. The *-i* switch is used to tell the tool that all the indexes should be created immediately after the import of the data. This switch is only set with the last batch of data imported into a table as the indexing should preferably be done after inserting all the data.

4.2 Perl Scripts

The Perl scripts make use of the DBI (Database independent interface) module which provides all the necessary methods to connect to the database and execute queries. In the following subsections two of the developed scripts are briefly discussed.

4.2.1 per_min_stats.pl

The *per_min_stats.pl* script calculates all the 'per-minute-statistics' described in Section 2.2.2. Queries on the tables deliver better performance when issued on at most one hour of data, as explained in Section 3.3.2. The script therefore first searches for the earliest start time of all flows with the query `SELECT MIN(start) FROM $TBLNAME`. In a next step the end of the first hour is calculated so that no minute is split between the first and the second hour.
The main query then looks as follows:

```
INSERT INTO per_min_stats_tmp SELECT count(fid) AS flows_per_min,
  sum(dPkts) AS packets_per_min, sum(dOctets) AS bytes_per_min,
  avg(dPkts) AS packets_per_flow, avg(dOctets) AS bytes_per_flow,
  count(DISTINCT addr) AS unique_ip_per_min, truncate(start/60000,0)
  FROM $TBLNAME WHERE prot=$PROT AND start BETWEEN $tmp_start_time
  AND $tmp_end_time GROUP BY truncate(start/60000,0)
```

All the flows with start time values between `$tmp_start_time` and `$tmp_end_time` (which are calculated as described before) are first sorted into groups which contain one minute by the `GROUP BY` command. In a next step all the values like the flows per minute, packets per minute, etc. are calculated with the count(), sum() and avg() functions of the database. The results are stored into a temporary table (`per_min_stats_tmp`) for later use.

4.2.2 num_flows_per_ip.pl

The *num_flows_per_ip.pl* script calculates the number of flows per IP address to a victim host during one minute with the following query:

```
INSERT INTO num_flows_per_ip_tmp SELECT count(*) AS num_flows_per_ip,
  addr, inet_ntoa(addr) AS addr_hr FROM $TBLNAME WHERE prot=$PROT
  AND start BETWEEN $start_of_min AND $end_of_min AND
  dstaddr=inet_aton($DSTADDR) GROUP BY addr ORDER BY 1 DESC
```

Here the data is first grouped by the different source addresses (`GROUP BY addr`) and then the number of flows in each of these groups is calculated with the count()-function. The corresponding addresses are stored as integer (`addr`) as well as in the standard IP address notation (`inet_ntoa(addr)`). All results are saved in a temporary table (`num_flows_per_ip_tmp`) for further processing.

4.2.3 num_flows_per_port.pl

The *num_flows_per_port.pl* script works similarly to the *num_flows_per_ip.pl* script. It computes the number of flows per destination port of the victim during one minute.
The corresponding query:

```
INSERT INTO num_flows_per_port_tmp SELECT count(*) AS
  num_flows_per_port, port FROM $TBLNAME WHERE prot=$PROT AND start
  BETWEEN $start_of_min AND $end_of_min AND dstaddr=inet_aton($DSTADDR)
  GROUP BY dstport ORDER BY 1 DESC
```

The records are first grouped by the destination port (`GROUP BY dstport`) and then the number of flows in each of these groups is calculated again with the count()-function. The number of flows per port and the corresponding ports are saved in a temporary table (`num_flows_per_port_tmp`) for further processing.

Chapter 5

Results

This chapter discusses the achieved results regarding database performance and localization accuracy of the proposed approach to detect anomalies in the network traffic.

5.1 Used Datasets

The following evaluation is based on NetFlow v5 data which has been collected on the SWITCH [3] border routers. This data comprises all aggregated packet headers (flows) which are sent between an ETH-internal host and hosts which are not part of the SWITCH backbone network. ETH-internal traffic as well as traffic between SWITCH-internal sites is not observed.
Five different datasets, each comprising at least one suspected attack have been selected to evaluate the detection capabilities of the developed attack localization scripts. The records consist of 3-4 hours of data and are stored stored in different tables of the database.
During the analysis, the two most interesting datasets were selected for further examination and are discussed in Section 5.3. For privacy reasons no actual IP addresses or hostnames are revealed and the selected attack scenarios are simply called Scenario I and Scenario II.
Scenario I represents a typical TCP SYN flooding attack and in Scenario II there is a combined TCP SYN and UDP flooding attack.

5.2 Database Performance

5.2.1 Import Performance

During the development of the import tool it was noted to be much faster to first insert the whole set of data into the database and create the required indexes thereafter. This is due to the tree structure which can be generated much faster after all the entries are added to the table rather than creating the tree at the beginning and updating it with each new entry. A further speedup of up to 25% was achieved by locking the table during the whole insert procedure which allows the database to handle the data more efficiently due to its exclusive access. Another effort to accelerate the import procedure was to decompress the (by default zipped) NetFlow data before insertion but this didn't result in a noticeably better performance and was therefore discarded.
The machine used for the following database operations was featuring two Opteron 275 (2.2GHz, Dual-Core) CPUs, 8GB of RAM and the raw data was read over a gigabit ethernet connection from a HW RAID 6 array and stored again on a local HW RAID 6 array. The hereby achieved performance for insertion of new entries is around 800'000 entries per minute which corresponds approximately to the rate of flows being recorded on one router during normal traffic conditions. As can be seen in Figure 5.1, the indexing performance is a bit lower (between 600'000 and 700'000 entries per minute). The main reason for the varying performance of both operations is very likely the load from other processes on the machine.
The total time needed for importing one hour of data varied betweeen about 2 1/2 (˜55M records) and 5 1/2 (˜116M records) hours depending on the number of flows as well as the load on the machine. This seems to be an acceptable amount of time as it allows to add 3-4 hours of traces for detailed analysis in at most one day.

