Flow-Based Intrusion Detection

Anna Sperotto

Graduation committee:

Chairman:	Prof. dr. ir. Anton J. Mouthaan
Promoter:	Prof. dr. ir. Boudewijn R. Haverkort
Assistant promoter:	Dr. ir. Aiko Pras

Members:

Prof. dr. Pieter H. Hartel	University of Twente
Prof. dr. Sandro Etalle	University of Twente
Prof. dr. Michel R.H. Mandjes	University of Amsterdam
Prof. dr. Gabi Dreo Rodosek	University of the Federal Armed Forces, Munich
Prof. dr. Jürgen Schönwälder	Jacobs University Bremen
Dr. Philippe Owezarski	Centre National de la Recherche Scientifique, Toulouse

CTIT

CTIT Ph.D.-thesis Series No. 10-180
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, NL – 7500 AE Enschede

ISSN 1381-3617
ISBN 978-90-365-3089-7

Publisher: Wöhrmann Print Service
Cover design: Gabriella Sperotto

FLOW-BASED INTRUSION DETECTION

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
Prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op donderdag 14 oktober 2010 om 15.00 uur

door

Anna Sperotto

geboren op 3 november 1982
te Belluno, Italië

Dit proefschrift is goedgekeurd door:
Prof. dr. ir. Boudewijn R. Haverkort (promotor)
Dr. ir. Aiko Pras (assistent-promotor)

Abstract

The spread of 1-10Gbps technology has in recent years paved the way to a flourishing landscape of new, high-bandwidth Internet services. As users, we depend on the Internet in our daily life for simple tasks such as checking e-mails, but also for managing private and financial information. However, entrusting such information to the Internet also means that the network has become an alluring place for hackers. To this threat, the research community has answered with an increased interest in intrusion detection. With the number of attacks almost exponentially increasing, and the attackers' motivations moving from ideological to economical, the researchers' attention is focused on developing new techniques to timely detect intruders and prevent damage. Our studies in the field of intrusion detection, however, made us realize that additional research is needed, in particular: the creation of shared data sets to validate Intrusion Detection Systems (IDSs) and the development of automatic procedures to tune the parameters of IDSs.

The contribution of this thesis is that it develops a structured approach to intrusion detection that focuses on (i) shared ground-truth data sets and (ii) automatic parameter tuning. We develop our approach by focusing on network flows. Flows offer an aggregated view of network traffic, by reporting on the amount of packets and bytes exchanged over the network. Therefore, flows drastically reduce the amount of data to be analyzed. In this thesis, we aim at detecting anomalies in flow-based time series, describing how the number of flows, packets and bytes changes over time.

Ground truth data sets are fundamental in the development phase, for validation purposes and, if publicly available, for comparison between different IDSs. We attack the problem of ground truth generation in two complementary manners.

First, we obtain ground truth information for flow-based intrusion detection by *manually* creating it. We do so by means of a honeypot-based data collection and monitoring setup, specifically tuned to (i) offer an attracting platform for attackers, and (ii) include enhanced logging capabilities to support the

labeling of the collected data. The outcome of our research has been a publicly released flow-based labeled data set. To the best of our knowledge, no such data set already exists.

Second, we generate ground truth information in an *automatic* manner. We do this by generating artificial flow, packet and byte time series for benign and attack traffic. In this thesis, we rely upon Hidden Markov Models (HMMs), which allow for probabilistic and compact representations of flow-based time series and can be used for generation purposes.

Finally, we approach the problem of automatic tuning of IDSs. The performance of an IDS is governed by the trade-off between detecting all anomalies (at the expense of raising alarms too often), and missing anomalies (but not issuing many false alarms). We developed an optimization procedure that aims to mathematically treat such trade-off in a systematic manner, by automatically tuning the system parameters.

Contents

CHAPTER 1

Introduction

As users, we are day by day increasingly dependent on the Internet. In Figure 1.1, we show the world average number of Internet users per 100 inhabitants in 2009, as reported by the International Telecommunication Union (ITU) [73]. Among the developed countries, Internet usage is pervasive. For example, the Netherlands has on average more than 89 Internet users per 100 inhabitants. If we consider that 90% of the Dutch population is between 5 and 80 years old[1], we may say that virtually everybody in a suitable age is nowadays using the Internet. Internauts all over the world are connected at work and at home, and e-mails have almost completely substituted envelopes and stamps. We browse the latest news on our phone and we smile when our favorite café offers free WiFi. Moreover, and for some a bit worryingly, the Internet is the virtual space in which we are nowadays used to manage our money and our personal data. We depend on the Internet not only at the personal level. Internet is by now such a ubiquitous technology that almost any company, university, governmental organization or, even, critical infrastructure such as power plants or water treatment plants, are globally connected.

At the basis of the situation that we have described, there is certainly the technological push at infrastructure level which we witnessed in the last decade. Nowadays an access speed of 1-10 Gbps is not unusual. There are huge investments being made in the creation of optical fiber infrastructures, such as, for example, Fiber-to-the-Home [49]. Where fiber was in the past a privilege reserved only to large Internet service providers and research institutes, now we have Gbps connectivity to our own neighborhood. Since bandwidth for wired connections is definitely not a problem anymore, more and more high-bandwidth services are being offered to the users.

We should realize, however, that, behind the scenes, this technological im-

[1] Source: Centraal Bureau voor de Statistiek, 2010

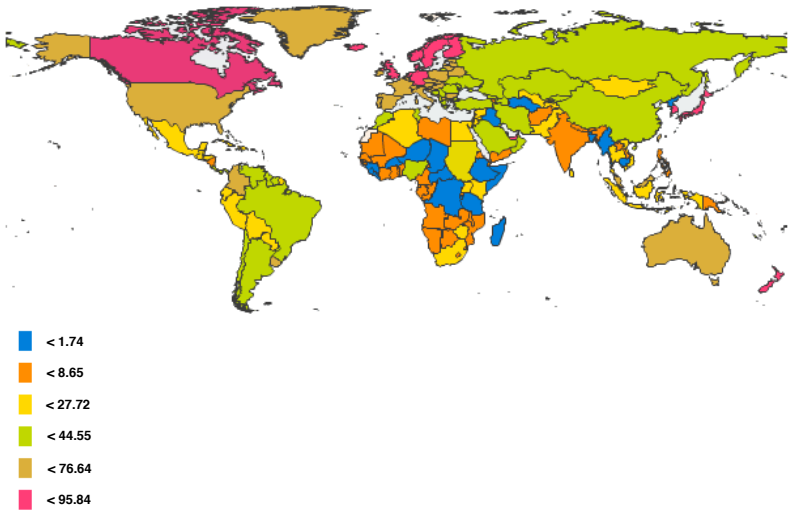


Figure 1.1: Percentage of Internet users in 2009 (source: ITU ICT Eye).

provement has paved the way to new challenges. First, the amount of Internet traffic, as well as the line speed, continues to grow. A university network, for example, reaches traffic averages in the order of hundreds of Mbps, with high activity peaks in the order of Gbps. On backbone networks, as for example Internet2 [74], the USA research and education backbone, the throughput is even higher, as we show in Figure 1.2. Internet2 shows an almost exponential growth over the period 2002-2010, with occasional peaks up to 50 Gbps. Such amounts of data need to be managed and monitored, and new strategies to cope with an average load of multiple Gbps have to be developed.

