PhD-FSTC-2012-10
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 30/03/2012 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Cynthia WAGNER
Born on 2nd August 1982 in Esch/Alzette (Luxembourg)

SECURITY AND NETWORK MONITORING BASED ON INTERNET FLOW MEASUREMENTS

Dissertation defense committee

Dr Thomas Engel, dissertation supervisor
Professor, Université du Luxembourg-SnT

Dr Chrisptoh Schommer, Chairman
Professor, Université du Luxembourg

Dr Vijay Gurbani, Vice Chairman
Professor, Illinois Institute of Technology and Bell Laboratories

Dr Radu State
Dr. habil., Université du Luxembourg – SnT

Dr Jean Hilger
Banque et Caisse d'Epargne de l'Etat (BCEE)

Acknowledgments

This doctoral thesis has been realized at the SECAN-LAB of the Interdisciplinary Centre for Security, Reliability and Trust (SnT) and the University of Luxembourg. Writing this doctoral thesis without help and support from kind people would not have been possible.

First of all, I would like to thank my supervisor, Prof. Dr. Thomas Engel, for giving me the opportunity of being a member of his research team for the last four years. I owe sincere and earnest thanks to my supervisor Dr.hab. Radu State for his support and advice. I want to thank Prof. Dr. Vijay Gurbani for being part in my CET committee. I owe sincere thanks to Dr. Jean Hilger and Prof. Dr. Christoph Schommer for participating in the jury of this thesis.

Furthermore, I would like to thank Dr. Gérard Wagener and Alexandre Dulaunoy from the Computer Incident Response Centre Luxembourg for providing relevant research data and especially for their scientific cooperation and support.

Special thanks are due to RESTENA Luxembourg, especially to Prof. Duhautpas and his team for supporting this thesis. I am indebted to my many of my colleagues for their support and especially to Dr. Jérôme François for collaborating in this work. Thanks to Dominic Dunlop for reviewing my works. Special thanks go to Sheila Becker, my office colleague and friend, thanks for making the last four years fun and unforgettable.

Besides the strong professional support, I owe my deepest gratitude to my family, especially to my mother and Tom for their strong belief in me. I know during this period you have given me a lot of patience and understanding. Without you this thesis would not have been possible.

Abstract

Today's networks face continuously arising new threats, making analysis of network data for the detection of anomalies in current operational networks essential. Network operators have to deal with the analysis of huge volumes of data. To counter this main issue, dealing with IP flows (also known as Netflows) records is common in network management. However in modern networks, even Netflow records still represent a high volume of data. Interest in traffic classification as well as attack and anomaly detection in network monitoring and security related activities has become very strong.

This thesis addresses the topic of Netflow record analysis by introducing simple mechanisms for the evaluation of large quantities of data. The mechanisms are based on spatially aggregated Netflow records. These records are evaluated by the use of a kernel function. This similarity function analyses aggregated data on quantitative and topological pattern changes. By the use of machine learning techniques the aim is to use the aggregated data and classify it into benign traffic and anomalies. Besides the detection of anomalies in network traffic, traffic is analyzed from the perspective of an attacker and a network operator by using a game-theoretical model in order to define strategies for attack and defence.

To extend the evaluation models, information from the application layer has been analyzed. An occurring problem with application flows is that in some cases, network flows cannot be clearly attributed to sessions or users, as for example in anonymous overlay networks. A model for the attribution of flows to sessions or users has been defined and related to this, the behaviour of attack and defence mechanisms is studied in the framework of a game.

Contents

1 Introduction **1**
- 1.1 A global picture . . . 1
- 1.2 Problem statement and methodology . . . 4
- 1.3 Contributions and thesis outline . . . 5

I State of the art **9**

2 Introduction **11**
- 2.1 IP flows from the network . . . 11
  - 2.1.1 What is an IP flow? . . . 11
  - 2.1.2 How to collect IP flows . . . 12
  - 2.1.3 A short historical summary: from Netflow records to IPFIX . . . 13
  - 2.1.4 Major issues with Netflow records . . . 14
- 2.2 Network anomalies . . . 14
  - 2.2.1 Technical anomalies . . . 14
  - 2.2.2 Categories of attack . . . 15

3 From IP flow monitoring to dependency identification **17**
- 3.1 Sampling methods . . . 17
- 3.2 Aggregation methods . . . 18
- 3.3 Storage of IP flow records . . . 20
- 3.4 Dependencies in IP flow information . . . 22
  - 3.4.1 Probability-based models . . . 22
  - 3.4.2 Rule-based and correlation-based models . . . 23
  - 3.4.3 Social behaviour-based models . . . 25
- 3.5 Approaches to Netflow record analysis and evaluation . . . 26
- 3.6 Limitations and advantages . . . 29

II Contributions **31**

Part A: Netflow record analysis using machine learning **33**

4 Single Netflow analysis **35**
- 4.1 A kernel function for Netflow processing . . . 35
- 4.2 The classification method . . . 38
  - 4.2.1 One-class SVM . . . 38
- 4.3 The architecture of the anomaly detector . . . 39
- 4.4 Experimental results . . . 39
  - 4.4.1 A grid search for σ estimation . . . 39
  - 4.4.2 OCSVM evaluation . . . 40
- 4.5 Summary and conclusions . . . 41

5 Spatio-temporal aggregation for Netflow records 43
5.1 Tree-like Netflow aggregation in space and time . . . 43
5.2 The model for spatio-temporal aggregation . . . 45
5.2.1 A kernel function for spatio-temporal aggregation . . . 45
5.2.2 The architecture . . . 46
5.3 Summary . . . 47

6 A game for aggregated Netflow records 49
6.1 Game-theoretical modeling for spatially aggregated Netflow records . . . 50
6.2 Attack and defence measures . . . 51
6.2.1 Defensive measures . . . 51
6.2.2 Attack measures . . . 51
6.2.3 Modelling of actions and computation of payoffs . . . 52
6.3 Experimental results . . . 53
6.3.1 Injection attacks . . . 53
6.4 Game-theoretical evaluation . . . 54
6.5 Summary . . . 56

7 Reconstructing missing dimensions in aggregated Netflow records 57
7.1 The model for DANAK . . . 58
7.1.1 The kernel function for traffic profiles . . . 58
7.1.2 The phase space method with delayed coordinates . . . 59
7.1.3 The classification algorithm . . . 60
7.1.4 The architecture of DANAK . . . 61
7.2 Experimental results . . . 62
7.3 Summary . . . 63

Part B: Analysis of Tor network flows using machine learning 65

8 Breaking Tor anonymity by flow analysis through data mining 67
8.1 Introduction to data-mining a Tor session . . . 67
8.2 The semi-supervised clustering algorithm . . . 69
8.3 Experimental results . . . 71
8.3.1 Passive attacks . . . 71
8.3.2 The hidden exit node . . . 71
8.4 Summary . . . 72

9 A game for attack and defence strategies in Tor flow analysis 73
9.1 Defence and attack strategies . . . 73
9.1.1 Advanced attacks . . . 74
9.2 Experimental Results . . . 75
9.2.1 Defence strategies: playing Tor games . . . 75
9.3 Summary . . . 77

III Conclusions and Future Work 79

IV Appendices **85**

A Data sets **87**
A.1 Simple Netflow Analysis 87
A.2 Aggregation-based evaluation of Netflow records 88

B The FLAME [12] attack injection tool **89**
B.1 General description 89
B.2 Attack data set description 90

C PeekKernelFlows — A visualization method for aggregated Netflow records **91**
C.1 Visualizing aggregated Netflow records 91
C.2 The visual representation of PeekKernelFlows 92

D The proof-of-concept Torinj architecture **93**

E Glossary **105**

F About the author **107**

List of Figures

1.1 60 Seconds - Things that happen on Internet every 60 seconds [15] 1
1.2 Estimate of traffic load until 2015 by Cisco [106] 2
1.3 Trend of technical intruder knowledge vs. attack complexity [79] 3
1.4 Thesis structure 7

2.1 Examples for Netflow records 12
2.2 Architecture for export and collection of IP flows [25, 24] 13

3.1 Aggregation cluster in [61] 19
3.2 Aggregation process in [68] 19
3.3 Main modules of NetStore (left) and processing steps (right) [50] 20
3.4 Multi-tank view storage process in DIPStorage [90] 21
3.5 Dependency graph in [7] 22
3.6 eXpose output from generated rules [117] 24
3.7 Dependencies in BLINC [70] 25
3.8 Relationship between predicates and packet rule classification [33] 26
3.9 Anomaly detection with histogram-detectors and association rules [11] 27

4.1 Example of Netflow records 36
4.2 Illustration for parameters `prefix` and `prefixlength` 36
4.3 Visualizing an attack by using the kernel function 37
4.4 Anomaly detector architecture 39

5.1 Graphical representation of spatio-temporal aggregation 43
5.2 Text output of aggregated destination traffic profile 45
5.3 Graphical representation of the kernel function on traffic proles 45
5.4 The architecture of the prototype 46

6.1 The Markov model transition diagram 52
6.2 Visualization of a service injection attack vs. normal traffic 53
6.3 Injection of a constant number machines over 30 seconds 54
6.4 Quantal response equilibrium evaluation 55

7.1 Kernel function evaluation for a violent DDoS UDP flood attack 57
7.2 Kernel function evaluation for a stealthy DDoS UDP flood attack 58
7.3 Stealthy DDoS UDP flood attack without phase space embedding analysis . . 60
7.4 DDoS UDP flood attack with phase space embedding analysis 60
7.5 General view of the DANAK architecture 61

8.1 Typical attack scenario 67
8.2 Injection attack 68
8.3 Flow A_2 interaction details 68
8.4 Semi-supervised learning algorithm 69

9.1 Tagging server attack with HTTP error message 75
9.2 Users changing User Agent 76
9.3 Quantal response equilibrium evaluation 77

B.1 Data set generation using Flame 89

C.1 The PeekKernelFlows monitoring framework 91
C.2 Colour representation of `PeekKernelFlows` 91
C.3 Figure for source (left) and destination (right) profiles 92

D.1 An overview of the Torinj framework 93

Chapter 1

Introduction

1.1 A global picture

Do you know what happens on the Internet in 60 seconds? Or have you ever asked the question, how many emails are sent in 60 seconds or even, how many voice calls are made in 60 seconds? To be honest, I did not know and therefore,...I googled... with the result that I finally found a web-blog [15] with an article covering exactly this topic and providing some facts about the daily use of the World-Wide-Web (represented in Figure 1.1). In that moment, I recognized that my search query on Google was one out of 694,445 made every 60 seconds....This set me thinking.

Figure 1.1: 60 Seconds - Things that happen on Internet every 60 seconds [15]

With the ever- increasing popularity of the Internet, a direct consequence is the increasing amount of content and traffic. Looking at Figure 1.1, it can be seen that 168 million emails are sent every 60 seconds, more than 70 new domains are registered and 370,000 voice calls

are made with Skype[1]. In a recent white paper by Cisco [106], the claim is made that by end of 2015, global IP traffic will reach one zettabyte per year and break the threshold of three billion Internet users. This increase is due to the growing connectivity in developing countries and also the increase of mobile devices using the Web. This means in Western Europe, in 2015, IP traffic will probably reach 19 exabytes per month, which represents around 32% of the global annual compound rate (GACR). Figure 1.2 by Cisco [106], shows an estimate of the evolution of traffic load from 1995 to 2015.

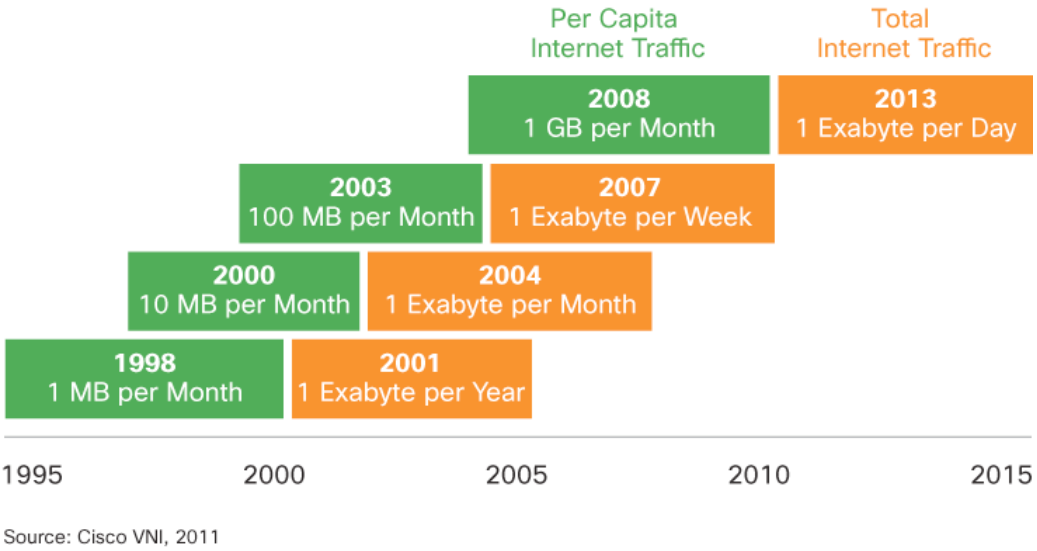

Figure 1.2: Estimate of traffic load until 2015 by Cisco [106]

This can be explained by the rapid increase in bandwidth over the last years. In the 90's most private Internet users had slow Internet connections (28.8/33.6/56 kbps modems) whereas nowadays a connection of 2-10 Mbits/s is quite common. The same is true for content: for example in 1995 the main activity was email and most people restricted their emails to simple text and low-bit images in order to keep email size below 1 Mbyte. Today, with high bandwidth connections, content has changed; common activities are social networking[2], Peer-to-Peer transmission, video and sound transmission for example with YouTube[3] and similar services.

This increase of Internet traffic also requires more traffic control on the network architecture operation side, because harmful traffic and attacks will increase as well. A recent example is the attacks launched against Sony's PlayStation Network[4]. A statement from Sony on the Forbes' Website[5] states that the overall costs for the attacks in April 2011 are more than $170M.

From a quantitative perspective, the 2010 Kaspersky Security Bulletin [75] states that they observed about 580 millions attacks on the Internet. These attacks were originated in different countries, but most malware detected by Kaspersky originated from 20 countries, for example the USA or China. A similar situation can be observed for spam and phishing emails. In a monthly report [129] from Symantec it is claimed that about 74% of the emails from September 2011 were spam, whereas only one in 188 emails contained malware.

[1] `http://www.skype.com/`
[2] examples: `http://www.facebook.com,http://twitter.com,http://www.linkedin.com/`
[3] `http://www.youtube.com`
[4] `http://lu.playstation.com/psn/news/`
[5] `http://www.forbes.com/sites/insertcoin/2011/05/23/sony-pegs-psn-attack-costs-at-170-million`

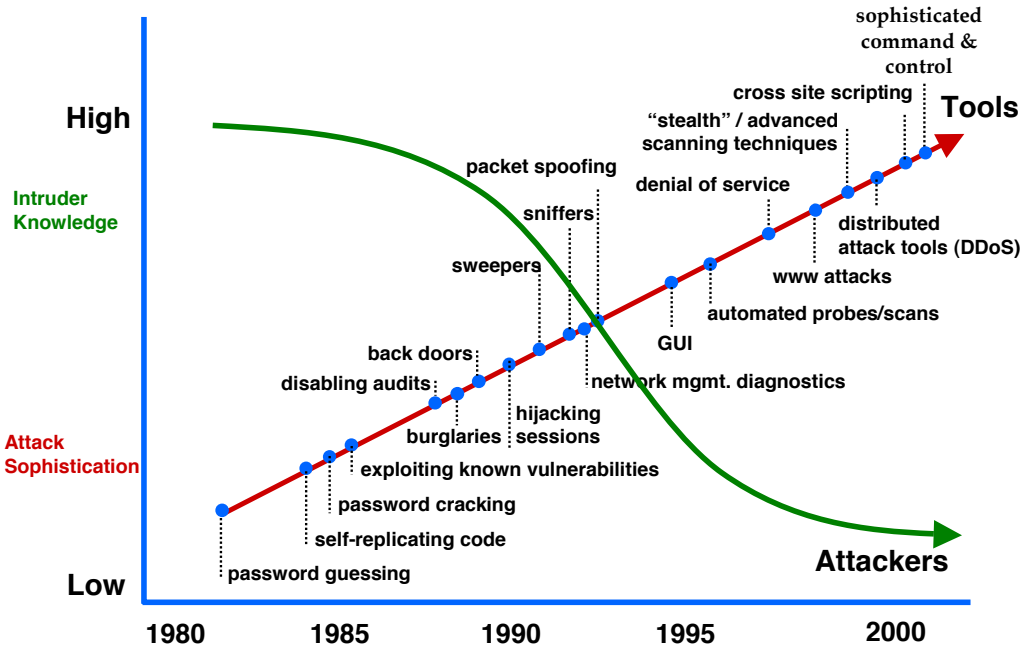

Figure 1.3: Trend of technical intruder knowledge vs. attack complexity [79]

Besides these threats, a recent study from Eurostat [41] showed that in 2010 in the European Union, nearly 60% of private users used security software to protect their equipment, but 55% of the individuals questioned also indicated that they only update their security products occasionally.

Besides the aspect on the quantity of attacks, their complexity and impact also increased. For example in Figure 1.3 from 2002 by [79], the complexity trend for attacks for the last 20 years is represented. It can be observed that due to the availability on the Internet of "off the shelf" attacks, an attack can be performed with two clicks nowadays. This shows that deep technical and programming knowledge is no longer really necessary to play the hacker. Such attackers are known as *script-kiddies*. Of course, highly sophisticated attacks and new attacks still demand technical knowledge from an intruder; for example to write new codes or 0-day exploits.

These arguments show that the need for strong monitoring tools network architectures is becoming indispensable, now and in the future, in order to reinforce the security of users, to minimize the impact of attacks and to neutralize possible threats in advance. Due to the large quantities of network data available, a realistic approach to monitoring is to use IP flow records. This kind of data provides useful information about communicating partners (who with whom), the quantity (how many) and additionally provides information about the type of exchanged data (what). For monitoring at the Internet Service Provider (ISP) level this provides a good solution due to its compact format. A formal definition of IP flow records is provided in section 2.1. The terms *IP flow record* and *Netflow record* will be used interchangeably for the rest of this document.

1.2 Problem statement and methodology

The aim of this thesis is to introduce simple mechanisms for the analysis of Netflow records in order to detect anomalies or attacks by combining a simple distance function with machine learning approaches. Over recent years, the domain of network monitoring became a popular research area in both academia and industry. Some of the landmark papers in this research area are, [129, 75, 76, 70, 86, 73, 7, 160, 33, 50, 97, 140], but with permanently increasing workloads and newly arising network threats, interest in further research in this domain remains. The information captured on the wire not only provides insights into the world of technical issues like machine crashes, but also provides precious information about anomalies, for example Denial-of-Service attacks, worms, etc. Huge quantities of such data are generated at the ISP level. These huge quantities of data should be analyzed and evaluated by introducing new evaluation metrics extracted from Netflow records for the generation of a new distance function. An initial research question is,

- What metrics can be extracted from IP flow record information in order to define a new distance function that is able to capture changes in patterns of traffic?

This research question deals with the analysis of Netflow record data in order to identify useful metrics, which can be extracted and used for the definition of a distance function, also known as a kernel function. The aim of this kernel function is to capture changes in traffic information in two different ways. In a first instance significant quantitative changes should be recognized and second, topological changes in traffic should be detected. Additionally, this function is defined to represent traffic information as an additional metric that can be applied to machine learning for the purpose of classification. For classification, different algorithms are evaluated in order to identify the most appropriate for the kernel function. To validate the approach, experiments are performed Netflow record time series are applied to the defined kernel function and subsequently evaluated using a variety of classification algorithms.

A limitation of the previous approach is that the analysis of a time series that includes all monitored Netflow records for a time period, is a tedious and complex task, as the storage of all Netflow records is nearly impossible, since peak rates of 60,000 flows/second at the ISP level are quite common. Therefore the previous research question is extended in order that a summarized form of Netflow records can be analyzed to reduce the storage problem. After studying different compression/summarization techniques for Netflow records, a decision was made to use tree-like aggregation of IP flow data together with a corresponding reformulation of the kernel function. Therefore, the second research question is

- How to detect anomalies and attacks in spatially and temporally aggregated IP flow information?

This second research question addresses the topic of detecting anomalies in aggregated Netflow records by reformulating the kernel function such that it is able to deal with spatially and temporally aggregated Netflow records. In this tree-like aggregation process small IP flow records are aggregated into larger ones. This provides traffic profiles for a given time period. In order to catch changes in the traffic profiles, the kernel function has to be adapted and reformulated to extract useful metrics. To validate these modifications, time series of traffic profiles are evaluated by the kernel function and machine learning algorithms and additionally a mechanism to reconstruct sparse or lost traffic time periods is modelled.

From a network operator or attacker perspective it is interesting to know which kind of defensive measure or, respectively, attack to choose, such that the result of the action is optimal. A game-theoretical model is introduced in order to study the behaviour of different strategies for attacking/defending a network architecture.

The research question that arises is,

- Which attack and defence strategy performs the best with the analysis of Netflow records?

This research question deals with the evaluation of different attack and defence strategies in a network by applying aggregated Netflow records and the kernel function to a game-theoretical model. Different strategies for an attacker and for the network operator are identified and evaluated by applying a game-theoretical model called Nash Equilibrium.

Different kinds of flows exist in a network. To extend the evaluation of flows, application layer flows are analyzed. The problem with application flows is that in some cases, flows cannot be attributed to users or sessions, as for example in anonymous overlay networks, for example Tor [133]. This last research question deals with the topic of,

- How may IP flow information from the application layer be attributed to a user or session?

This research question addresses the topic of analyzing anonymous network flows in order to reconstruct sessions by controlling an exit node, so allowing evaluated flows to be attributed to users. For this approach a semi-supervised learning algorithm is introduced. To enhance the model, a game-theoretical framework is modelled to study the behaviour of attacker and defender strategies such that the optimal strategies for both parties can be identified.

1.3 Contributions and thesis outline

The structure of this thesis follows the order of the main research questions and is represented in outline form in Figure 1.4. It is divided into two parts, State of the art and contributions.

The first part, State of the art, includes relevant work done in the area of network monitoring and while dealing with Netflow records, different kinds of threats have to be managed. Therefore, it is considered relevant to regroup the state of the art part for the analysis of IP flow records into different chapters. These chapters are organized by considering different aspects related to IP flow records. The state of the art is structured as follows:

- **Chapter 2** gives an introduction to Netflow records and describes the research domain of Netflow records and anomaly detection.

- **Chapter 3** presents relevant research work done for the capture and storage of IP flow/packet information. This chapter is divided into four sections, the first section describes recent sampling and aggregation methods for IP flow/packet information. The second section discusses approaches for the storage of network data. The third section highlights relevant work for the detection of dependencies between flows and applications. The fourth section presents state of the art work performed in the area of anomaly detection for IP flow records while using machine learning approaches.

The second part, Contributions, focuses on the several research contributions that are made by this thesis. The Contribution part is divided into two parts, A and B, which deal with two kinds of network information: Part A covers the Netflow record data from an ISP that has been used as a basis for the main contributions (chapters 4 to 7), while Part B uses Tor data for the evaluation of the attribution of flows to users or sessions (Chapters 8 and 9).

Part A:

- **Chapter 4** presents the analysis of Netflow records from a large-scale network. This chapter contributes a new approach for analyzing large volumes of IP ow-related data by defining a kernel function that is able to track topological and quantitative changes in traffic. The objective is to detect traffic anomalies and to characterize network traffic by applying machine learning.

- **Chapter 5** presents an aggregation technique that preserves topological and quantitative changes in a network without losing much data. To evaluate the aggregated data, the kernel function for Netflow records is reformulated to be applicable to the new data format.

- **Chapter 6** presents a game-theroretical model to identify the best strategy mainly from the perspective of an attacker having complete knowledge about the defence system, but also from the perspective of a network operator.

- **Chapter 7** deals with the topic of sparse data or lacks of data series in time series. This has an effect on the evaluation of the traffic time series. In this chapter, the main contribution is a new model that uses an accurate method for the embedding of time series into a n-dimensional space such that missing dimensions in aggregated traffic profiles can be reconstructed and evaluated by applying a classifier in order to separate attacks from benign traffic.

Part B:

- **Chapter 8** presents a new model for the reconstruction of Tor sessions by applying a semi-supervised learning algorithm. A new attack model is presented that leverages a data mining technique together with aspects of the HTTP protocol (as only Web traffic is considered) to cluster and extract individual HTTP sessions relayed over a malicious exit node.

- **Chapter 9** contributes an efficient game-theoretical model that on one hand deals with user privacy by presenting a variety of defensive measures and, on the other, presents attack strategies for the attacker to remain undetected such that a maximum number of sessions can be reconstructed.

- The **Conclusions and Future Work** chapter concludes the thesis and presents possible future work.

- The **Appendix** provides information about the data sets and tools used for the experiments , as well as information that was not considered as scientific contributions per se. An example is the implemented prototype, `PeekKernelFlows`, a vizualization tool for the evaluation of kernel function results.

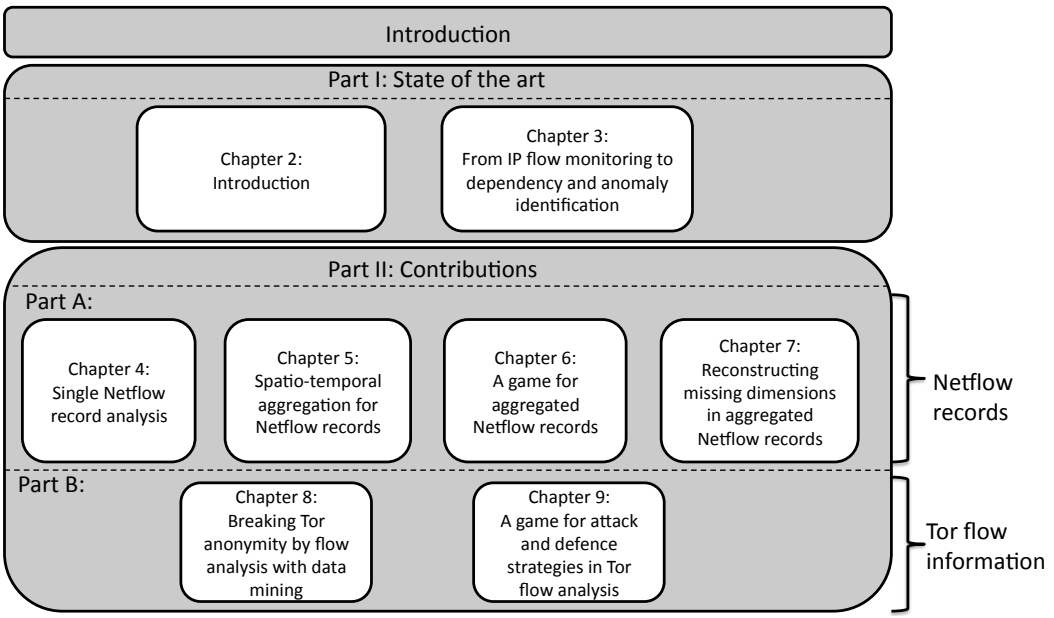

Figure 1.4: Thesis structure

Part I

State of the art

Chapter 2

Introduction

State of the art in Netflow monitoring

The monitoring of network architectures and observation of network activities is highly important. To dig into the world of network data, this chapter provides background information about IP flow related data. Basic information about Netflow related data is explained to introduce the area of network monitoring.

2.1 IP flows from the network

2.1.1 What is an IP flow?

In network monitoring, IP flow records are an easily available source of information. For the definition of an IP flow the reader is referred to the RFC 3954 [24] and current IETF standard, IPFIX (RFC 5101) [25], where an IP flow is defined as,

- IP Flow or Flow: *An IP Flow, also called a Flow, is defined as a set of IP packets passing an Observation Point in the network during a certain time interval. All packets that belong to a particular Flow have a set of common properties derived from the data contained in the packet and from the packet treatment at the Observation Point.*

- Flow record: *A Flow Record provides information about an IP Flow observed at an Observation Point.*

The common properties of flows are also called `flow keys` and can be defined as fields in the packet header, particular packet properties and fields resulting from packet treatment itself. Flow keys commonly used to identify a flow are: `(Source IP address, Destination IP address, Source port, Destination port, IP protocol, byte count)`.

Depending on the required level of granularity, flows can have more detailed definitions. In Figure 2.1, some fictive IP addresses are shown to illustrate the structure of a Netflow record. For privacy reasons, real IP addresses from the data set cannot be shown.

In this case flow records have additional fields and can be written as *(Date, Start Time, Duration, Protocol, Source IP Address, Source Port, Destination IP Address, Destination Port, Packets, Bytes, Flows)*. The example of Figure 2.1 shows not only a sample of Netflow records, but also shows the case of a spamming attack. This can be identified by observing the multiple incoming messages on port 25.

To typify traffic, sources, destinations, traffic direction and symmetry must be analyzed. Traffic can be classified between unidirectional and bidirectional modes. In bidirectional mode traffic can be sent in both directions. This is the most commonly used communication for two entities. Unidirectional mode means that traffic is only sent in one direction, as for example in a broadcast application. In addition to the direction of flows, the symmetry of the flow must be established. Some applications require flows to be bidirectional and symmetric, which means that traffic flows are sent at the same rate in both directions. Other applications need bidirectional asymmetric flows, for example in a client-server case

```
2010-12-24 15:15:25.430  465 TCP  123.123.12.123: 1234 -> 99.987.54.67:25   8484     7.8 M     1
2010-12-24 15:15:25.430  465 TCP  123.123.12.123: 1234 -> 75.642.53.42:25   8484     7.8 M     1
2010-12-24 15:15:25.432  465 TCP  123.123.12.123: 1234 -> 96.63.31.123:25   8484     7.8 M     1
...
2010-12-24 15:15:25.435  465 TCP  123.123.12.123: 1234 -> 75.642.31.20:25   8484     7.8 M     1
2010-12-24 15:15:26.000  517 TCP  158.269.40.199:80  -> 917.836.14.225:128  6499     8.6 M     1
2010-12-24 15:15:26.000  389 TCP  154.65.123.234:443 -> 97.222.111.136:108  5276     7.4 M     1
2010-12-24 15:15:26.005  741 TCP  140.50.160.170:26038 -> 123.456.789.8:216 4648     6.1 M     1
2010-12-24 15:15:26.010  734 UDP   321.654.987.123:53-> 151.41.71.81:327      420     7.2M      1
2010-12-24 15:15:26.010  591 TCP  136.257.369.33:80   ->123.654.987.789:539 1838     1.0 M     1
2010-12-24 15:15:26.442  465 TCP  123.123.12.123: 1234 -> 152.263.374.12:25   8484   7.8 M     1
2010-12-24 15:15:26.450  465 TCP  123.123.12.123: 1234 -> 163.301.444.148:25   8484 7.8 M     1
```

Figure 2.1: Examples for Netflow records

[98], where a client sends small queries that requests large answers (e.g. a document). In the case of a broadcasting application, the flow is unidirectional and asymmetric, since no answers are requested.