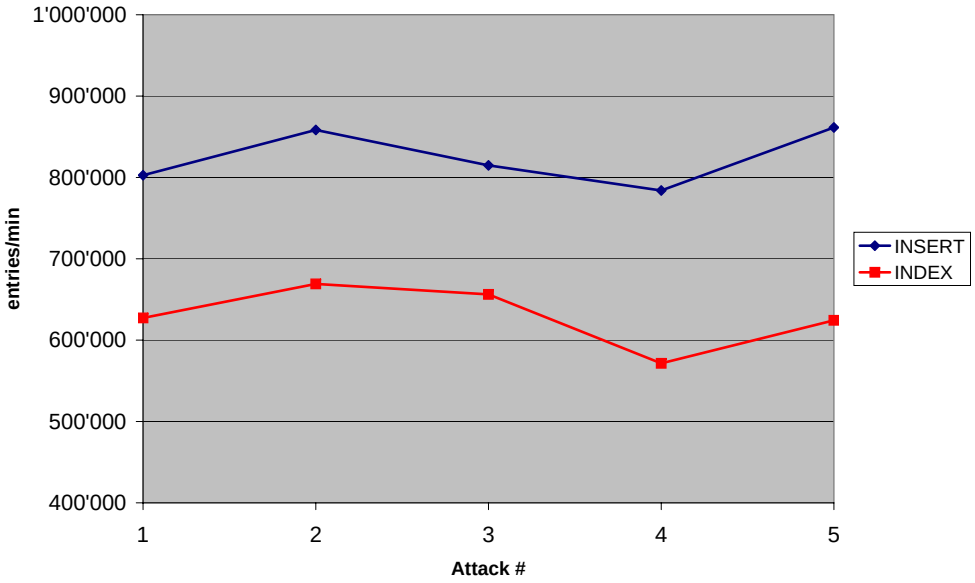


Figure 5.1: MySQL insert tool performance

The size of the resulting database tables compared to the size of the raw compressed data in bzip2 format is shown in table 5.1. There is a substantial memory overhead in the database structure mainly caused by the different indexes and the missing compression.

Attack #	Raw size (bzip2)	Table size
1	3'652 MB	33'591 MB
2	5'209 MB	38'263 MB
3	2'255 MB	17'869 MB
4	2'518 MB	21'085 MB
5	4'036 MB	32'306 MB

Table 5.1: NetFlow raw data size vs MySQL table size

5.2.2 Lookup Speed

Thanks to the well indexed database and handling only one hour of data at a time, all the required queries could be completed in a short amount of time. Only about 30 minutes are needed to get all the per minute statistics discussed in Section 3.3.2 on even the largest table (which has been Scenario II) which contains over 340 million entries. For the less extensive table of Scenario I (containing about 305 million records) the process takes less than 20 minutes. Smaller queries, like for example the number of flows per distinct IP addresses during one minute of attack, are handled in less than two minutes and subsequent queries on different minutes of the same attack take only one minute due to caching. This speed combined with the versatility of the database proves the suitability of this approach for detailed traffic analysis.

5.3 Localization Performance

5.3.1 Scenario I

In Scenario I **two consecutive TCP SYN flooding attacks** have been launched against two different hosts. To detect and analyze these attacks the approaches described in Section 3.3.2 have been taken.

First the number of TCP flows per minute has been calculated and is plotted as a blue, dotted line in Figure 5.2. Shown as red lines are the calculated thresholds which are equal to the mean number of flows per minute over the last 20 minutes plus/minus five times its standard deviation.

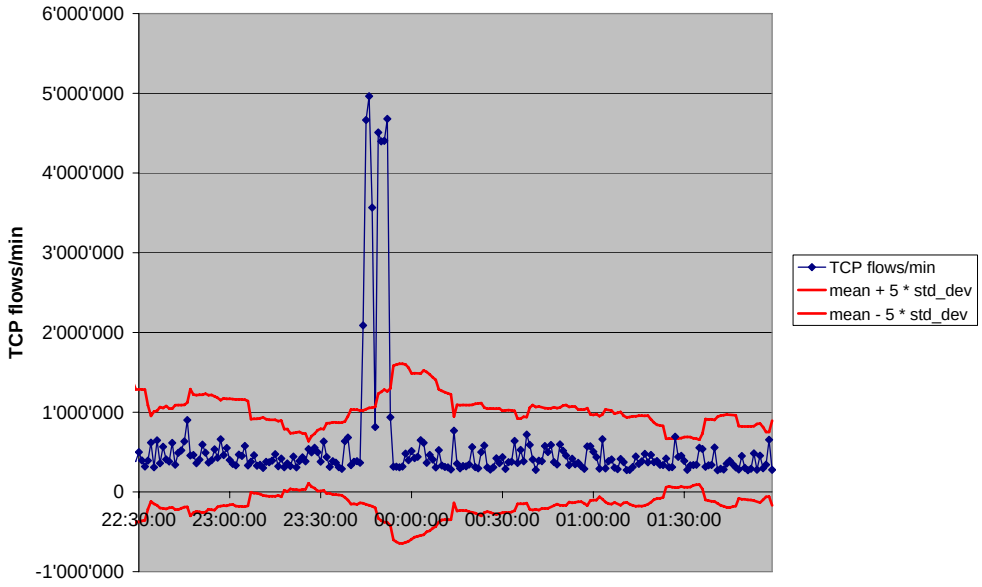


Figure 5.2: Number of TCP flows per minute for Scenario I

The graph clearly shows the striking rise in the number of flows per minute during the attacks which occured between 23:44 and 23:47 and again immediately thereafter between 23:49 and 23:52. The implemented threshold proves to be adequate as it identifies all the relevant minutes for the two attacks. It only produces one false positive at 01:29 which is caused by a very stable number of flows directly preceeding this minute. This leads to a smaller than usual standard deviation and results in a threshold which is a bit too low.
Figures 5.3 and 5.4 show the number of TCP packets and bytes per minute for comparison with Figure 5.2. There is no noticeable peak or anomaly visible during the attacks. This demonstrates the advantage of using flow statistics to detect this kind of attack at the backbone network level.