Second, the number of attacks does also continue to grow. The reason behind this is in itself very simple: attacks are getting economically more and more profitable. Let us take SPAM as an example. We all receive unsolicited mail, ranging from more or less obvious commercials to phishing messages specifically designed to resemble legitimate mails. SPAM represents a form of pervasive advertisement, but it is, increasingly often, also a means to gather personal information (by mimicking a message from your bank, for example). Experts estimate that 90% of the worldwide sent mail messages are SPAM

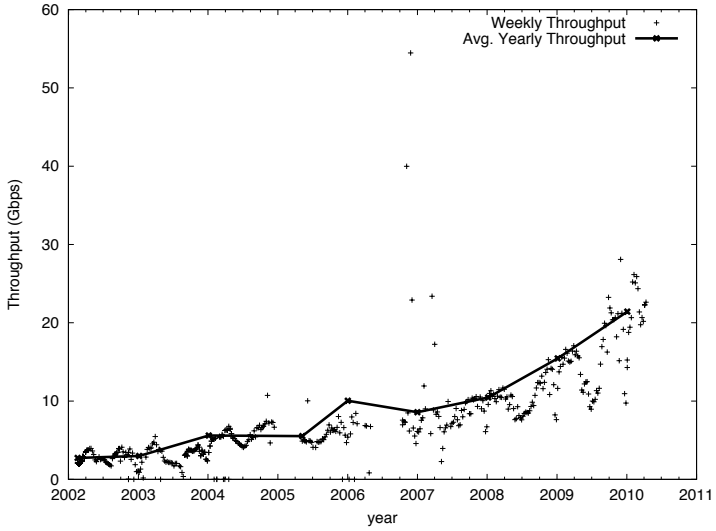


Figure 1.2: Network throughput (Gbps) for the network Abilene [74].

[144, 134, 150], and the phenomenon does not seem to decrease.

The combination of growing network load and attack frequency is challenging if we are aiming to effectively detect intruders. The network monitoring community reacted to the ever growing amount of data by focusing on network flows, rather than individual network packets [19, 121, 22]. A flow is defined as a set of packets that have common properties, as, for example, having the same source and the same destination (see Section 2.1.1). Measuring flows offers an aggregated view of traffic information and drastically reduces the amount of data to be analyzed. Flows are therefore a possible solution to cope with scalability issues in IP monitoring. However, from a security perspective, we do not yet see a definite answer to the problem of intrusion detection in situations, as high-speed networks, in which the traditional packet-based solutions may no longer be feasible. Flows therefore appear as a promising approach that may lead to improved results in the field of intrusion detection in high-speed networks.

In the following, we first introduce the topics of intrusion detection and flow-based intrusion detection (Section 1.1), and then point out what we consider the main open issues in the field (Section 1.2). The analysis of such open

issues leads us to present the goal and the research questions addressed in this thesis (Section 1.3). Finally, we present the outline of this thesis (Section 1.4).

1.1 Intrusion Detection

According to Krügel *et al.* [81], "intrusion detection is the process of identifying and responding to malicious activities targeted at computing and network resources". An *intrusion attempt*, also named *attack*, refers to a sequence of actions by means of which an intruder attempts to gain control of a system. Citing Halme *et al.* [66], the aim of an *Intrusion Detection System (IDS)* is therefore to *discriminate intrusion attempts and intrusion preparation from normal system usage*.

Since research on intrusion detection started in the 1980s, many flavors of IDSs have been proposed (see Chapter 2). Traditional IDS taxonomies, such as Debar *et al.* [32, 33] and Axelsson [5], classify IDS according to several characteristics, e.g., the data the IDS analyzes (log, application or network data), the type of analysis (real-time or offline) or the type of data processing (centralized or distributed). However, the most widely known classification feature regards how an IDS identifies intrusions, i.e., *misuse-based* or *anomaly-based* IDS.

A misuse-based IDS, also known as signature-based or knowledge-based IDS, performs detection by comparing new data with a knowledge base of known attacks. An alarm is generated if a previously specified pattern is recognized. A famous example of this technique is the use of pattern matching algorithms in packet payload analysis (e.g., Snort rules [123]). The strength of a misuse-based IDS lies in being highly accurate, i.e., it rarely raises an alarm due to a benign activity. On the other hand, its effectiveness depends on the completeness of the signatures. Therefore, a misuse-based system cannot recognize new attacks.

An anomaly-based IDS, also known as behavior-based IDS, compares input data with a *model of normality*, which describes the expected behavior of the system. A significant deviation from the model is marked as an anomaly. Examples of suitable techniques for the creation of a behavioral model are neural networks, statistical analysis techniques and Markov models, as surveyed in the works of Patcha *et al.* [115] and Estevez-Tapiador *et al.* [45]. The main advantage of an anomaly-based IDS is that it can potentially detect also attacks that have never been seen before [114]. Note however that, while an attack is often an anomaly, there exist cases in which events that deviate from the model of normality are not necessarily malicious. The classical example is a so-called flash crowd, i.e., a surge of traffic towards a web-server, caused by the public

release of some content, for example a new Linux release [8]. While dealing with the overwhelming amount of traffic, the web-server is likely to become unresponsive. A flash crowd is usually not malicious, but it can be mistaken for a Denial of Service (DoS) attack.

An IDS aims to discriminate between intrusion attempts and normal activities. In doing so, however, an IDS can introduce classification mistakes. A *false positive* is a benign input for which the system erroneously raises an alert. A *false negative*, on the other hand, is a malicious input that the IDS fails to report. The correctly classified input data are usually referred to as *true positives* (attacks) and *true negatives* (normal traffic). There is a natural trade-off between detecting all malicious events (at the expense of raising alarms too often, i.e., having high false positives), and missing anomalies (i.e., having high false negatives, but not issuing many false alarms). We graphically show this trade-off in Figure 1.3. It is usually the case that we can control the system performance by tuning specific IDS parameters, as schematically suggested in Figure 1.3 by the dashed line: the region to the left of the line results in low false positive rate, but an increasing false negative rate; similarly, the region to the right favors a low false negative rate, but it has a higher false positive rate. Which component of the trade-off is more important is a case-specific decision, and ideally, we would want to optimize both components. We might want to identify all malicious attempts, because this would make our network safer. However, this would be of no use if the number of alerts would overload the IT specialist responsible for handling them.

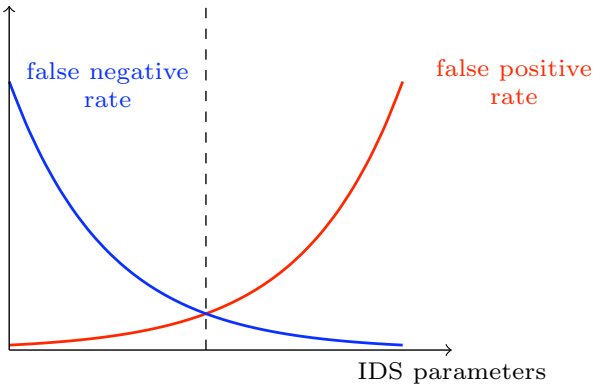


Figure 1.3: Trade-off between false positive and false negative rates.