2.1.2 How to collect IP flows

The process of collecting IP flows from the network is illustrated in Figure 2.2. The monitoring of IP flows involves two steps, *export* using metering tools and *collection* of the flows. On the border of a network, routers with flow export functionality (exporters) are placed to tap network traffic. The *exporter* monitors packet headers on the monitoring interface and timestamps are set for each header. Depending on the use of the flows, they may be filtered or sampled. The update function generates a new flow entry for a packet header if no matching can be found. These steps are called metering [24, 25]. Expired flow records are transmitted to the *collector* which can parse, compress and store the records. Referring to [24, 25], a flow is considered expired if

- the flow-end has been received,
- the flow is inactive for a certain time,
- a long-lasting flow has been running for longer than a defined timeout,
- or if there are problems in the Exporter.

From the collector, flows can be stored or used for further monitoring purpose or for analysis.

Several network solutions are for flow capture available on the market and the top leaders in this area also provide their own flow formats. For example Cisco[1] developed `NetFlow` the most popular IP flow format. `sflow` [104] is mainly used by D-Link[2] or HP[3]. Smaller players in network monitoring have also introduced their own formats, for example `Jflow` or `cflowd` by Juniper[4] or `NetStream` by Huawei Technology[5] to cite only a few. In the open-souce domain there are also many tools available, such as `nfdump`[6] or `Softflowd`[7].

[1] `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`
[2] `http://www.dlink.com`
[3] `http://www8.hp.com/us/en/software/software-solution.html?compURI=tcm:245-936973`
[4] `http://www.juniper.net`
[5] `http://www.huawei.com/en/`
[6] `http://nfdump.sourceforge.net/`
[7] `http://www.mindrot.org/projects/softflowd/`

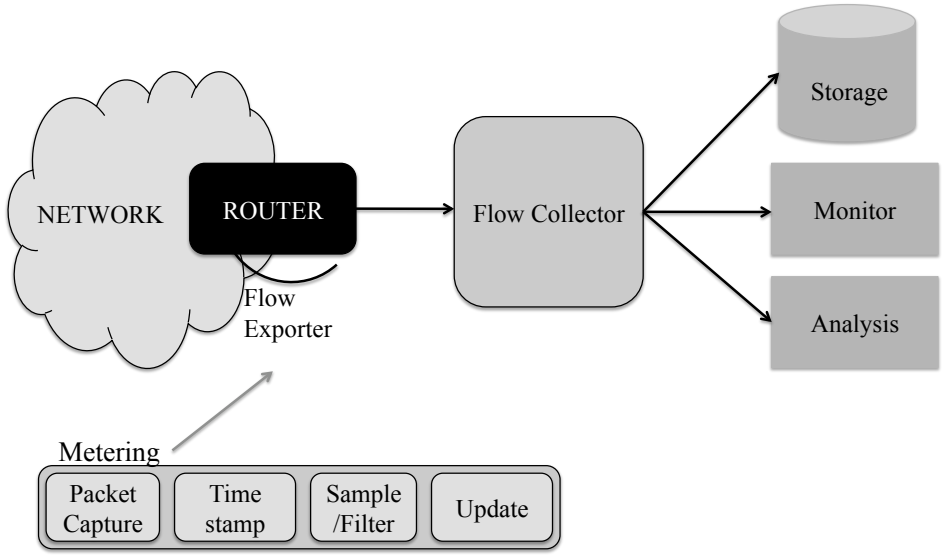

Figure 2.2: Architecture for export and collection of IP flows [25, 24]

2.1.3 A short historical summary: from Netflow records to IPFIX

In 1996, Cisco designed NetFlow as a layer 3 switching path speedup, where its router and switch caches were re-organized on the basis of per-flow information. This had the side-effect of evolving into a caching technique for the collection of flow information and statistics without penalizing overall performance [157, 89]. The currently used version of Netflow records is NetFlow version 9.

> Previously released versions were V1— the original version, V5 — including Border Gateway Protocol (BGP) information and sequence numbers, V7— supports Cisco Catalyst series switches in hybrid or native mode and V8— supporting aggregation of Cisco IOS router platforms.

A new feature in NetFlow version 9 is that it is template-based, making it more flexible and providing new benefits, such as the export of any information from routers and quick implementation of new features. Furthermore, NetFlow version 9 serves as standard for the IETF work IPFIX and PSAMP (Pack Sampling) working group.

The mission of the Internet Engineering Task Force, IETF [2], is to enhance the work and process quality in Internet technologies. The main activity of the IETF is to publish highly relevant documents for the management, design and use of the Internet and its related technologies. The IETF combines research and engineering to produce new standards, which embody best practice for a given domain of research.

The IETF IPFIX working group was created in 2002 to specify a universal standard for the export of IP flow information from all vendors' routing devices. The main argument for this was the need for a universal standard protocol that can be used for measurement, management and accounting processes for flows. This protocol standard defines the format and transfer for IP flow information.

NetFlow version 9 (RFC3954) serves as the basis of the new IPFIX standard, but is more flexible and allows new features, such as the Stream Control Transmission Protocol on Layer 4, and template use. Templates make the collecting process more flexible, since templates enhance IPFIX messages to be processed without interpretation. In [25] a template is defined as:

> *A Template is an ordered sequence of <type, length> pairs used to completely specify the structure and semantics of a particular set of information that needs to be communicated from an IPFIX Device to a Collector. Each Template is uniquely identifiable by means of a Template ID.*

By introducing templates, more information can be provided to the Collector than with Netflow records alone, due to the options afforded by the template format. For more detailed information about Netflow and IPFIX, the reader is referred to the corresponding RFCs, RFC 3954 [24] and RFC 5101 [25].

The IETF IPFIX working group submitted the IPFIX Specification to the Internet Engineering Steering Group (IESG) for validation in 2006 and recent advances discussed in active and pending IPFIX working group documents are published on the IETF website for IPFIX[8].

2.1.4 Major issues with Netflow records

The monitoring of large ISP networks often faces a wide range of problems. The main activity of network monitoring consists in the detection of unusual and malicious events of many different types, be they attacks from outside or network problems inside. Monitoring is required in order to take appropriate countermeasures. From a technical perspective, the only potentially available network traffic information is Netflow records, since the storage of full packet captures is impossible. Even storing and analyzing all Netflow records is a difficult task, because of the huge data volumes (i.e. there might be 60,000 flows/second) on the network and in near future this problem will become even worse (see Figure 1.2).

A solution to this problem is to use efficient storage methods or to apply strong compression schemes such that more data can be stored, like presented in [50, 91]. For compressing Netflow records, one popular approach is sampling used for example in [101, 36], but this discards data, so an alternative technique is aggregation, where data is simply compressed into other data, as presented in [68]. The use of IP flows and their storage is a crucial problem, since Netflow records carry sensitive information about the communication of two entities. This is a major issue since privacy should be respected, so anonymization techniques are quite often used to mask sensitive data.

Besides the sensitive data in IP flows, concerning who is communicating with whom, the data also includes information about what happened on the network. This information is relevant to detect dependencies and relations between flows and applications [7, 125, 66, 70]. Besides their use in distinguishing dependencies between different kinds of IP flow information, an important additional feature is that Netflow records can be used for the identification of anomalies in network traffic.

2.2 Network anomalies

A major threat for network architectures is the various incidents which may occur. This thesis distinguishes between two types of anomaly: technical anomalies and attacks.

2.2.1 Technical anomalies

From the network management perspective, technical problems may occur at a number of levels. Anomalies can occur at the

- Hardware level (switch, router, firewall, etc.)
- Software and protocol level (program error, overflow, ...)
- Host level (machine crash, not responding, cable unplugged,...)

[8] `http://tools.ietf.org/wg/ipfix/`

As well as being able to identify technical issues, the analysis of IP flow information can also identify attacks on the network infrastructure. The type of attack can be anything from simple spam to highly destructive attacks, such as physical attacks or denial-of-service.

2.2.2 Categories of attack

Before presenting different categories of attack, an attack has to be defined. In [3] from 1980, a *threat* is defined as,

> *the potential possibility of a deliberate unauthorized attempt to access information, manipulate information and to render a system unreliable or unusable*

and an *attack* as

> *a specific formulation or execution of a plan to carry out a threat.*

Ten years later in [59], a similar definition is introduced for an intrusion or attack, which can be defined as:

> *Any set of actions that attempt to compromise the*
>
> - *Integrity,*
> - *Confidentiality or*
> - *Availability*
>
> *of a resource.*

In literature [57, 62, 127, 122], there are different taxonomies for attacks available. The authors describe several attacks, the most prominent being:

- Worms — programs propagating by self-replication: very fast propagation through the network, by means of mass mailing or network-aware worms
- Confidential information gathering such as scanning/sniffing activities — information is gathered by means of sniffing or scanning the system for information that can be used to attack further targets.
- Network attacks — this group includes attacks related to session hijacking or spoofing, etc.
- (Distributed) Denial of Service (D)DoS — Attack that makes a service unavailable or reduces the level of service available for users
- Botnets — Groups of machines infected with malicious programs that propagate content without the owner knowing, in order to carry out spam or DoS attacks.

These taxonomies cannot be seen as individual classes for malicious activities, but more as a simple summary of malicious resources. For example, a worm is malicious code in a useful program that can perform actions like participating in a (Distributed) Denial of Service, stealing confidential information, downloading files, etc. This example makes it clear that the previous taxonomy is only a summary of malicious resources and that one class may include another.

In addition to determining categories of attacks, two classes of intrusion have to be distinguished, *network-based* and *host-based*. According to [3, 122], a *host-based* intrusion targets only specific computers within an architecture. In general, this kind of attack is intended to consume resources or to crash the host or the applications it runs. *Network-based* intrusions target whole network architectures of computers in order to prevent networks from providing services by degrading network connectivity and consuming all available bandwidth.

The papers presented in the following chapters are only a small selection of papers considered relevant for a particular research domain. There are of course many other significant papers related to the highlighted topics, but due to space restrictions only a few were selected in this thesis.

Chapter 3

From IP flow monitoring to dependency identification

Collecting network data is essential for good management and surveillance of a network. With the increasing size of networks, collection and storage of all network data becomes infeasible. For a network operator, 60,000 Netflow records per second are common at peak times. In order to overcome this problem, sampling or aggregation of network data is used. Sampling [36, 101, 34, 60, 109] is based on the selection of a subset of all network data whereas in Aggregation [61, 68, 125], many small data items are assembled into fewer larger items. Besides the aspects of storing data, dependencies between applications and network components can be detected by analyzing flow data. Another application area for IP flow data is intrusion detection and anomaly detection. This chapter illustrates some state of the art methods for these different research topics.

3.1 Sampling methods

In sampling and aggregation approaches, it has to be distinguished between two kinds of sampling: *Flow* and *Packet*-Sampling. In *Packet*-sampling, each individual packet is sampled with a given probability. In *Flow*-sampling, packets are sampled on the basis of flow granularity.

As presented in [60, 109, 21, 101, 127] and others, the sampling of packets, it can be divided into *systematic* and *random* sampling. *Systematic* packet sampling, as presented by Cisco [23], became popular due to its simplicity and low overhead. Systematic sampling is explained in [21] as a method that is based on a 1-out-of-k approach, with a random selection of one element out of the first k elements followed by every k^{th} element thereafter, using a random number generator and a counter. Popular systematic approaches can be event-driven, as in [109, 119] or time-driven, e.g. [21]. *Random* sampling uses randomly selected packets, where the sampling probability is fixed or uses an n element selection, e.g. [22].

In [36], Adaptive NetFlow, a new method for sampling is presented, using for example an optional Flow counting extension or adaptive sampling rates. This approach introduces measures for Netflow that overcome some of its drawbacks, for example by using adaptive linear sampling to optimize memory use. At the beginning a large sampling rate is fixed, but this will self-adapt if, an overflow occurs. This allows Adaptive NetfFlow to stay below a fixed memory usage and remains easily configurable since statistical sampling rates are no longer necessary. [60] refers to non-linear sampling rates for passive measurement. The sampling rate adjusts its value by observing the number of packets that have been counted for a given flow. The method uses high sampling rates for small flows and low sampling rates for large flows. Consequently, the flow size distribution has only a low impact on flow estimation accuracy. Another size-based sampling approach is presented in [119], where

class-based sampling is performed. A new data structure is introduced, composite Bloom-Filters, which filter traffic, allocated it to different classes and optimize memory-usage.

In [38], an approach called *Sample and Hold* is presented. This approach is dependent of the packets that are sampled at the router by performing a cache lookup for each packet. If a packet matches, the flow statistics are updated; otherwise a new cache entry is generated with a certain probability for a given byte size.

[34] proposes a dynamic control mechanism for sampling that utilizes a threshold-based sampling technique. This technique relies on size-dependent sampling and estimates the sum of sample sizes in bytes in order to assess the relative accuracy of sampling. [109] presents *FlexSample*, a framework that performs per-packet sampling based on the membership of packets in sub-populations previously fixed by an operator. This allows traffic to be sampled over towards a certain class of subpopulations.

Another kind of approach, called Sampled NetFlow, is presented in [125], which presents a study of the accuracy and information loss for Netflow records and packet traces. The authors of [125] use a systematic sampling of 1-in-250 for Netflow records. In addition to this, they captured all packets passively. This allowed them to evaluate the accuracy of Netflow records. They found in [125] that bandwidth usage grows linearly with the number of flows and that the overall performance results of systematic and random sampling are similar.

To conclude this section about sampling, it can be said that the most important points in sampling are:

- Optimization of memory usage and bandwidth consumption
- Definition of an optimal sampling rate (threshold, random, class-based,....)
- Accurate flow estimation

3.2 Aggregation methods

Aggregation methods for flow records are based on an approach where small flow entries are assembled into larger entries. Aggregation can be based on flow record entries or on packets, as in [68, 61, 125] for example. A major advantage of aggregation is that data is assembled into other data, but is not lost. In [61, 68], two different methods for aggregation of flows are presented, the first being based on cluster aggregation, and the second on prefix-tree aggregation.

In [61], a new method for the adaptive aggregation of flows is presented, where traffic clusters are identified in real-time. A cluster is represented by a 5-tuple having different combinations of flow information, source/destination IP address, source/destination ports and number of flow entries.

Figure 3.1 shows the cluster aggregation used by the authors of [61]. For example the largest cluster (cluster A) holds records having the same source IP address. Inside these clusters, smaller clusters can be observed holding different combinations of the 5-tuple information. By aggregating information from different clusters, priorities can be given, the resolution loss be minimized and the overall memory requirement be optimized by using a 2-dimensional hash-table. A similar method, but for post-processing, has been implemented in [37], where traffic is grouped into multi-dimensional clusters. The aim of [37], as in [61] is to identify clusters with the greatest similarity.

An aggregation method that is applied in this thesis relies on prefix-tree aggregation as presented in [68, 20]. This is a packet-based aggregation method that relies on the approach to reduce entries over space and time. Aguri [68, 20] is a flow monitoring tool that also allows aggregation-based traffic profiling in near real-time. Aguri can profile traffic by referring to prefix-based trees which continuously generate summaries of the monitored network traffic. An advantage is that Aguri supports both IPv4 and IPv6 traffic.

To generate time-window based trees, packets sharing the longest common IP prefix for source and destination are matched by using a pre-order traversal of the tree. Patricia trees

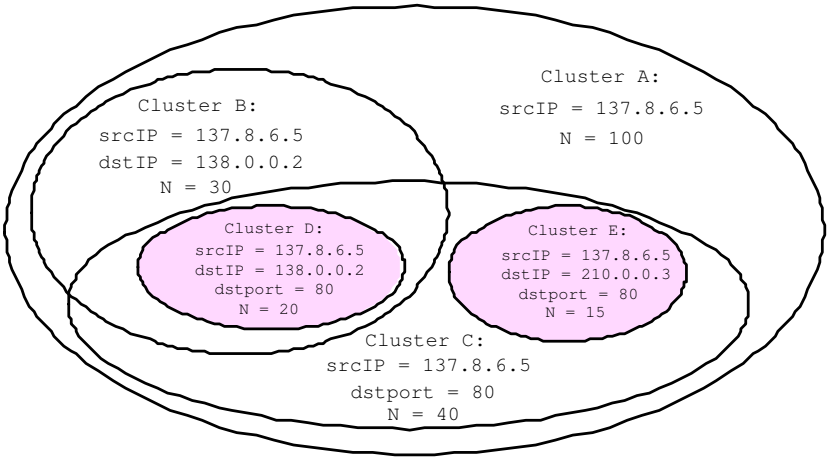

Figure 3.1: Aggregation cluster in [61]

[92], fully binary radix trees with a fixed tree size are used for the creation and update processes. An advantage of Patricia trees is that memory use and the search time for keys having variable length are limited. At the end of a time period, trees are optimized by post-order traversal, where smaller nodes are assembled into their parents such that they represent IP subnets with smaller prefix sizes.

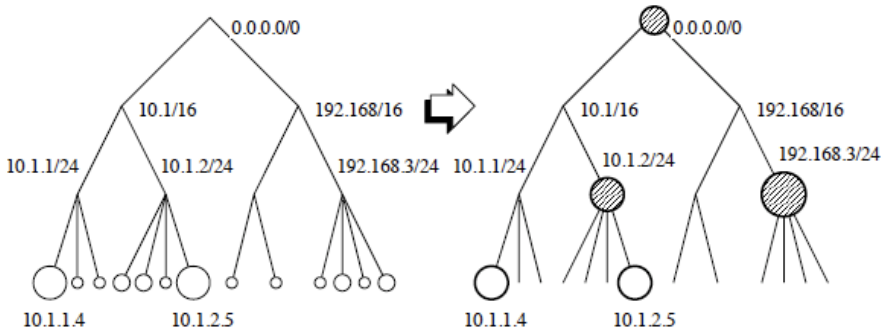

Figure 3.2: Aggregation process in [68]

Figure 3.2 shows, how the entries are aggregated into larger ones. To evaluate the monitored traffic, four separate profiles are established: a profile for source addresses, destination addresses, source protocols and destination protocols. To improve the accuracy of the algorithm in real-time operation, a variant of the Least-Recently-used replacement policy is applied. The different profiles reflect key information such as, the source address profile, which can identify popular ftp[1] servers, whereas the destination profile is more relevant to the client side. Aguri can be used for different tasks like monitoring data, analyzing previously captured data, etc. The major finding of [68, 20] is that the algorithm runs in real-time with $O(log(n))$ and that it can generate summaries for time-windows by aggregation of traffic over small time-windows.

[1]File transfer Protocol [105]

3.3 Storage of IP flow records

An efficient storage technique is essential for further processing of network information. Such a system should have fast data access methods, provide high insertion rates and have dependable storage methods. Referring to [40], it is distinguished between *row-* and *column-*based techniques, where row-based techniques are considered *write*-optimized such that high write performance can be obtained. Column-oriented systems are *read*-optimized, meaning that, for a given query, only the necessary columns are read, making the methods optimized for fast lookups. A recent trend is to use column-oriented storage systems and several are presented in this chapter.

A new method, NetStore, is presented in [50]. It is an efficient storage infrastructure for large quantities of network flow data. It uses a column-oriented approach to reduce storage requirements, query processing and data load times. Column orientation means that data is stored in columns and not in rows: a column is used for each attribute, where an attribute is an element of the 5-tuple of a flow record.

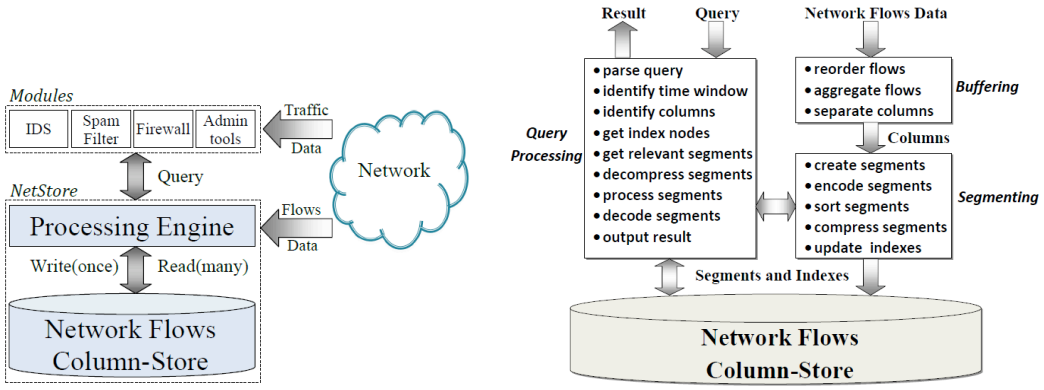

Figure 3.3: Main modules of NetStore (left) and processing steps (right) [50]

This kind of structure allows faster compression and access. To enable physical storage of data and flexible compression, columns are subdivided into fixed segments, also providing a finer granularity for traffic capture. These segments are compressed individually since the traffic distribution (over working hours, weekends,...) is not uniform and a variety of compression techniques are used, for example variable byte encoding or even no compression at all. The left of Figure 3.3 shows the modules of NetStore, while on the right illustrates the process steps of the approach. [50] suggests the use of aggregated data to avoid bias resulting from micro-variations in traffic. NetStore takes its inspiration from Aguri [68, 67] which is also able to manage large quantities of IP flow related data by using spatial and temporal aggregation. The design of NetStore allows to outperform simple row-based systems in terms of compression and fast lookup of queries.

In earlier work, for example [40], a hybrid architecture is presented. This approach uses a column-oriented physical storage structure that incorporates a row-based system for query processing. Indexed and sorted data is divided into column sets, which are known as projections. This step is similar to [50]. C-Store aims to lower disk accesses per request and to save disk space by storing each column separately. Light-weight compression is used to improve the read performance. [48], another column-oriented storage system for large-scale data, performs high-speed storage and data querying of flow records, by using on-the-fly compression and indexing. This real-time compression technique uses ComPAX (compressed adaptive index), an on-line compressed column principle and a bitmap to designate a column.

This approach is based on a codebook of words, which can reduce the bitmap index size by performing logical operations on these bitmaps. A preprocessing step can be used in order to sort flows by using a locality-sensitive hash table in order to improve compression, similar to that of gzip[2] .

A scalable storage system for IP flow records is presented in [90]. The authors aim to prove that P2P can be used to improve storage scalability and increase query performance through decentralization. DIPStorage is a platform where both processes and the storage of IP flow records are distributed/shared across multiple nodes. The main module in DIPStorage is a distributed hash table that is responsible for storing flow record subsets. To store the data, a *tank-based* view is used: a group of peers stores the network data by referring to specific conditions, for example by using hash functions. To optimize the process, multi-

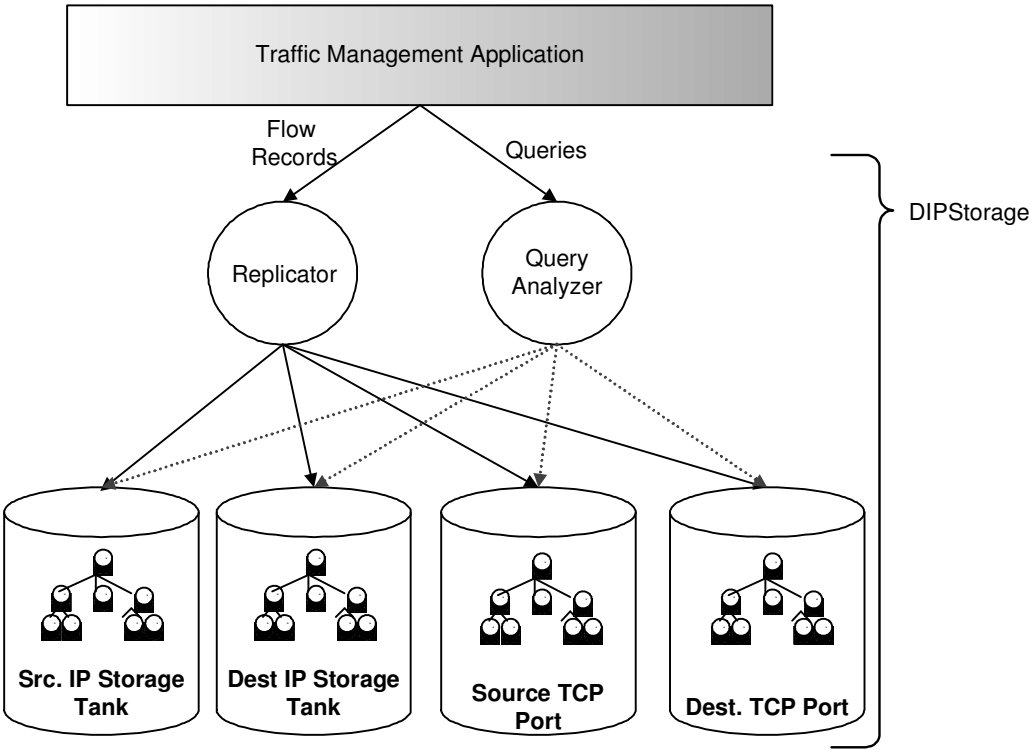

Figure 3.4: Multi-tank view storage process in DIPStorage [90]

tanks are used under different rules in order to search attributes of queries. Figure 3.4 shows the the multi-tank view for the different attributes of IP flow records. The query analyzer module handles the queries and tries to make an optimal decision for answering the query by forwarding the query to a particular tank. The flow replicator module distributes the flows across the tanks. P2P-based approaches are eminently feasible nowadays and allow storage to be increased simply by adding new nodes to the platform. This has the additional advantage of providing more computational resources for processing.

[2]gzip: software for file compression [31]

3.4 Dependencies in IP flow information

Knowledge of interactions between network flow records is essential for evaluation purposes. One means of evaluation is to dig into IP records in order to detect dependencies between flows, because IP-related flows and Netflow records can be used for the detection of failures or anomalous events in a network. This section presents the state of the art in relevant techniques for the identification of dependencies between flow information and also dependencies between applications or network components. Much research has been undertaken over recent years, and can be divided into groups, discussed in the subsections that follow.

3.4.1 Probability-based models

In this section probabilistic models are presented, which use a variety of techniques to detect failures in networks by analyzing dependencies. In [7], inference graphs are used to exploit multi-level dependencies for fast, accurate problem localization. The paper describes a performance tool, Sherlock, used for the detection of faults and performance problems. The Inference graph itself is used to display dependencies between all components, while the Ferret algorithm, a ranking algorithm using scoring functions, generates the necessary dependency vectors. The inference graph shows the different dependencies in the IT infrastructure. The dependencies are estimated by the Ferret algorithm, which calculates the dependency probabilities that explain the observations. The chosen inference graph provides

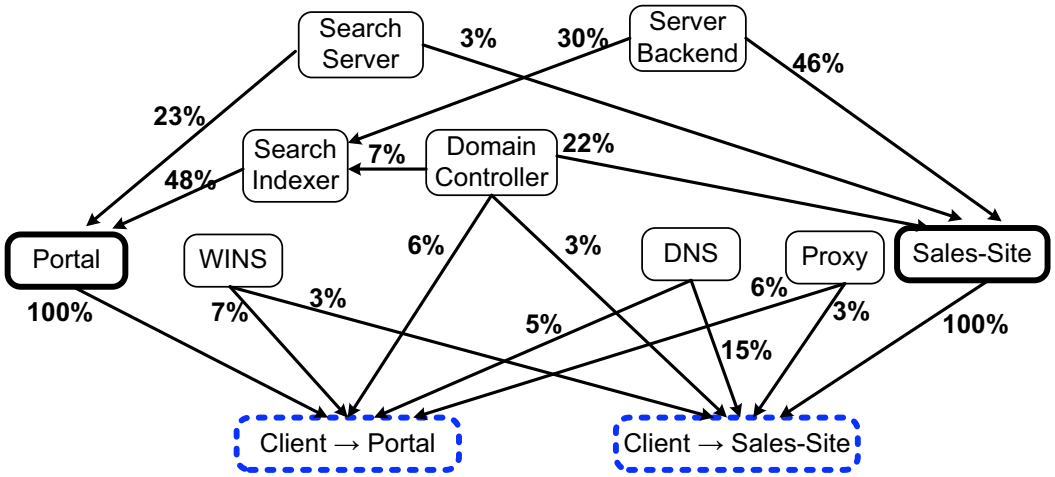

Figure 3.5: Dependency graph in [7]

the input for the Ferret algorithm, which calculates the vectors and applies the established confidence levels to output a list of ranked vectors. Sherlock is a centralized inference engine with distributed agents. It is responsible for the creation of the service-level dependency graphs. The agents monitor traffic sent/received by one/more hosts and calculate for each service the dependencies and response-times. Data aggregation provides the service-level dependency graph. The final step is to combine this service-level dependency graph with the IT topology in an inference graph. The service-level dependency graph for two web portals is presented In Figure 3.5. Arrows indicate the main services, and the edges are weighted with the dependency probabilities. The observations are three-fold. First, the service dependencies vary considerably, some servers redirect requests, while others handle them locally. Second, even where service dependencies seem very similar, it has been proven that the dependency strength is not identical. As a last argument the authors can prove in [7] that service dependencies vary over time. In summary, [7] introduces a complex tool for heterogenous network components that is able to catch failures and service problems by

using a multi-level probabilistic model.

In another work, [6], Leslie graphs represent dependencies of components in networks at different granularities, for example at the granularity of IP-addresses, hosts, etc. Leslie graphs are used for fault localization, reconfiguration planning, helpdesk optimization and anomaly detection. The aim is to present a generic representation of the web of dependencies. Another module, called the Constellation module, monitors packets and maps them onto Leslie graphs, while the AND module extends these graphs by adding some additional data, such as layer 3 trace route probabilities. Dependencies are derived by passively monitoring network traffic, activity patterns being deduced by the Constellation and AND tool with the Leslie graphs being approximated on demand afterwards. The outcomes are twofold. First, network topologies can be discovered and second, dependencies can be estimated through the use of Leslie graphs. This method is also feasible in larger networks, but has the disadvantage of needing full packet traces.

[116] presents a new model for failure diagnosis in IP networks. Shrink is a top-down approach for network diagnosis, using Bayesian networks. By modifying the Bayesian network, diagnosis time and accuracy can be improved. The main aim is to build a Bayesian network with data and apply an inference model to it in order to detect failures based on probabilities. The idea works well for link failures, but it has not yet been proven that it also works for TCP connections.