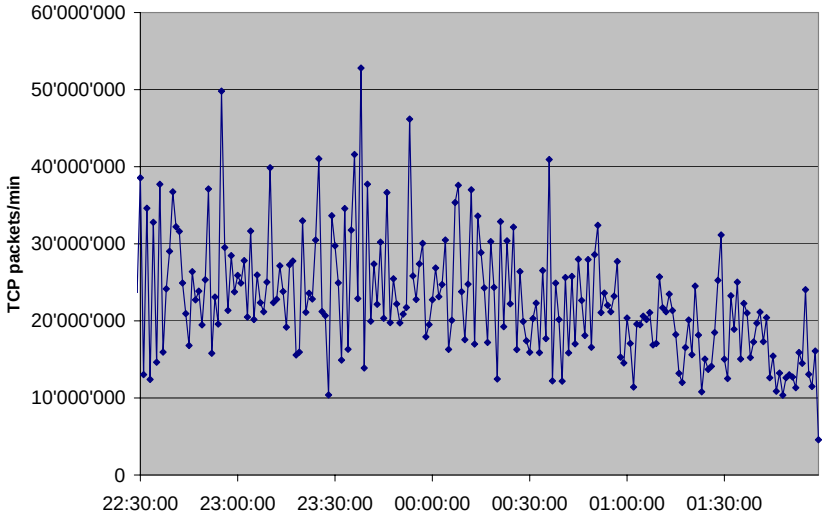


Figure 5.3: Number of TCP packets per minute for Scenario I

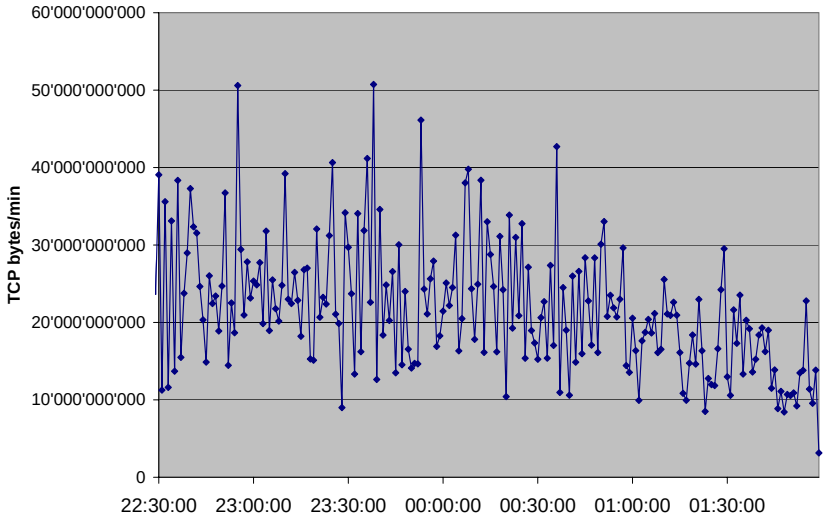


Figure 5.4: Number of bytes per minute sent with the TCP protocol for Scenario I

The next statistic that has been studied is the average number of packets in a TCP flow. This average has again been taken over each minute and is shown in Figure 5.5.

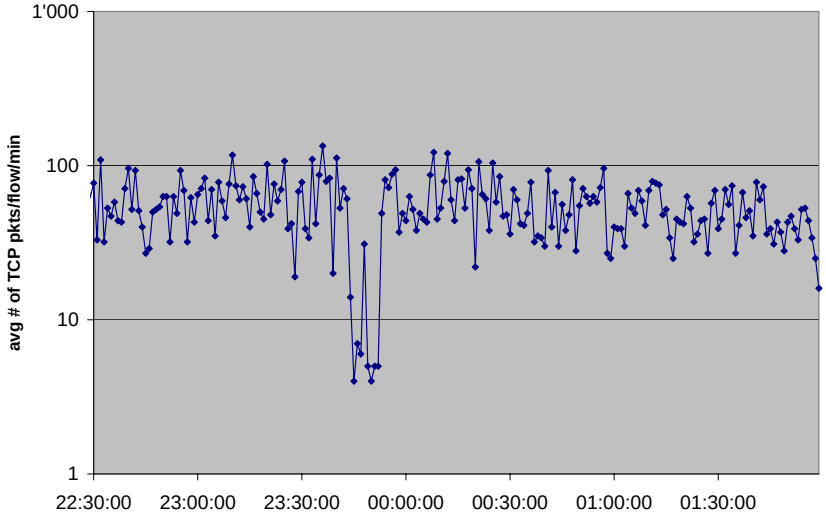


Figure 5.5: Average number of packets per TCP flow during each minute of Scenario I

The attack is clearly visible with the used log-scale in this plot but the threshold used in the flows per minute graph didn't deliver the expected results here and therefore isn't shown. The reason why it couldn't be used to determine the attack is the substantially smaller difference between the attack traffic and the normal traffic. Similar observations could be made on the graph showing the average number of bytes in a TCP flow. This value also drops distinctively during the attacks and is shown in Figure 5.6.

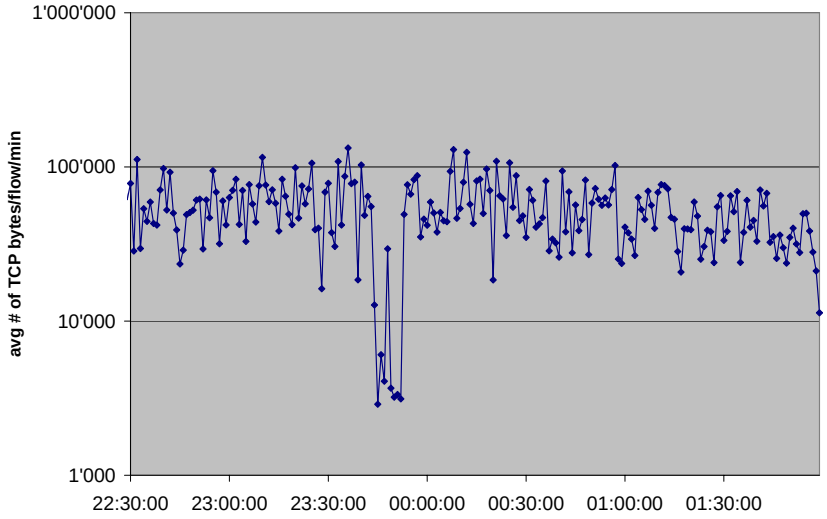


Figure 5.6: Average number of bytes per TCP flow during each minute of Scenario I

The last attempt to locate and characterize the attack has been to study the number of unique IP addresses seen in the traffic during each minute. Like in the previous statistics only the TCP traffic has been taken into account. The corresponding plot is presented in Figure 5.7.