1.1.1 Flow-based Intrusion Detection

An IDS would need to be able to handle the growing number of attacks, the rise in the amount of traffic as well as the increase in line speed [137]. However, researchers assess the payload-based IDSs processing capability to lie between 100 Mbps and 200 Mbps when commodity hardware is used [82, 55], and close to 1 Gbps when dedicated hardware is employed [31, 151]. Well-known systems like Snort [123] and Bro [116] exhibit high resource consumption when confronted with the overwhelming amount of data found in today high-speed networks [36]. In addition, the spread of encrypted protocols poses a new challenge to payload-based systems. An example is described in the work of Taleb *et al.* [46, 145], where the authors propose an IDS based on per-packet inspection that relies only on header information in order to identify misuses in encrypted protocols.

Given these problems, flow-based approaches seem to be promising candidates for intrusion detection research. Flows are monitored by specialized accounting modules usually placed in network routers. These modules are responsible for exporting reports of flow activity to external collectors (see Section 2.1). Flow-based IDSs will analyze these flows to detect attacks. Compared to traditional IDSs, flow-based IDSs have to handle a considerably lower amount of data. For example, in the case of the University of Twente (UT) network, we calculated that the ratio between packets exported by NetFlow (containing the flow records) and the packets on the network is on average equal to 0.1%. Moreover, considering the network load measured in bytes, the overhead due to Netflow is on average 0.2%. Flow-based intrusion detection is therefore a logical choice for high-speed networks.

The question remains whether flows do carry enough information, compared to payload inspection, to be useful for intrusion detection. Flow measurements are by nature aggregated information. They, therefore, do not provide the detection precision of payload-based inspection. However, information on the safety status of a monitored network can be obtained from flows, for example by studying evolutions of network interactions. Flow measurements provide an aggregated view of the data transferred over the network and between hosts, in terms of number of packets, bytes and measured flows themselves. In this context *time series* are a powerful tool to describe network evolution patterns. In network monitoring, time series are usually accepted as the natural way to look at network traffic, i.e., in a streaming manner. The popularity of this approach is reflected by the widespread use of tools like, among others, Multi Router Traffic Grapher (MRTG) [111] and, lately, the Netflow Sen-

sor (NfSen) suite [64]. NfSen, in particular, has been specifically developed to collect and visualize flow data. In this thesis, we combine time series analysis with flow-based intrusion detection. We aim to investigate how it is possible to describe anomalous events, by mathematically considering the evolution over time of flows, packets and bytes.

In any case, it is important to underline that flow-based intrusion detection is not supposed to substitute the packet-based one, but rather complements the approach by allowing early detection in environments in which payload-based inspection is not scalable.

1.2 Open issues in Intrusion Detection

Flow-based intrusion detection is a relatively new research field, the first contributions of which, as we will see in Chapter 3, date back to the first decade of this century. Flow-based intrusion detection builds upon previous experience in intrusion detection, but faces different challenges, for example the absence of payload. Like payload-based intrusion detection, however, it has a distinct problem-oriented attitude: with the number of attacks almost exponentially increasing [137], and the attackers' motivations moving from ideological to economical, the researchers' attention is focused on developing new techniques to timely detect intruders and prevent damage. Our studies in the field of flow-based intrusion detection, however, made us realize that other research directions are possible, especially if we consider that flow-based intrusion detection is a discipline placed at the intersection between network monitoring and security. Here we point out two problems that, in our opinion, need urgent attention.

First, in the network monitoring community, the importance of sharing network traces for development, validation and comparison purposes is well understood. Examples are repositories like Caida DATA [147], DatCat [30], Crawdad [25], Simpleweb [132] and the MOME project [103]. In intrusion detection, data sets also play a central role, with the difference being that researchers usually evaluate new approaches by testing them on data sets for which the malicious or benign nature of the data is known. We refer to these traces as *ground-truth* data sets. Considering the current situation, research on IDS generally suffers from a lack of shared ground-truth data sets. High-quality ground-truth data sets are time consuming to create and often rely on privacy-sensitive data. Therefore, most publications use non-public traffic traces for evaluation purposes, and we have no knowledge of any publicly available ground-truth

flow-based data set. This is generally an obstacle to the comparison of different IDS approaches.

Second, we may certainly say that each new approach to intrusion detection brings us a step forward towards having a safer network. However, we often forget that, for an intrusion detection system to be deployed, we keep "the man in the loop". With this, we mean that, once a new approach has been developed, it is generally assumed that expert IT personnel will take care of the operational aspects in a specific network. This assumption is motivated by the fact that any IDS, to be effective, needs to be tuned according to the specific characteristics of the monitored network. However, this also means that only the expertise of the security operator ensures us that an IDS is tuned to be used in the best way possible. We wonder therefore whether the problem of parameter tuning for IDS could not be treated in a more systematic manner, leaving the security expert free to focus on high-level policies (such as maximum allowed false positive rate, or relative importance assigned to false positive and false negative rates) instead of understanding how the IDS works. This reasoning relates to the wider concept of *autonomic management*. Citing Horn [69], who proposed in 2001 the idea of the Autonomic Computing Initiative (ACI), an autonomic system allows "users to concentrate on what they want to accomplish rather than figuring how to rig the computing systems to get them there." Applied to an IDS, this means that we want to achieve the desired (optimal) performance without knowing the details of the underlying system. The task of the security expert would therefore be to specify security policies, and the system should implement these policies to provide optimal detection.

1.3 Goal, research questions and approach

In light of the reasoning so far, the goal of this thesis is to develop a structured approach to intrusion detection that focuses on (i) shared ground-truth data sets and (ii) automatic parameter tuning. We develop our approach in the context of detecting anomalies using flow data and time series. To achieve this goal, we will answer the following research questions:

Research Question 1: What is the state of the art in the field of flow-based intrusion detection?

Research Question 2: Are time series a good approach for intrusion detection at flow level? If yes, how can they be exploited best?

Research Question 3: How can we determine ground-truth information for flow-based intrusion detection?

Research Question 4: How can we tune the parameters of a flow-based IDS based on high-level policies?

Flow-based intrusion detection is a relatively new research field that has only recently attracted the researchers' attention. The objective of Research Question 1 is therefore to identify the main contributions and research trends in flow-based intrusion detection so far. In order to do this, we perform a literature study that presents a structured overview of the research field.

The starting point for the research in this thesis are real network measurements. Therefore, Research Question 2 aims at gaining domain knowledge about how anomalies look like "in the wild", in cases where only flow data are available. To answer this question, we perform extensive data analysis on flow data from the University of Twente and SURFnet, the Dutch national research and education network. We focus, in particular, on how anomalies affect metrics as the number of flows, packets and bytes, in the form of time series. As a result, we acquire an in-depth understanding of which of the aforementioned metrics is suitable for intrusion detection.