The aim in [112] is to design a tool to overcome bottlenecks in systems by using black-box nodes. Distributed systems are represented as a graph of communicating nodes with a causal path, a series of node traversals caused by external system requests. The tool finds high-impact causal path patterns and identifies most the nodes which add the most latency to the pattern. The method includes three main steps, first, online discovery and tracing of patterns of communication; second, inferring causal paths and patterns offline; and finally visualization of the results. The trace contains at least three parameters (time-stamp, sender, receiver) for each message. The method uses the "nesting algorithm" and the "convolution algorithm". The tool shows promising results, even if the technique is complex, due to its use of traces that are converted into time signals.

[113] describes a method to avoid bottlenecks by caused delays in distributed systems. The model is divided into four stages: the first traces the socket API calls per application of each host, which the second reconciles into single trace with one message per machine (time-stamp and clock of sender receiver are parameters). The third step performs a causality analysis leading to final presentation of the results as trees or timelines. The paper sets out a new causal path model and the corresponding algorithm. The results are estimated by finding and scoring parent messages represented in tree form with probabilities. The problem is that capturing the traces for sender and receiver can easily become complex with multi-homed hosts.

Paper [9] discusses a solution for automatically identifying traces of dependent messages among web services. The system is based on correlations between messages that are exchanged between services. Message dependencies are discovered through causal dependencies which result from a probabilistic dependency graph, with the aim of representing the final results in form of a pruned graph or path.

3.4.2 Rule-based and correlation-based models

Another approach in this field is to use rule-based or correlation-based models for the analysis of dependencies in traffic information.

In [163], interaction between central system binaries is used to create dependency graphs. The paper distinguishes between two kinds of dependencies, data and call dependencies, where call dependencies are calls between functions and sites and data dependencies represent the definition and use of values. The method relies on "network analysis" techniques known for human networks by describing the network neighbourhood just as in social contexts. Correlations are established by statistical methods through the use of regression. By splitting the data set into training and test sessions, [163] the paper aims to predict defects

by applying network measures on dependency graphs. Network measures on dependency graphs have been shown to be effective in finding defects and perform better than more complex measures.

In [117], a rule mining technique is applied to time series of traffic to extract communication patterns from traces, without focussing on any particular trace. The hypothesis is that groups of flows occurring together are likely to be dependent. Traces are portioned out into time windows which are then analyzed. Candidate rules are generated and are those that remain following an information theoretic signal processing step. By applying the generated

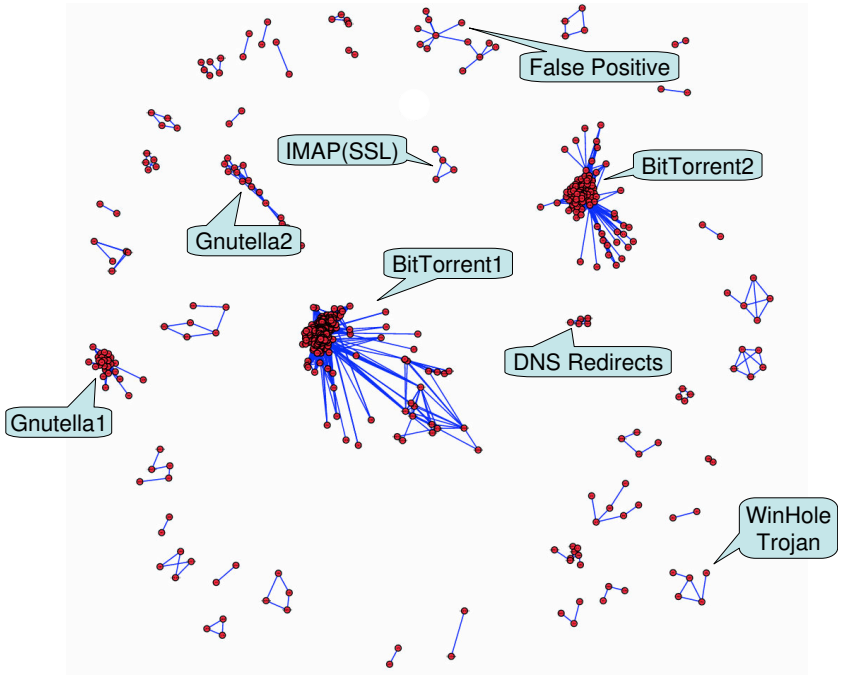

Figure 3.6: eXpose output from generated rules [117]

rules, it is possible to detect dependencies among flow groups and applications. An example is given in Figure 3.6, where the rules learnt from the activities are represented. Here, nodes in the figure are flow activities and edges the interaction of the activities involved in rules. The results of the eXpose technique provide some interesting insights into dependencies between traces.

[69] presents a semi-automated method to mine large sets of network connection logs and extract the apparent structure of application sessions in those connections. It processes connection-level traffic traces and operates in two stages: first, session extraction and second, structure abstraction. Session extraction is a statistical algorithm for reducing streams of connections to a stream of sessions. The paper notes that it is limited to the capture of a pair of hosts but could be generalized for multiple hosts. Structure abstraction relies on reduction/generalization rules to abstract the sessions and to provide insights about them. A session can be singleton (itself), homogenous (consecutive invocations of same applications) or mixed (involving different types of connection). The session descriptors for the abstraction structure are based on regular expressions and are used to derive a graphic visualization, by applying the tool GraphViz [4].

In [74], a dependency discovery algorithm for Netflow data is introduced. This algorithm detects correlations between flow events by using a semantic table. By applying rule mining, a dependency is composed from four association rules based on semantics set out in the table. A major advantage of this method is that the semantic classes can also explore the type of an identified dependency. This method provides an accurate tool for business and support

decisions.

[30] introduces a new approach for the discovery component and network dependencies, by applying IP flow information attributes to a flexible inference mechanism. A major characteristic in their model is that they use start/end timestamps in Netflow time series in order to detect whether one Netflow triggers another one. The inference mechanism uses timestamps and confidence correlations to determine the degree of each dependency.

3.4.3 Social behaviour-based models

Examining to the "social behaviour" of hosts is another approach to determining dependencies in Network traffic. The premise of BLINC, described in [70], is to classify traffic flows by application, where internet hosts are associated with particular applications. The process is operating without foreknowledge: the behaviour of hosts is observed on several levels and an evaluation is made by referring to statistical methods for classification. The classification has different levels: social, functional and application levels. The difference between those levels is the way they capture traffic and associate an application with a host. The method how the traffic flows are classified is interesting. The method differentiates

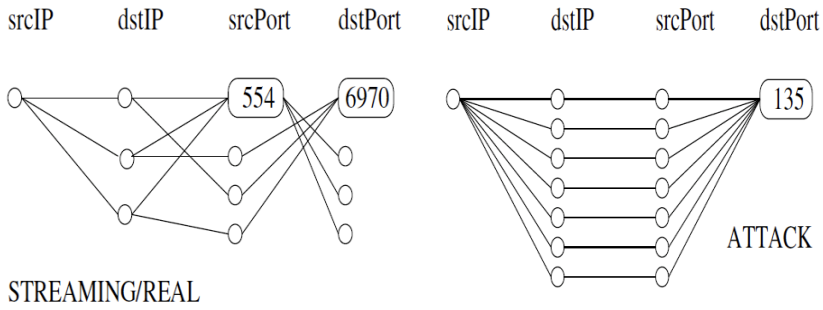

Figure 3.7: Dependencies in BLINC [70]

between social, functional and application levels it is possible to detect anomalies only by the different classification levels. For example by looking at the social level classification, it is possible to detect the behaviour of a host, whether it is using web, peer-to-peer, etc.If this is the case, it can be observed whether the host participates in communities, having the same group of destination IPs, which means that the probability is quite high that the host is participating in gaming or peer-to-peer activities. Two examples are given in Figure 3.7. The right part of the Figure shows an attack, where a malign host scans on the active address space in order to find vulnerabilities at port level. The left part of the Figure 3.7 illustrates streaming and its interactions. Additionally, findings in [70] show that if, a special type of community called clique is detected, it is highly probable that malware activity, caused for example by a worm, is taking place. At the functional level, it can be observed if a host is either a service provider or a consumer. The application level then can use these functional levels to associate applications to hosts by the associate flows. Heuristics are used to refine the classification by means of these "graphlets", which represent the behaviour of an application.

Another hypothesis, also relying on social behaviour of hosts is described in [63], which introduces a new type of graph for monitoring, analyzing and visualizing network traffic flows. The main difference from BLINC is that here, not only the host level is considered. Monitoring host interactions by applying traffic dispersion graphs (TDGs) derived from Netflow records can detect abnormal phenomena like legacy port abuse or malicious network activity. A TDG is a graphical representation of the interaction between nodes. New in this approach is that traffic visualization is not done by volume on a per flow basis, but instead represented at different levels like packet, flow and host level. The TDGs

themselves represent the interaction pattern. The main goal of this new graph type is to represent network-wide social behaviour and to distinguish the nature of applications through vizualization in TDG form.

[1] presents a method for detecting communities of interest by examining the popularity of target hosts and the frequency of their interactions with peers. From the behaviour of the communities detected, the authors aim to detect, whether there is a defect or anomaly.

3.5 Approaches to Netflow record analysis and evaluation

A recent trend in the evaluation of IP flow data relies on intelligent evaluation methods developed in other domains like bioinformatics, natural language processing etc., for example by applying machine learning approaches.

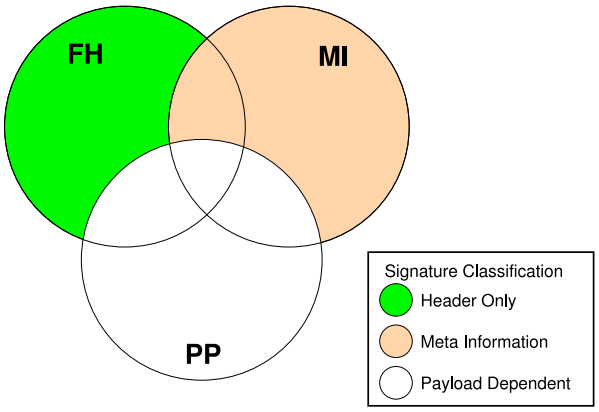

Figure 3.8: Relationship between predicates and packet rule classification [33]

The first machine learning approach is the rule-based method, as presented in [33]. The authors present a machine learning approach that learns flow rules from packet signatures. Rules are defined as sets of predicates, which are extracted from flow and packet data. These predicates are flow-header (FH), packet payload (PP) and meta-information (MI). Disjoint classes for rules are established for example for *header only rules* or *meta-information* only rules. Figure 3.8 shows the classification of the packet rules and predicate relationships. A machine learning approach is applied, called Adaboost [47] which is a meta-algorithm that can be used with other learning approaches to improve accuracy. The classifier can raise alerts on Snort [114] on detecting an anomalous event in IP flow records.

Another work [11] has an approach based on association rules for extracting anomalies from backbone networks. The authors refer to meta-data from histogram-based detectors and a machine learning approach to classify anomalous events occurring during a time interval.

The histogram detectors record extracted features and provide relevant meta-data from flows, for example, port, IP address, etc. These histograms are binned and the similarities for the given time interval calculated. The motivation in [11] for association rules is that suspicious data has similar characteristics. The Apriori algorithm is used for the generation of the association rule mining. This algorithm can be used for the generation of association rules since it discovers frequent item-sets in data sets.

Machine learning take advantage of kernel methods, strong mathematical tools, which have found their utility in the evaluation of large and complex data sets. Kernel methods were introduced as a mathematical tool in early the 1900's by Hilbert, and were first applied

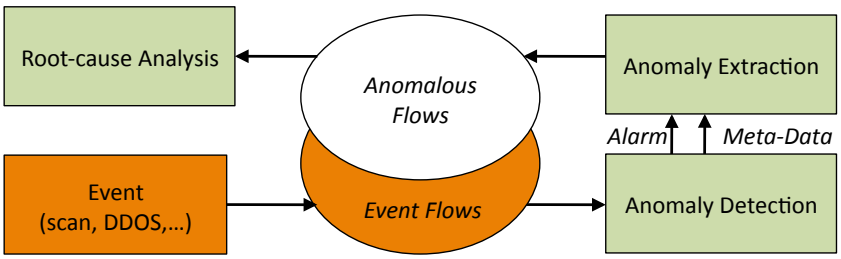

Figure 3.9: Anomaly detection with histogram-detectors and association rules [11]

as an element of machine learning techniques in the late 1990's by Vapnik [137], then further developed in the early 2000's by Schölkopf et al. [121]. In the recent past, kernel functions have been applied in many domains, e.g. Bioinformatics [139], Natural Language Processing [26], and can be applied to many different input formats, whether structured (e.g. graphs, trees) or unstructured (e.g. text).

There is a great deal of literature concerning the application of support vector machines (SVMs) in many disciplines. For example [28] is a work based on kernel methods. A similarity measurement is calculated for input trees consisting of parsed and pre-processed sentences from news reports. These are then decomposed into tree-like structures. Support Vector Machines (SVMs) with tree kernel functions are then applied to classify sequences according to their similarity values. Another work that refers to SVMs is presented in [115]. Here, SVMs are used to classify Internet traffic, into different categories of applications by extracting parameters from packet headers. Different kernels for the SVMs are evaluated to obtain the best classification results. A major disadvantage of using these supervised learning algorithms is that that they require (labelled) training data.

For the classification of flows, another approach is to use Clustering, as presented in [86]. Different parameters from header traces resulting from various applications are extracted and applied to the Expectation-Maximization (EM) algorithm, such that data is classified into different clusters. The EM-algorithm belongs to the soft clustering algorithm class, where one data point can be attributed to several clusters, depending on the probability density distribution. With aid of this algorithm, data is clustered by application type and the evaluation shows that good results can be achieved for most, but not all traffic types.

In [65], Netflow-level statistics are evaluated by using the Naive Bayes classifier. The aim of applying this machine learning algorithm to Netflow records is to provide a tool for the classification of Netflow records according to the corresponding applications. In [71], classifiers such as Naive Bayes, k-Nearest Neighbours (KNN) and SVMs are applied to statistics from flows in order to detect anomalous traffic sources. Another approach is presented in [80] which analyzes Botnet traffic. By extracting parameters such as IP address, port, TCP flags, etc., from packet headers. After cleaning the flow samples, machine learning algorithms as Naive Bayes, decision trees (C4.5) and Bayesian networks are applied to classify into normal and botnet traffic. [156] reports a performance comparison of five classification algorithms for the evaluating IP flow traffic. The evaluated algorithm are the most poplar ones: C4.5, Naive Bayes, Bayesian networks and Naive Bayes Tree algorithm. The results of this evaluation are that the classification accuracy is similar for all algorithms, but there are significant variations in performance.

Semi-supervised learning approaches focus on data sets with only a small amount of labelled data and many unlabelled data samples. Methods such as k-means [58], a traditional clustering method, assume that adjacent data samples tend to have similar labels with the result that they propagate their labels to unlabelled data samples and so, cannot detect new types of attack.

In [161], the authors present a new label propagation algorithm targeted at classification. The algorithm is based on the idea of the semi-supervised clustering algorithm. A schematic

description of the label propagation algorithm is presented in Chapter 8, Figure 8.4. The main idea is to construct a fully-connected graph, in which only some nodes are labelled. Each label represents the class name of the node. The edges between two nodes have associated weights, which depend on the distance between two nodes and are controlled by the weight factor σ adjusts the label-propagation quality. All labels are propagated iteratively to unlabelled regions and nodes are either labelled or unlabelled data samples. Unlabelled nodes are assigned iteratively estimated class-fellowship probabilities that give the probability that an item is associated with a class. At the end of the iterations, each unlabelled node is allocated to the most probable class, that is the class with the highest class-fellowship score. Further relevant work related to semi-supervised learning is presented in [35]. The authors present a new semi-supervised algorithm which classifies information extracted from full packet traces into different types and applications. A major advantage of using semi-supervised algorithms is that both known and unknown anomalies can be detected, as this kind of algorithm can use either labelled and unlabelled data.

Other works, for example in [97] or [140] are based on the use of entropies. In [97], the authors set out to identify anomaly patterns in traffic distributions by applying entropy-based approaches. They evaluate pair-wise correlations between entropy time-series to highlight correlated features. In [140], the authors provide a proof-of-concept of worm detection in fast IP networks by applying entropy-based calculations. In [76], anomalies are tracked by referring to traffic feature distributions; a direct application of statistics and entropy is used to observe changes in the distributions.

Simpler evaluation methods use statistics, heuristics and visualization models or a combination of these three domains. An example is Aguri [67] presented in section 3.2. In a subsequent publication [68], the authors present a simple approach for the identification of Denial-of-Service and Flooding attacks. Through aggregation and statistical evaluation (number of IP addresses, ports, traffic volume etc.) of traffic summaries, the authors were able to detect these two types of attack without further processing simply by analyzing the generated traffic profiles.

A common approach is to use visualization techniques for the evaluation of traffic data. In the field of network intrusion detection, visualization has also become a popular complementary instrument, with approaches that range from simple statistical value representation to highly complex machine learning based schemes. An example is Foresti et al. [43]. The authors use visualization to represent network alerts. Visualization can also be used at lower levels. In [52], the authors refer to a Netflow record repository as input that is transformed and stored in a database. A network operator can then choose between different visualization techniques, including various histograms or flows, which are represented as a graph in a circular manner. [52] presents a tool called FlowViz that uses a similar coloured rectangular representation for showing the usage of ports. Gonzalez-Arevalo et al. [51] propose that a flow is modeled as set of packets and a segment in the image represents a single connection in a two-dimensional space. To avoid collisions of parallel connections, randomly chosen heights are used. Patole et al. [102] use Self-Organizing Maps (SOM), while Mansmann et al. [9] represent data in the form of TreeMaps, tree-like maps for dynamic intrusion detection.

3.6 Limitations and advantages

The dependency models presented in this chapter rely mainly on data that is provided by business enterprises. The previously-presented approaches are content-dependent; for example social-behaviour based approaches use applications as a basis for identifying dependencies. Other approaches use enterprise network traffic for the identification of bugs and machine crashes. The advantages of these approaches is that they cover the needs of enterprise networks and allow quick reaction when an incident occurs. This is also possible due to the restricted quantity of network traffic in an enterprise network. Since technical problems can also be identified with these techniques, system crashes or printer failures can easily be repaired. This simplifies life for network operators.

The major problem with these approaches that they cannot be applied in Internet Service Providers (ISPs) due to scalability issues. The workload in terms of traffic at the ISP level cannot be compared to an enterprise workload. A variety of approaches are used to analyze full packet captures in order to identify service dependencies between applications and services. This provides deep insights into the activities on a network. What is more, the presented approaches are specifically designed for security purposes such as the detection of worms. Approaches at the ISP level have to cover a wider range of different anomalies and attacks and provide useful results when faced with new and unknown incidents.

From the perspective of evaluation, machine learning algorithms perform well in the area of intrusion detection and traffic analysis. Depending on the traffic information used, a good choice of the algorithm is essential for achieving acceptable classification accuracy and computation efficiency. Supervised learning approaches achieve good results, but a major drawback is that unknown attacks cannot be detected, since this techniques requires labelled data. A remedy to this is to use unsupervised or semi-supervised learning, since here only a small amount of labelled data or even none at all is enough to deliver good classification results. A further advantage is that such algorithms are able to detect previously unseen anomalies.

Part II

Contributions

Part A:
Netflow record analysis using machine learning

Introduction

This part of the thesis presents the research contributions that have been achieved. It is based on the evaluation of flow information by applying several different evaluation strategies. The first approach is to analyze simple sequences of Netflow records, searching for anomalies. A new method for the processing of simple Netflow records that combines kernel functions with machine learning techniques for detecting anomalies is presented. A new kernel function that operates over sequences of Netflow records by analyzing contextual and quantitative information has been designed. To detect IP flow record anomalies, analysis results from the distance function are used in a machine learning module that leverages the Support Vector Machines for classification.

This initial approach is extended by a second, which introduces a tree-like summarization technique for Netflow records. The aggregated Netflow record structures are analyzed by focussing on topological and quantitative aspects. In a first evaluation method, Netflow records are analyzed for anomalies by using a data mining algorithm for classification. In a follow-up chapter, Game Theory is used to model the attack strategies an attacker can use with knowledge of defence strategies. The aim is to evaluate the kernel function defence strategies available to network administrators. The final chapter, the kernel function approach is evaluated on sparse data sets by using Phase Space Analysis, such that missing dimensions can be reconstructed through time series.

Chapter 4

Single Netflow analysis

Anomaly detection using machine learning

This chapter addresses the topic of using machine learning techniques to analyze Netflow records from a large-scale network. Peak rates of 60,000 flows per second are a common phenomenon in large-scale networks. This huge quantity of data makes analysis difficult, both in terms of computational resources and scientific methodology. A new approach for analyzing large volumes of IP ow related data by defining a kernel function is described that is able to track topological and quantitative changes in traffic. The objective is to detect trafc anomalies and to characterize network trafc by applying for example a machine learning algorithm, as the support vector machine algorithm (SVM). Therefore the contribution of the current chapter is two-fold,

- Definition of a kernel function to compare Netflow records
- Use of an SVM-based approach for detecting anomalies in Netflow records.

4.1 A kernel function for Netflow processing

The detection of anomalies in Netflow record data requires a sequence of steps that are presented in this chapter. Netflow records have been described in section 2.1; to summarize they include all relevant network traffic information in a compressed format. A first step in the analysis of network traffic, is to manipulate the time-series of Netflow records such that they can be compared with each other. To achieve this, Netflow records are divided on the basis of a time interval, also known as time window. A time window can be defined as a fixed time slice (e.g. 5 seconds) during which Netflow flows are recorded. A major challenge for the analysis was to define an approach that can capture traffic pattern changes. Support vector machines analyze and identify patterns by using complex methods to map data in order to make it distinct in a higher dimensional space. Therefore, a mathematical transformation of Netflow records that is able to represent quantitative and topological information of these flows in a numerical form is needed.

In [137], a kernel function K is defined as a mapping of $K : X \times X \in [0, \infty[$ from an original input space X to a similarity score $K(x, y) = \sum_i \phi_i(x)\phi_i(y) = \phi(x) \cdot \phi(y)$, where $\phi_i(x)$ describes a feature function over a data snippet x. A general property of a kernel function is *symmetry*, such that $[K(x, y) = K(y, x)]$ and *positive-definiteness*[1]. The kernel function $K(W_n, W_m)$ determines the similarity between two Netflow windows W_n and W_m, each of t seconds. For the definition of the kernel function, two parameters extracted from the Netflow records are used in the definition of the function.

[1] A real-valued, continuous differentiable function called f is positively definite in a neighbourhood of the origin $(0, 0)$ called D, if $f(0) = 0$ and $f(x) > 0$ for every non-zero $x \in D$ [138]

The first parameter manages the topological part of the kernel function and uses the IP address space for source (`src`) and destination (`dst`). The IP addresses can be decomposed into two parts, $\texttt{IP}_{src,dst}$ = `(prefix, prefixlength)`. The `prefix` is the longest common sequence of bits of two IP-addresses, while the non-matching remaining bits of these IP-addresses are the `prefixlength`. The second parameter manages the quantitative part of the kernel function and holds traffic volume, `vol` in Bytes. To remind the structure of a Netflow record, see Figure 4.1.

With these notations, a Netflow record window can be modeled as a set of n Netflow records $W = \{f_1, \ldots, f_n\}$, where a record f_i is described as a three-tuple f_i =(`prefix(src,dst)`$_i$, `prefixlength(src,dst)`$_i$, `vol(src,dst)`$_i$).

Date	Start	Duration & Protocol	SrcIPAddress:Port	DstIPAddress:Port	Packets	Bytes	Flows
2010-12-24	15:15:25.428	382 TCP	138.146.155.74:80 ->	97.254.27.34:3269	1288	815286	1
2010-12-24	15:15:25.430	465 TCP	138.146.47.199:80 ->	97.254.55.41:1272	8484	7.8 M	1

Figure 4.1: Example of Netflow records

To illustrate what prefix and prefixlength mean, consider the Netflow record example in Figure 4.2. Comparing the binary form of source IP addresses of the two records,

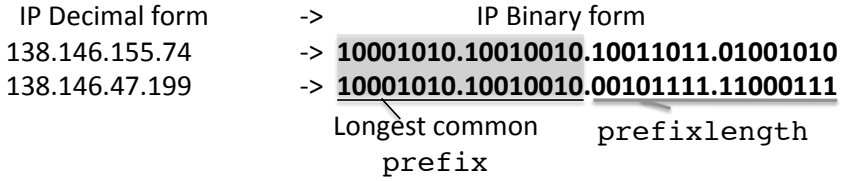

Figure 4.2: Illustration for parameters `prefix` and `prefixlength`

it can be seen that the longest common prefix for the IP addresses is 138.146. This means both IP addresses have a prefix of 16 bits in common and the remaining 16 bits are considered as prefixlength.

The kernel function, $K(W_n, W_m)$, aims to detect anomalous traffic patterns by detecting crucial topological or quantitative changes over time. To achieve this, a similarity function capturing topological changes and a matching function tracking quantitative changes is introduced. This kernel function produces a similarity score that is the sum over the matching and similarity functions for source and destination over all Netflow records in a window. A high similarity score characterizes similar Netflow windows. The kernel function K for two windows W_n and W_m is defined as,

$$K(W_n, W_m) = \sum_{i \in N_{W_n}, j \in N_{W_m}} s_{src,dst}(i,j) \times v_{src,dst}(i,j) \tag{4.1}$$

with Netflow records denoted N_{W_n} and N_{W_m}.

The similarity function $s_{src,dst}(i,j) \in [0, \infty[$ tracks topological network changes and models the extracted information from IP addresses for source and destination. It can be defined for flows i and j by,

$$s_{src,dst}(i,j) = \begin{cases} \frac{2^{prefixlength(src,dst)_j}}{2^{prefixlength(src,dst)_i}} & \text{if } prefix(src,dst)_i \text{ prefix of } prefix(src,dst)_j \\ \frac{2^{prefixlength(src,dst)_i}}{2^{prefixlength(src,dst)_j}} & \text{if } prefix(src,dst)_j \text{ prefix of } prefix(src,dst)_i \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

The matching function $v_{src,dst}(i,j) \in [0,1]$ handles traffic information, expressed in bytes and is used to track large changes in transfer volumes. The matching function $v_{src,dst}(i,j)$ can be defined by

$$v_{src,dst}(i,j) = exp\big(-\frac{\mid vol(src,dst)_i - vol(src,dst)_j \mid^2}{\sigma^2}\big) \tag{4.3}$$

where σ is the width scaling factor for the Gaussian kernel [14, 49] that has been estimated experimentally (see section 4.4.1).

In order to identify anomalous events, the experiments compared successive windows $K(W_i, W_{i+1})$ with all others to assess their similarities. To illustrate the effectiveness of the implemented kernel function, a real example is given in Figure 4.3. The representation shows the similarity values for successive Netflow windows obtained by the kernel function. No additional tool is used in this case. It can be observed that by tracking pattern changes in topology and traffic quantities with a kernel function, anomalous network activities can be detected, as here, a UDP flooding attack between Netflow record window W84 and W86.

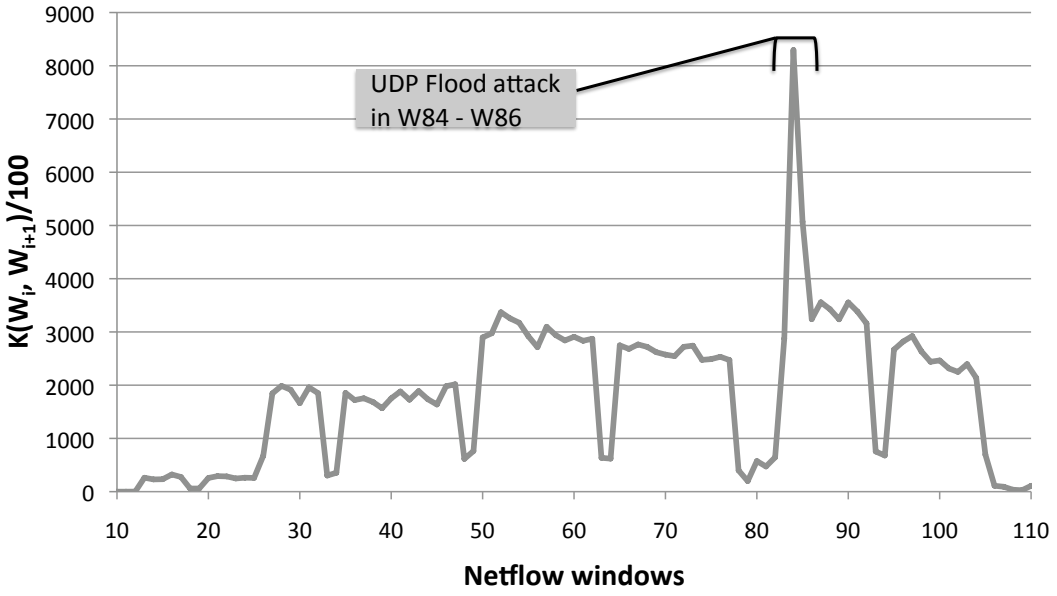

Figure 4.3: Visualizing an attack by using the kernel function

Flooding attacks generate large amounts of traffic and thus can be identified by the use of the kernel function, but more stealthy attacks probably cannot be detected. To strengthen the approach presented, a machine learning classification algorithm is applied in order to classify the similarity values obtained by the kernel function into benign or anomalous traffic patterns and so, help to identify stealthy attacks.

Since the range of possible network anomalies can include Denial of Service (DoS), Scanning, Botnets, etc., the anomaly class must describe these different kinds of data points. Due to this, multi-class classifiers do not meet the requirements as they need ground-truth, which means that they require labelled samples for each expected class of anomaly. Additionally, this kind of classifier is not able to detect new and unknown anomalies, which are the most interesting and potentially dangerous phenomena in current networks and the Internet.

Therefore, a very specific kind of algorithm, a *one-class* classification is used. The aim is to build a classifier that is able to detect new and unknown anomalies, corresponding to data points that do not follow a general traffic pattern and that can be used to train the classifier. More precisely SVMs are used, to achieve high accuracy with low complexity [152].

There is a specific one-class method, which was initially proposed in [120], known as One-class SVM. In this section, OCSVM is combined with the similarity scores from the kernel function to detect anomalies.

4.2 The classification method

4.2.1 One-class SVM

A major requirement in deploying an OCSVM is to use a data set that represents the single class of data points. This set is denoted X. In general the problem can be defined as follows: assuming that points from an original space S follow an underlying probability distribution P, the goal is to define a subset space of this original space S such that the probability that a point from P lies outside S is equal to some previously-chosen value between 0 and 1 [120].

This means that during the learning phase, the goal is to determine a function that is positive when applied to a point from S and negative otherwise. Subsequently, during the testing phase the sign of this function indicates, whether a point can be classified to the single class or not.