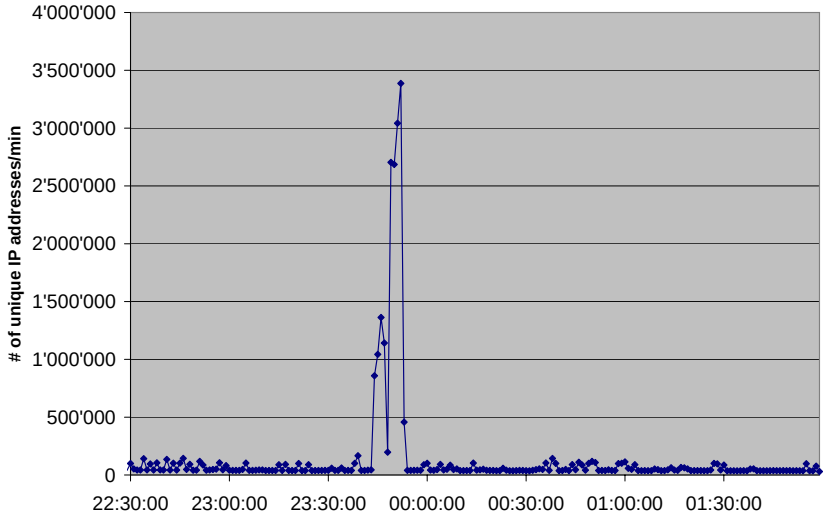


Figure 5.7: Number of unique IP addresses per minute for Scenario I

The graph shows that there is a massive increase of the number of different IP addresses observed during the attacks. The increase is well distinguishable from the normal traffic for both the first and especially for the second attack. The peak in the second attack is more than 50 times higher than the average value for one hour of normal traffic. This suggests that the attacker most likely randomly spoofed the source addresses - it is improbable that he had control over a botnet containing over three million hosts. It is possible that a smaller botnet was used in which all the hosts spoofed their IP address, or the traffic could have come from just one machine. Further analysis would have to be done to determine these details.

5.3.2 Scenario II

Scenario II features a **TCP flooding attack combined with a UDP flooding attack**. This combined attack is followed by another TCP flooding attack about one hour later. Both attacks were launched at one victim. The UDP flooding was not repeated in the second attack as the attacker seems to have learned that the UDP traffic has been filtered by a router and did not affect the victim host.
For the TCP attacks the same types of anomaly detection like the ones used in Scenario I have been applied. Figures 5.8 to 5.11 show the obtained results which are briefly discussed in the following.

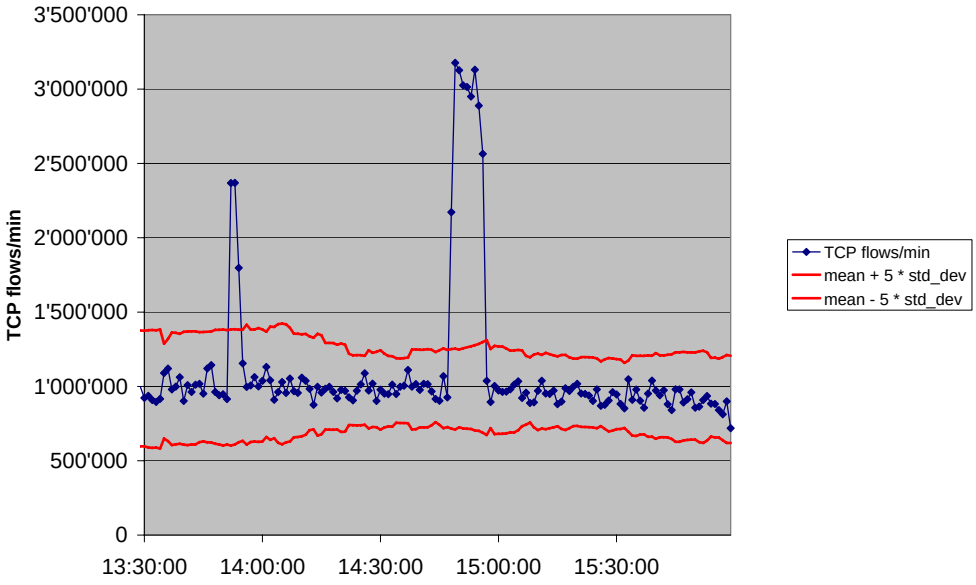


Figure 5.8: Number of TCP flows per minute for Scenario II

The number of TCP flows per minute reaches up to three times the usual value during the two attacks lasting from 13:52 to 13:54 and from 14:48 to 14:56. The anomalies can be well detected by the applied thresholds plotted in red. Compared to the first scenario there is no false attack identification occuring in this scenario.

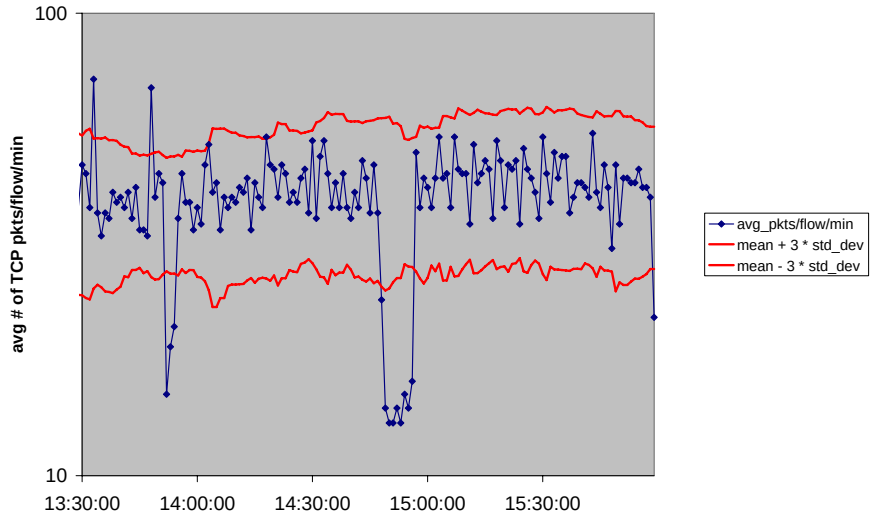


Figure 5.9: Average number of packets per TCP flow during each minute of Scenario II

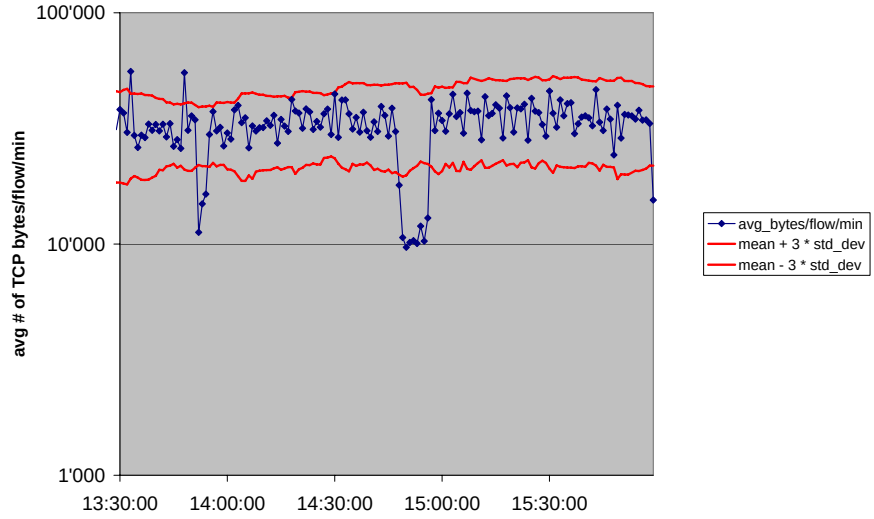


Figure 5.10: Average number of bytes per TCP flow during each minute of Scenario II