The goal of Research Question 3 is to shed light on a basic problem that all IDSs as well as other classification systems have in common: the need for labeled data, or *ground truth*. We are not able to evaluate a system if we do not know the nature of the data it has processed. Ground-truth data sets are fundamental in the development phase, for validation purposes and, if publicly available, for comparison between different IDSs. We identify two viable solutions to answer Research Question 3. First, we obtain ground-truth information for flow-based intrusion detection by *manually* creating it. In order to do so, after identifying the requirements a ground-truth data set should meet, we inspect possible network infrastructures suitable for the task. Following the outcome of this phase, we proceed by creating a flow-based labeled data set by tracking malicious activities on a monitoring point that has been optimized for collecting security information. Finally, we release the flow-based data set in anonymized form for public use.

A second approach to answer Research Question 3 is to create ground truth in an *automatic* manner. To answer this part of Research Question 3, we model malicious and benign flow information at time series level. We rely upon the well-known framework offered by HMMs [122], since they allow a probabilistic, compact representation of (flow-based) time series and they can be used for

generation purposes. We validate our approach by verifying that the generated time series statistically approximate the original ones.

Finally, Research Question 4 deals with the topic of tuning the parameters of an IDS according to high-level policies. The general expectation for an IDS is to have a high true positive rate while maintaining a low false positive rate. However, the tuning of the system parameters is typically left to the experience and the skill of IT personnel and little work has been done in proposing structured approaches to address this problem. To answer Research Question 4, we propose an optimization procedure for tuning the parameters of a flow-based, time-series based intrusion detection system. We approach the parameter tuning by solving a non-linear optimization problem that addresses the trade-off between true positive and false positive in a probabilistic manner. Moreover, we explicitly take into account the fact that the optimal solution may depend on the situation, meaning that it can change according to specific network and user requirements. Finally, we support our approach by extensively validating it on synthetic and original data sets.

1.4 Thesis outline

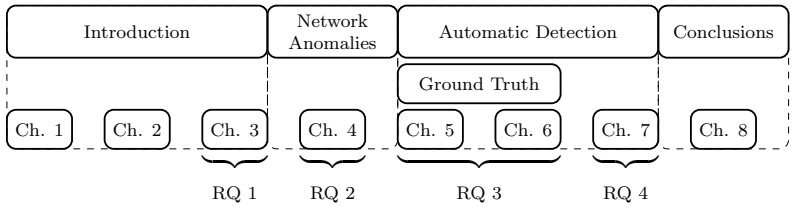


Figure 1.4: Thesis structure.

The thesis structure closely follows the Research Questions. Figure 1.4 presents a sketch of the thesis outline, where the chapters have been grouped according to their topic and the research question they will answer. The remainder of this thesis is therefore organized as follows:

- **Chapter 2** ("Background") provides the background information needed for the research presented in this thesis. The chapter covers three topics: *flows* and their creation and collection process; *attacks* and *intrusion detection*, in the form of basic taxonomies.

- **Chapter 3** ("State of the art in flow-based intrusion detection") presents a structured overview of the main contributions so far in the field of flow-based intrusion detection. Moreover, the added value of Chapter 3 lies in the classification of the current solutions according to the intrusion detection taxonomy in Chapter 2. This analysis allows us to point out the main trends in the intrusion detection field. Chapter 3 therefore answers Research Question 1.
- **Chapter 4** ("Detecting attacks using flow data") presents an extensive data analysis on flow-based traffic time series. The analysis is conducted by performing a protocol breakdown of the total traffic. This operation makes it easier to detect the presence of anomalies. Moreover, the chapter analyzes which numerical flow information, in terms of flow, packet and byte time series, is needed for intrusion detection purposes. Chapter 4 answers Research Question 2.
- **Chapter 5** ("Manual ground truth generation") explores the possibility to generate ground truth for flow-based intrusion detection in a manual manner. First, Chapter 5 investigates which requirements should be put upon a flow-based labeled data set. Second, it studies which network infrastructures are suitable for collecting meaningful data for the data set creation. Finally, the chapter describes how to add ground-truth information to the collected data. Chapter 5, together with Chapter 6, answers Research Question 3.
- **Chapter 6** ("Automatic ground truth generation") investigates how to create ground truth in an automatic manner. The chapter presents models for malicious and benign Secure Shell Protocol (SSH) traffic time series as a case-study. The description of the models is followed by a validation step, where we verify that the generated labeled time series respects the main statistical characteristics of the original traffic. Chapter 6 completes the research needed to answer Research Question 3.
- **Chapter 7** ("Tuning intrusion detection systems ") focuses on automatic parameter tuning. The chapter investigates, by means of an example in the form of a simple anomaly-based, probabilistic IDS, how we can tune the IDS parameters. The tuning procedure aims to approach the trade-off between true positive rate and false positive rate as an optimization problem. Finally, Chapter 7 presents an extensive validation of the proposed optimization procedure by means of synthetic and real labeled time series. Chapter 7 answers Research Question 4.

- Finally, in **Chapter 8** ("Conclusions") we close this thesis by drawing our conclusions and identifying directions for future work.

CHAPTER 2

Background

In Chapter 1 of this thesis, we outlined the trends in Internet development and we motivated the need of having fast and scalable intrusion detection. In this chapter, we provide background information on the topics that play a central role in this thesis: *flows*, *attacks* and *intrusion detection*. In this thesis, we base our research on network flows. Section 2.1 therefore gives the definition of flows and describes the flow creation and collection process. The section also presents an overview of the flow export protocols and of the sampling methods applied to create flows. Since it is also important to be aware of which attacks are threatening our networks, we introduce, in Section 2.2, a basic taxonomy of network attacks. Similarly, in Section 2.3, we provide a taxonomy of intrusion detection systems.

2.1 Network flows

In the last decade, flows have become quite popular in IP network monitoring, since they help to cope with the scalability issues introduced by the increasing network speeds. Nowadays all major vendors offer flow-enabled devices, such as, for example, Cisco routers with Netflow [19]. The Internet Engineering Task Force (IETF) is currently working on an IP flow standard, IP Flow Information eXport (IPFIX).

2.1.1 Flow definition

In the literature, several flow definitions can be found [19, 52, 51, 21]. We present the definition of *IP flow* as it is described by the IPFIX working group within the IETF [121, 22]:

> "A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties."

In the IPFIX terminology, the *common properties* are called *flow keys*. An example of flow keys commonly used for characterizing a flow is:

$$(Source\ IP, Destination\ IP, Source\ port, Destination\ port, IP\ protocol).$$

Aggregated views on the network traffic can be obtained by choosing coarser grained flow definitions, according to the need of the network administrator. An overview of this process is given by Fioreze *et al.* [52, 51]. It is analogously possible to have more detailed flow definitions. For example, additional fields can be introduced as extensions for diverse applications. These additional fields are described in IPFIX and they have been recently included in Flexible Netflow [18].

> **Note:** There are important differences between *flows* and *Transmission Control Protocol (TCP) connections*. A TCP connection determines a pair of flows: one from the initiator of the connection to the destination, and one from the destination to the initiator. However, a flow should not necessarily be due to the TCP protocol. For example, a stream of User Datagram Protocol (UDP) packets between a source host A and a destination host B will result in a flow. Moreover, a flow does not have size restrictions: each communication between source and destination hosts will generate a flow, even if a single packet has been exchanged. Traditionally, flows are also unidirectional, whereas TCP connections are by definition bidirectional. However, IETF has recently introduced a definition of bidirectional flows [148], since bidirectional data can further improve export and collection efficiency.