Assume a labelled sample $X = \{x_1, \ldots, x_n\}$ with n instances. The objective is to capture a small region enclosing these points by projecting them into a higher dimensional space, such that a subset of X can be better separated from the origin. The goal is to determine a hyper-plane with maximum margin, meaning that the distance to the origin is maximized. This projection is made using the $\phi(x)$-function and the proportion of points to separate is defined by $1 - \nu$ with $\nu \in [0, 1]$. The function can be defined as follows,

$$min_{w,\rho,\xi_i \ldots n} \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}(\xi_i - \rho) \tag{4.4}$$

with $< w, \phi(x_i)) > \geq \rho - \xi_i, \xi_i \geq 0$.

The optimization problem must be solved to identify two variables, w and ρ. ξ_i variables are slack variables that prevent points of S from being located on the correct side of the hyper-plane. This avoids problems with erroneous points. Since the definition of a projection function is not obvious, support vector methods traditionally rely on kernel functions. From a general point of view, kernel functions can be considered as similarity measures that respect the properties of finite positive semi-definite functions[2] and consequently satisfying the Mercer condition[3]. The kernel function $K(x_i, x_j)$ is equal to $< \phi(x_i), \phi(x_j) >$. Based on this function and by transforming the problem into its dual form, it can be deduced,

$$min_{\alpha_i \ldots n} \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j K(x_i, x_j) \tag{4.5}$$

with $0 \leq \alpha_i \leq \frac{1}{\nu n}, \sum_{i=1}^{n} \alpha_i = 1$.

Once this optimization problem is solved, the decision function can be defined as

$$f(x) = sgn(\sum_{i=1}^{n} \alpha_i K(x_i, x) - \rho) \tag{4.6}$$

This function is applied to each data point that is to be tested. If the sign of the function is positive, the point belongs to the class; otherwise it is considered as anomalous. In

[2] In [137, 121, 16] a positive semi-definite function is defined as: A function $\kappa : X \times X \rightarrow \mathbb{R}$ satisfies the property, if it is a symmetric function for which the matrices formed by restriction to any finite subset of the space X are positive semi-definite.

[3] Mercer's condition [128]: There exists a mapping function and expansion such that $K(x, y) = \sum_i \Phi(x)_i \Phi(y)_i$ iif $\forall g(x) : \int g(x)^2 dx$ is finite such that $\int K(x, y)g(x)g(y)dxdy \geq 0$

σ	K(T_n,T_{n+1})	σ	K(T_n,T_{n+1})
0.001	0.0000	10	0.9624
0.100	0.0920	25	0.9856
1.000	0.5168	50	0.9888
5.000	0.8944	100	0.9892

Table 4.1: Illustrating the effect of σ by a grid search

conclusion, it can be said that the main parameter ν represents the maximal proportion of points in X that are placed on the incorrect side of the hyperplane. The details of this method are presented in [120, 152].

4.3 The architecture of the anomaly detector

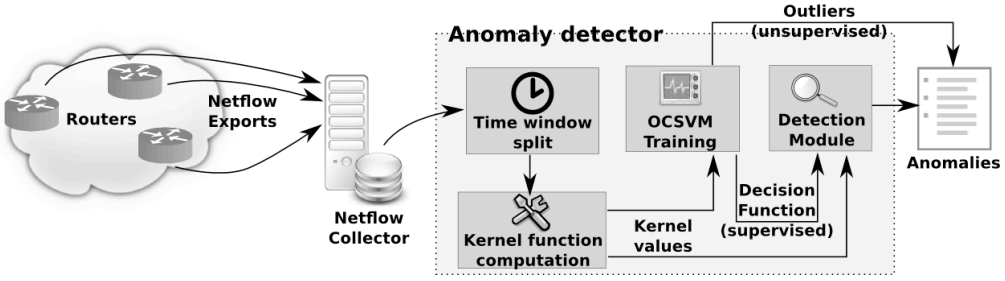

Figure 4.4: Anomaly detector architecture

The Anomaly Detector has three main components. The first module takes the captured Netflow records and divides them into time windows of t seconds. These Netflow windows are fed into the kernel function module, which compares successive windows and computes similarity scores. Here, a preliminary insight into the network activity can be obtained by representing the similarities scores graphically. In order to improve the evaluation of the similarity scores, a machine learning module is used. In Figure 4.4, two machine learning modules are illustrated, but only the supervised version of OCSVM has been evaluated.

4.4 Experimental results

4.4.1 A grid search for σ estimation

In a preliminary experiment, the aim was to study the influence of the σ factor on the Gaussian kernel function element, $v_{src,dst}(i,j) = exp\big(-\frac{|vol(src,dst)_i - vol(src,dst)_j|^2}{\sigma^2}\big)$.

σ is the width-scaling factor of the function and was estimated by performing a grid search on the kernel function for values ranging between $\sigma = 0.001, ..., 100$.

Table 4.1 shows the outcomes for the best σ values. For values of $\sigma \leq 1$, the impact on the kernel function is very high, whereas for values of $\sigma \geq 10$ only a low impact on the kernel function can be observed. This can be explained by the convergence property of the matching function element $v_{src,dst}(i,j)$.

Type of Attack	**Results**		
	Accuracy	False Positive rate	True Negative rate
Nachi scan	0.896	0.004	0.996
NetBIOS scan	0.938	0.000	1.000
Popup spam	0.915	0.023	0.970
Ssh scan + TCP flood	0.917	0.011	0.989
DDoS UDP flood	0.915	0.022	0.978
DDoS TCP flood	0.907	0.033	0.967
Stealthy DDoS UDP flood	0.938	0.000	1.000
DDoS UDP flood + traffic deletion	0.934	0.000	1.000

Table 4.2: Classification results by using OCSVM

4.4.2 OCSVM evaluation

As already stated, anomaly detection is a challenging topic and evaluating innovative solutions is a difficult task due to the lack of freely available and particularly of labelled data sets. There are some freely available resources such as, as the very well-known Lincoln data set[4], but this is considered to be out-dated nowadays. A recent data set based on Netflow records is provided in [126], but it contains only attack-related traffic from a honeypot. Furthermore, this kind of architecture collects attack traces from end-user point of view rather than that of the network.

Therefore, as starting point a real data set provided by a network operator from Luxembourg was used. It is assumed that it is free of malicious network activities, because a secondary semi-automated traffic screening [95] was made with an ISP-specific solution. The data set is shown in Table A.1 in Annex A. This data set is extended by injecting several types of attacks in form of Netflow records by using the Flame tool [12]. Both Flame and the generated attacks are described in Table B.1 in annex B. Flame can inject synthetic traces into an existing Netflow record data set. The tool includes a set of real attack models, which was extended to include stealthier attacks, as described in Table B.1. Even though the duration of the data set is short, it is sufficient for evaluating the approach presented, since all attacks were limited to last for no more than 30 seconds.

For the experiments, Netflow windows of a duration of 5 seconds were generated. The kernel function was then applied in order to evaluate the contextual and quantitative information of each Netflow record window. Using the kernel function to shape contextual and quantitative information allows intense attacks to be detected without further processing, as shown in Figure 4.3. To strengthen the kernel function method and to detect stealthy attacks, the OCSVM algorithm has been applied to these similarity values to determine whether or not, the method is able to detect the attacks. 20% of the Netflow record windows were used to train the OCSVM classifier, while the remaining 80% of the data set were used for the testing phase. The outcomes of the experiments are presented in Table 4.2.

The results of applying OCSVM are very promising and it can be observed that the false positive rates are very low, except for some types of attack. A possible explanation for this can be that the method is not well-suited for this kind of attack. What is more, the overall average classification accuracy for all attack classes is around 0.92.

[4] `http://www.ll.mit.edu/mission/communications/ist/index.html`

4.5 Summary and conclusions

This chapter has summarized the work that in [144]. A new approach for the evaluation of sequences of Netflow record windows is presented. To validate the approach, several types of synthetic attacks were added to the data set using the tool Flame.

The first contribution consists in the evaluation of Netflow records by analyzing their quantitative and contextual information through the introduction of a new kernel function that calculates the similarities between Netflow windows.

Secondly, a SVM algorithm was applied to the output of the kernel function in order to detect attacks in data sets. The classification results are very promising, with most attacks being identified.

From the point of view of performance, it must be stated that it is impossible to use this algorithm in near real-time. This is because it requires that n Netflow windows are compared with each other, so with a window of size m flows, complexity easily reaches $O(n^2)$, as each Netflow record must be compared to all the other Netflow records in successive windows. Consequently, a large m per window results in longer runtimes. In practice this means that, for the small ISP data set, $n = 150$ Netflow windows having on average $m = 9,000$ flows per window must be compared to each other with the result that 10^{10} operations are carried out. Runtime can be reduced by varying the Netflow window size and by this the number of Netflow records in the window.

To lower the runtime (in the sense of operations to calculate), sampling methods like presented in [36] or [101] can be used. The identification of effective sampling rates remains an open issue. Another possibility is to apply aggregation, where Netflow records are aggregated over space and time in order to reduce storage problems and complexity while losing only a small amount of data.

The following chapter, presents a spatio-temporal aggregation technique for Netflow records, which allows the task of processing Netflow records to be accomplished more quickly.

Chapter 5

Spatio-temporal aggregation for Netflow records

Aggregation-based anomaly detection

The analysis of time-windowed Netflow records has shown that, while kernel function evaluation provides good detection results, it needs long run-times. Sampling or compression could decrease run times by reducing the quantity of Netflow records that are analyzed. An additional argument in favour of compression, is the storage requirement for logging every Netflow record. The question for this chapter is: can anomalies be detected by combining the kernel function approach with a tree-like compression representation for the Netflow data?

This chapter addresses this problem and proposes a compression technique that is able to track topological and quantitative changes producing a tree-like representation with minimal data loss. To evaluate the aggregated data, the kernel function for Netflow records introduced in Chapter 4 is extended so that it can be applied to the new data format.

5.1 Tree-like Netflow aggregation in space and time

Analyzing every Netflow record is a time-consuming task that only works in a delimited framework, as shown in the previous chapter. In this chapter, a tree-like aggregation approach is presented for the processing of Netflow records. Spatial and temporal aggregation of IP related data was first presented in [20, 68] for full-packet captures and was applied to Netflow records in [144, 45].

The task of spatial aggregation is performed by extracting host IP address information (for source or destination) and aggregating volume information from Netflow records (measured as proportion of bytes/packets). This information is then spatially aggregated and results stored in tree-like traffic profiles. A graphical illustration is given in Figure 5.1.

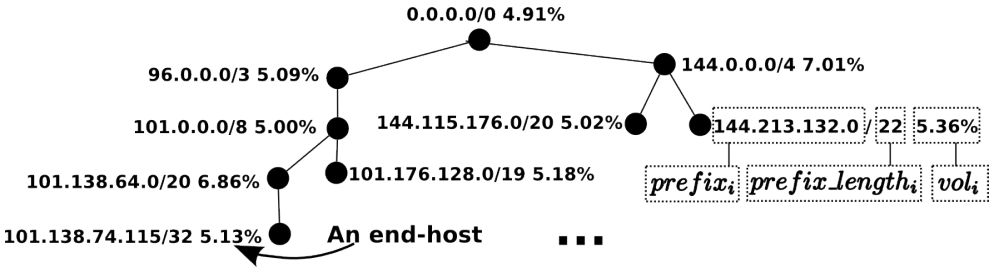

Figure 5.1: Graphical representation of spatio-temporal aggregation

In a traffic profile, a node represents an IP subnet, which can also be a single address, with the corresponding volume (excluding volume from child subnets), vol_i. The volume for a node is expressed as a proportion of the total volume. For example, assume $\#bytes_i$ is the number of bytes corresponding to node i, then the volume for a node i can be expressed as,

$$vol_i = \frac{\#bytes_i}{\sum_j \#bytes_j}$$

An aggregation threshold α is applied and nodes with $vol_i \geq \alpha$ are stored, while the volumes the others are added into their parents. The IP address information is decomposed into $prefix_i$ and those of $prefix\_length_i$ (see section 4.1). Thus, the tree structure respects the hierarchy of the IP address space. Clear advantages of this aggregation method firstly are that a traffic overview on a subnet level can be represented and second, data can be stored in compact traffic profiles.

When a new Netflow record is analyzed, the process matches source or destination IP address with the most similar node ip_{sim}, which is the one that shares the longest common IP prefix. This step is accomplished by a pre-order traversal of the tree. If a match is found, the corresponding volume (vol_i) is updated. Otherwise, a new node is created, together with a new branching point between the parent of this new node and the last visited unmatched node.To do this, Patricia trees [92], full binary Radix trees with a fixed tree size (N_{MAX} nodes) are used. Each time a new node is needed, the least recently used (LRU), node not having an associated volume higher than α, is reclaimed. An advantage of Patricia trees is that memory use and search time for variable-length keys are bounded. At the end of each time window, the tree is processed in post order to aggregate nodes with $vol_i < \alpha$ into their parents, which represent IP subnets with shorter prefix lengths. This description omits details, since the process is similar to that described in [68, 20]. The approximation of using an LRU algorithm is managed in $O(1)$ and does as established in [68, 20] not have a significant impact on the aggregation quality. To complete the definition of a spatial aggregation tree T, a traffic profile can be described as follows:

- A set of N nodes, where $T = \{n_1, ..., n_N\}$ and $n_i =< prefix_i, prefix\_length_i, vol_i >$
- A relation, called *child*, where $child : T \rightarrow \mathcal{P}(T)$, returning the set of child nodes for a given node[1]

Temporal aggregation consists of a sequence of traffic profiles over time:
$\{< T_1^{src,byt}, T_1^{dst,byt}, T_1^{src,pkt}, T_1^{dst,pkt} >, \ldots, < T_M^{src,byt}, T_M^{dst,byt}, T_M^{src,pkt}, T_M^{dst,pkt} >\}$
where $T_i^{src,byt}$ is the spatial traffic profile within a time window i source IP address-based, and $T_i^{dst,byt}$ the traffic profile for destination information. Both are based on volume in bytes. For temporal aggregation, a time parameter is introduced to set time window sizes (β in seconds). If packets, rather than flows are to be evaluated, the notations should be adjusted to $T_i^{src,pkt}$ and $T_i^{dst,pkt}$.

This chapter is focussed on anomaly detection, where the aim is to track traffic changes in traffic profiles by controlling the granularity α. The higher this threshold is set, the more Netflow records are aggregated into larger profiles, producing a coarse-grained view of traffic, whereas a small aggregation threshold provides fine-grained traffic views. In the case of anomaly detection, spatial aggregation is very useful for discarding small proportions of traffic that may be highly variable and particular for numerous end hosts. Temporal aggregation is applied to avoid distraction resulting from irrelevant concentrated short-term peak traffic patterns.

[1] a power set $\mathcal{P}(T)$ is the set of all subsets of T

```
Total =  11008757
0.0.0.0/0  540220 (4.91% / 100.00%)
└─96.0.0.0/3  560312 (5.09% / 27.21%)
   │  101.0.0.0/8  550880 (5.00% / 22.12%)
   │  │ 101.138.64.0/20  754834 (6.86% / 11.99%)
   │  │   101.138.74.115/32  564682 (5.13% / 5.13%)
   │  │ 101.176.128.0/19  564851 (5.13% / 5.13%)
     144.0.0.0/4  771170 (7.01% / 67.88%)
       144.115.176.0/20  552664 (5.02% / 5.02%)
       145.213.132.0/22  590328 (5.36% / 5.36%)
       145.213.144.0/20  712268 (6.47% / 6.47%)
       147.186.128.0/18  737222 (6.70% / 17.24%)
         147.186.144.0/21  599586 (5.45% / 5.45%)
         147.186.160.0/21  561074 (5.10% / 5.10%)
       147.186.192.0/18  559543 (5.08% / 26.78%)
         147.186.192.0/20  629860 (5.72% / 5.72%)
         147.186.208.0/21  561773 (5.10% / 5.10%)
         147.186.240.0/21  608873 (5.53% / 5.53%)
         147.186.248.0/21  588617 (5.35% / 5.35%)
```

Figure 5.2: Text output of aggregated destination traffic profile

5.2 The model for spatio-temporal aggregation

5.2.1 A kernel function for spatio-temporal aggregation

The kernel function designed for the traffic profiles is similar to that presented for simple Netflow records in Chapter 4. The main difference is that aggregated traffic profiles for source and destination are used as input. The metrics for the kernel function namely the source or destination IP addresses and volume information can be extracted from the traffic profiles. Figure 5.1 illustrates metrics vol_i, $prefix_i$ and $prefixlength_i$ (see section 5.1). Similarity ($s_{src,dst}$) and matching ($v_{src,dst}$) functions are reused from Section 4.1.

The kernel function for source and destination $K_{src,dst}(T_n, T_m)$ for profile tree T_n and T_m with the set of nodes N_{T_n} and N_{T_m} can be defined as follows,

$$K_{src,dst}(T_n, T_m) = \sum_{i \in N_{T_n}, j \in N_{T_m}} s_{src,dst}(i,j) \times v_{src,dst}(i,j) \tag{5.1}$$

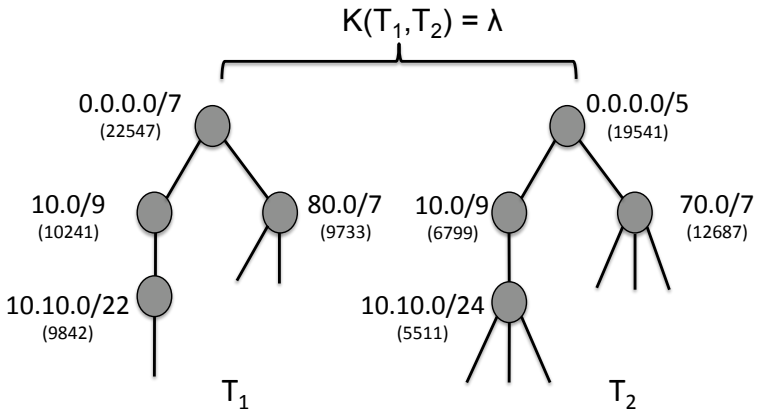

Figure 5.3: Graphical representation of the kernel function on traffic proles

To recall, the difference between the kernel function modelling here, and the kernel function presented in Chapter 4 mainly is the input that has changed. In Chapter 4, time window-based Netflow records were used. Here, Netflow records aggregated into traffic profiles are used, but the kernel metrics have not changed. Figure 5.3 illustrates, the process for the kernel function that is applied to the traffic profiles.

5.2.2 The architecture

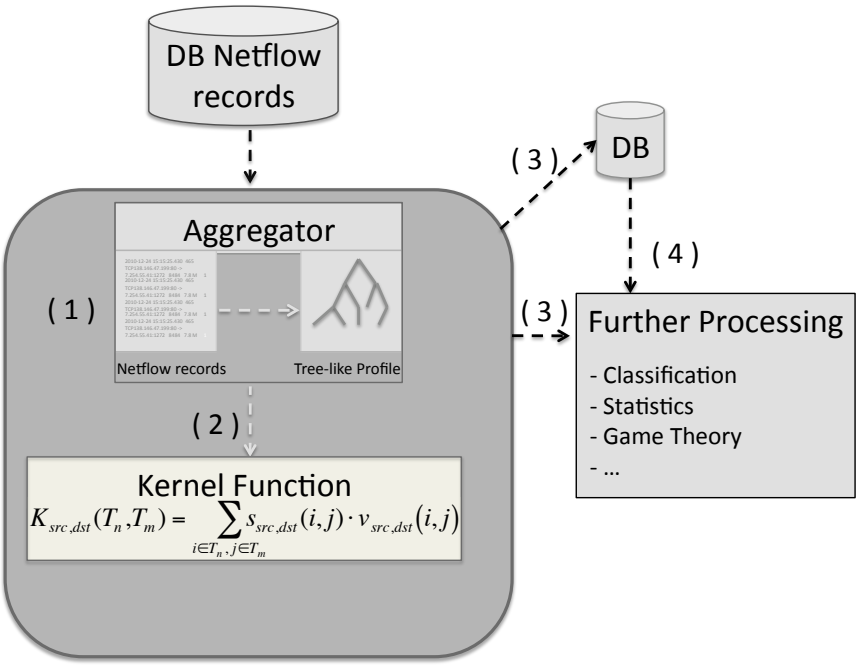

Figure 5.4: The architecture of the prototype

The general architecture is composed of three main modules. The first module, Aggregator, takes the captured Netflow records from a database and processes them into the spatially-aggregated shape. The aggregation task is performed by Aguri [20, 68], which implements aggregation-based traffic profiling. The original implementation of Aguri only supported full-packet captures, not the direct import of Netflow records, so it was necessary to extend it by implementing a custom import interface for Netflow records.

A traffic profile summarizes flow records for a period s of η seconds by spatially aggregating subnets with their hosts and activities, with η being defined by the network operator. An aggregation threshold t is fixed in order to adjust the level of aggregation. Aguri outputs four distinct traffic profiles. source and destination IP addresses and source and destination ports/protocols. Only the source and destination profiles for IP addresses are used in this thesis. A generated Aguri profile looks similar to Figure 5.2.

In Figure 5.4 the architecture of the proof-of-concept is shown: (1) Netflow records are used as input and the tree-like traffic profiles are generated as output. The kernel function module uses successive traffic profiles for comparison and computes a similarity value for these profiles. These kernel values can either be stored in a database (3) or immediately used for further processing, for example in this thesis to use classification, statistics, game theory, etc. Another possibility is to store the kernel values in a database (3) and process them later only (4).

5.3 Summary

The work presented in this chapter has previously been published in [148, 150, 151]. This chapter presented a new evaluation approach for Netflow record analysis that applies spatio-temporal aggregation. A major advantage of using aggregation trees is that the lookup operation when having N nodes runs in $O(logN)$, while management[2] of the list runs in $O(1)$. This allows the traffic aggregation method to be used in near real time [68, 20]. In the following chapters, this aggregation model will be applied to different methods for anomaly detection.

The kernel function discussed in the previous chapter has been extended to compute similarities between tree-like traffic structures for destination and source information in order to detect anomalies. From a runtime perspective this approach performs faster than the method presented in Chapter 4, as data is now aggregated and reducing the quantity of data to be processed. Additionally, the complexity of the new approach is at $O(n^2)$. In the data used for the experiment, the traffic profile has a constant size of 42 nodes that must be compared to each other within successive traffic profiles. This means while having $n = 150$ traffic profiles with $m = 42$ nodes, one requires 10^5 operations with the new method. Compared to the Netflow record windows approach (10^{10} operations), the new method is much more efficient in terms of run time.

To complement and assist the work set out in this chapter, a visualization module, `PeekKernelFlows`, was implemented. This tool enables a network operator to visualize network attacks and anomalies by using a mapping of kernel similarity values into the RGB[3] color space such that outliers and anomalies can easily be seen. A short description of `PeeKernelFlows` is given in Appendix C.

[2] insert, delete and update operations
[3] RGB: Red-Blue -Green color space

Chapter 6

A game for aggregated Netflow records

This chapter presents a somewhat other perspective to anomaly detection in Netflow records. The methods presented so far assume that a malicious entity is not aware of the network security mechanisms that are in place. This chapter addresses the problem where each entity has full knowledge about the other. This means that an attacker exactly knows which security mechanisms are being used and the network operator knows about the attacks being used. Additionally each party is has full knowledge about the other party's knowledge.

This chapter models a game that is able to define the best strategy for an attacker launching an attack against a network while exactly knowing its defence mechanisms, but at the same time defining the best defending strategy for a network operator. To model this strategic game for the kernel function based evaluation of aggregated Netflows, game-theory is applied.

The contribution of this chapter is twofold,

- The kernel function based model for evaluating Netflow records within
- a game-theoretical model that embodies the definition of a set of actions for attackers and defender

One application domain of game theory is the use of strategic games, for example the Nash Equilibrium [94]. The Nash Equilibrium represents a concept of a game that involves two to n players and aims to compute the optimal strategy profile for each player. If a player cannot obtain a higher payoff by deviating his profile unilaterally, this set of actions and payoffs constitutes a Nash Equilibrium. A well-known example for a Nash Equilibrium is the *prisoner's dilemma* [110, 5]. The prisoner's dilemma can be explained as follows: two people are arrested, but it cannot be proven who is guilty. The two are separated with no possibility of communication and the police officer offers both the same deal,

- If one prisoner incriminates the other and the other remains silent, then the defecting prisoner is released while the other gets one year of prison
- If both remain silent, then both get one month of prison
- If both incriminate each other, then both get three months of prison

In this situation, the prisoners must optimize their choice and cannot cheat. Table 6.1 illustrates the choices and its related consequences. A prisoner can choose one of the two strategies to play, but the rational consequence, the time to spent in prison varies greatly. For example, if prisoner 1 remains silent and the prisoner 2 defects, then prisoner 1 is imprisoned for a year while prisoner 2 is freed, but if both prisoners defect then the time in prison for each is only three months.

	Prisoner 2 is silent	**Prisoner 2 defects**
Prisoner 1 is silent	1 month each	Prisoner 1: 1 year Prisoner 2: free
Prisoner 1 defects	Prisoner 1: free Prisoner 2: 1 year	3 months each

Table 6.1: Prisoner's dilemma matrix

The optimal solution for both prisoners is to remain silent, but this case is not stable, because one prisoner can take benefit from this situation by defecting, so condemning the other to a year in prison. The only stable solution in this case for both prisoners to defect, because neither can take advantage in this situation. This is known as a Nash Equilibrium, even though the jail time for both prisoners is not minimized.

6.1 Game-theoretical modeling for spatially aggregated Netflow records

In this thesis, the Nash Equilibrium is used for the evaluation of different attacks and their respective defence strategies in network monitoring analysis. In this framework, a game between an `attacker` and a `network operator` is modeled. The different definitions and notations are taken from [53]. A game can be formally defined as follows,

Let the game be a 3-tuple $\Gamma = (\mathcal{N}, (A_i, R_i)_{1 \leq i \leq n})$ between `network operator` and `attacker`, where $\mathcal{N}$ is a set of n players, A_i is a finite strategy set ($a_i \in A_n$) and R_i: A $\rightarrow \mathbb{R}$ is a payoff function, with $A = A_1 \times \cdots \times A_n$.

In general, a Nash Equilibrium leads to either a pure strategy or a mixed strategy. A pure strategy can be explained as a set of actions reflecting a complete definition of a player's game, showing what moves a player can make in any situation. In a mixed strategy, players use a probability distribution over a set of possible actions. This means that a random choice is made between strategies in order to avoid being exploited by the adversary. A good example of a mixed strategy is the *Rock-Paper-Scissors* game [42], because there is no pure strategy for that game, as moves are unpredictable. In this case, a good strategy is to switch between strategies in order to avoid being exploited by the adversary. Nash showed in his work [94][1] that at least one mixed strategy Nash Equilibrium exists in any game with a finite set of actions, including zero-sum games.

According to [53], the set of probability distributions over a strategy set A_i is a mixed strategy set for a player i,

$$\Delta(A_i) = \left\{ q_i : A_i \rightarrow [0,1] \mid \sum_{a_i \in A_i} q_i(a_i) = 1 \right\} \tag{6.1}$$

where $\Delta(A_i) \equiv Q_i$, $Q = \prod_i Q_i$. The expected payoffs for a player i from a strategy profile q are

$$\mathbb{E}_{a \sim q} = \sum_{a \in A} q(a) R_i(a), \tag{6.2}$$

such that $q(a) = \prod_{i=1}^{\mathcal{N}} q_j(a_j)$ iff a Nash Equilibrium results from $\forall q_i \in Q_i$, $\mathbb{E}[R_i(q_i^*, q_{-i})] \geq \mathbb{E}[R_i(q_i, q_{-i})]$. In the context of the approach presented, a Nash equilibrium means that neither the `network operator` nor the `attacker` can increase their expected payoff, while it has been assumed that neither player changes the strategy during the game.

[1] `http://www.jstor.org/stable/1969529`, last accessed 02/01/2012

6.2 Attack and defence measures

6.2.1 Defensive measures

From a defensive perspective, the goal is to monitor the sequence of values $K(T_n, T_{n+1})$ (denotes the kernel function for traffic profiles of Netflow records, as defined in Chapter 5.2) and decide if at a moment n an anomaly or an attack is present. Most existing approaches are based on identifying outliers or trends in time series and leverage threshold-based methods. Several approaches are analyzed which can be leveraged in our framework:

- The network operator manually defines a threshold that computes the mean μ of observed normal traffic when no attack is taking place. If the kernel value for an Aguri profile
$$K(T_n, T_{n+1}) \geq \mu \tag{6.3}$$
then this profile is declared as an incident.

- The network operator defines a threshold ($\beta\sigma$) based on the Chebychev's inequality[2], where
$$|K(T_n, T_{n+1}) - \mu| \geq \beta\sigma \tag{6.4}$$
with β a real number, μ being the mean value and σ the standard deviation. A value exceeding this threshold is considered to indictate an incident.

- The network operator sets a threshold θ which is the maximal observed value of $K(T_n, T_{n+1})$ for normal profiles. This is defined as,
$$\theta = \max\Big(K(T_1, T_{1+1}, ..., K(T_n, T_{n+1})\Big) \tag{6.5}$$
If a new $K(T_n, T_{n+1}) \geq \theta$ then this event is declared as an incident.

- The network operator defines the threshold by using an Exponential Weighted Moving Average (EWMA) [18] that applies exponentially decreasing weighting factors.
$$EWMA_n = \Big(K(T_{n-1}, T_n) - EWMA_{T_{n-1}}\Big) \times \alpha + EWMA_{T_{n-1}} \tag{6.6}$$
with $\alpha = \frac{2}{P+1}$ and P is the number of Aguri profiles. If $K(T_n, T_{n+1}) \geq EWMA_n$ then there is an incident.

6.2.2 Attack measures

An attacker has two main choices. He can first decide when to attack, that is the time interval during which the attack will be launched. Once this decision has been made, he can apply the chosen attack method. It is assumed that the first choice can be made in two modes. The simplest mode corresponds to a memoryless and random attacker. Formally, this is modelled by a binomial probability distribution function, where p is the attack probability and q the non-attack probability, with $p+q = 1$. An advanced attacker can use a more complex model, where the choice is dependent on the immediate past. Such an attacker is modelled using a Markov chain [84]. This Markov chain model has a finite set of states $S = \{A, N\}$, with A being an attack state and N a non-attacking state. The transition probabilities are defined as $p(A \rightarrow A) = p$ being the probability of staying in the attacking state A, $p(N \rightarrow N) = q$ the probability of remaining in in state N, the transition probabilities $p(N \rightarrow A) = 1 - q$ for moving from N to A and $p(A \rightarrow N) = 1 - p$ the transition probability of moving from A to N (see Fig. 6.1). This model starts in state N, because it is assumed that the first Aguri profile corresponds to normal network operation.