Figure 5.9 and Figure 5.10 show the average number of packets and bytes per TCP flow respectively. The two observed attacks cause the averages to dip down since most of the attack traffic contains only one or two packets and correspondingly only a few kbytes per flow. In this scenario an adapted version of the threshold used in the 'flows-per-minute-statistics' can be applied. The only difference to the previously used threshold is that only three times the standard deviation is added/subtracted from the mean value. With these settings an attack detection is possible although some false identifications are present.

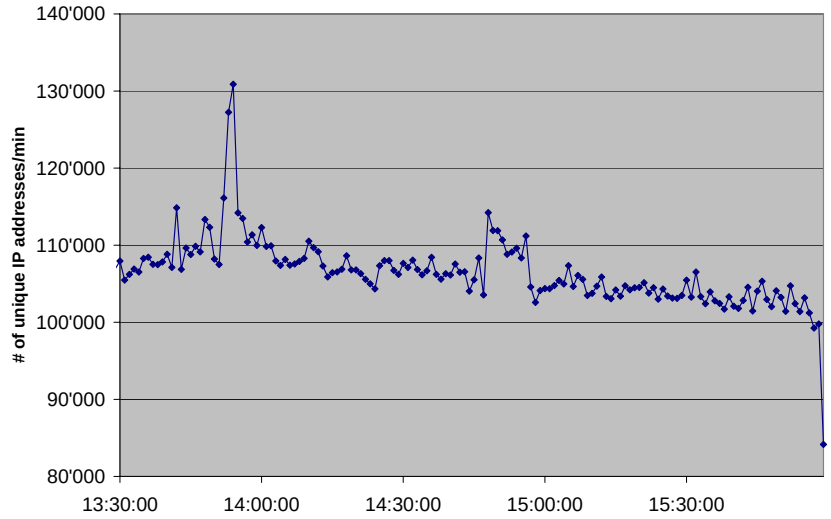


Figure 5.11: Number of unique IP addresses per minute for Scenario II

As shown in Figure 5.11 the number of unique IP addresses per minute shows only a weak increase when compared to Scenario I. The peak observed during the first attack represents an increase of about 30'000 addresses compared to the average number of unique addresses. This characteristic suggests that a botnet without spoofed addresses has been used for the attacks. To further examine this matter, the number of flows per IP address to the victim has been studied. Figure 5.12 shows this statistic for one minute of the attack (at 14:54).
Note: For privacy reasons the IP address ranges have all been permuted and do not reflect the real attacking addresses.

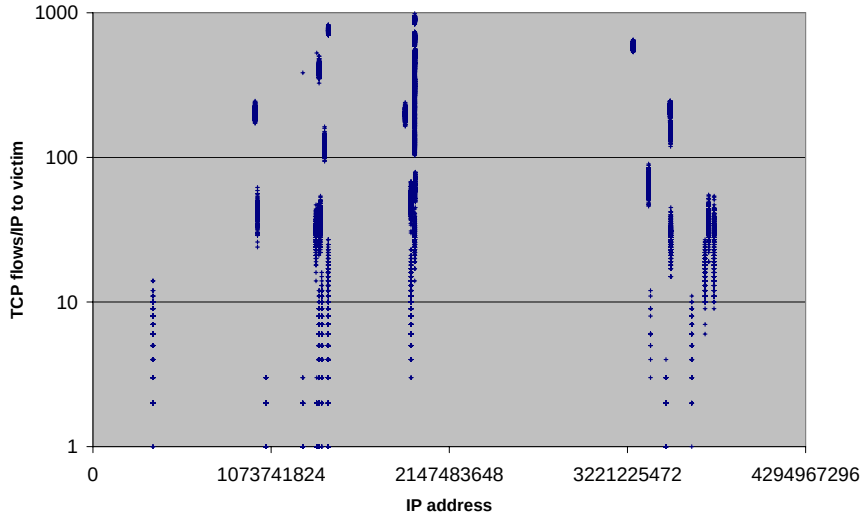


Figure 5.12: Number of TCP flows per source IP address to the victim during one minute (14:45)

The obtained results show that the distribution of the addresses is far from random. There are several distinct subnets involved in the attack. Further studies showed that the same subnets were present during both the TCP and the UDP attack as well as during the second TCP-only attack. These facts support the assumption that a botnet may have been used for the attacks.

Figure 5.13 shows the number of flows per minute for the UDP protocol.

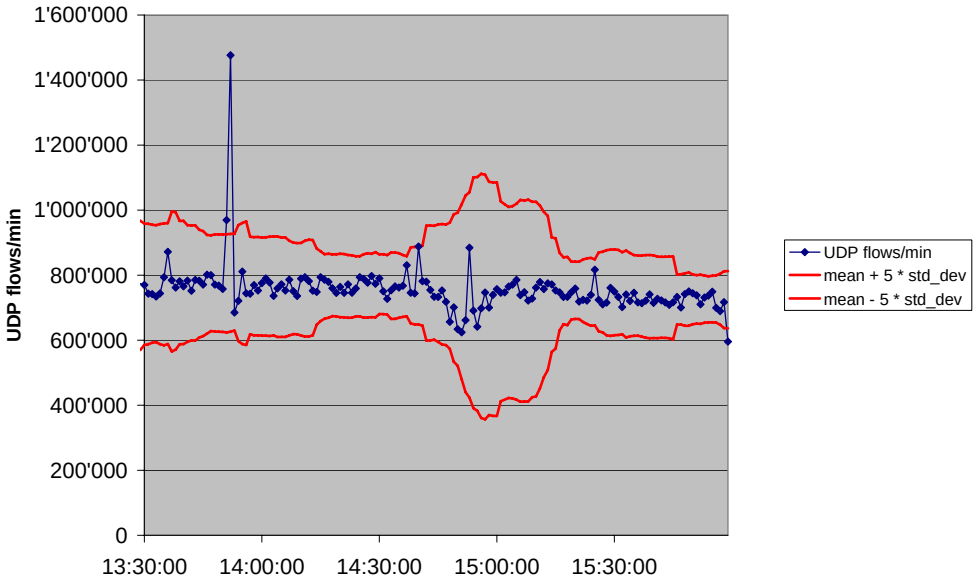


Figure 5.13: Number of UDP flows per minute for Scenario II

What can be seen in this plot is that only the first attack of Scenario II included UDP flooding. During the second attack the number of UDP flows per minute even decreases slightly. This is most probably caused by the high TCP attack traffic which suppresses the UDP traffic.
The last statistic which has been studied is the number of TCP flows originating from each source port of the attack and the number of flows towards each destination port of the victim during one minute of the attack at 13:52. The corresponding plots are shown in Figure 5.14 and Figure 5.15.