2.1.2 The metering and collection process

Monitoring flows entails a two-step process: *flow exporting* and *flow collection*. These tasks are respectively performed by two components: the *exporter* and the *collector*. Figure 2.1 presents an overview of the metering and collection process.

The *flow exporter*, or *monitoring point*, is usually a router or a different flow-enabled device. It is responsible for the *metering* process, i.e., the creation of

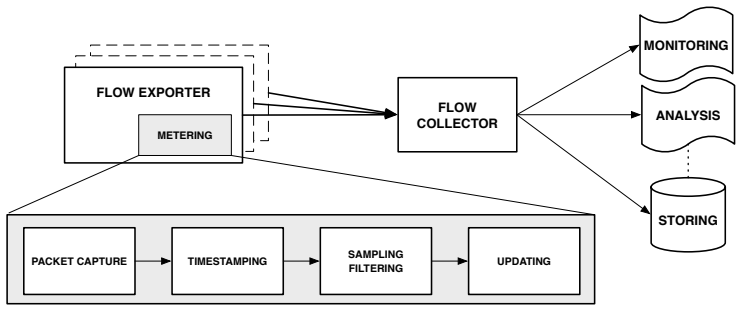


Figure 2.1: IP Flow exporting and collecting architecture [22, 19].

flow records from observed traffic. The *flow exporter* extracts the packet header from each packet passing through the monitoring interface. Each packet header is marked with the *timestamp* when the header was captured. The header is then processed by a *sampling-filtering* module, where it can be *sampled* (see Section 2.1.4) and *filtered* according to specific administrative requirements (e.g., a specific protocol or IP range). The final step is the *update* module. Each incoming packet header triggers an update to a flow entry in the *flow cache*. If there is no flow matching the packet header, a new flow entry is created.

A flow record is exported to the flow collector when it is considered *expired*. In the case of Cisco NetFlow [19] and similarly in IPFIX [121, 22], a flow expires when:

- the flow was idle (no packets belonging to the flows have been observed) for a time interval longer than a given threshold. This threshold is known as *inactive timeout*. The default value for the inactive timeout for Cisco Netflow [19] is 15 seconds. However, it can be tuned according to the operators' requirements. At the University of Twente and for SURFnet, for example, the inactive timeout is set to 30 seconds. Géant [59] uses an inactive timeout of 60 seconds;

- the flow reaches a maximum allowed lifetime, known as *active timeout*. For Cisco Netflow [19], the active timeout is 30 minutes, but our experience showed that shorter timeouts are also common. At the University of Twente, for example, the active timeout is set to one minute. SURFnet [143] and Géant [59] both use an active timeout of 5 minutes;

- the FIN or RST flags have been seen in a TCP flow, indicating the end of

a TCP connection;

- the flow-cache memory is exhausted. In this case, a subset of the flows in the cache is marked as expired and exported to the collector. Least Recently Used algorithms may be used to free the flow-cache memory, as well as heuristic algorithms.

The aim of the *flow collector* is to receive the flow records from the flow exporter and to store them in a form suitable for further monitoring or analysis. Examples of flow collector and analysis tools are `flow-tool` [54], `nfdump` [63], sFlowTrend [72], IsarFlow [75] and DiCAP [106, 107].

2.1.3 Flow export protocols

A *flow export protocol* defines how expired flows are transferred by the exporter to the collector. The information exported to the collector is usually referred as *flow record*.

> **Note:** The terminology *flow* and *flow record* usually raises the question about the actual difference between the two. A flow is the complete unidirectional stream of packets between a source and destination in a network, while a flow record is the information stored in the flow exporter cache. A flow can coincide with a flow record, if the flow duration is shorter than the exporter active timeout. Flows longer that the active timeout will be split into several flow records. In other words, a flow record is the information describing (part of) a flow as we obtain it directly from a flow exporter. In many contributions (Chapter 3), and concerning this thesis, the difference is often subtle and the terms are interchangeably used.

Cisco Netflow version 5 [19] is a simple protocol that exports flow records of fixed size (48 bytes in total). Each export datagram will contain up to 30 flow record. The fields of a Netflow Version 5 flow record are summarized in Table 2.2.

Cisco Netflow version 9 and IPFIX [121, 22] propose flexible protocols in which flow record formats can be defined by using templates. These protocols allow also a larger set of parameters to be exported, such as, for example, sampling rate and algorithm, source and destination VLAN identifiers, MAC addresses and autonomous system numbers [21, 120]. An IPFIX packet is logically divided into sections known as *sets*. A message can normally consist of

Field Description
Source IP address
Destination IP address
Next hop router IP address
SNMP input and output interfaces indexes
Total number of packets in the flow
Total number of Layer 3 bytes in the flow packets
Start of flow timestamp
End of flow timestamp
Source and destination port number
Cumulative OR of TCP flags
IP protocol (for example, 6 = TCP, 17 = UDP)
IP Type of Service
Source and Destination Autonomous system
Source and destination address prefix mask bits

Table 2.2: Cisco NetFlow Flow Record Fields.

three kinds of sets, namely *template sets* (format template exchange), *data sets* (flow records) and *options template sets* (necessary for the correct interpretation of a template set). For a more detailed treatment of the IPFIX message format, we refer to [22]. In Netflow v9 terminology, template sets are referred as *template FlowSet* and data sets as *data records*.

2.1.4 Sampling

The metering process requires state information to be kept for each active flow. The IETF Packet Sampling (PSAMP) working group [119] is currently discussing the creation of a sampling standard. On high-speed links, with millions of packets per second and hundreds of thousands of active flows, the metering process can easily become a bottleneck. A heavy demand is indeed put on the CPU and memory resources of the flow exporter, with a consequent degradation of the performance in terms of both high CPU load and packet loss.

To relieve the load, sampling methods can be deployed. Sampling can be applied at different levels. Firstly, at *packet* level, meaning that each packet independently undergoes the sampling procedure. Secondly, at *flow* level, where, if a flow is sampled, all packets belonging to it are accounted.

Sampling techniques are divided into *systematic* and *random* ones [76, 155,

68]. In *systematic* packet sampling, for example, a packet is selected based on a time interval (*time-driven sampling*) or a sequence of packet arrivals (*event-driven sampling*). If randomness is introduced, we can have an *n-in-N sampling* schema. The traffic is split into sequences of N packets. Out of each of these, each time n are randomly selected. Examples of sampling rates for *n-in-N sampling* are 1-in-100 and 1-in-1000, often used in Cisco Netflow-enabled routers.