[2]From [54]: Chebychev's inequality for a random variable (X) with a finite expected value (μ) and a non-zero variance (σ^2) can be defined for a real number k with $k > 0$ as $Pr(|X - \mu| \geq \mathrm{k}\sigma) \leq \frac{1}{k^2}$

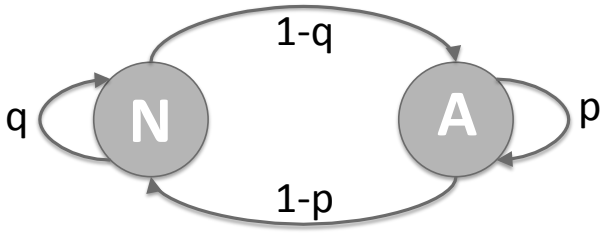

Figure 6.1: The Markov model transition diagram

The attacker can choose between several attacks. This thesis mainly considers attacks based on the injection of additional nodes and traffic (see section 6.3.1) into the network. Such attacks encompass distributed denial-of-service (increase in services, e.g. nodes and traffic), illegal repository (increase in traffic volume) or backdoors (new nodes and traffic). These attacks can be directly performed, with an attacker steadily injecting services into the network over a single time-slot, repeating this behaviour only a few times. In a more stealthy attack, an attacker continuously adds new services and/or traffic volume to the network over time, such that both of network size and traffic grow constantly. Such an approach is useful for deceiving anomaly-based monitoring approaches tuned only to detect sudden changes.

6.2.3 Modelling of actions and computation of payoffs

In this game, an attacker wants to find the optimal attack probabilities in order to remain undetected while running her attack. The game model can be defined as follows: The game has two players, $N = \{$*network operator*, *attacker*$\}$. Additionally, it is assumed that all *network operators* act as a single collective player and all *attackers* act as a single collective player. The set A_1 corresponds to the actions of the *attacker* and A_2 to the *network operator*'s actions.

The *attack* strategy is defined as a set of actions $A_1 = \{a_{1,0}, \ldots, a_{1,3}\}$, where

- $a_{1,0}$: binomial distribution with a simple injection scheme — an attacker injects a number of services into the network in one time-slot.
- $a_{1,1}$: binomial distribution with a stealth method — an attacker continuously adds services over a period of time.
- $a_{1,2}$: Markov Model with a simple injection scheme — an attacker injects a number of services into the network in one time-slot.
- $a_{1,3}$: Markov Model with a stealth method — an attacker continuously adds services over a period of time.

The *defence* strategy for a network operator can be defined as a set of actions $A_2 = \{a_{2,0}, \ldots, a_{2,3}\}$, where

- $a_{2,0}$: The network operator uses a threshold μ computed from observed normal traffic.
- $a_{2,1}$: The network operator applies the Chebychev inequality.
- $a_{2,2}$: The network operator sets a threshold that is the maximal observed value $K(T_i, T_{i+1})$ without attacks.
- $a_{2,3}$: The network operator applies the EWMA method.

A payoff should model each player's gain as faithfully as possible. This means that a network operator requires measures for the sensitivity (s_{sen}) and the specificity (s_{spec}) of the detection method. Sensitivity, also known as recall rate, is given by

$$s_{sen} = \frac{correctly\ identified\ attacks}{all\ identified\ attacks} \tag{6.7}$$

while specificity is a measure for the correctly identified negatives given by

$$s_{spec} = \frac{correctly\ identified\ attacks}{all\ correctly\ identified\ attacks} \tag{6.8}$$

Thus, the payoff function for the network operator is,

$$p_n = \frac{s_{sen} + s_{spec}}{2} \tag{6.9}$$

For the *attacker*, the payoff measures the time intervals that were not properly classified by the network operator's detection method, with

$$p_a = 1 - p_n = 1 - \frac{s_{sen} + s_{spec}}{2} \tag{6.10}$$

6.3 Experimental results

6.3.1 Injection attacks

This subsection describes how quantitative measurements were made in order to study the similarity of traffic profiles (of the ISP data set) by using the kernel function (see Appendix A.2). The sensitivity of this method was studied by injecting events that imply changes in the traffic matrix, like the appearance of new services in a network (while keeping the overall traffic volume constant) or spontaneous traffic volume changes (while keeping the number of services constant). Since the similarity function takes both the topology and the labelling of the underlying aggregated traffic profiles into account, the analysis examined how traffic changes impacted this metric.

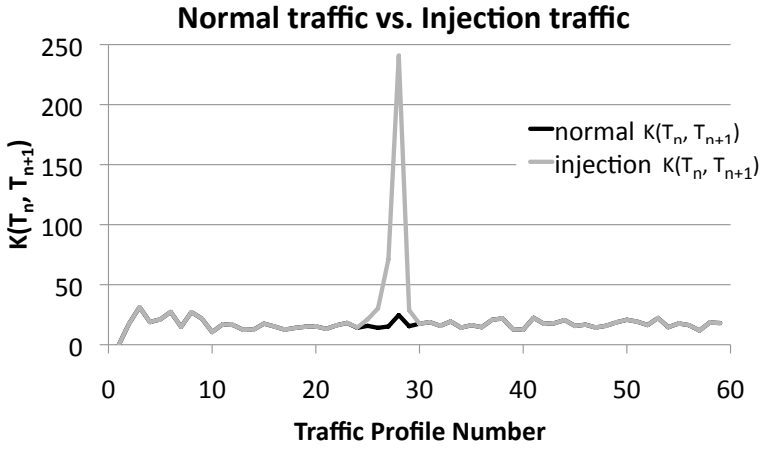

Figure 6.2: Visualization of a service injection attack vs. normal traffic

The first experiment simulated an injection attack. The intention was to show the impact on $K_(T_n, T_{n+1})$ of injecting new services into source traffic profiles by increasing the number of services in the profile, but without changing the traffic load in the network. To achieve this, a traffic profile was modified by adding n services (nodes). In Figure 6.2,

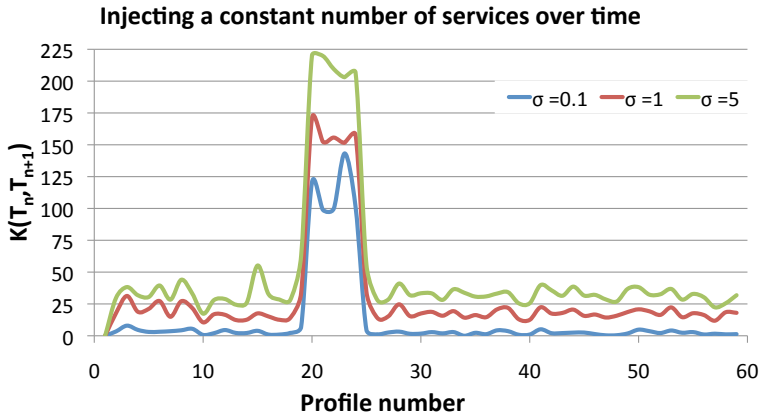

Figure 6.3: Injection of a constant number machines over 30 seconds

the $K_(T_n, T_{n+1})$ values for normal traffic profiles and the same profiles augmented by the injection of 10 services are highlighted.

The second experiment simulated a stealth injection attack . An attacker injects services over a number of profiles in the hope of remaining undetected. Starting in profile 20, an attacker injects 20 new services over a time period of 5 profiles (see Figure 6.3 for different σ-values, $\sigma = \{0.1, 1, 5\}$). It can be seen that the kernel function is able to detect stealthy incidents, since the stealthy injection provokes a peak throughout the injection period, whatever the value of σ.

6.4 Game-theoretical evaluation

This section presents the evaluation of the application of a game-theoretical model for injection attack/response simulations in order to determine optimal strategies for both attacker and network operator. Several scenarios were generated, using both binomial and Markovian distributions, but only one scenario for each distribution is presented in this section to illustrate the outcomes for this approach.

The binomial strategy used a probability of $p = 0.4$ to simulate stealth and injection attacks, the Markovian strategy used probabilities of $p = 0.4$ and $q = 0.6$ for its simulation of the injection and stealth attacks. The defence strategy used experimentally-estimated threshold values as defined in the action set $A_2 = \{a_{2,0}, \ldots, a_{2,3}\}$.

	Payoff values for attacker p_a and network p_n							
	$a_{2,0}$		$a_{2,1}$		$a_{2,2}$		$a_{2,3}$	
	p_n	p_a	p_n	p_a	p_n	p_a	p_n	p_a
$a_{1,0}$	0.58	0.15	0.71	0.29	0.41	0.59	0.72	0.28
$a_{1,1}$	0.71	0.29	0.72	0.28	0.57	0.43	0.65	0.34
$a_{1,2}$	0.71	0.29	0.65	0.35	0.81	0.19	0.80	0.19
$a_{1,3}$	0.71	0.28	0.85	0.15	0.58	0.41	0.71	0.29

Table 6.2: Payoffs for attacker p_a and network operator p_n

The payoff results for attacker and network operator are given in Table 9.1. The Nash Equilibrium was estimated by using a game-theory tool, Gambit [136]. A mixed-strategy equilibrium was obtained (see Table 9.2) by playing the different games. It can be seen that the probabilities for the different attack strategies vary, and similar observations can be made for the network operator. The best strategy for an attacker would be to apply the Markovian distribution scenario while performing a simple injection attack ($a_{1,2}$), because

this scenario reached a probability of 0.63. From the network operator's point of view, a good strategy would be to apply the simple mean (μ) threshold ($a_{2,0}$) strategy for network monitoring, as this scenario reached a probability of 0.53.

Attacker				Network operator			
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
0.04	0	0.63	0.33	0.53	0.24	0.23	0

Table 6.3: Results for Nash Equilibrium

A major assumption for regular finite games, such as the previous experiment, is that all players are rational. This means that each player wants to optimize its received payoff. However, players may make ill-judged decisions by selecting 'bad' strategies; alternatively they may take decisions they consider to be 'good', despite theory showing that another strategy is 'optimal'. To investigate the effectiveness of the different strategies for each player from the previous experiment, a quantal response equilibrium (QRE) analysis was performed. [87] introduces a precision factor λ to represent imperfections in payoffs. The QRE can be derived from a log-Weibull distribution, is a probability distribution for a failure occurrence time. Failures that is, poor decisions by a player, are described by the factor λ, which is inversely related to the level of error. This means that if $\lambda = 0$, all actions are erroneous, while$\lambda = \infty$ signifies no errors.

The QRE can be defined as

$$\pi_{ij} = \frac{e^{\lambda x_{ij}}}{\sum_{k=1}^{A_i} e^{\lambda x_{ij}}} \tag{6.11}$$

with x_{ij} being the expected payoff for player i using strategy j and π_{ij} the probability of selecting a particular action [135]. The outcome of the QRE analysis is represented in Figure 9.3. The y-axis describes the probabilities of selecting a strategy and the x-axis represents a grid search for λ.

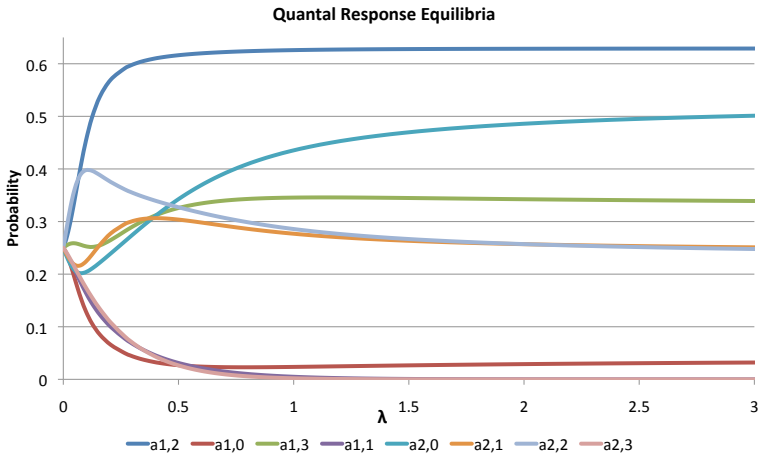

Figure 6.4: Quantal response equilibrium evaluation

Setting $\lambda = 0$ results in an equal distribution of strategies, meaning that a probability of 0.25 is given to each strategy. Increasing λ makes the obtained value tend towards the obtained Nash Equilibrium value (see Table 9.2 and Figure 9.3) and stabilizes. Additionally, it can also be observed that, as more bad decisions are made, the attacker strategy converges faster than the network operator strategy.

6.5 Summary

The work presented in this chapter was initially published in [147]. This chapter has applied game-theory to aggregated Netflow record profiles in order to use the kernel function to identify good attack and defence strategies for attacker and network administrators.

An advantage of the game-theoretic approach is that each party to a game has full knowledge about the other party' knowledge and all parties know that the other party has full knowledge about the own knowledge. In a preliminary experiment, it was shown that the kernel function for aggregated Netflow records is able to track both flooding and more stealthy attacks.

The Nash Equilibrium was used to model the game for the attacking and defending parties. A set of actions for attacker and defender were defined and the Nash Equilibrium applied to them in order to determine the best strategies for attacker and defender. Several approaches were tested and evaluated for the definition of the set of actions in the game. For the attacker actions, different attack schemes were evaluated to identify the most profitable solutions. The evaluation of the game model showed no pure strategy exists for the attacker or defender, as the Nash Equilibrium was mixed. To strengthen and evaluate the efficiency of this result, a quantal response equilibrium (QRE) was performed by adding an error value to the payoff function. This simulation confirmed the outcomes of the Nash Equilibrium.

The game-theoretic approach presented shows some limits. A reasonable approach would be not to use collective players. Collective players act by following a defined set of actions, whereas in network security, both attacker and defender can take a wide variety of possible actions. This chapter has used, only a restricted set of actions has been used to model the game, even though, in reality, many other actions are available to each party. The model does not use learning games; players' knowledge is not increased as the game progresses. Another limitation is the used of fixed payoffs, because a payoff should principally reflect the motivation of players and model the profit for each player.

Chapter 7

Reconstructing missing dimensions in aggregated Netflow records

DANAK: Finding the odd!

Previous chapters have analyzed anomalies in successive traffic profiles in a time series: ordered sequences of traffic profiles were analyzed at equally-paced time intervals[1]. Figure 7.1 represents the visualization of a time series with n traffic profiles into which a violent DDoS UDP flood attack has been injected between traffic profile 60 to 120. By evaluating this time series with the kernel function set out in Chapter 5.2.1, this strong attack can be detected and visualized in this graphical representation (annotated in Figure 7.1).

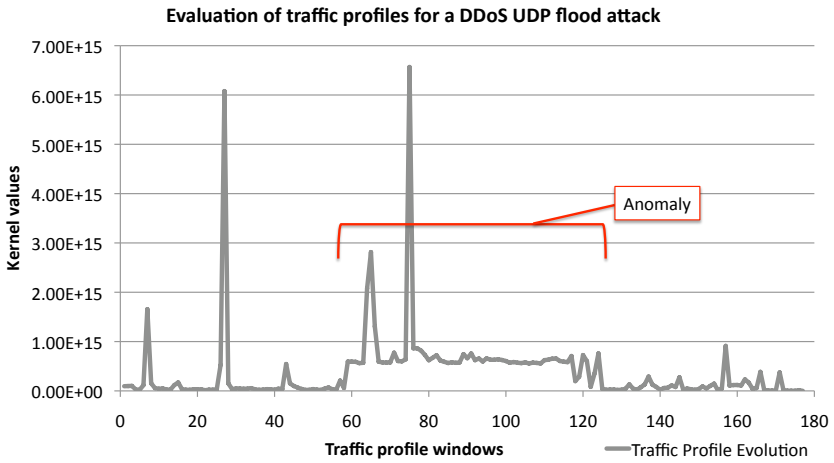

Figure 7.1: Kernel function evaluation for a violent DDoS UDP flood attack

In Figure 7.2 on the following page, a stealthy DDoS flood attack has been injected into the same time series and evaluated with the kernel function. This time, it can be seen that the kernel function is not sufficient to identify the anomaly, with the result that the graphical representation is unable to reveal it to a human expert.

[1]Definition of Time Series: An ordered sequence of values of a variable at equally spaced time intervals [96]

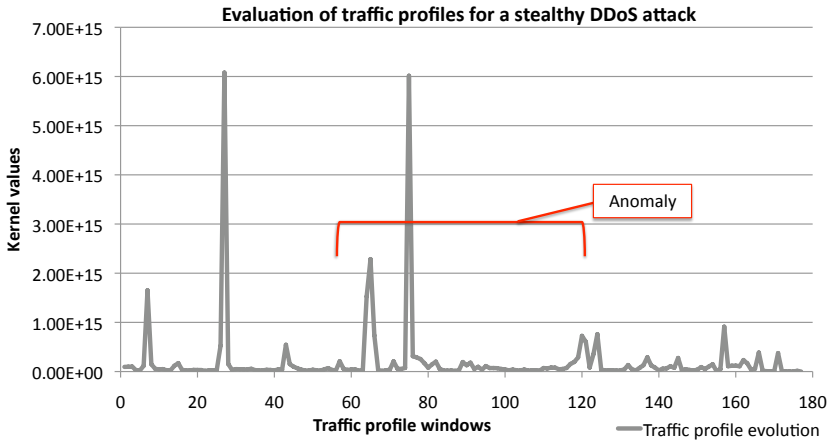

Figure 7.2: Kernel function evaluation for a stealthy DDoS UDP flood attack

Network workload changes or other disturbances impact network monitoring and reporting, such that large data sets may include periods where data is sparse or even completely absent. This had a major influence on the evaluation of the traffic time series. This chapter introduces DANAK (**D**etecting **A**nomalies in **N**etflow records by spatial **A**ggregation and **K**ernel methods), a new model that applies an accurate method for the embedding of time series into a n-dimensional space to reconstruct missing dimensions in traffic profiles. The model is evaluated by using a classifier in order to separate attacks from benign traffic. The contribution of this chapter is two-fold:

- A phase space embedding model for traffic profiles is introduced
- Evaluation of model by use of a classifier to detect anomalies

7.1 The model for DANAK

7.1.1 The kernel function for traffic profiles

In a first step, the kernel function of Chapter 5.2.1 is slightly extended by introducing a weight parameter for DANAK. The kernel function is calculated for input profiles by counting the number of common nodes and not by performing an exhaustive computation over the entire space for source and destination. The similarity and the matching function part ($s_{src,dst}(i,j)$, $v_{src,dst}(i,j)$) remain as defined in Chapter 5.2.1. For two traffic profiles denoted $T_n =< T_n^{src}, T_n^{dst} >$ and $T_m =< T_m^{src}, T_m^{dst} >$ (see details in section 5.2.1), the output of the extended kernel function $K(T_n, T_m)$ gives a similarity score over source and destination in only one function and can be defined as

$$K_{src,dst}(T_n, T_m) = \frac{1}{2} \sum_{i \in T_n^{src,dst}, j \in T_m^{src,dst}} s_{src,dst}(i,j) \times v_{src,dst}(i,j) \tag{7.1}$$

The output of the kernel function is used for the embedding and in the Phase Space analysis that is presented in the following section. The weighting of kernel function does not have an influence on the kernel function per se, but has been introduced for representation purposes, such that graph values become more readable.

7.1.2 The phase space method with delayed coordinates

This section presents the phase space embedding of time series for nonlinear systems. Time-delay reconstructions of a phase space are an accepted technique for viewing the dynamics in a system, as presented in [159, 130, 81, 72, 118]. Phase space analysis with delayed coordinates allows the reconstruction of missing dimensions by using preceding and delayed information (function values) as additional coordinates [130, 81, 159]. So enabling exploration of historical profile knowledge (including missing dimensions). Takens embedding theorem[2] [130] is used to apply time-delayed observed scalars as coordinates in the phase space, according to $x(t_0 + n\Delta t) = x(n)$ [72]. Multivariate vectors in d-dimensional reconstruction space are used to trace the system orbit such that, with T being the time delay,

$$y(n) = (x(n), x(n+T), ...x(n+(d-1)T) \tag{7.2}$$

with the time evolution of y denoted as $y(n) \rightarrow y(n+1)$ (see [130, 72]).

In this section, the time series resulting from the traffic profiling is modelled to allow the reconstruction of missing dimensions by using delayed coordinates. These traffic profiles summarize network activity for time intervals of η seconds. The modeling applies the model used in [159]. Here, Phase Space Analysis is applied to network-based sequence number generator quality in a system representing an n-dimensional space that fully describes a n-variable system. [159] uses a three-dimensional representation of one-dimensional data that is generated by calculating the first-order differences between traffic profiles. The obtained delayed coordinates are also known as function attractors [81]. Referring to [159], the model for the Phase Space Analysis with delayed coordinates for traffic profiles can be defined as follows:

Taking an input set s, and x, y, z as the point coordinates, the equation for a sample n can be defined as,

$$x[n] = s[n-2] - s[n-3] \tag{7.3}$$
$$y[n] = s[n-1] - s[n-2] \tag{7.4}$$
$$z[n] = s[n] - s[n-1] \tag{7.5}$$

where $s[n] = K_{src,dst}(T_n, T_{n-1})$. To model traffic profiles in this space, the first-order difference equations are applied to preceding, delayed and actual traffic profiles kernel function values in order to obtain an accurate and extended view on the evolution of the network topology and its related traffic. Unlike direct historical comparisons of one sample with prior ones, this allows the dynamism of changes to be captured. The results obtained from the phase space analysis are used in the classification task to detect anomalies.

In the introduction it was shown that the simple use of kernel functions can fail in the identification of anomalous network behaviour, for example, with stealthy attacks (see Figure 7.2) or in sparse data sets. Therefore, phase space embedding was applied to build historical profile knowledge (including missing dimensions).

An example showing a stealthy DDoS UDP Flooding attack in its traffic profiles is given in Figure 7.3. The Figure is a 3-dimensional graph that includes a simple history of actual traffic profiles T_i (expressed in kernel values) compared to the three preceding traffic profiles T_{i-1}, T_{i-2} and T_{i-3}.

[2] A summary of Takens' Theorem is given in [123]: Given a continuous time dynamical system with a compact invariant smooth manifold A , s.t. A,

- is of dimension d_A
- has a finite number of equilibria
- has no periodic orbits of period T_d or $2T_d$
- has only a finite number of periodic orbits of period $pT_d, 3 \leq p \leq m$ and those have distinct eigenvalues
- then there exists a delay coordinate function H which is a differentiable embedding from A to H(A) for $k > 2d_A$

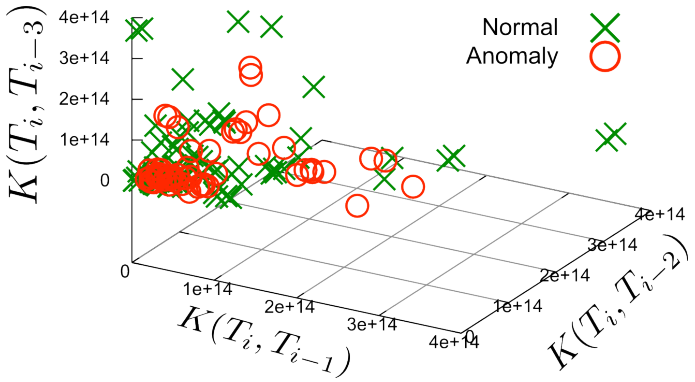

Figure 7.3: Stealthy DDoS UDP flood attack without phase space embedding analysis

The attack remains undetected in this representation, as the data points corresponding to the anomaly and to the benign traffic are not separable using the kernel function value history.

Figure 7.4 shows the same stealthy DDoS UDP flood attack, but this time when phase space embedding is used. The axis in the graph represent: the x, y, z -coordinates obtained by applying the first-order difference equations, presented in section 7.1.2.

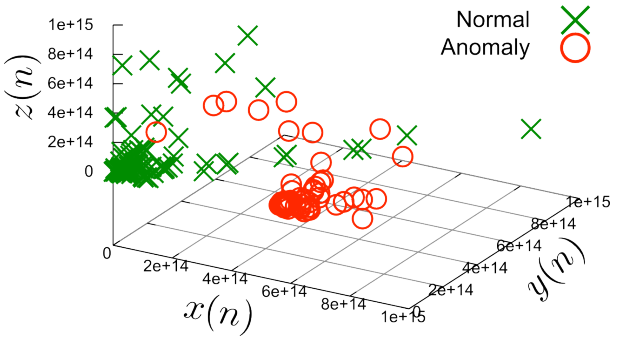

Figure 7.4: DDoS UDP flood attack with phase space embedding analysis

Comparing the two figures (Figure 7.3 and 7.4), shows that the anomalies can be visualized by applying Phase Space Analysis. In Figure 7.4, anomalies are now largely grouped in one part of the figure and disjoint to normal profiles. This demonstrates that combining kernel functions and phase space analysis, different types of attacks can be visually detected. However, to strengthen the validation of the approach presented, a classification algorithm that uses these three-dimensional features in order to identify anomalous and benign network activities is required.

7.1.3 The classification algorithm

The CART (Classification And Regression Trees) algorithm [13] is used to classify the three-dimensional features generated by the phase space embedding. CART algorithm is a supervised learning algorithm. The difference between classification and regression tree analysis is that, in classification tree analysis, the outcome is usually a prediction of membership of a class, whereas in regression tree analysis the outcome is a prediction of a real number.

Referring to the description in [13], the CART algorithm methodology can be divided into three steps,

- Building decision tree
- Pruning
- Classifying new instances

This paragraph briefly describes the CART algorithm, for a more detailed description, the reader is referred to [13, 124]. The algorithm generates its binary decision tree by applying recursive splitting on the sample space, using sample impurity to estimate the split threshold from the set until the stopping condition is reached. This impurity index, known as the Gini index, identifies the largest data class in a sample set and isolates it. A common problem in decision trees is that they are overfitted and complex. Therefore, pruning is performed in order to calibrate the decision tree and build an optimal tree. One common pruning method is minimal cost pruning, which is based on the estimation of the optimal balance between tree complexity and misclassification errors/costs.

In this work, the CART algorithm is applied to the phase space embedding results obtained with the kernel function values $K_{src,dst}(T_n, T_m)$. The aim of using the algorithm is to create a simple two-class classifier which is able to distinguish attacks/anomalies from benign traffic profiles.

7.1.4 The architecture of DANAK

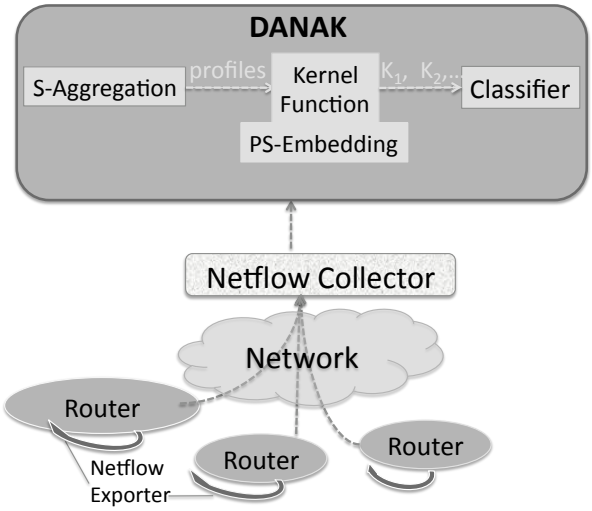

Figure 7.5: General view of the DANAK architecture

Figure 7.1.4 shows a general overview of the architecture of DANAK. The *S-Aggregation* processes Netflow records, aggregating them as shown in Figure 5.2 in Chapter 5, to generate traffic profiles for source and destination information. *PS-Embedding and Kernel Function* calculus are performed, representing the main component of DANAK. This method refines the kernel values for profiles by leveraging historical knowledge by means of phase space embedding. The results are passed to the classifier, which applies a machine learning algorithm, with the aim of classifying the values into anomalous or conventional traffic.

7.2 Experimental results

The ISP operator data set described in Appendix A, Table A.1, was used for the experiments. The parameters for the aggregation module were set to $\alpha = 5\%$ for the aggregation threshold, such that a fine profile granularity is preserved and only small Netflow record information omitted. The capture time for a Netflow record windows was set to $\eta = 5$ seconds.

To validate the DANAK monitoring framework, the Netflow data set was been extended by injecting synthetic attack traces into real traces using Flame [12]. A description of the Flame tool is given in Appendix B. The injected attacks were limited to time periods of five minutes; a shorter would not allow evaluation of attack detection in the face of varying background traffic. An overview of the implemented synthetic attacks and their description can be found in Appendix B.2, list B.1.

Classification results

The open-source WEKA machine learning tool [56], version 3.6.4[3] was used for the classification task. This tool implements a collection of machine learning algorithms (both supervised and unsupervised) for use in real scenarios.

The SimpleCART algorithm was used to evaluate the Phase Space Analysis with the kernel function values obtained by the traffic profile processing. WEKA's SimpleCART implements a minimal-cost pruning method [13]. A cross-validation parameter of 5 folds was set to fine-tune the experiments. Cross-validation is the equivalent of dividing the data set into n equally sized sub sets (in this case $n = 5$). A decision tree was generated using four of the five subsets, then the remaining subset was classified using this tree. The procedure was repeated for each of the five possible groups of four subsets, and the overall average classification performance calculated. Each kind of injected attack was evaluated with the SimpleCART algorithm. Table 7.1 summarizes the classifier results giving the true-positive and the false-positive rate for each attack. Finally, it is important to note that, while it was assumed that the data set was free of malicious traffic, there is no ground truth. Hence, some results classified as false-positives may actually be real attacks in the original data set. It can be observed that this classifier can accurately detect the various attacks injected

Type of attack	**Results**	
	True Positive Rate	False Positive Rate
Nachi scan	0.912	0.222
NetBIOS scan	0.941	0.185
Popup spam	0.882	0.361
Ssh scan + TCP flood	0.882	0.028
DDoS UDP flood	0.923	0.077
DDoS TCP flood	0.887	0.027
DDoS UDP flood + traffic deletion	0.932	0.072

Table 7.1: Classification results by using WEKA's SimpleCart classifier

into the data set. This confirms that the use of kernel functions for traffic profile evaluation is valid, even if weak attacks could not be detected visually in a graph (see Figure 7.2). The same conclusions hold for the phase space analysis. The true-positive rates were in the region of 88% to 94% for the attacks. The false-positive rates were acceptable (2 — 20%), even if for one particular attack, the rate was more than 30%. In Chapter 4, the false-positive rates were lower because unaggregated Netflow records were used as input and compared in time windows, which is equivalent to computing pairwise distances for approximately 22,000 flows per window. By inspecting the data set in more depth, it can be observed that the size of traffic profiles is constant, giving an average size of 42 nodes. This

[3] `http://www.cs.waikato.ac.nz/ml/weka`, last accessed: March 2011

means that in the worst case, the DANAK method must compute 42^2 kernel functions to compare the two traffic profiles. This is very low compared to the $22,000^2$ operations needed to compare two simple Netflow record windows. From the perspective of complexity, the situation is the same as in Chapter 4, $O(n^2)$. Obviously, the aggregation process needs some additional optimization, as proposed in [68, 20], especially for bounding the size of a tree during its construction phase. By introducing tree-optimization, trees can be constructed on the fly. In other words, aggregation is a good tradeoff for improving the scalability of the presented approach by maintaining a high true-positive rate with only a low cost in terms of false-positives.