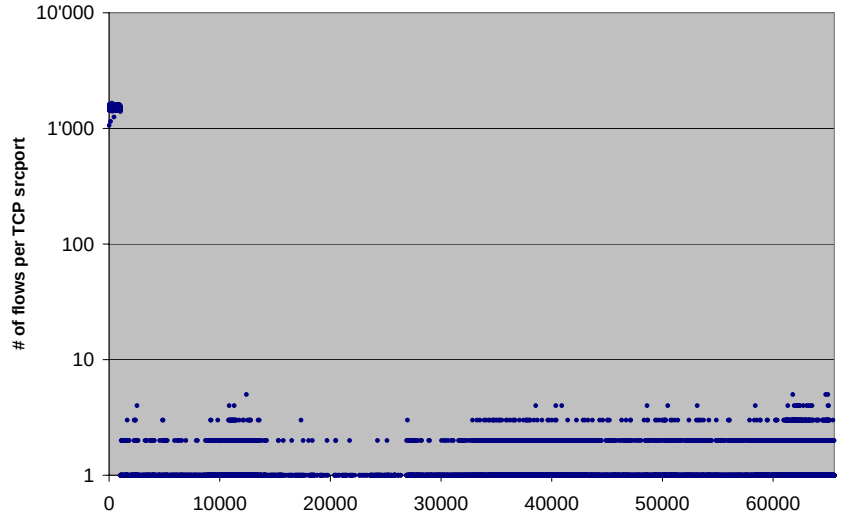


Figure 5.14: Number of TCP flows from each source port of the attack during one minute (13:52)

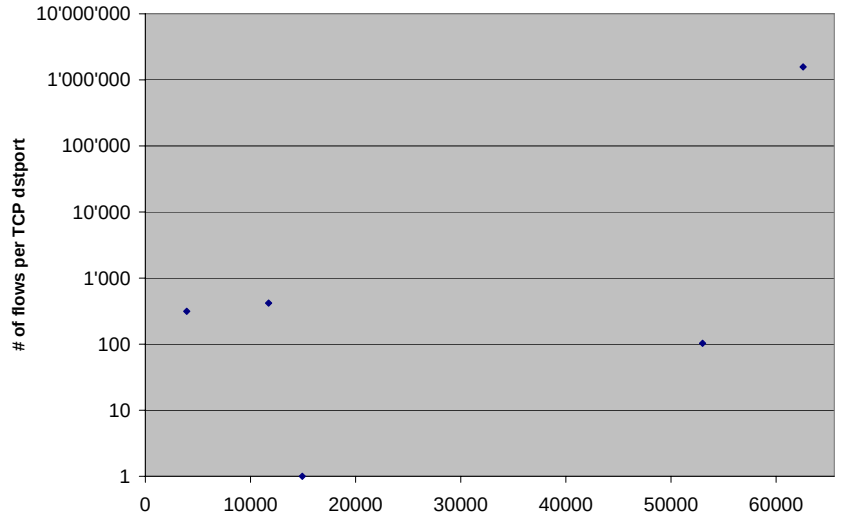


Figure 5.15: Number of TCP flows to each destination port of the victim during one minute (13:52)

This port analysis shows that most of the attack traffic is sent from the well-known ports ranging from 0 through 1023. On the registered and dynamic ports (1024-65535) there are only few recorded flows per port but they are well distributed over the whole range. Data has been sent from a total of 13481 different ports. On the receiving side however only 5 different ports are recorded with one port used for almost all of the traffic (99.947% go to port 62568).

Chapter 6

Conclusion and Outlook

6.1 Problems

A tedious problem encountered during the development of the Perl scripts was the following: Once a query was submitted to the database it couldn't be interrupted anymore since only the script got killed by sending a SIGINT signal (by pressing ctrl+c) but not the query itself. To stop the query, one had to connect manually to the database and kill it with the mysql tool. This was problematic because the manual kill is quite cumbersome and in the first phase several different types of queries have been tested which partly were too complex and therefore took way too long to finish. The first and unsuccessful attempt to solve this problem was to use the Perl signal handler to call a subroutine which interrupts the running query upon arrival of a SIGINT signal. This didn't work since the signal handler defers the interrupt until a strategic 'safe' points is reached, which is only after the query from the database returns. The solution to the problem was to use the POSIX interface to directly call the needed subroutine in case of a SIGINT which finally worked flawlessly.

6.2 Conclusion

The evaluation of the proposed traffic anomaly detection approach showed some promising results. The localization capabilities of both TCP SYN flooding and UDP flooding attacks have been demonstrated on two different scenarios which used real network traces recorded at the SWITCH border routers. It was shown in Section 5.3 that such massive flooding attacks can be fairly well detected.
Throughout the evaluation the database proved to be a very helpful tool in analyzing different aspects of NetFlow traces. It offers a versatile interface to the data with which various kinds of statistics can easily be generated.
Several other measures could still have been studied. This includes for example a more detailed study of the attack patterns by analyzing the flow statistics on a per second basis. Other interesting information would be whether the attacker performed indirect attacks to gather information about the victim.
The two remaining attacks discussed in Section 2.2 (namely the ICMP flooding and the DNS reflector attacks) couldn't be studied due to the lack of known incidents on the recorded data. It is assumed that the proposed detection methods would work evenly well if the respective attacks are as extensive as the ones studied in the previous section.