When *probabilistic sampling* is used, on the other hand, each packet is sampled with a certain probability. An example of probabilistic sampling at flow level is *Sample and hold*, proposed by Estan *et al.* [44]. When the exporter detects a new packet that does not belong to any already existent flow, it creates a new flow with probability p. Duffield *et al.* [40, 41] and Alon *et al.* [2] extend this idea by making the probability of sampling a flow depending on its size. The rationale behind this approach is that, since the distribution of bytes per flow is heavy tailed [48], a random omission of large flow can introduce a substantial error in the estimation of the original traffic volume. In a recent contribution, Liu *et al.* [95] observe that a flow sampling procedure based on the flow size poses computational limitations in high speed networks: all packets need to be seen before any decision on the sampling is taken. On the other hand, packet sampling is more scalable, but it may introduce errors in the estimation of statistical properties of the traffic. Liu *et al.* suggest a two-stage sampling procedure that combines both random packet sampling and random flow sampling.

It is important to realize, however, that although sampling indeed reduces the demands on the flow exporter, it makes decision making harder, since the statistical properties of the traffic need to be estimated based on the sampled measurements. Several studies discuss the impact of sampling on intrusion detection and flow creation: examples are Brauckhoff *et al.* [15], Mai *et al.* [98] and Zseby *et al.* [162].

2.2 Attack classification

The previous section gave an overview of how flows are created. To understand how flows can be used for intrusion detection, we are now going to give a brief overview of the attacks present in our networks.

Several attack classifications have been described in the literature [67, 71, 156], and outlined in the extensive attack taxonomy survey by Igure *et al.* [71]. We are aware, however, that, considering the nature of the topic, several new threats constantly appear on our network. We outline here the basic attack

categories in Table 2.4, following the contributions of Hansman *et al.* [67] and Weaver *et al.* [156]. The categories we present, however, should not be regarded as mutual exclusive classes of attacks. For example, buffer overflows and information gathering attacks, as for example port scans, can be regarded as separate categories of attacks, but also as specific techniques used by worms or as prelude to Denial of Service (DoS) attacks. Rather, these categories describe general "concepts" of attacks that have been frequently observed in practice. Moreover, not all taxonomies provide a classification like the one reported here. For example, Howard [70] focuses on a process-driven taxonomy, based on the objective of the attacker, the used tools, etc. Hansman *et al.* [67] summarize under the category "Network attacks" various other attacks, such as, for example, spoofing, session hijacking and parameter tampering.

Nowadays, an additional threat has evolved pertaining to Botnets. Botnets are groups of computers "infected with malicious program(s) that cause them to operate against the owners' intentions and without their knowledge", as defined in Lee *et al.* [90]. Botnets are remotely controlled by one or more *bot-masters*. Moreover, Botnets are the perfect infrastructure for setting up and supporting any kind of distributed attack, such as, for example, Denial of Service attacks and SPAM (unsolicited e-mail) campaigns. Infected hosts unknowingly become part of Botnets, and take part in malicious activities [26, 142]. The threats posed by Botnets are such that, even though they were not considered in the classifications in [67, 156], we decided to include them in our analysis.

2.3 Detection classification

Since the first papers on intrusion detection appeared in the 1980s, several taxonomies of intrusion detection techniques have been proposed. Our study identifies two main contributions to the field, the work of Debar *et al.* [32, 33] and that of Axelsson [5].

Debar *et al.* [32, 33] were among the firsts to propose an IDS taxonomy. Their classification focuses on the following elements:

- **Detection Method**: if a system bases the detection on a definition of *normal* behavior of the target system, it is called *behavior-based*. If it matches the input data against a definition of an attack, it is known as *knowledge-based*. In the literature, the community usually refers to these classes with the names of *anomaly-based* and *misuse-based* solutions [109, 5, 34, 91, 57].

Attack	Description	Example
Physical attacks	Damage to the computers and network hardware.	Material damage; Electromagnetic emanations
Buffer overflows	Attacks that gain control or crash a process on the target system by overflowing a buffer of that process.	Stack buffer overflows; Twilight hack
Password attacks	Attacks trying to gain passwords, keys, etc., for a protected system.	John the Ripper
DoS attacks	Attacks which lead to situations in which legitimate users experience a diminished level of service or cannot access a service at all.	Ping of Death, TCP/UDP floods
Information gathering attacks	Attacks that do not directly damage the target system, but gain information about the system, possibly to be used for further attacks in the future. This category comprises network traffic sniffing and scans.	SSH dictionary attack; packet sniffing
Trojan horses	Programs disguised as a useful application, which deliberately perform undesirable actions.	Beast; Graybird
Worms	Program that self-propagates across a network. Self-propagation is the characteristic that differentiates worms from viruses. A worm spread can be extremely fast: an example is the Sapphire/Slammer worm, which is known to have infected 90% of the vulnerable hosts in 10 minutes [104].	Code Red; Sapphire/Slammer; Conficker
Viruses	A virus is regarded as a worm that only replicates on the (infected) host computer. Hence, it needs user interactions to propagate to other hosts. Often, the definition also requires a virus to attach itself to files on the host, e.g., executable files, in order to be activated. As a consequence, the speed of spreading cannot be compared with a worm spread.	CIH/Chernobyl Virus; Melissa
Botnets	A group of compromised hosts, know as bots or zombies, that are remotely controlled by a command and control center, the bot-master. Botnets are a suitable platform for launching attacks like, among others, spam campaigns, DoS and worms.	Storm; Conficker

Table 2.4: Attack categories, following [156, 67], and extended with the category Botnets.

- **Behavior on detection**: a system can be proactive and act against the intruder (*active system*) or can generate alerts that will be later processed by a different system or a human operator (*passive system*).
- **Audit source location**: the data processed in order to detect intrusion can be *host* or *application logs*, *network traffic* or *alerts* generated by other detection systems. The original taxonomy by Debar *et al.* [32, 33] refers only to *network packets*. However, we believe that this type of audit data can be naturally extended to include other kind of network information, such as *packet headers* or *flows*. We therefore refer to it as *network traffic* data. Moreover, packet-based network IDSs can perform their analysis per packet (*stateless* system) or based on previously collected information, for example performing TCP session reconstruction (*stateful* system).
- **Detection Paradigm**: the IDS can detect the current status of the target system (secure or insecure) or can alert on a state transition (from secure to insecure).
- **Usage frequency**: the system can perform its task in real-time (*continuous monitoring*) or post-mortem (*periodic analysis*).

Axelsson [5] builds his taxonomy upon the one proposed in Debar *et al.* [32, 33], but extends and completes it. In particular, besides the previously described characteristics, an IDS is described also according to the following categories:

- **Locus of data-processing**: an IDS can be *centralized* or *distributed*, irrespectively of the origin of the data.
- **Locus of data-collection**: the data collection can be *centralized* or *distributed*.
- **Security**: the IDS is or is not *resilient* to security threats of which it is the target.
- **Degree of inter-operability**: a system can be built to work in *conjunction* with other systems (exchanging data) or *stand-alone*.