7.3 Summary

In this chapter, a spatial temporal aggregation method of Netflow records was applied to phase space embedding in order to gain historical knowledge about traffic profiles. This contribution was published in [143]. It shows the benefits of combining phase space analysis with a kernel function to evaluate Netflow records, which is only possible if the records are aggregated. Various attacks on a real data set were analyzed with this new monitoring model to validate the approach. It was shown that applying only the kernel function to aggregated Netflow records was insufficient for tracking some kinds of stealthy attacks. Therefore a new technique phase space embedding was applied to the traffic profile kernel function values. This enabled stealthy attacks to be detected by generating historical reconstructions of traffic profiles. From the point of view of classification, it can be said that the method is well-suited to the detection of most types of injected attacks, although, in some cases, the false-positive rates are high.

As stated in the experimental results section, false-positives may be reduced by using simple Netflow record windows without aggregation, this would result in impossibly high run time costs (see section 7.2). Therefore, a trade-off must made: either the analysis can be performed quickly, or false-positive rates can be minimized at the cost of very long runtimes. Here, it is argued even if false-positive rates are a bit higher than for the Netflow record window analysis, the method is able to catch most attacks. It is also advantageous from a storage perspective, because one has to store on average just 52 lines for a profile, compared with 22,000 lines for a Netflow record window.

A major problem in the analysis of spatio-temporal data is to find a good representation for the data dynamics. The phase space embedding approach, allows reconstruction of missing data from estimates, so building a historical profiles for traffic. A limitation in this approach is that in the model presented only three successive profiles are used for the reconstruction of missing dimensions, allowing insight into traffic dynamics over only a brief period. Another limitation remains the calculation cost for the kernel function applied to Netflow record profiles, which results in long runtimes.

Part B:
Analysis of Tor network flows using machine learning

Introduction

Anonymous communication systems have emerged as potential solutions to the challenges of providing privacy and anonymous web access. The initial implementation of this kind of network architecture was provided by state-funded system for the government [133] that enhanced confidential and privacy-conserving information transmission by implementing strong security services.

Tor [32, 133, 19] is the best known overlay network approach, because it has a large deployment base and provides good security services. Tor is based on onion routing, which encrypts data and then randomly distributes information through a network of volunteer nodes, known as relays. Onion routing [111] uses layered encryption, where each layer represents a relay. This encrpytion scheme can be compared to the layers of an onion. The source system encrypts multiple times, then sends it through the relays, where each relay removes one layer of encryption before sending the data to the next relay, until it eventually reaches its destination. The aim of Tor is to conserve the anonymity and privacy of the communication as it passes through the relays.

A major drawback of such secured systems is that they are often abused for malicious activity by hackers, paedophile rings or terrorists. A recent example is the terrorist attacks in Norway on July 22, 2011. The author of these attacks, Anders Breivik urges potential candidates to use Tor for communicating their activities, as shown in the following verbatim extract from his doctrine, *A European declaration of Independence*[4] published after the attacks:

" *Use Tor network (while following all Tor guidelines) or use an equivalent service which will mask your PC/IP identity when browsing the net. Alternatively, you can use an anonymous PC from a LAN network provided by Mc Donalds or any available commercial services as long as they do not require you do ID yourself...*".

This is only one example, showing that Tor networks provide a strong architecture for hiding the identity and location of users that is unfortunately often misused with harmful intentions.

The main idea behind this kind of system is an overlay network, which is used to mix traffic and so defeat attacks based on traffic correlation analysis. Despite the strong security services provided by Tor, some vulnerabilities were discovered in recent years. Several articles [93, 19, 78, 99, 162, 10, 153, 154, 158] have shown Tor's vulnerability to a class of disclosure attacks which are able to detect whether one host is communicating with another. In most research, the threat model assumes that a malicious entity can control either a Tor entry node or a pair (entry and exit) of nodes.

[4] `http://bundes.blog.de/2011/07/24/2083-european-declaration-of-independence\ \-manifest-behring-breivik-11534460/`, last accessed, 27.09.2011

In some cases, attackers inject data or tags into user-related flows in order to trigger information disclosure [132]. Several protection mechanisms have been proposed, such as probing exit nodes [134] or forging browser-related information.

This part of the thesis presents two contributions:

- A data mining driven solution to recover the browsing history of Tor users (in Chapter 8).
- Optimal configuration settings based on game theory for Tor users and Tor operators, taking malicious nodes into account (in Chapter 9).

Following the discussion in [99], only trusted entry nodes should be selected when using a Tor network. Therefore, in Chapter 8, Torinj, proof-of-concept malware (see Appendix D and also [146, 149]) is proposed that attempts to infect trusted Tor nodes with the aim of reconstructing overall network activity. The purpose of this part of the thesis is to detect the number of distinct users and to reconstruct the browsing history for each user. Torinj, targets Tor exit nodes because a larger vulnerable population of exit nodes than entry nodes is expected and they are presumed to be more vulnerable than entry nodes. The main purpose of Torinj is its ability to recover a user's browsing history even when a trusted entry node is used.

The disclosure of users' identities or locations is not specifically addressed; rather, the focus is on extracting user-specific behaviors. A practical attack that leverages a semi-supervised learning algorithm and a simple traffic injection scheme for this purpose is described. Several defensive measures exist, including enhancing user privacy by manipulating user agent fields or by testing the exit node. These actions can be countered by an attacker — he can become more stealthy in order to capture and reconstruct as many sessions as possible. Chapter 9 tests the efficiency of these strategies with the aid of game-theoretical concepts.

Chapter 8

Breaking Tor anonymity by flow analysis through data mining

8.1 Introduction to data-mining a Tor session

It is challenging to reconstruct Tor sessions when only one exit node can be controlled. The attacker can observe pairs of outgoing requests and incoming replies but is unable to differentiate individual sessions. Thus, anonymous browsing under this threat model is analogous to hiding in the crowd. Consequently, it is natural to attempt to isolate individual browsing histories and build individual user-related trace histories. This chapter describes a new attack that leverages a Data Mining technique together with aspects of the HTTP protocol to cluster and extract individual HTTP sessions relayed over a malicious exit node. In this analysis, only Web traffic is considered. The new attack scenario is represented in Figure 8.1.

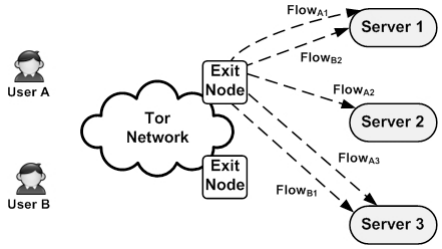

Figure 8.1: Typical attack scenario

Several users (in the example, for the sake of clarity only two users are considered) use two Tor entry nodes, which build tunnels ending at an exit node. It is assumed that this exit node is controlled by an attacker, so he can observe the traffic (five flows) between the exit node and the three servers. The objective of the attack is first to establish that only two users (user A and user B) are currently routed through the exit nodes, and second to reconstruct the browsing history for each of them. In this case, $flow_{A1}$, $flow_{A2}$ and $flow_{A3}$ can be tagged as belonging to user A, while $flow_{B1}$ and $flow_{B2}$ are tagged for user B. The practical attack on Tor is represented in Figure 8.2 and is executed as follows:

1. User A issues one (or several) HTTP request(s) to server 1. This traffic is contained in one flow: $flow_{A1}$. The malicious exit node forwards the request(s) to server 1 and injects a static tag. The static tag is a fixed image that is introduced in HTTP responses. The image has a size of 1x1 pixel and is invisible, so as to avoid distracting the user while browsing the HTML page. The image URL is unique for every injection. This is done by generating a universally unique number that is used to name the image

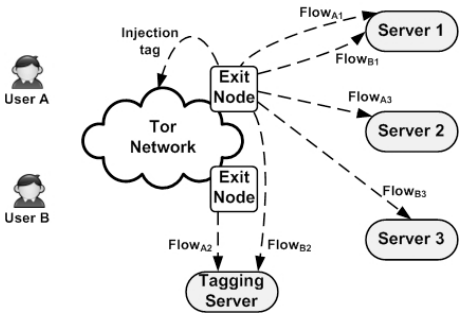

Figure 8.2: Injection attack

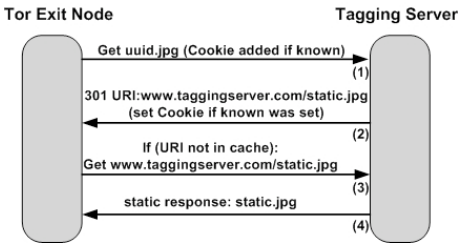

Figure 8.3: Flow A_2 interaction details

file. We assume that the browser cache on the user's machine is working correctly and that the image is downloaded only once. The file containing the image is hosted on an attacker-controlled server, the *tagging server*. For illustration, we assume the server URL to be `www.taggingserver.com`.

2. The browser of user A requests the tag by connecting to the tagging server to download it. Even if another Tor exit node is used to retrieve the tag, the incoming traffic ($flow_{A2}$) is intercepted by the attacker. This step in the attack (see Figure 8.3) is essential to logically link $flow_{A2}$ and $flow_{A1}$. The attacker uses the image URL to link the initial injection performed on the flow $flow_{A1}$ with the incoming request. The tagging server replies with a redirection URI `www.taggingserver.com/static.png` — which is a static URL. This reply is provided via the *301 Moved Permanently* HTTP response code [39]. It will also set a unique cookie for the domain `www.taggingserver.com` if no cookie was provided in the request.

   The browser of user A will retrieve the file *static.png* and cache the new URI, which is the same for all replies performed in this stage. The advantage of this redirection is that, when an URI is retrieved via *301 Moved Permanently*, its response codes are locally cached by the browser. Setting the cookie depends on the browser's capability to accept third party cookies, but it can also work without setting the cookie, if an additional synchronization step between the tagging server and the exit node is performed. This step is needed to associate the flow used to download the image tag with the initial request/response. Obviously, these cookies are not taken into account when clustering the traffic. This injected traffic is not considered when assessing the performance of the Data Mining algorithm.

3. User A issues one (or several) HTTP request(s) to server 2, contained in $flow_{A3}$. The malicious exit node forwards the request(s) to server 2 and performs a tag injection. The tagging server replies with a redirection URI `www.taggingserver.com/static.png`. In this redirection, no cookie is set for the domain `www.taggingserver.com` because the request already includes the cookie received in the previous step. This reply is provided via the *301 Moved Permanently* HTTP response code. However, the browser of user A will not retrieve the file *static.png* because it is already cached locally — the user agent will use the locally-cached version instead. Due to the missing download and the provided cookie, the attacker learns that the incoming traffic and the associated flow $flow_{A3}$, are related to a user that has already been observed.

4. User B issues one (or several) HTTP request(s) to server 1. This traffic is contained in $flow_{B1}$. The malicious exit node forwards the request(s) to server 1 and performs a tag injection.

5. The browser of user B requests the static tag by connecting to the tagging server to download it. The incoming traffic ($flow_{B2}$) is intercepted by the attacker. The processing is similar to the previous case.

   Here, a download of `www.taggingserver.com/static.png` is performed and no cookie for the domain `www.taggingserver.com` is included, the attacker can infer that $flow_{B1}$ and $flow_{B2}$ are associated with a user that has not been observed yet.

6. Both users (A and B) continue their browsing sessions and will be tracked.

The key idea behind this attack is to have a set of flows for which it is definitely known that the associated users are different: these are the flows targeted at the tagging server, from which the 1x1 image is downloaded and the cookies are issued. Additionally, it is assumed that connections by one user will share a set of features despite the Tor infrastructure [100]. Such features relate to user agent-specific settings (e.g. accepted language, version, name), and to outgoing HTTP requests (cookies, destination URIs). Finally, it is presumed that users follow a normal browsing behavior — i.e. after retrieving a HTML page, the next connection will be targeted towards an URI which is included in the initial web page.

The problem that has to be solved is: *A large amount of data must be clustered, where it is certain that only a small subset belongs to each cluster.* To solve this problem, a relevant distance function defined over pairs of flows is calculated. This function considers content-related similarities, user agent-specific settings and HTTP-specific protocol elements. A semi-supervised clustering algorithm takes this distance into account when performing the clustering.

8.2 The semi-supervised clustering algorithm

The methodology used to assign membership of flows to users is based on the idea of a semi-supervised clustering algorithm, known as a label propagation algorithm [161] presented in Chapter 3.5. We assume C different classes. Each class is initially represented by a flow that was tapped at the tagging server. All labels are propagated iteratively to unlabelled regions. At the end of the iterations, each unlabelled node is allocated to the most probable class — see Figure 8.4. Initially, there are only a few flows for which the user is known. These are the flows that have been used to retrieve the image file from the tagging server and where cookies have been set. The aim is to classify the remaining flows by clustering them on a per-user class basis. This is done by the clustering algorithm, which uses a distance function between pairs of flows to compare them.

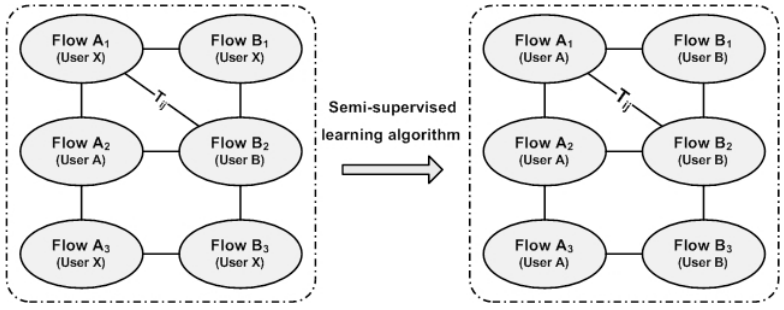

Figure 8.4: Semi-supervised learning algorithm

There is a set R^D composed of labelled data (x_1, y_1),...,(x_l, y_l) and unlabelled data (x_{l+1}, y_{l+1}),..., (x_{l+u}, y_{l+u}) with $l \ll u$, where Y_L=$\{y_1, ..., y_l\}$ are the class labels of the labelled data and Y_U=$\{y_{l+1}, ..., y_{l+u}\}$ of those yet to be assigned. D is the dimension of the input space. It is assumed that the number of classes C is known and that all classes are represented in the labelled data samples [161].

Let $X=\{x_1, ..., x_{l+u}\}$ be the different flows, and assume that each x is a D-dimensional vector, that is $x_i \in R^D$. Now estimate the class labels of the unlabelled samples Y_U from the data items X and their class labels Y_L.

The various distances used in this algorithm have to be defined. Assume two flow-class tuples (x_i, y_i) and (x_j, y_j) which must be compared. y_i or y_j represent the respective class labels and $D = 3$. To calculate the distances, the flow x_i is decomposed into a set of requests and responses, $Flow_i$=$\{(req_{i1}$,$rep_{i1})$,...,$(req_{ik}$, $rep_{ik})\}$. The same is done for flow x_j, $Flow_j$=$\{(req_{j1}$,$rep_{j1})$,...,$(req_{jl}$, $rep_{jl})\}$.

$d(i, j)$ is the distance between the two pairs (req_{ip}, rep_{ip}) with p = (1 to n) and (req_{jq}, rep_{jq}) with q = (1 to m).

$$d_{ij} = d(Flow_i, Flow_j) = d_1(req_{ip}, req_{jq}) + d_2(req_{ip}, rep_{jq}) + d_2(req_{jq}, rep_{ip}) \tag{8.1}$$

where $d_1(req_{ip}$,$req_{jq})$ is the Levenshtein distance[1] d_L of particular parameters: the content of Cookie (Co), URL, User agent (UA) and Accepted Languages (AL) fileds.

$$\begin{aligned} d_1(req_{ip}, req_{jq}) = d_L\Big((req_{ip[Co]}, req_{jq[Co]}) + d_L(req_{ip[URL]}, req_{jq[URL]}) \\ + d_L(req_{ip[UA]}, req_{jq[UA]}) + d_L(req_{ip[AL]}, req_{jq[AL]})\Big) \end{aligned} \tag{8.2}$$

Equation 8.2 calculates the Levenshtein distance between the requests ip and jq, which is the sum of the numerical distances of the fields listed above.

Distance $d_2(req_{ip}, rep_{jq})$ reflects whether or not a requested URL from ip is contained in reply jp,

$$d_2(req_{ip}, rep_{jq}) = \begin{cases} \alpha & \text{if } req_{ip[URL]} \text{ substring of } req_{jq} \\ 0 & \text{otherwise} \end{cases} \tag{8.3}$$

with $\alpha \in \mathbb{N}$, and in this case α=1. α can be selected in order to tune the algorithm and considers the weight given to a logical link of browsing activity. The formula for $d_2(req_{ip}, rep_{jp})$ can also be applied for distance $d_2(req_{jq}, rep_{ip})$, to see whether or not a requested URL from jp is in reply ip.

The labelled and unlabelled data samples are represented in a fully-connected graph, where the edge between nodes i, j is weighted. To calculate the edge weight w_{ij}, the distances and estimated weights are scaled by σ, a function width scaling parameter.

$$w_{ij} = exp(-\frac{d_{ij}^2}{\sigma^2}) = exp(-\frac{\sum_{d=1}^{D}(x_i^d + x_j^d)^2}{\sigma^2}) \tag{8.4}$$

Node labels are propagated through the edges to all other nodes. As in [161], we define a $(l + u) \times (l + u)$ transition matrix T, where T_{ij} gives the probability of a transition from node i to j.

$$T_{ij} = P(i \to j) = \frac{w_{ij}}{\sum_{k=1}^{l+u} w_{jk}} \tag{8.5}$$

$(l + u) \times C$ is defined as label matrix Y, where a row reflects the label probability distribution of a node. For instance, the element Y_{ic} is the probability Y_i of flow i belonging to class $c \in C$. Initially these probabilities are initialized to $1/C$ for the unlabelled data samples. The label propagation algorithm proposed by [161] has three steps,

- Propagate $Y \leftarrow TY$. All nodes propagate their labels.
- Row normalization of Y. This maintains a probability distribution.
- Clamping of labelled data. The label distribution of labelled data is clamped to Y_{ic}=1, if item Y_i had an initial label of c. This step assures that initial labels are maintained.

[1]Levenshtein distance [77]: Edit distance for measuring the difference between two strings or how many operations are needed to change one string into another.

These steps are repeated until Y converges. As proved in [161], the label propagation algorithm always converges.

To evaluate the labeling algorithm, class sensitivity, class specificity and the number of correct predictions, a technique set out in [8] is used. In a multi-class prediction problem with C classes, a $C \times C$ contingency matrix $Z{=}z_{ij}$ is used, where z_{ij} gives the number of times a sample belonging to class i is put in class j.

Class sensitivity gives the value of correctly-predicted samples belonging to class i to $x_i{=}\sum_j z_{ij}$, the total number of samples associated with class i.

$$Q_i = \frac{z_{ii}}{x_i} \tag{8.6}$$

Average class sensitivity ϕ scales the sensitivity for all classes according to the number of classes C.

$$\phi = \frac{\sum_1^c Q_i}{C} \tag{8.7}$$

Class specificity gives the ratio of correctly-predicted samples in class i to $y_i{=}\sum_j z_{ji}$, the total number of samples predicted to be in class i.

$$Q_i = \frac{z_{ii}}{y_i} \tag{8.8}$$

Average class specificity ψ scales the specificity of all classes by the number of classes C.

$$\psi = \frac{\sum_1^C Q_i}{C} \tag{8.9}$$

The final evaluation parameter is the quality Q_{Total}, the value of all correct predictions made.

$$Q_{Total} = \frac{\sum_i z_{ii}}{N} \tag{8.10}$$

where $N = \sum_{ij} z_{ij} = \sum_i x_i = \sum_i y_i$.

8.3 Experimental results

8.3.1 Passive attacks

As a passive attack a Tor exit node was run for a period of 28 hours and captured HTTP headers passively. Results similar to those reported in [85] were observed: 96% of traffic is HTTP and only 4% of traffic is end-to-end encrypted with HTTPS. The injection attack works only for HTTP replies that have the associated MIME [46] type set to *text/html* and consequently the proportion in real traffic was measured. As shown in Table 8.1, about 30% of all HTTP replies have the MIME type set to *text/html*. Over this time frame, 627 reply messages were injected and 879 unique download requests from the tagging server were observed. The exit node provided 2Mbit/s bandwidth and it was assumed that the attack was undetected because the exit node did not get blacklisted.

8.3.2 The hidden exit node

The accuracy and precision of the implemented attack were assessed. To accomplish this, a hidden exit node was installed and students in a class were asked to use it. Ten experiments were performed with total control of the browsing history. Thus, it was exactly known which URLs belonged to which user session and thus, it was possible to compare the results obtained with the clustering algorithm with reality, as shown in Table 8.2. Additionally, the impact of the tagging process was investigated. While tagging all HTTP requests having the MIME type set to *text/html* is possible, an advanced attacker could be less invasive and tag only a subset of the HTTP requests. For instance he could use a probabilistic

	General Information per Data set									
Data set	1	2	3	4	5	6	7	8	9	10
Request /Response pairs	402	411	404	462	416	401	408	408	420	406
User-Agents	11	10	10	8	12	11	12	7	6	4
Cookies	43	62	48	50	81	51	66	44	95	48
URLs	56	160	133	109	167	144	182	128	148	156
text/html	0.520	0.198	0.227	0.380	0.450	0.185	0.228	0.215	0.231	0.211
text/css	0.280	0.001	0.042	0.005	0.027	0.015	0.045	0.030	-	0.075
application /x-javascript	0.055	0.061	0.070	0.017	0.042	0.053	0.128	0.110	0.058	0.067
application /shockwave-flash	0.045	0.005	0.022	-	0.022	-	0.032	0.038	0.010	0.052
image/jpeg /png/gif/x-icon	0.031	0.670	0.601	0.413	0.381	0.555	0.463	0.545	0.564	0.491
text/javascript	0.020	0.015	0.002	0.012	0.020	0.035	0.010	-	0.014	0.035
text/plain	0.018	0.022	0.017	0.014	0.030	0.020	0.020	0.012	0.091	0.022
application/xml	0.005	0.007	0.007	-	-	0.010	0.005	0.013	0.002	0.012
text/xml	0.002	0.005	0.005	0.007	0.015	0.002	0.002	0.010	0.002	0.005
Miscellaneous Content	0.004	0.007	0.002	0.131	0.011	0.002	0.010	0.009	0.004	0.004

Table 8.1: Global statistical information

	Sensitivity, Specificity and Q_{total} for 3 values of θ								
Set (C)	$\theta = 0.1$			$\theta = 0.2$			$\theta = 0.3$		
	ϕ / ψ	Q_{total}	N	ϕ / ψ	Q_{total}	N	ϕ / ψ	Q_{total}	N
1 (7)	0.47/0.52	0.80	357	0.53/0.82	0.89	306	0.62/0.82	0.88	285
2 (9)	0.35/0.49	0.65	357	0.45/0.71	0.71	320	0.54/0.72	0.74	285
3 (8)	0.47/0.54	0.54	358	0.52/0.75	0.64	323	0.55/0.75	0.68	275
4 (6)	0.45/0.54	0.64	364	0.48/0.54	0.69	332	0.52/0.55	0.78	273
5 (5)	0.55/0.66	0.72	357	0.60/0.70	0.81	315	0.71/0.79	0.84	290
6 (8)	0.54/0.53	0.85	352	0.52/0.60	0.87	314	0.56/0.73	0.88	289
7 (11)	0.31/0.59	0.61	362	0.33/0.69	0.70	308	0.42/0.69	0.71	270
8 (7)	0.32/0.49	0.63	349	0.31/0.51	0.72	317	0.41/0.51	0.76	259
9 (4)	0.49/0.63	0.73	371	0.45/0.92	0.70	313	0.58/0.95	0.81	271
10 (4)	0.51/0.69	0.82	360	0.64/0.69	0.82	302	0.69/0.72	0.92	269

Table 8.2: Sensitivity ϕ, Specificity ψ, Q_{total}, the number of classes C and the number of samples N for different thresholds θ

scheme driven by a threshold, i.e. for each HTTP reply, a random number between 0 and 1 was generated and if it is less than a predefined threshold then an HTML injection was performed. Tagging was performed for three different threshold values, the injection probabilities θ. The outcomes in Table 8.2 validate the clustering algorithm and represent average results obtained by multiple simulation runs. It can be observed that 30% of the HTTP responses contained HTML documents, so an attacker can set $\theta = 0.3$ ($\sigma = 1$) as maximal value, which gives the best prediction quality. The explanation for this is, that more request/responses are known. To show the robustness of our method, the standard deviation for the ten data sets is calculated and found to be very low, at 0.08.

8.4 Summary

This chapter has presented a new inference attack against anonymous communication systems that combines an active tag injection scheme with a Data Mining algorithm to cluster observed traffic flows. The label propagation algorithm provides results with an accuracy of 92% for attributing a flow to a user. The overall aim of this chapter was to present a method for identifying flows that are related to specific users and not to disclose a users identity. A proof-of-concept application described in Appendix D was developed to implement the attack.

Chapter 9

A game for attack and defence strategies in Tor flow analysis

To enhance privacy, a user of a Tor network can deploy several defensive measures such as modifying fields or testing the exit node. Knowing about such measures, an attacker can avoid them by being more stealthy in order to capture and reconstruct as many sessions as possible. In this chapter, the efficiency of these defensive/attacking strategies is investigated with the aid of a game-theoretical concept, called Nash Equilibrium.

9.1 Defence and attack strategies

A Tor user can deploy two main types of defensive measure to counter an injection attack. These can detect the malicious node and denounce it as a rogue in the global Tor directory. To do this, a user connects to a known web server and retrieves a particular web page. A hash function checks whether the reply has been tampered with. A previously-stored hash value is compared with the current one and, if a mismatch is detected, then the injection is revealed. This idea is proposed in [103] in the wider context of security improvements for the Tor network. Another proposal consists in carefully distributing Tor exit node usage so as to use disjoint IP networks. The latter proposal is not considered in the current game. Regarding the case of bad Tor exit nodes, Torscanner [134] is in use. The current process for adding a *BadExit* flag is implemented by the authorities managing the Tor directories, but at the time of writing, no router marked with a *BadExit* flag has been found.

Once a rogue node has been announced, no other user will use it as an exit node and thus, for an attacker the game is literally over. The attacker can however minimize the probability of being detected by performing fewer injections to improve his stealth, e.g. instead of injecting the image tag into all HTTP replies, the attacker can use a probabilistic scheme and tag only a subset, like injecting only into 10 to 50% of the replies. The presumption is that the probability of being detected is indirectly proportional to injection probability.

Another set of measures directly attacks the data mining approach presented in Chapter 8. Users can manipulate the HTTP headers that disclose information about their browser. For example, the user agent field can be spoofed or completely removed. Privoxy [107] allows the specification of a list of user agents that can be used for this purpose. Another header that can be changed or removed is the accepted languages, but impractical to provide a completely random choice, since popular web sites use this field to provide customized content/layout. For example, English-speaking users almost never access Google's russian website, except in order to remain anonymous.

The game between an attacker and Tor users is modelled by using the Nash Equilibrium notations from Chapter 6 which are set out in [53].

A game for Tor user and attacker be defined as a 3-tuple $\Gamma = (N, (A_i, R_i)_{1 \leq i \leq n})$, where N is a set of n players, A_i a finite strategy set ($a_i \in A_n$) and R_i: A→$\mathbb{R}$ a payoff function, where $A = A_1 \times ... \times A_n$. The game has two players, N={Tor user, attacker}. All Tor users act as a single collective player. The set A_1 corresponds to the Tor user actions and A_2 to the attacker's actions. The Tor user strategy is defined as a set of actions $A_1 = \{a_{1,0}, ..., a_{1,3}\}$, where

$a_{1,0}$: Every user works normally — the web client is used without any privacy protection mechanisms

$a_{1,1}$: Everybody modifies the user agent — for each request, a user agent is randomly selected and spoofed in the requests

$a_{1,2}$: Everybody deletes user agent — no information whatever is leaked to the attacker

$a_{1,3}$: Everybody deletes accepted languages — no language-specific information is leaked

The attacker strategy is defined as a set of vectors A_2={$(a_{2,\theta})$} where θ={0.1,0.2, ...,0.5} is the injection probability. An attacker plays by choosing the probability of injecting a frame into a given HTTP reply.

A Nash equilibrium in the context of a Tor system game means that neither the Tor user nor the attacker can increase their expected payoffs, assuming that neither player changes his strategy during the game.

Computing payoffs

The payoff for each player should model his gain as accurately as possible. Two distinct goals are important for an attacker. First, he should reconstruct sessions as accurately as possible by maximizing Q_{Total}. Secondly, he should remain as stealthy as possible and thus perform as little tagging as possible.

In the previous paragraphs, it was explained that the more an attacker injects tags, the higher the probability of being detected. The payoff function for the attacker, p_a, can be defined as $p_a = (1-\theta) \times Q_{Total}$. For the Tor user, the payoff p_u is a measure of the achieved privacy and can be defined as $p_u = (1 - Q_{Total})$.

9.1.1 Advanced attacks

When a Tor user deploys a user agent-changing strategy, the impact on clustering is immediate. Due to the user agent field, the distance component will be biased and members belonging to the same class will be missclassified as a result of large distances. Thus, for an attacker, it makes sense to learn the list of user agents used by an individual user. This would bias the distance function towards weighting out large differences in the user agent field.

There is a straightforward extension that an attacker can implement, leveraging the processing of HTTP error messages. According to the HTTP specification [39], a server can reply with a *HTTP Error 302 - Moved temporarily* error message, which includes an alternative URL to which redirection should occur. The web browser immediately retries the alternative URL. If this happens, the user agent supplied by a user who is spoofing this field will change.

The attacker can use this behaviour to discover the list of user agents in use by a user. As the basic attack, an image tag (i.e. `www.taggingserver.com/uuid.jpg`) is injected into a reply that is being relayed by the exit node. The user's browser does not have the image in the cache and will connect to the tagging server to retrieve it. To achieve this, the client spoofs the user agent header by using one of the values from its list. The tagging server receives a request for the resource *uuid.jpg*. Instead of sending back the corresponding file, the server will generate a universally unique identifier *uuid1*, reply with a *HTTP Error 302 Moved temporarily* message and provide the alternative URL: `www.taggingserver.com/uuid1.jpg`.