6.3 Outlook

Future work continuing the studies of this thesis could further refine the proposed detection methods or focus on the localization of new attack types. These could for example include the detection of host or port scans, which are often a good indicator that someone may be planning an attack.
An easy way to quickly evaluate the output of different database queries would be to create a PHP script running on a webserver which would prompt for a query, or certain parameters of a query, and then directly output the results on a webpage (as a table or in a graph). If implemented universally enough, this would allow rapid prototyping of different detection approaches.
Another idea could be to automate the whole detection procedure. If adequate automated thresholds can be evaluated, it is possible to write more advanced scripts which check several different attack types. The output of such a script could then for example be a summary of suspected anomalies along with information about all the involved hosts and possibly even more details about the spotted attack(s).
If the database approach was used for more comprehensive analyses, a change from the MySQL to a PostgreSQL database could be considered. This is due to the supported data types, which natively include network address types on the PostgreSQL database. This allows for a more convenient handling of operations on IP addresses, subnets and also MAC addresses since input error checking and several specialized operators and functions are available for these data types.

Acknowledgment

I would like to thank Prof. Dr. Plattner for the given opportunity to write a semester thesis at the Communication Systems Group and for the supervision of the whole project.
Special thanks go to my tutors Daniela Brauckhoff and Arno Wagner for their continuous support as well as the helpful advice and inspiring discussions during the whole work.
I would also like to thank the TIK services group for providing the whole infrastructure and technical equipment.

Appendix A

Schedule and Work Progress

Week	Date	Primary Task	Deliverables
1	25.09.-28.09.	Familiarization with NetFlow and the cluster	
2	01.10.-05.10.	Study of different papers discussing DoS attacks	
3	08.10.-12.10.	Derivation of suitable flow signatures, familiarization with DDoSVax tools	Schedule
4	15.10.-19.10.	Begin implementation of trace database	
5	22.10.-26.10.	Finish implementation of trace database	
6	29.10.-02.11.	Begin of script programming	
7	05.11.-09.11.	Preparation of informal presentation, continuing script programming	Informal presentation, ToC of report
8	12.11.-16.11.	Completion of scripts	
9	19.11.-23.11.	Evaluation of scripts	
10	26.11.-30.11.	Further evaluation and optimization of scripts	
11	03.12.-07.12.	Buffer (Automated Threshold Determination)	
12	10.12.-14.12.	Composition of report	
13	17.12.-21.12.	Completion of report and final presenation	Report, Final presentation

Table A.1: Schedule

Table A.1 shows the schedule as planned in the beginning of the project. The actual outcome was different because of several factors which are explained hereafter.
The first phase of getting familiar with the NetFlow data and the computing cluster used for the thesis did not raise any problems. The theoretical assessment of different DoS attacks and the familiarization with the tools from the DDoSVax project went well as planned. Then came the implementation of the trace database which was the first big challenge for me as I never had worked with MySQL and had only little experience in C-programming. After reading some information about the C API offered by MySQL I was able to write a C program which could provide the needed access to the database. The next step was to define the structure of the trace database. As my experience with databases was marginal it took quite a bit of time and support of my tutors to eventually get to the finally used structure.
Next up was the writing of Perl scripts to implement the attack localization with the help of the database. Since I had never used Perl before, I had to first read up on the basic syntax before I could start to write the database interface. After finishing a first version of the attack detection script I was already in the second week of the originally scheduled evaluation. Therefore the testing and evaluation of the developed attack detection approach was rescheduled and done in the remaining time before the end of the semester. As I was quite busy during the last week before Christmas with work from other subjects including an exam and a paper presentation, we decided to postpone the final presentation until January 2008. This also gave me time to write this report.
For me personally the project was a big success as I could gain a broad insight into several different subjects which surely will be very valuable in any further studies. This includes, besides the whole network-related knowledge, a very beneficial introduction to database management and Perl scripting.

Appendix B

Task Description

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Semester Thesis

for

Niklaus Frey

Supervisors: Daniela Brauckhoff, Arno Wagner

Issue Date: 25.09.2007
Submission Date: 21.12.2007

Localization of Known Anomaly Types in NetFlow Traces

1 Introduction

Using signatures for attack detection is very common in packet-level intrusion detection systems. Packet-level signatures describe patterns in payload and/or header data that are characteristic for a specific attack or exploit. On the flow-level something similar can be done. In this case, signatures describe the characteristic flow patterns of attacks (e.g., flows/min or flows/IP). Such attack patterns can be local, e.g., for single hosts (denial of service, scan) or single flows (heavy hitter), but also distributed over multiple hosts and flows (distributed denial of service, distributed scan).

This semester thesis aims at developing a tool that applies flow signatures for automatic localization of attacks that are present in captured NetFlow data. Therefore, an efficient and flexible mechanism for identifying local and distributed flow signatures is to be designed and implemented by the student. Moreover, since most of the patterns will apply some threshold value (e.g., maximum number of bytes per flow), a mechanisms for automated threshold determination from given data is to be implemented if time permits. Last but not least, the performance and reliability of the prototype needs to be evaluated.

The data used during this Master thesis is NetFlow [2] data which is collected on the SWITCH (Swiss Education Backbone Network) [1] border routers. This data comprises all aggregated packet headers (flows) which are sent between an ETH-internal host, and hosts which are not part of the SWITCH backbone network. Consequently, ETH-internal traffic and traffic between SWITCH-internal sites is not observed.

2 The Task

This thesis is conducted at ETH Zurich. The task of this Semester Thesis is to develop and implement a system that automatically localizes known types of anomalies in recorded NetFlow traffic traces. The student needs to analyze attack properties which can serve as flow signatures, implement an efficient and flexible framework for localizing flow signatures in the NetFlow data, and evaluate and test the developed prototype. If time permits, a methodology for determining pattern-specific thresholds is to be defined.

2.1 Theoretical Assessment of Attack Patterns

For this task the student needs to study literature on different types of attacks and anomalies, e.g., denial of service attacks, heavy hitter flows, as well as port- and host scans, in order to identify characteristics that can be used to (unambiguously) identify them in NetFlow data.

2.2 NetFlow Trace Database

Databases are a valuable tool for efficient searching in NetFlow traces since they provide a simple interface and indexing functionality. However, insert and delete operations are slow and the added memory overhead is considerable. Therefore, the student needs to develop a flow trace database which optimizes this trade-off. The goal is to hold up to 3 hours of data simultaneously in one or multiple databases allowing for fast data access and consuming as little memory as possible.

2.3 Script-based Attack Localization

The idea is to develop specialized scripts (in Python or Perl) which localize the attack patterns in the NetFlow data. Therefore, the scripts execute database queries, e.g., get all flows with a specific source port, and then further process the query results. Additionally, the scripts should seperate all important information about a localized attack (contributing flows, hosts, etc.) for further studies. This task is to be conducted for one or two selected anomaly types.