In his work, later followed by Almgren *et al.* [1], Axelsson refines the definition of anomaly- and misuse-based system. An *anomaly-based* system can be described as *self-learning* or *programmed*. A self-learning system is able to automatically build a model of the normal behavior of the system to protect, usually

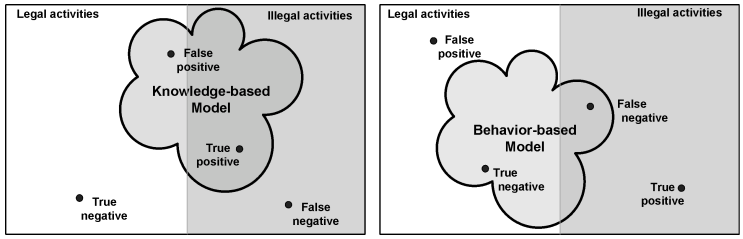


Figure 2.2: Detection capabilities of different intrusion detection models [109].

through a period of training. Examples of self-learning systems are Hidden Markov Models and Artificial Neural Networks. In the case of a programmed system, on the other hand, the definition of normality has to be provided by the system developer. A simple example is the case in which the system developer describes the conditions under which the observed system functions safely, and provides rules to flag the deviations from these conditions (for example providing thresholds over statistical values). A *misuse-based* system, on the other hand, is always defined as *programmed*. The system is provided with a knowledge-base of attacks, against which it matches the inputs, such as in the case of attack string matching.

Figure 2.2 graphically displays the problem of intrusion detection in the case of knowledge-based and behavior-based systems. A knowledge-based model is supposed to describe only illegal activities. In some cases, however, if the system is not *accurate* enough, legal activities can be flagged as intrusions; such events are called *false positives*. At the same time, if the model is not *complete*, it will not be able to report all malicious activities; unflagged illegal activities are known as *false negatives*. A behavior-based model, on the other hand, is supposed to describe only legal activities (*normality*). Also in this case, incompleteness and inaccuracy can lead to false positive and false negatives.

Finally, in [5], Axelsson introduces a third class of systems in which both anomaly-based inspired characteristics and misuse-based ones coexist. In his work, such systems are known as *compounds*.

2.4 Summary

This chapter gave background information, on the one hand, on the flow creation and collection process and, on the other hand, on attacks and detection

mechanisms.

In Section 2.1, we provided the definition of flow, according to the IPFIX documentation [121, 22] and we sketched the main phases of the flow creation process: the metering process, that perform the actual flow cache update, the exporting process and the collection process. In the same section, we described the principal flow export protocols and the sampling procedure introduced to deal with the increase of the network speed and load.

Sections 2.2 and 2.3 introduced the attack and intrusion detection taxonomies, respectively. Classification of Internet attacks is a field in constant evolution. On the contrary, classification of intrusion detection systems seems to rely on few widely accepted contributions: Debar *et al.* [32, 33] and Axelsson [5].

The following chapter will present the state-of-the-art solution in the field of flow-based intrusion detection.

CHAPTER 3

State of the art in flow-based intrusion detection

According to the definition in Section 2.1.1, flow records provide an aggregated view of network traffic. From an intrusion detection point of view, this means that it is necessary to rely on information other than the packet payload to identify malicious activities. In Chapter 2, we summarized the most important classes of attacks that can be observed. However, flow-based intrusion detection, since it cannot rely on packet content, can deal with only a subset of these attacks. In particular, our survey of the state of the art shows that the research community currently focuses on detecting *Denial of Service attacks*, *Scans*, *Worms* and *Botnets*.

In Section 3.1 we present the main contributions in *Denial of Service* detection, whereas we discuss the state of the art in *scan detection* in Section 3.2. Section 3.3 addresses the problem of *worm detection*, and finally Section 3.4 give the state of the art on a recent and fast evolving threat, *Botnets*. In Section 3.5 we classify and analyze the state-of-the-art solutions according to the intrusion detection taxonomy introduced in Chapter 2. In the end, Section 3.6 concludes the chapter.

3.1 Denial of Service

As defined in Section 2.2, a Denial of Service (DoS) attack aims to bring legitimate users to experience a diminished level of service or no service at all. Denial of Service is a frequent attack on the Internet. An overview of how often a system is the target of a DoS attacks is given in Moore *et al.* [105]. The authors analyze multiple one-week traces covering over three years from 2001 to 2004, and they conclude that on average each hour 24.5 different IP addresses all over

the world are the target of a DoS attack. The findings of Moore *et al.* clearly show that DoS attack detection is, still in these days, a problem that requires experts' attention.

> **Note:** Flow-based research is mostly addressing the problem of *brute force* DoS attacks, i.e., a type of DoS that relies on resource exhaustion or network overloading. Unfortunately, it is almost impossible to directly detect *semantic DoS* attacks, i.e., attacks in which the service interruption is caused by the payload contents. For example, let us consider the (nowadays out-of-date) Ping of Death attack. In such attack, the attacker sends malformed or otherwise malicious ping packets, which causes the victim system to crash. Since, following the Cisco Netflow definition, this attack will be accounted as a single Internet Control Message Protocol (ICMP) flow, the attack would most likely be undetected.

The work of Gao *et al.* [56] approach the problem of Denial of Service detection by means of aggregate flow measures accounted in appropriate data structures, named *sketches*. A sketch is originally a one-dimensional hash table suitable for fast storage of information [128]: it counts occurrences of an event. Sketches permit to statistically characterize how the traffic varies over time. An anomaly-based engine triggers alarms based on a statistical forecast of the values the sketches are storing: a sharp variation from the expected forecast values is flagged as an anomaly. A simple example of the use of sketches in DoS attacks is the detection of SYN Flooding attacks [135], as described in Gao *et al.* [56]. In this case, the sketch stores, for each time frame and each key *(dest_IP, dest_port)*, the difference between the number of SYN packets and the number of SYN/ACKs, as shown in Figure 3.1. The model indeed assumes that in a normal situation the number of observed SYN and SYN/ACK packets would be almost balanced. If this is not the case, a DoS SYN Flooding attack is detected. The sketch-based approach could potentially also be deployed without the use of flows, relying in this case on header inspection. However, in this case the data reduction gain provided by flows would not be achieved. Gao *et al.* developed a prototype that receives exported flows from a netflow-enabled router in real time.

A similar approach is proposed by Zhao *et al.* [159]. In this case, a data-streaming algorithm is used to filter part of the traffic and identify IP addresses that show an abnormal number of connections. The authors consider both the case in which a host is the source of an abnormal number of outgoing connections (*large fan-out*), as well as the case in which a host is the destination of an

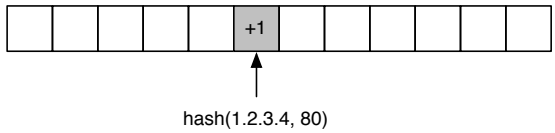


Figure 3.1: Example of sketch.

unusual number of connection attempts (*large fan-in*). The first case it is likely to match a scanning host, while the second is used for detecting DoS victims. The method is based on 2D hash tables, clearly resembling the contributions of Gao *et al.* [56] and, in Section 3.2, Li *et al.* [92]. In their paper, Zhao *et al.* also apply a flow *sampling* algorithm. Sampling reduces the amount of data to be processed and significantly raises the processing speed. However, it can introduce measurement errors since not all the flows are considered. To solve this issue, the authors develop statistical techniques to accurately estimate the *fan-in/fan-out* of the considered hosts.