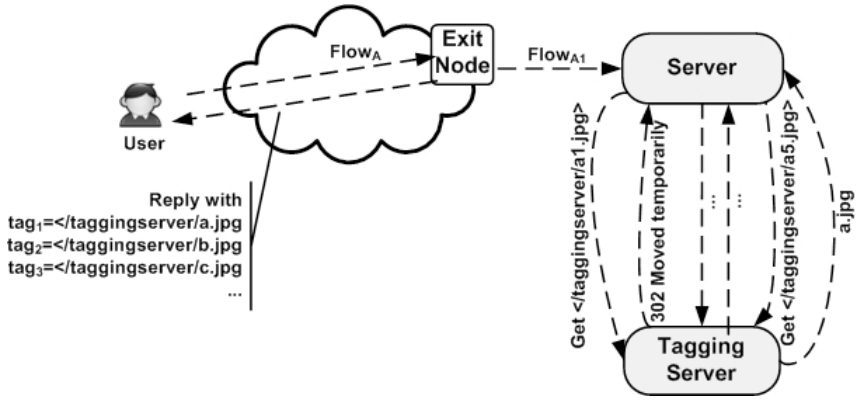

Figure 9.1: Tagging server attack with HTTP error message

Consequently, the client will contact the tagging server `www.taggingserver.com` and ask for *uuid1.jpg*. For this request, another value for the user agent list is used and hence disclosed. The tagging server replies with one more *HTTP Error 302 Moved temporarily* error and an additional alternative URL: `www.taggingserver.com/uuid2.jpg`. Browsers allow this process to be repeated up to five times. The fifth request can be answered by sending back the image file. The use of a universally-unique identifier is required to differentiate user-initiated requests. Obviously the file *uuid**x**.jpg* must not actually exist and x is used by the tagging server to count the *HTTP Error 302 Moved temporarily* messages that have already been sent back.

To differentiate between new users and users who have already been observed, the final download of the image file and a cookie-tracking mechanism is required. New users will download the file and will have a cookie set, while previously observed users will rely on the locally-cached data and will use the domain cookie in all requests, as represented in Figure 9.1.

When the procedure is complete, the attacker has learned five values used to spoof the user agent header. Large lists can be retrieved by generalizing the attack and having more than one tag injection. Starting with the first request, n different image tags are injected. If these are not cached, the client will retrieve them from the tagging server. Each individual request will reveal one value used to scrub the user agent field. For each individual image tag up to five alternative URLs will be provided and thus disclose a total of $5n$ user agent values.

9.2 Experimental Results

9.2.1 Defence strategies: playing Tor games

The game theoretical model was applied to data set 7 and the Nash equilibrium was computed using the Gambit library [136]. As the clustering values of Q_{Total} are not constant, this step was repeated three times and the average value taken. Table 9.1 shows the average payoff values for experiments with data set number 7. An Equilibrium with two pure strategies was obtained and is represented in Table 9.2.

A pure strategy in the context of Tor security is defined as the set of actions providing a complete definition of a user's strategy, allowing the prediction of moves a player can make in any situation. This result shows that under game assumptions, the best strategy is to randomly change the user agent. This is unexpected, since a different outcome was anticipated. It had seemed far better to delete the user agent field and thus provide no information at all. It can be deduced that the random spoofing of user agents adds more noise and so disrupts the clustering process.

θ	Payoff values for attacker p_a and user p_u							
	$a_{1,0}$		$a_{1,1}$		$a_{1,2}$		$a_{1,3}$	
	p_u	p_a	p_u	p_a	p_u	p_a	p_u	p_a
0.1	0.42	0.52	0.77	0.20	0.42	0.53	0.45	0.45
0.2	0.36	0.51	0.70	0.24	0.32	0.55	0.32	0.54
0.3	0.23	0.54	0.65	0.24	0.29	0.5	0.27	0.51
0.4	0.30	0.42	0.62	0.23	0.26	0.45	0.22	0.47
0.5	0.23	0.38	0.53	0.22	0.23	0.39	0.26	0.37

Table 9.1: Payoffs for attacker p_a and Tor user p_u

Nb	Attacker					Tor User			
	0.1	0.2	0.3	0.4	0.5	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
1	0	0	1	0	0	0	1	0	0

Table 9.2: Nash Equilibrium table

In the light of this unexpected outcome, the impact of changing the user agent list for an end-user population was studied. In data set 7, the proportion of users changing their user agent is varied. A list of eleven user agents was used. The changing of the user agent was implemented by randomly choosing one value on a per-request basis. A simulation of a strategy using real data by directly changing the user agent field in the raw data with $\theta = 0.3$ ($\sigma = 1$) was run. It was observed that the accuracy was strongly impacted when all users changed their user agent, resulting in a loss in classification quality (Q_{total}), as presented in Figure 9.2.

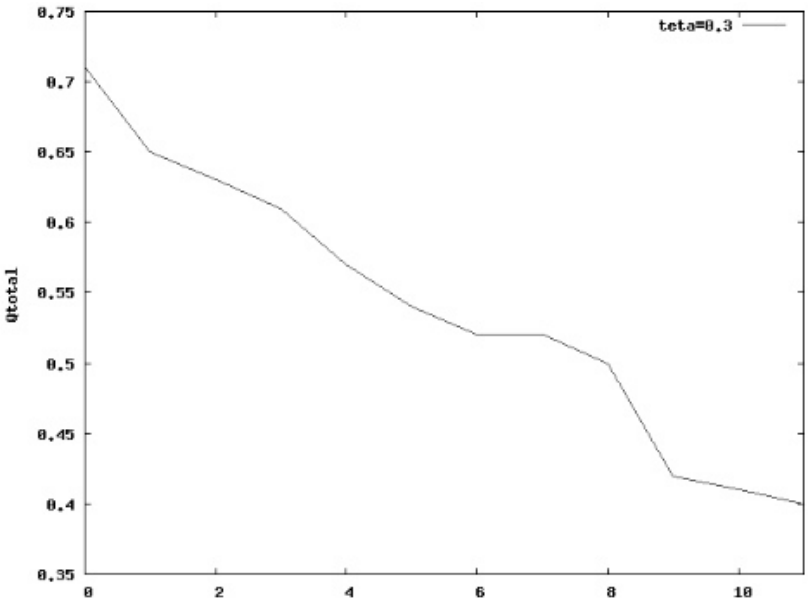

Figure 9.2: Users changing User Agent

To deepen the analysis, a quantal response equilibrium (QRE) method was introduced in order to show what happened if players made bad moves. The reader is referred to Chapter 6.4 for a description of the QRE approach. The QRE analysis was performed in order to show the effectiveness of the players' various strategies, because it can provide quite different results as the Nash Equilibrium analysis.

Figure 9.3 shows the results of the evaluation. Setting $\lambda = 0$ gives an equal distribution for the different strategies: there is an initial probability of 0.2 for all five strategies. By increasing λ we observe that the behaviour of the curves tends towards the previously obtained Nash Equilibrium (as presented in Table 9.2) and stabilizes.

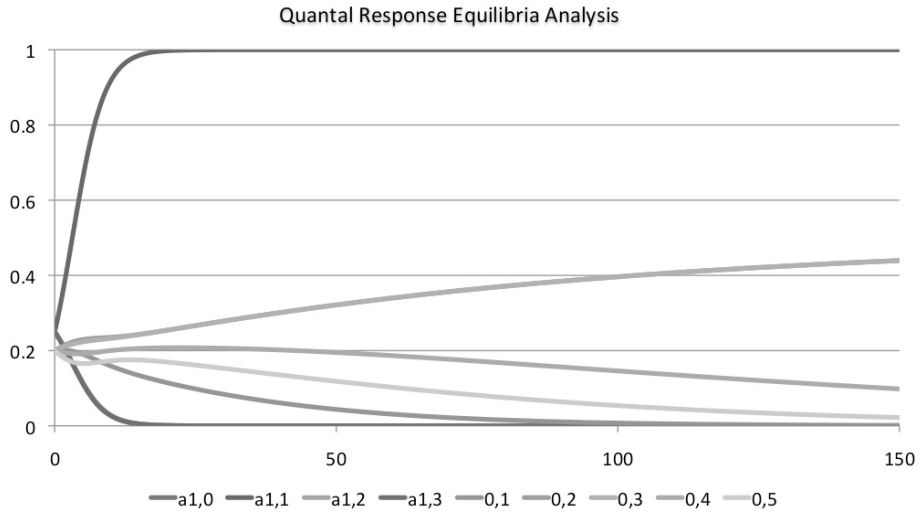

Figure 9.3: Quantal response equilibrium evaluation

This result confirms the results already obtained with the simple Nash Equilibrium, namely that the best strategy for an attacker is to use a tagging probability of 0.3, whereas the best strategy for a user is to apply strategy $a_{1,1}$, where every user modifies his user agent.

9.3 Summary

This chapter has modelled a game for attacker and Tor user in order to discover which strategies perform the best. This work has previously been presented in [146, 149]. Several possible attack and defence strategies were described in order to make the Nash stategies more comprehensible. To strengthen the evaluation, the quantal response equilibrium approach, a more advanced method was applied. This showed was shown that the results obtained confirm the Nash Equilibrum outcome.

Part III

Conclusions and Future Work

Conclusions and Future Work

The first part of this thesis described different models and their related proof-of-concepts for the analysis of Netflow records on attacks and anomalies. Netflow records provide valuable information not only about traffic content, but also about the communicating partners. Netflow records also provide insights into attack attempts, because particular Netflow record patterns appear in the Netflow record history: for example a very large number of logs for the same host differing only in their port numbers.

From an ISP perspective, Netflow records are the only available information, since it is impossible to store full packet captures of data. Compact formats for Netflow records are commonly used, for example by applying aggregation of Netflow records as described in this thesis. In this tree-like aggregation process, small Netflow records are aggregated into larger ones, such that they gain in relevancy and so can represent larger entities, for example subnets. A major advantage of this aggregation process is that data per se is not simply lost, but added to a larger entity.

The initial approach in this thesis dealt with the analysis of Netflow record windows and required storage of all Netflow records. The results obtained by applying a kernel function for exploring Netflow record content to machine learning showed that this approach is very useful in the detection of anomalies. There is a considerable body of work describing the use of to machine learning techniques for the evaluation of network data but, unlike the work described in this thesis, these machine learning approaches are used without a special shaping of network data.

A major problem with this approach is that in practical applications, this method is not feasible, since the storage of all Netflow records is not possible for a network operator. A further limitation of this approach is that all Netflow records of a Netflow record window (each of an average size of 22,000 lines) must be compared to each other, making the calculation of the kernel function between Netflow record windows was a very time-consuming task.

This showed the need to introduce a more concise model for the storage of Netflow records. A possible method would have been to use sampling methods presented in Chapter 3.1; another possibility was to apply aggregation (see Chapter 3.2), where Netflow records are aggregated over space and time into tree-like structures. Netflow record profiles can be analyzed for anomalies by combining the kernel function with a machine learning approach, which classifies them into benign and unusual traffic. By applying the kernel function to the traffic profile, a major outcome is that strong attacks can be identified simply by using the kernel function without additional processing methods.

To illustrate the effectiveness of this approach, a prototype was implemented that visualizes the outcomes for the kernel function values, allowing service injection or stealthy attacks to be easily represented in image form (see Appendix C).

An advantage and at the same time a limitation of the experimental analysis is that the data set was extended by injecting synthetic traffic traces, rather than using traces from known real-world attacks. On the other hand this allowed accurate analysis, as it was known exactly where the attacks were located. To extend this work, attack and defence strategies for attacker and network operator respectively were identified and explored by applying a game-theoretical model to the aggregated Netflow records.

A major limitation of the machine learning approach is false-positive rates which vary considerably for the attack models, with stealthy attacks showing higher false-positive rates than scanning attacks for example.

The outcomes of the previous contributions showed that the kernel function for aggregated Netflow records is not able to track stealthy attacks accurately. Another major drawback of Netflow records is that sometimes, due to technical problems, Netflow record time windows are lost or not recorded. These two arguments motivated the development of a new technique that is able to detect stealthier attacks and identify attacks in sparse data sets, through a reconstruction of missing dimensions in traffic profile histories. By applying the method of phase space embedding, missing profiles in a time series can be reconstructed by estimation.

Part B describes the analysis of application flows in order to associate flows to users. This part describes a new inference attack against anonymous communication systems that combines an active tag injection scheme with a semi-supervised data mining algorithm to cluster observed traffic flows. Each cluster corresponds to the browsing history of one user. The performance of this attack was assessed on several realistic data sets and a formal modelling framework based on game-theoretical concepts is proposed.

More advanced attack and defence strategies were evaluated with this framework and an optimal set of strategies was identified (in the context of the Nash equilibrium). The aim of this approach was not to reveal the user location or identity, but this could be achieved in future work by going beyond simple HTML injection through injecting malicious JavaScript, like Beef [55] or XSSProxy. This would build permanent connections (for the lifetime of a browsing session) and allow advanced recognition actions against users' privacy.

The approach presented has some limits. If all users use end-to-end encryption such as ssl, this method is not able to correlate pairs of requests and replies to reconstruct browsing histories. However, based on our experiments, less than 3% of all traffic is encrypted. Another limitation of this method results from the maximum life-time of an established Tor tunnel, which currently defaults to ten minutes.

The described attack is particularly suited for Web browsing reconstructions and it was observed that. User agent-specific headers and HTTP parameters drive the clustering phase.

Several browser plugins already block image tags. PithHelmet[1], an optional ad-blocker for Safari[2] includes this option and also blocks images matching a (fixed) list of common banner ad sizes, e.g. 1x1 pixel. It blocks them by displaying a transparent graphic of the same size after first downloading them, in order to define the image size, so our attack still works. ICab[3] mobile, a little-known third-party browser for iOS devices has a user-editable list of sizes to block.

But again, even if 1x1 images are blocked, they have to be downloaded first. The more general ad-block functionality, by Adblock Plus[4] for FireFox, simply does not download content from URLs matching any of a list of patterns. This offers no protection against the attack presented in this thesis unless the image-hosting site gets blacklisted.

Future Work

This section presents some possible future research directions. This thesis has presented, a variety of approaches to aspects on Netflow record analysis by introducing a kernel function that is able to capture changes in network traffic. Netflow record windows as well as aggregated forms of Netflow records have been analyzed for anomalies. Since the results obtained by the kernel function after applying aggregation provided good results, a possible topic for future work would be to optimize the aggregation process by estimating the optimal thresholds for aggregation.

[1] `http://www.culater.net/software/PithHelmet/PithHelmetSampleAdBlocking.php`
[2] `http://www.apple.com/safari/`
[3] `http://www.icab.de/`
[4] `http://adblockplus.org/en/`

By optimizing this process, long term analysis could be performed such that larger data sets can be used for evaluation. Introducing additional features in future work could make the approach more scalable, allowing analysis of traffic from ISPs and other institutions to validate the presented approach on different kinds of traffic, since for example ISP traffic cannot be compared to internal enterprise network activity.

Additionally some short term future work can be identified, aiming mainly to identify evaluation techniques similar to the method presented. The evaluation of processed Netflow records has mainly applied machine learning techniques, in particular classification mechanisms, such as support vector machines. Future work could investigate alternative machine learning approaches including semi-supervised learning like presented in the chapter concerned with the evaluation of Tor data. Another activity could be to implement a kernel function for Netflow records into a support vector machine algorithm, so reducing the overall processing time. A possible method could be to adjust the support vector kernel such that it is able to extract topological and quantitative information from Netflow records.

Relevant future work can be done in the area of the game-theoretical models since the model presented here, does not learn during games. The work could develop new game-theoretical models that leverage machine learning techniques for the generation of repeated interactions between an attacker and a defence system.

From a computational perspective, the approaches presented need long runtimes and show high complexities. These complexities could be reduced by optimizing the algorithms for the kernel calculus. An alternative or additional mean of reducing runtime would be to apply distributed computing techniques, for example Hadoop[5]. These split large computing tasks into smaller tasks and distribute them over many machines. This improves runtime, so speeding the processing of large data sets.

A possible extension of this work would be to include analysis and cross-analysis with a variety of types of network-level data. Another extension would be to combine Netflow records with DNS entries. DNS data provides relevant information about data on the wire. The captured data can be evaluated by combining the presented approaches with DNS data to identify attacks on a network, ranging from the detection of worms to fast-flux[6] DNS attacks. This would require fast, high-capacity storage as both DNS data and Netflow records would need to be stored.

An interesting future work topic would be to extend the work on the Tor network to count Tor users . This is particularly difficult, since exit nodes have major differences in terms of bandwidth, service policies and location. A further challenge is to extend the Tor approach to the more general case of any application using anonymous overlay networks

[5] `http://hadoop.apache.org/`

[6] Fast-flux DNS [29, 155]: a commonly used botnet DNS approach for hiding malicious sites behind compromised hosts acting as proxies.

Part IV

Appendices

Appendix A

Data sets

This section describes the different data sets that were used for performing the experiments described in this thesis. A major problem for the validation of the approaches presented was to find a labelled data set. Many previous works have used the Lincoln data set [1], but nowadays this is considered outdated. Therefore the work presented here is based on three data sets, the first two provided by a large network operator in Luxembourg and the third from a legitimate honeypot. The ISP data set is assumed to be free from malicious network traffic as a secondary semi-automated traffic screening has been performed with aid of a network operator specific-solution [95].

A.1 Simple Netflow Analysis

The following table summarizes the data set from Luxembourg ISP operator. This data set was used to perform the experiments of unaggregated Netflow records in Chapter 4.4.2. Additionally, this data set was extended later by injecting attacks with Flame (see appendix B.2). To this extend, this data set was then used to generate aggregated Netflow records profiles in Chapter 7.2.

Flows	1,371,194
IP addresses	128,781 (source), 125,723 (destination)
Duration	13min 31sec
Bytes	7.5GB
Avg. bytes/flow	5,492
Packets	11.5M
Avg. packets/flow	8.36
UDP Flows	983,511
TCP Flows	375,132
ICMP Flows	11,347
Other protocols Flows	1,204

Table A.1: ISP data set statistics

[1] `http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html`, last accessed: March 2011

A.2 Aggregation-based evaluation of Netflow records

The following table shows a second data set records from an ISP operator. This data set includes aggregated Netflow records, and was used to validate the experiments in Chapter 6.3.

ISP data set	
Average number of nodes	42
Number of flows	3,733,680
Total bytes	19.36G
Global capture duration	300 s
Average bandwidth	528Mbit/s

Table A.2: ISP network monitoring data set description

A High interaction honeypot, exposing a vulnerable ssh-server, was operated for 24 hours on one public IP-address. All traffic related to this host was recorded and considered suspicious by definition. The follwoing data set was used for the validation of the PeekKernelflows tool, see Appendix C.

Honeypot data set	
Number of addresses	47,523
Exchanged TCP packets	1,183,419
Operation time	24 hrs
Bandwidth used	64 Kbits/s
Colour (bit)	24

Table A.3: Honeypot monitoring data set description

Appendix B

The FLAME [12] attack injection tool

B.1 General description

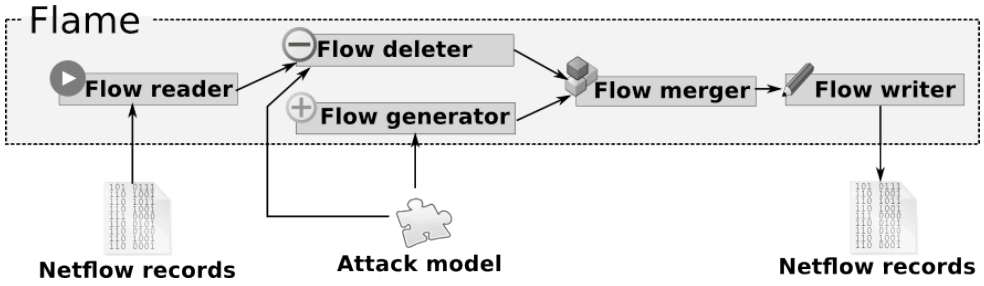

Figure B.1: Data set generation using Flame

The data sets used in the experiments are extended by injecting synthetic attack traces into real traces using the Flame tool [12]. Written in C++, Flame is a tool for evaluating anomalies at the flow level. Flame is based on Netflow records and takes Netflow record files without malicious activity as input (flow reader in Figure B.1), then injects synthetic attacks by generating (flow generator) or deleting (flow deleter) flow records based on various attack models.

The main idea is that attacks will generate traffic, but also affect normal traffic through activities such as DDoS. New attack records are combined with the modified input stream by the flow merger to produce an output file. Figure B.1 shows this process.

A key task in Flame is the generation of the attack models for the simulation of attacks. The default attack models, which were derived from real observations, are described on the Flame website[1]. In an initial trial, the existing attack models were used to generate attacks and later on, the experiments were extended by the creation of better adapted and stealthier attacks.

[1] `http://flame.ee.ethz.ch/publications.html`

B.2 Attack data set description

This table (B.1) shows the attack models that were used for the validation of the monitoring approaches presented in this thesis. It presents both the default and newly generated attack models generated using Flame. These attack models were applied to the data set A.1 to perform the experiments in Chapter 4.4.2 and 7.2. The attack models annotated with a * are integrated in Flame.

Nachi scan*	The Nachi/Welchia worm was released in 2003. It tests the reachability of hosts using an ICMP scan, which is considered in the attack model. One host of the original data set sends single ICMP packets of 92 bytes to destination IP addresses defined by the following properties: an IP address shift of between 40 — 45 between consecutive scans, a positive or negative shift of 400 every 200 scans and a shift between 45 — 110 every 800 flows. The inter-arrival time of flows is generally around 2 milliseconds but there is a break of around 61 milliseconds after 5 scans
NetBIOS scan*	This is a traditional scan for finding vulnerabilities. The corresponding UDP flows contain a single packet to port 137, with an inter-arrival time generally between 60 and 70 milliseconds. The destination hosts are scanned sequentially while sometimes keeping one IP address. After, between 100 and 200 scans there is a shift in the destination IP address of between 60 and 70
DDoS UDP flood*	A host receives UDP packets on several ports and also sends from multiple IP addresses with various ports. The attack generates a burst of 40 flows followed by a break of between 60 and 120 milliseconds
DDoS TCP flood*	This denial-of-service attack targets web servers running on TCP port 80 with 3 packets of 128 bytes. A burst of 10 packets is sent before a break of between 60 and 120 milliseconds
Stealthy DDoS UDP flood:	Normal UDP flood, having more randomness in flow characteristics (number of packets, size, duration) is applied, while the inter-arrival time can reach 1 second, representing a very stealthy flooding attack, the attack modelled is more generic in order to improve the completeness of the tests
DDoS UDP flood + traffic deletion	This is equivalent to the DDoS UDP flood, but each additional flow originated by the victim has a probability of 0.2 of being deleted due to victim overload
Popup spam*	This kind of spam is similar to the sending of undesired Windows Messenger popups by using UDP port, 1026 and 1027. Only one packet of 925 bytes is needed. The victim IP addresses do not show a regular pattern because two consecutive IP addresses have a gap of 200 addresses. The inter-arrival time is generally lower than 1 millisecond, except that every 200 flows, it is around 64 milliseconds, and every 550 it is 250 milliseconds
Ssh scan + TCP flood	The goal of TCP scan is to probe an ssh server by trying to log in. This is by far the most popular attack that occur in the wild. Each flow contains between 1 and 4 packets. The inter-arrival time oscillates between 1 to 50 milliseconds. The destination IP addresses are scanned in a sequential manner until approximately 400 scans have been executed. There is then a shift of between 200 — 400 IP addresses. In order to test the approach in a real scenario, 5% of the attacks are considered successful and the corresponding victims trigger a TCP flood attack

Table B.1: Attack models generated with Flame

Appendix C

PeekKernelFlows — A visualization method for aggregated Netflow records

C.1 Visualizing aggregated Netflow records

The purpose of the PeekKernelFlows tool is to simplify the detection of anomalies on the network by representing network activity in an intuitive colour scheme. Its main focus is on scanning activities and dominant (e.g. ssh-brute force attack) or long-lasting TCP sessions on the network. An additional feature is that it provides insights into the traffic of a single host. Figure C.1 shows the architecture of PeekKernelFlows. The tool's input is aggregated Netflow records(see section 5.1), which are processed with the kernel function scheme (see section 5.2.1).

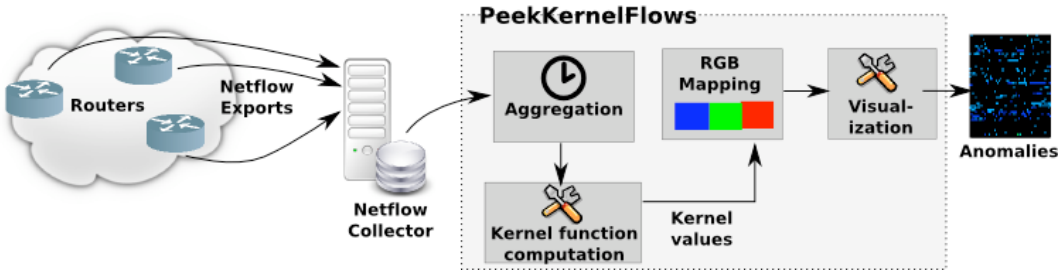

Figure C.1: The PeekKernelFlows monitoring framework

The main task is the mapping of a kernel function value $K_{src,dst}(T_n, T_{n+1})$ onto an RGB [27] colour scheme image.

When traffic fluctuations are large, the similarity between two profiles is quite low, resulting in bits having a low value representation by black. Small profile similarities (K_i) are displayed in bluish colours and high similarities in reddish colours. When all bits are high (implying a very high similarity) the colour tends to white. The colour scheme is given in Figure C.2.

Figure C.2: Colour representation of `PeekKernelFlows`

C.2 The visual representation of PeekKernelFlows

Various parameters can be adjusted by the network operator, for example the monitoring time for aggregated Netflow record profiles (η), the Brightness (B) or the Intensity factor (I). As well as generating a graphical representation, PeekKernelFlows allows statistical information to be retrieved in text-form.

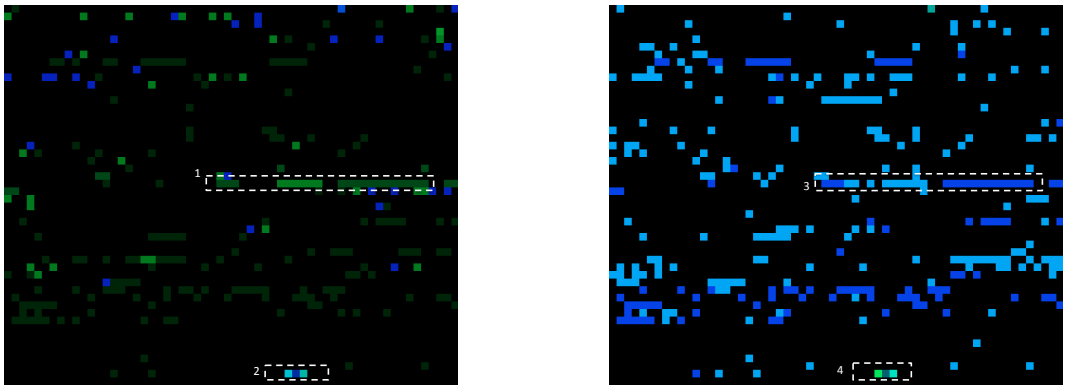

Figure C.3: Figure for source (left) and destination (right) profiles

The sample display of Figure C.3 shows the output generated for the data set from a honeypot, see Appendix A.3. The sample display in Figure C.3 has a resolution of 1,200 × 1,000 pixels and holds 4,000 aggregated Netflow record profiles, the equivalent of four hours monitoring. Relevant patterns can be observed. The three successive green lines on the left frame, annotated by 1, represent ssh brute-force attacks. At the bottom of the source profile representation a coloured line, annotated by 2, can interpreted as the scanning activities of a legitimate honeypot against other hosts. Attackers used nearly the full bandwidth for scanning entire sub-networks. These activities can be observed because they continue over a longer time period.

The destination image gives a more fine-grainedinsight into attacker targets. The same patterns as for the source profile, annotated by 3 and 4 respectively can be observed, representing the communication intensity of both parties. Different patterns as the coloured segments (4) represent the durations that the attackers stay at a single target, and the amount of exchanged traffic. An observation is that dominant TCP sessions, such as ssh brute-force attacks, are shown by bright colours, whereas scanning activities result in dark colours. This can be explained by the kernel function, which plays a dominant topological role in the volume/traffic matching function part.

Appendix D

The proof-of-concept Torinj architecture

A proof of concept malware called Torinj was implemented. As shown in Figure D.1, the tool is composed of three main components: a unmodified Tor client [133], an embedded intercepting proxy, and a hidden C&C (command and control) channel. A standard, unmodified Tor client is integrated into Torinj, providing access to the Tor network layer. Torinj behaves like any other Tor client and provides similar services, including relay or exit functions. It includes a small HTTP proxy used to intercept and relay HTTP requests. Interception and relaying are activated by the attacker using the hidden C&C channel, which relies on the hidden service protocol [108] available in Tor to provide some anonymity [99] to the C&C interface and its user. The attacker accesses the C&C channel of each Torinj bot through the Tor network. For testing we used three machines.

The first machine, M_1, ran an unmodified Tor exit node (v0.2.1.14-rc). M_2 ran BIND [64] (v.9.4.2), as DNS server with tcpdump [131] to capture all DNS queries and responses. M_3 had an Apache [44] web server (v.2.2.6), hosting the transparent file image simulating a malicious payload. Due to legal and ethical considerations, the proof-of-concept did not inject malicious JavaScript payloads like XSS-proxy[1] or BEEF [17]. The machines were synchronized with NTP [88] to ensure precise timestamps. Connected to the Tor network, we set up a web proxy implemented in Perl[2](v.0.23) and extended it to inject tags. Iptables[3] was set up to reroute traffic originated from the Tor exit node to the Perl proxy server via the Internet. The web proxy was used to generate tags.

[1]XSS- proxy: `http://sourceforge.net/projects/xss-proxy/`
[2]Perl HTTP proxy-module: `http://search.cpan.org/dist/HTTP-Proxy/`
[3]`http://www.iptables.org/`

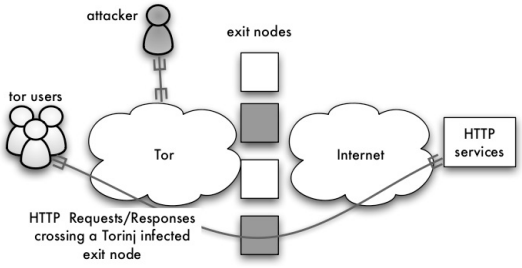

Figure D.1: An overview of the Torinj framework

Bibliography

[1] W. Aiello, C. Kalmanek, P. McDaniel, S. Sen, O. Spatscheck, and J. Van der Merwe. *Analysis of Communities of Interest in Data Networks*. Springer Berlin/Heidelberg, 2005. vol. 3431, pp.83-96.

[2] H. Alvestrand. A Mission Statement for the IETF, 2004. RFC 3935.