2.4 Evaluation

The developed prototype will be evaluated on at least one week of the SWITCH data. Therefore, the results of the signature matching prototype will be compared against security alert logs provided by the SWITCH network administrators.

2.5 Optional: Automated Threshold Determination

Most if not all of the flow signatures need to apply a threshold which decides between normal and abnormal behavior. The goal of this optional task is to develop a mechanism for automatic threshold determination based on a given data set. This mechanism is to be implemented for at least one specific attack signature.

3 Deliverables

The following results are expected:

- *Attack Pattern Survey.* A study of releated work that identifies possible flow characteristics/signatures for different types of attacks and anomalies such as denial of service, heavy hitter flows, as well as port- and network scans. This survey should also provide exact definitions for each analyzed attack.
- *Anomaly Localization Prototype.* A prototype for anomaly localization in NetFlow traces, including an optimized round-robin database, as well as attack localization scripts for at least two types of anomalies.
- *Documentation.* A concise description of the work conducted in this thesis (task, related work, problem statement, implementation, results and outlook). The abstract of the documentation has to be written in both English and German. The original task description is to be put in the appendix of the documentation. One sample of the documentation needs to be delivered at TIK. The whole documentation, as well as the source code, slides of the talk etc., needs to be archived in a printable, respectively executable version on a CDROM, which is to be attached to the printed documentation.

4 Organizational Aspects

4.1 Documentation and Presentation

A documentation that states the steps conducted, lessons learned, major results and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the thesis, a presentation will have to be given at TIK that states the core tasks and results of this thesis. If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network and security conference.

4.2 Dates

This Semester thesis starts on September 25th 2007 and finishes on December 21st 2007. It lasts 13 weeks in total. At the end of the second week the student has to provide a schedule for the thesis. It will be discussed with the supervisors.

After six weeks the student should provide a draft of the table of contents (ToC) of the thesis. The ToC suggests that the documentation is written in parallel to the progress of the work.

An intermediate informal presentation for Prof. Plattner and all supervisors will be scheduled 7 weeks into this thesis.

A final presentation at TIK will be scheduled close to the completion date of the thesis. The presentation consists of a 10 minutes talk and reserves 5 minutes for questions. Informal meetings with the supervisors will be announced and organized on demand.

4.3 Supervisors

Daniela Brauckhoff, brauckhoff@tik.ee.ethz.ch, +41 44 632 70 50, ETZ G97

Arno Wagner, wagner@tik.ee.ethz.ch, +41 1 632 70 04, ETZ H83

References

[1] SWITCH - The Swiss Education and Reserach Network. http://www.switch.ch/.
[2] Cisco Systems Inc. Netflow services and applications - white paper.

August 21, 2007

Bibliography

[1] Estonian 'Cyber Riot' Was Planned, But Mastermind Still A Mystery,
http://www.informationweek.com/showArticle.jhtml?articleID=201202784,
27.01.2008

[2] Cisco IOS NetFlow White Papers,
http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html,
09.01.2008

[3] SWITCH - The Swiss Education and Research Network,
http://www.switch.ch,
10.01.2008

[4] Sven Dietrich, Neil Long and David Dittrich,
Analyzing Distributed Denial of Service Tools: The Shaft Case,
Proceedings of the 14th USENIX conference on System administration, p. 329-340, 2000

[5] Alefiya Hussain, John Heidemann and Christos Papadopoulos,
A Framework for Classifying Denial of Service Attacks,
Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, p. 99-110, 2003

[6] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari and Darrell Kindred,
Statistical Approaches to DDoS Attack Detection and Response,
Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'03), p. 303-314, 2003

[7] Jelena Mirkovic and Peter Reiher,
A Taxonomy of DDoS Attack and DDoS Defense Mechanisms,
ACM SIGCOMM Computer Communication Review, Volume 34, Issue 2, p. 39-53, 2004

[8] Constantinos Kotsokalis, Demetrios Kalogeras and Basil Maglaris,
Router-based detection of DoS and DDoS attacks,
Journal of Network and Systems Management, Volume 12, Number 1, p. 73-94, 2004

[9] Daniel Reichle,
Analysis and Detection of DDoS Attacks in the Internet Backbone using Netflow Logs,
Diploma Thesis DA-2005.06, TIK, ETH Zurich, 2005

[10] Thomas Dübendorfer, Arno Wagner, Bernhard Plattner,
A Framework for Real-Time Worm Attack Detection and Backbone Monitoring,
Proceedings of First IEEE International Workshop on Critical Infrastructure Protection (IWCIP 2005), Darmstadt, Germany, 3-4 November, 2005

[11] Smurf Amplifier Registry (SAR),
http://www.powertech.no/smurf/,
11.01.2008

[12] CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks,
http://www.cert.org/advisories/CA-1996-21.html,
11.01.2008

[13] Slashdot effect,
http://en.wikipedia.org/wiki/Slashdot_effect,
28.01.2008

[14] Daniela Brauckhoff, Martin May, Bernhard Plattner,
Flow-level Anomaly Detection: Blessing or Curse?,
IEEE INFOCOM 2007, Student Workshop, Anchorage, Alaska, USA, May, 2007

[15] Daniela Brauckhoff, Martin May, Bernhard Plattner,
Comparison of Anomaly Signal Quality in Common Detection Metrics,
ACM SIGMETRICS 2007, MineNet Workshop, San Diego, CA, USA, June, 2007

[16] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May and Anukool Lakhina,
Impact of Packet Sampling on Anomaly Detection Metrics,
ACM Internet Measurement Conference 2006, IMC 2006, Rio de Janeiro, Brazil, October, 2006

[17] Arno Wagner, Thomas Dübendorfer, Lukas Hämmerle, Bernhard Plattner,
Flow-Based Identification of P2P Heavy-Hitters,
International Conference on Internet Surveillance and Protection (ICISP) 2006,
August 26 - 29, 2006 Cap Esterel, Côte d'Azur, France

[18] Thomas Dübendorfer, Arno Wagner, Theus Hossmann, Bernhard Plattner,
Flow-Level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone,
GI SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2005), Vienna, Austria, July 7-8, 2005

[19] DDoSVax research project,
http://www.tik.ee.ethz.ch/~ddosvax,
09.01.2008

[20] DBI - Database independent interface for Perl,
http://search.cpan.org/ timb/DBI/DBI.pm,
16.01.2008