[3] J.P. Anderson. Computer security threat monitoring and surveillance, 1980. Technical Report, Fort Washington, `http://csrc.nist.gov/publications/history/ande80.pdf`, last accessed 25/11/2011.

[4] AT&T and Bell Labs. Graphviz- graph visualization. `http://www.graphviz.org/`, last accessed: 01/12/2011.

[5] R.M. Axelrod. *The evolution of cooperation*. Basic Books, 1984.

[6] P. Bahl, P. Barham, R. Black, R. Chandra, M. Goldzmidt, R. Isaacs, S. Kandula, L. Li, J. MacCormick, D.A. Maltz, R. Mortier, M. Wawrzoniak, and M. Zhang. Discovering dependencies for network management. *5th ACM Workshop on Hot Topics in Networking (HotNets)*, 2006.

[7] V. Bahl, R. Chandra, A. Greenberg, S. Kandula, D.A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. *SIGCOMM Comput. Commun. Rev.*, pages 13–24, 2007.

[8] P. Baldi, S. Brunak, Y. Chauvin, C.A.F. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics review, vol.16 no.5 2000, 412-424*, 2000.

[9] S. Basu, F. Casati, and F. Daniel. Toward web service dependency discovery for SOA management. *IEEE Int. Conf. on Services Computing, Vol. 2.*, pages 422–429, 2008.

[10] K. Bauer, D. Maccoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against anonymous systems. *Proceedings of ACM Workshop on Privacy in the Electronic Society*, 2007.

[11] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly extraction in backbone networks using association rules. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 28–34, New York, NY, USA, 2009. ACM.

[12] D. Brauckhoff, A. Wagner, and M. May. FLAME: a flow-level anomaly modeling engine. In *Proceedings of the conference on Cyber security experimentation and test*. USENIX Association, 2008.

[13] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. Classification and Regression Trees. Belmont, California, Wadsworth International Group, 1984.

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

[15] Infographic by Shanghai Web Designers. 60 Seconds Things That Happen On Internet Every Sixty Seconds, 2011. `http://www.go-gulf.com/blog/60-seconds`, last accessed 30/12/2011.

[16] G. Camps-Valls, J.L. Rojo-Alvarez, and M. Martinez-Ramon. *Kernel Methods in Bioengineering, Signal and Image Processing*. Idea Group Pub, 2007.

[17] R. Cannings, H. Dwivedi, and Z. Lackey. Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions. *McGraw-Hill Osborne Media*, 2007.

[18] P.K. Carson and A.B. Yeh. Exponentially weighted moving average (EWMA) control charts for monitoring an analytical process. *Ind. Eng. Chem. Res.*, pages 405–411, 2008.

[19] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM, vol.4, no.2*, 1981.

[20] K. Cho, R. Kaizaki, and A. Kato. Aguri: An aggregation-based traffic profiler. *LNCS 2156*, 2001. QofIS2001, pp.222-242, Springer Verlag.

[21] B.Y. Choi and S. Bhattacharyya. On the accuracy and overhead of Cisco Sampled Netflow. In *In Proceedings of ACM Sigmetrics Workshop on Large Scale Network Inference*, 2005.

[22] B.Y. Choi, J. Park, and Z.L. Zhang. Adaptive packet sampling for flow volume measurement, 2002. University of Minnesota, MA, Technical report, TR02-040.

[23] Cisco. Cisco Sample NetFlow. `http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html`.

[24] B. Claise. Cisco Systems NetFlow Services Export Version 9, 2004. RFC 3954, `http://www.ietf.org/rfc/rfc3954.txt`.

[25] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Information, 2008. RFC 5101, `http://tools.ietf.org/html/rfc5101`.

[26] M. Collins and N. Duffy. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14*, pages 625–632. MIT Press, 2001.

[27] M.F. Cowlishaw. Fundamental requirements for picture presentation. *vol. 26 no.2, pp.101-107*, 1985.

[28] A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. In *42nd Annual Meeting on Association for Computational Linguistics*, 2004.

[29] D. David, P. Niels, L. Christopher, and L.Wenke. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of NDSS'08*, 2008.

[30] D. Dechouniotis, X.A. Dimitropoulos, A. Kind, and S.Denazis. Dependency detection using a fuzzy engine. In *Proceedings of the Distributed systems: operations and management 18th IFIP/IEEE international conference on Managing virtualization of networks and services*, volume 4785 of *Lecture Notes in Computer Science*, pages 110–121. Springer Verlag, 2008.

[31] L.P. Deutsch. GZIP file format specification version 4.3, 1996. RFC 1952, `http://www.gzip.org/zlib/rfc-gzip.html`.

[32] R. Dingledine and N. Dingledine. TOR: The second-generation onion router. In *TOR: The second-generation onion router*, pages 303–320, 2004.

[33] N. Duffield, P. Haffner, E. Krishnamurthy, and H. Ringberg. Rule-based anomaly detection on IP flows. In *International Conference on Computer Communications (INFOCOM)*. IEEE, 2009.

[34] N. Duffield, C. Lund, and M. Thorup. Learn more sample less: Control of Volume and Variance in network measurement. *IEEE Transactions on Information Theory*, 51(5):1756–1775, 2005.

[35] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64:1194–1213, 2007.

[36] C. Estan. Building a better Netflow. In *In Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, 2004.

[37] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *SIGCOMM'03: Proceedings of the 2003 conference on Application, technologies, architectures and protocols for computer communications*, 2003.

[38] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 SIGCOMM*, 2002.

[39] Fielding et al. HTTP/1.1, 1999. RFC2616, `http://www.w3.org/Protocols/rfc2616/rfc2616.html`.

[40] M. Stonebraker et al. C-store: a column-oriented DBMS. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 553–564. VLDB Endowment, 2005.

[41] Eurostat. Eurostat-Data Explorer. `http://appsso.eurostat.ec.europa.eu/nui/setupModifyTableLayout.do`, last accessed: 12/10/2011.

[42] L. Fisher. *Game Theory in Everyday Life*. Basic Books, 2008.

[43] S. Foresti, J. Agutter, Y. Livnat, S. Moon, and R. Erbacher. Visual correlation of network alerts. *IEEE Comput. Graph. Appl.*, 26:48–59, 2009.

[44] The Apache Software Foundation. Apache. `http://www.apache.org`.

[45] J. François, C. Wagner, R. State, and T. Engel. SAFEM: Scablable Analysis of Flows with Entropic Measures and SVM. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2012*. IEEE, 2012.

[46] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Bodies, 1996. RFC2045, `http://www.isi.edu/in-notes/rfc2045.txt`.

[47] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

[48] F. Fusco, M.Ph. Stoecklin, and M. Vlachos. NET-FLi: On-the-fly Compression, Archiving and Indexing of Streaming Network Traffic. In *Proceedings of the VLDB Endowment*, 2010.

[49] T. Gartner. A survey of kernels for structured data. *SIGKDD. Explorations Newsl., Vol. 5, nb.1.*, pages 49–58, 2003.

[50] P. Giura and N. Memon. NetStore: An Efficient Storage Infrastructure for Network Forensics and Monitoring. In *Recent Advances in Intrusion Detection (RAID)*, Lecture Notes in Computer Science. Springer, 2010.

[51] B. Gonzalez-Arevalo, F. Hernandez-Campos, J.S. Marron, and C. Park. Visualization challenges in internet traffic research. *Graphics of a Large Set, Visualizing a Million*, pages 203–226, 2006.

[52] J.R. Goodall and D.R. Tesone. Visual analytics for network flow analysis. In *Cybersecurity Applications and Technology Conference For Homeland Security*, pages 199–204, 2009.

[53] A. Greenwald. Matrix Games and Nash Equilibrium. *Lecture in Game-theoretic Artificial Intelligence, Brown University CS Dep. Providence, US*, 2007.

[54] C.M. Grinstead and J.L. Snell. *Introduction to Probability*. American Mathematical Society, 1997. 2nd Edition.

[55] J. Grossmann, R. Hansen, and P. Petkov. Cross site scripting attacks: Xss exploits and defense, 2007.

[56] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The weka data mining software: An update, 2009. SIGKDD Explorations, Volume 11, Issue 1.

[57] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24:31–34, 2005.

[58] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28, 1979.

[59] R. Heady, G. Luger, and A. Maccabe. The architecture of a network level intrusion detection system, 1990. Technical Report, `http://www.cs.unm.edu/~treport/tr/90/tr.pdf`, last accessed 25/11/2011.

[60] C. Hu, S. Wang, J. Tian, B. Liu, Y. Chen, and S. Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *The IEEE Infocom 2008 proceedings*, 2008.

[61] Y. Hu, D.-M. Chiu, and J.C.S. Lui. Adaptive flow aggregation - a new solution for robust flow monitoring under security attacks. In *NOMS 2006*, pages 424–435, 2006.

[62] V. Igure and R. Williams. Taxonomies of attacks and vulnerabilities in computer systems. *Communications Surveys & Tutorials*, 10:6–19, 2008.

[63] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs. *ACM SIGCOMM Internet Measurement Conference IMC'07, San Diego, CA.*, 2007.

[64] ISC. Bind. `https://www.isc.org/software/bind/`.

[65] H. Jiang, A.W. Moore, Z. Ge, S. Jin, and J. Wang. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, pages 299–304, New York, NY, USA, 2007. ACM.

[66] W. Jinsong. P2P traffic identification based on NetFlow TCP Flag. *Proceedings of the 2009 International Conference on Future Computer and Communication*, pages 700–703, 2009.

[67] R. Kaizaki, K. Cho, and O. Nakamura. Detection of denial of service attacks using Aguri. *International Conference Telecommunications ICT2002, Beijing, China*, 2002.

[68] R. Kaizaki, O. Nakamura, and J. Murai. Characteristics of Denial of Service Attacks on Internet Using Aguri. In *Information Networking*, volume 2662 of *Lecture Notes in Computer Science*, pages 849–857. Springer Berlin / Heidelberg, 2003.

[69] J. Kannan, J. Jung, V. Paxson, and C.E. Koksal. Semi-automated discovery of application session structure. In *SIGCOMM Internet Measurement Conf., IMC'06, Rio de Janeiro, Brazil*, 2006.

[70] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In *ACM Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2005.

[71] R. Kawahara, N. Kamiyama, S. Harada, H. Hasegawa, and S. Asano. Identifying anomalous traffic sources using flow statistics. In *Global telecommunications Conference (GLOBECOM)*. IEEE, 2008.

[72] M. B. Kennel, R. Brown, and H. D.I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A*, 45:3403–3411, March 1992.

[73] L. Khan, M. Awad, and B. Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal*, 16(4):507–521, 2007.

[74] A. Kind, D. Gantenbein, and H. Etoh. Relationship discovery with netflow to enable business-driven management. In *Proceedings of Business-Drive IT Management*, pages 63–70, 2006.

[75] Kaspersky Lab. Kaspersky Security Bulletin 2010. `http://www.securelist.com/en/analysis/204792162/Kaspersky_Security_Bulletin_2010_Statistics_2010`, last accessed: 10/10/2011.

[76] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM'05*, 2005.

[77] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10, 1966.

[78] B.N. Levine, M.K. Reiter, C. Wang, and M. Wright. Timing attacks in low-latency mix-based systems. In *Proceedings of the 8th International Financial Cryptography Conference (FC 2004)*, volume 3110 of *Lecture Notes in Computer Science*, pages 251–265, 2004.

[79] H.F. Lipson. Tracking and tracing cyber-attacks: technical challenges and global policy issues, 2002. Special Technical Report: CMU/SEI-2002-SR-009 , CERT Coordination Center.

[80] C. Livadas, B. Walsh, D. Lapsley, and T. Strayer. Using machine learning techniques to identify botnet traffic. In *Conference on Local Computer Networks*. IEEE, 2006.

[81] B.B. Mandelbrot. *The Fractal Geometry of Nature*. W.H. Freeman and Company, NY, 1982.

[82] S. Marchal, J. François, C. Wagner, and T. Engel. Semantic exploration of dns. In *10th International IFIP TC 6 Conference on Networking*, Lecture Notes in Computer Science. Springer, 2012.

[83] S. Marchal, J. François, C. Wagner, R. State, A. Dulaunoy, T. Engel, and O. Festor. DNSSM: A large scale passive DNS security monitoring framework. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2012*, pages 1–7. IEEE, 2012.

[84] A.A. Markov. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *reprinted Appendix B: R. Howard, Dynamic Probabilistic Systems, vol. 1: Markov Chains*, 1971.

[85] D. Mccoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the Tor network. In *Proceedings of the 8th international symposium on Privacy Enhancing Technologies*, PETS '08, pages 63–76, Berlin, Heidelberg, 2008. Springer-Verlag.

[86] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. *PAM 2004, Antibes Juan-les-Pins, France*, 2004.

[87] R.D. McKelvey and T.R. Palfrey. Quantal response equilibria for normal form games. *Games and Economic Behaviour*, 10:6–38, 1995.

[88] D.L. Mills. Network time protocol (version 4) specification, implementation and analysis, 2010. `http://www.ntp.org/rfc.html`, RFC1305.

[89] M. Molina. Flow export an visualization (Flowviz), 2007. 1st EMANICS summer school, Bremen, Germany, `http:/www.geant2.net/upload/pdf/flowviz_MM-v2.pdf`, last accessed: 17/11/2011.

[90] C. Morariu, T. Kramis, and B. Stiller. DIPStorage: Distributed Storage of IP flow records. In *16th IEEE Workshop on Local and Metropolitan Area Networks LANMAN2008*, pages 108–113, 2008.

[91] C. Morariu, P. Racz, and B. Stiller. Design and Implementation of a Distributed Platform for Sharing IP Flow Records. In *Integrated Management of Systems, Services, Processes and People in IT*, Lecture Notes in Computer Science, pages 1–14. Springer Verlag, 2009.

[92] D.R. Morrison. PATRICIA- - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *ACM Journal*, 15:514–534, 1968.

[93] S.J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2005. IEEE Computer Society.

[94] J. Nash. Non-Cooperative Games. In *Annals of Mathematics*, volume 54, 1951. No. 2.

[95] Arbor Networks. Peakflow SP: Traffic anomaly detection. `http://www.arbornetworks.com/`, last accessed: 29/11/2011.

[96] NIST/SEMATECH. e-handbook of statistical methods, 2003. `http://www.itl.nist.gov/div898/handbook/`, last accessed: 9/11/2011.

[97] G. Nychis, V. Sekar, D.G. Andersen, H. Kim, and H. Zhang. An empirical evaluation of entropy-based traffic anomaly detection. In *Conference on Internet measurement*. ACM SIGCOMM, 2008.

[98] P. Oppenheimer. *Characterizing Network Traffic*. Cisco Press, 2004. 2nd Edition.

[99] L. Overlier and P. Syverson. Locating Hidden Servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2006. IEEE Computer Society.

[100] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 18th Workshop on Privacy in the Electronic Society (WPES)*. ACM Press, 2011.

[101] I. Paredes-Oliva. Portscan Detection with Sampled NetFlow. In *Lecture Notes in Computer Science, vol. 5537*, 2009.

[102] V.A. Patole, V.K. Pachgare, and P. Kulkarni. Self-organizing maps to build intrusion detection system. *International Jounral of Computer Applications*, 1, 2010.

[103] M. Perry. Securing the Tor Network, 2007. Black Hat USA, Defcon, `http://www.freehaven.net/~arma/SecuringTheTorNetwork.pdf`.

[104] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, 2001. RFC 3176, `http://tools.ietf.org/html/rfc3176`.

[105] J. Postel and J. Reynolds. File transfer protocol, 1985. RFC 959, `http://tools.ietf.org/html/rfc959`.

[106] Cisco Press. Entering the Zettabyte Era, 2011. White paper, `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/VNI_Hyperconnectivity_WP.pdf`, last accessed: 11/10/2011.

[107] Privoxy. `http://www.privoxy.org/`.

[108] Tor Project. Tor: Hidden Service Protocol. `http://www.torproject.org/hidden-services.html.en`.

[109] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani. Fast monitoring of traffic subpopulations. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 257–270, New York, NY, USA, 2008. ACM.

[110] A. Rapoport and A.M. Chammah. *Prisoner's dilemma: a study in conflict and cooperation*. University of Michigan Press, 1965.

[111] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16:482–494, 1998.

[112] P. Reynolds, J.L. Wiener, J.C. Mogul, M.K. Aguilera, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. *In 19th ACM Symp. on Operating Systems Principles, SOSP'03, Bolton Landing, New York*, 2003.

[113] P. Reynolds, J.L. Wiener, J.C. Mogul, M.K. Aguilera, and A. Vahdat. WAP5: Black-box performance debugging for wide-area systems. *Int. World Wide Conf. Committee Iw33C2, WWW2006, Edinburgh, Scotland*, 2006.

[114] M. Roesch and Stanford Telecommunications. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX conference on System administration (LISA'99)*, pages 229–238, 1999.

[115] Y. Ruixi, Z. Li, X. Guan, and L. Xu. An SVM-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers*, 12:149–156, April 2010.

[116] D .Katabi S. Kandula and J.-P. Vasseur. Shrink: A tool for failure diagnosis in IP networks. *SIGCOMM'05 Workshops, Philadelphia, PA.*, 2005.

[117] R. Chandra S. Kandula and D. Katabi. What's going on? Learning communication rules in edge networks. *SIGCOMM'08, Seattle Washington.*, 2008.

[118] T. Sauer, J.A. Yorke, and M. Casdagli. Embedology. *J. Stat. Pys.*, 65:579–616, 1991.

[119] M. Saxena and R.R. Kompella. CLAMP: Efficient class-based sampling for flexible flow monitoring. *Comput. Netw.*, 54:2345–2356, 2010.

[120] B. Schölkopf, J.C. Platt, J.C. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, July 2001.

[121] B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[122] C. Simmons, S. Shiva, D. Dasgupta, and Q. Wu. AVOIDIT: A cyber attack taxonomy, 2009. Technical Report: CS-09-003, University of Memphis.

[123] L. A. Smith. Takens' Theorem. `http://dime.lse.ac.uk/lsecats/CourseMaterial/node16.html#foot1428`, last accessed: 11/10/2011.

[124] K.P. Soman, S. Diwakar, and V. Ajay. *Insights into Data Mining - Theory and Practice*. Prentice Hall of India, 2006.

[125] R. Sommer. NetFlow: Information loss or win? In *In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, 2002.

[126] A. Sperotto, R. Sadre, D. F. van Vliet, and A. Pras. A labeled data set for flow-based intrusion detection. In *IP Operations and Management (IPOM 2009)*. Springer, 2009.

[127] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview of IP Flow-based Intrusion Detection. In *Communications Surveys & Tutorials*, volume 12, pages 343–356. IEEE, 2010.

[128] Support Vector Machines (SVMs). Mercer's condition. `http://www.svms.org/mercer/`.

[129] Symantec.com. Symantec Intelligence Report: September 2011. `http://www.symantec.com/connect/blogs/symantec-intelligence-report-september-2011`, last accessed: 10/10/2011.

[130] F. Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381. Springer Verlag, 1981.

[131] tcpdump. `http://www.tcpdump.org/`(accessed on 02/05/09).

[132] A.G. Timothy, K.J. Lai, M.R. Lieberman, and E.C. Price. Browser-based attacks on Tor. In *Proceedings of the 7th international conference on Privacy enhancing technologies*, PET'07, pages 184–199, Berlin, Heidelberg, 2007. Springer-Verlag.

[133] Tor-project. `https://www.torproject.org/about/overview.html.en`.

[134] Torscanner. `http://code.google.com/p/torscanner/`.

[135] T.L. Turocy. A dynamic homotopy interpretation of the logistic quantal response equilibrium correspondence. *Games and Economic Behavior, vol. 51, Issue 2, pp. 243-263, Elsevier*, 2005.

[136] T.L. Turocy. Gambit, 2007. `http://gambit.sourceforge.net/download.html`, last accessed: 03/2011.

[137] V.N. Vapnik. *Adaptive and Learning Systems for Signal Processing, Communication and Control*. Wiley, 1998.

[138] F. Verhulst. *Nonlinear Differential Equations and Dynamical Systems*. Springer Verlag, Berlin Heidelberg, 2000. 2nd Edition.

[139] J.P. Vert. A tree kernel to analyze phylogenetic profiles. In *Proceedings of 10th International Conference on Intelligent Systems for Molecular Biology (ISMB02)*, 2002.

[140] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast IP networks. In *Proceedings of International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WET ICE)*. IEEE, 2005.

[141] C. Wagner and T. Engel. Detecting anomalies in netflow record time series by using a kernel function. In *Proceedings of Autonomous Infrastructure, Management and Security Conference (AIMS) 2012*, Lecture Notes in Computer Science. Springer, 2012.

[142] C. Wagner, J. François, A. Dulaunoy, G. Wagener, R. State, and T. Engel. SDBF: Smart DNS Brute Forcer. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2012*. IEEE, 2012.

[143] C. Wagner, J. François, R. State, and T. Engel. DANAK: Finding the Odd! In *Proceedings of 5th International Conference on Network and System Security*, pages 161–168, Milano, Italy, 2011. IEEE.

[144] C. Wagner, J. François, R. State, and T. Engel. Machine learning approach for IP-flow record anomaly detection. In *LNCS: NETWORKING 2011*, volume 6640, pages 28–39, Valencia, Spain, 2011. Springer.

[145] C. Wagner, G. Wagener, A. Dulaunoy, R. State, and T. Engel. Malware Analysis with Graph Kernel and Support Vector Machines. In *Proceedings of the 4th International Conference on Malicious and Unwanted Software conference (MALWARE)*, pages 63–68. IEEE, 2009.

[146] C. Wagner, G. Wagener, R. State, A. Dulaunoy, and T. Engel. Breaking Tor Anonymity with Game Theory and Data Mining. In *Proceedings of the 4th International Conference on Network and System Security*, pages 47–54. IEEE, 2010. Best Paper Award.

[147] C. Wagner, G. Wagener, R. State, A. Dulaunoy, and T. Engel. Game theory driven monitoring of spatial-aggregated IP-flow records. In *Proceedings of 6th International Conference on Network and services Management*, pages 463–468, Niagara Falls, Canada, 2010.

[148] C. Wagner, G. Wagener, R. State, A. Dulaunoy, and T. Engel. PeekKernelFlows: Peeking into IP Flows. In *Proceedings of 7th International Workshop on Visualization for Cyber Security*, pages 52–57, Ottawa, Canada, 2010. ACM.

[149] C. Wagner, G. Wagener, R. State, A. Dulaunoy, and T. Engel. Breaking Tor Anonymity with Game Theory and Data Mining. *Concurrency and Computation: Practice and Experience*, 2011. John Wiley & Sons, Ltd.

[150] C. Wagner, G. Wagener, R. State, and T. Engel. Monitoring of spatial- aggregated IP-flow records. In *Computational Intelligence in Security for Information Systems 2010*, volume 85 of *Advances in Intelligent and Soft Computing*, pages 117–124. Springer, 2010.

[151] C. Wagner, G. Wagener, R. State, and T. Engel. Digging into IP flow records with a visual kernel method. In *Computational Intelligence in Security for Information Systems 2011*, volume 6694 of *Lecture Notes in Computer Science*, pages 41–49. Springer, 2011.

[152] L. Wang. *Support Vector Machines: Theory and Applications*, volume 177 of *Studies in Fuzziness and Soft Computing*. Springer, 2005.

[153] X. Wang, S. Chen, and S. Jajodia. Network flow watermarking attack on low-latency anonymous communication systems. In *Proceedings of the IEEE Security and Privacy Symposium*, 2008.

[154] X. Wang, J. Luo, M. Yang, and Z. Ling. A novel flow multiplication attack against Tor. In *Proceedings of the 13th IEEE International Conference on Computer Supported Cooperative Work on Design*, 2009.

[155] Y. Wang, M. Hu, B. Li, and B. Yan. Tracking anomalous behaviors of name servers by mining DNS traffic. In *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, volume 4331 of *Lecture Notes in Computer Science*, pages 351–357. Springer Berlin / Heidelberg, 2006.

[156] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36:5–16, 2006.

[157] www.caligare.com. Netflow Portal: Tutorial. `http://netflow.caligare.com/`, last accessed: 17/11/2011.

[158] W. Yu, X. Fu, B. Graham, D. Xuan, and W. Zhao. DSSS-based flow marking technique for invisible traceback. In *Proceedings of the IEEE Security and Privacy Symposium*, 2007.

[159] M. Zalewski. Strange Attractors and TCP/IP Sequence Number Analysis, 2001. `http://lcamtuf.coredump.cx/newtcp/`, last accessed: 25/01/2012.

[160] B-Y. Zhang, J-P. Yin, J-B. Hao, D-X. Zhang, and S. Wang. Using support vector machine to detect unknown computer viruses. *International Journal of Computational Intelligence Research*, 2(1), 2006.

[161] X. Zhu and Z.Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University, 2002. Technical Report CMU-CALD-02-107.

[162] Y. Zhu, Y. Fu, B. Graham, R. Bettati, and W. Zhao. Timing attacks in low-latency mix-based systems. In *Proceedings of Workshop on Privacy Enhancing Technologies (PET)*, 2004.

[163] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. *30th Int. Conf. on Software Engineering, ICSE'08, Leipzig, Germany.*, 2008.

Appendix E

Glossary

AL: Accepted language

CART: Classsification and Regression Trees

Co: Cookie

DANAK: **D**etecting **A**nomalies in **N**etflow records by spatial **A**ggregation and **K**ernel methods

DNS: Domain Name System

(D)DOS: (Distributed) Denial-of-Service

FTP: File Transfer Protocol

Honeypot: A monitored computer or system that aims to attract intruders to attack or compromise the system

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

IETF: Internet Engineering Task Force

IP: Internet Protocol

IPFIX: IP Flow Information Export

ISP: Internet Service Provider

IT: Internet Technology

Malware: Malicious standalone software or embedded code in legitimate applications

ML: Machine Learning

NTP: Network Time Protocol

P2P: Peer-to-Peer

QRE: Quantal Response Equilibrium

RGB: Color space based on red, green and blue

Scanning: Explore network on active hosts

Spam or spam email: Unsolicited bulk mail

SVM: Support Vector Machine

Tor network: Onion routing based network for anonymous communication

TDG: Traffic Dispersion Graph

UA: User Agent

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

Worm: Self-replicating program that propagates through the network, e.g. by means of mass mailing

0-day exploit: A security vulnerability that is exploited on the same day it is disclosed. 0-day, since there are 0 days between attack and detection.

0-sum game: A case in which the gain/loss of a player is dependent of the losses/gains of other players. By adding up the all gains and subtracting all losses the sum will be zero.

Appendix F

About the author

Cynthia Wagner was born in Esch-sur-Alzette, Luxembourg, on August 2nd, 1982. She qualified with an Industrial Engineer degree in Applied Computer Science from the University of Luxembourg, FSTC in 2006. Then in 2008, she graduated as a Master in in Computer Science (MICS) with a main focus on Intelligent Systems. At the end of 2008, she enrolled as a PhD candidate in Network Security and Anomaly Detection at the SECAN-LAB, part of the University of Luxembourg's Computer Science and Communications Research Unit (CSC). Here, she worked for the EU-project entitled EFIPSANS for two years. Her research has covered broad areas of network security such as Netflow record, Malware and DNS analysis. Her research has also covered the analysis of flows in anonymous networks.

List of Publications

- C. Wagner, G. Wagener, R. State and T. Engel, *Malware Analysis with Graph Kernel and Support Vector Machines*, [145], 4th International Conference on Malicious and Unwanted Software (MALWARE), pp. 63-68, Montreal, QC, October, 2009.
- C. Wagner, G. Wagener, R. State, A. Dulaunoy and T. Engel, *Breaking Tor Anonymity with Game Theory and Data Mining*, [146], 4th International Conference on Network and System Security (NSS), pp. 47-54, Melbourne, Australia, September 2010. (Best Paper Award)
- C. Wagner, G. Wagener, R. State, A. Dulaunoy and T. Engel, *PeekKernelFlows: Peeking into IP Flows*, [148], 7th International Symposium on Visualization for Cyber Security (VizSec), pp. 52-57, Ottawa, ON, September 2010.
- C. Wagner, G. Wagener, R. State, A. Dulaunoy and T. Engel, *Game Theory driven monitoring of spatial-aggregated IP flow records*, [147], 2010 International Conference on Network and Service Management (CNSM), Mini-conference, pp. 463-468, Niagara Falls, ON, October 2010.
- C. Wagner, G. Wagener, R. State and T. Engel, *Monitoring of spatial aggregated IP-Flow records*, [150],3rd International Conference on Computational Intelligence in Security for Information Systems (CISIS), AISC, Springer, vol. 85/2010, 117-124, Leon, Spain, November 2010.
- C. Wagner, J. François, R. State and T. Engel, *Machine Learning Approach for IP-Flow Record Anomaly Detection*, [144], 10th International IFIP TC 6 Conference on Networking, LNCS, vol. 6640/2011, pp. 28-39,Valencia, Spain, May 2011.
- C. Wagner, G. Wagener, R. State and T. Engel, *Digging into IP Flow Records with a Visual Kernel Method*, [151], 4th International Conference on Computational Intelli-

gence in Security for Information Systems (CISIS), LNCS, vol. 6694/2011, pp. 41-49, Torremolinos, Spain, June 2011.

- C. Wagner, G. Wagener, R. State, A. Dulaunoy and T. Engel, *Breaking Tor Anonymity with Game Theory and Data Mining*, [149], Concurrency and Computation: Practice and Experience, John Wiley and Sons, Ltd, 1532-0632, July 2011.
- C. Wagner, J. François, R. State and T. Engel, *DANAK: Finding the Odd!*, [143], 5th International Conference on Network and System Security (NSS), pp. 161-168, Milano, Italy, September 2011.
- J. François, C. Wagner, R. State and T. Engel, *SAFEM: Scalable Analysis of Flows with Entropic Measures and SVM*, [45], IFIP NOMS 2012, Maui, Hawaii, April 2012.
- C.Wagner, J. François, R. State, A. Dulaunoy and T. Engel, *SDBF: Smart DNS Brute-Forcer*, [142], IFIP NOMS 2012, Maui, Hawaii, April 2012.
- S. Marchal, J. François, C. Wagner, R. State, A. Dulaunoy, O. Festor and T: Engel: *DNSSM: A Large Scale Passive DNS Security Monitoring Framework*, [83], IFIP NOMS 2012, Maui, Hawaii, April 2012.
- S. Marchal , J. Franois, C. Wagner, T. Engel, *Semantic Exploration of DNS*, [82], 10th International IFIP TC 6 Conference on Networking, LNCS, Prague, CZ, May 2012.
- C.Wagner and T. Engel, *Detecting Anomalies in Netflow Record Time Series by Using a Kernel Function*, [141], Autonomous Infrastructure, Management and Security Conference (AIMS) 2012, LNCS, Luxembourg, June 2012.