Kim *et al.* [78] describe several different types of DoS attacks in terms of *traffic patterns*. A traffic pattern is an attack signature expressed in terms of the number of flows and packets, the flow and packet sizes, as well as the total bandwidth used during the attack. The authors present as example the pattern differences between instances in the class of "flooding attacks": SYN Flooding (exploiting the resource exhaustion in old TCP stack implementation in presence of half open TCP connections), ICMP flooding (provoking ICMP replies from an unaware network towards the victim) and UDP flooding (a stream of UDP packets aiming to exhaust the resource on the victim and possibly also the connection bandwidth towards the victim. The attack pattern produced by a SYN Flooding attack is characterized by a large flow count, yet small packet counts, as well as small flow and packet sizes and no constraints on the bandwidth and the total amount of packets. The pattern is significantly different from the one generated by an ICMP or UDP flooding attack, in which we observe large bandwidth consumption and intensive packet transfer. Kim *et al.* clearly identify the metrics they are interested in and formalize them into *detection functions*, which give the likelihood of an observed traffic sequence to be malicious.

In the context of DoS monitoring and detection, it is important to cite also the work of Münz *et al.* [110], who propose a general platform for DoS detection. The system, known as TOPAS (Traffic flOw and Packet Analysis System), acts as a flow collector for multiple sources and locations. The platform sup-

ports several real-time detection modules and it can be customized by the network administrator. Examples of modules are a *SYN flood detection module*, a *traceback module* (to allow identification of the network entry point of spoofed packets) and a *Web Server overloading module* (focusing on DoS attacks using HTTP requests). The work has been developed within the context of the European Diadem Firewall project, which specifically focuses on DoS and Distributed DoS detection [35].

Attention must also be given to the work of Lakhina *et al.* [83, 85, 86, 84]. The analysis is conducted on flow aggregated measures, that is on Origin-Destination (OD) flows between Points of Presence (PoP) on the Abilene [74] and Sprint-Europe [139] networks. An OD flow aggregates all the flows having a certain entry PoP and a certain exit PoP. The authors apply a mathematical eigenvector-based approach, the principal component analysis, to this small set of pairs (only n^2, where n is the number of PoPs). In this way, it is possible to decompose the traffic flowing through the backbone in traffic trends, called *eigenflows*. Lakhina *et al.* identify three types of eigenflows: deterministic eigenflows that show a periodical trend (day-night pattern), spike eigenflows that show isolated values that strongly deviate from the average and noise eigenflows that appears to be roughly Gaussian. The spike components reveal the presence of a traffic anomaly. The proposed method is general enough to capture various kinds of anomalies, spanning from hardware failures to attacks. Moreover, the method is appropriate for almost all the attack classes we are interested in (DoS, scans and worms).

3.2 Scans

A second category of network attacks is scans. Scans are usually characterized by small packets that probe the target systems. Due to their nature, scans can easily create a large number of flows, since the attacker may contact several different destination hosts using many source or destination ports. There are three categories of scans: (i) a host scanning a specific port on many destination hosts (*horizontal scan*); (ii) a host scanning several ports on a single destination host (*vertical scan*); (iii) a combination of both (*block scan*). Figure 3.2 depicts the possible scan categories, displaying on the x-axis the IP addresses and on the y-axis the victim destination ports.

In the literature, scans have generally been investigated by considering their most evident characteristic: the scanning source shows an unnaturally high number of outgoing connections, *cf.* Zhao *et al.* [159]. Looking at host be-

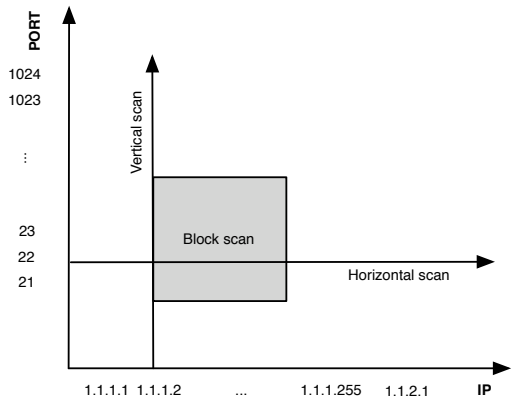


Figure 3.2: Categories of scans [79].

havior from an incoming/outgoing connections perspective allows addressing DoS and scan attacks as different faces of the same problem: hosts with a suspicious and unusual fan-in/out. Similarly, Kim *et al.* [78] describe a scan in terms of traffic patterns, as already proposed in the case of DoS. The authors differentiate between network (horizontal) scans and host (vertical) scans.

Li *et al.* [92] extend the approach of Gao *et al.* [56], in Section 3.1, introducing 2D sketches, a more powerful extension of the original ones. 2D sketches are suitable not only for DoS detection, but also for scan detection. The authors hash a different key for each dimension of the sketch, improving in this way the overall detection capabilities of the system. For example, we can think of a 2D sketch with keys *(dest_port)* and *(src_IP, dest_IP)*. The first key is used to capture information about the target service, while the second one describes pairwise interactions. In Figure 3.3, for example, we store in a sketch information about a connection from host 1.2.3.4 to port 80 of the destination host 5.6.7.8. Such a sketch can be used for detecting SYN Flooding attacks, vertical scans and horizontal scans. The first key can be used to select a column in the sketch, as indicated in Figure 3.3. The values stored in such a column allow to infer the nature of the attack, in this case discriminating between a SYN Flood or a vertical scan.

Wagner *et al.* [154] propose to use the probabilistic measure of entropy to disclose regularity in connection-based traffic (flows). Entropy has been introduced in Information Theory in 1948 [131] and, generally speaking, is a mea-

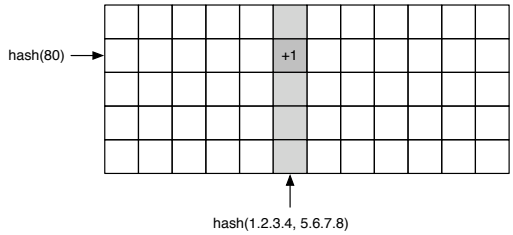


Figure 3.3: Example of 2D sketch, as in [92].

sure of randomness and *uncertainty* of a stochastic process. Entropy is also related to loss-less data compression: the theoretical limit of the compression rate of a sequence of bits is the entropy of the sequence. Starting from this well-known result, Wagner *et al.* created an efficient analysis procedure based on compression of sequences of network measurements. They observe that, in the case of a scanning host, the overall entropy in a specific time window will change. In particular, the presence of many flows with the same source IPs (the scanning host) will lead to an abrupt decrease of the entropy in the distribution of the source IP addresses. At the same time, the scanning host will attempt to contact many different destination IPs on (possibly) different ports, generating an increase in these entropy measurements. The combined observation of multiple entropy variations helps in validating the presence of an attack. Other approaches are based on logistic regression [58] and distances from baseline models [140].

3.3 Worms

The third category that we take into consideration is worms. Worm behavior is usually divided into a target discovery phase (the worm explores the network in order to find vulnerable systems) and a transfer phase (the actual code transfer takes place) [4, 89]. Code Red [161] and Sapphire/Slammer [104] are examples of this mechanism. Flow-based detection systems usually focus on the target discovery phase, since the transfer of malicious code cannot easily be detected without analyzing the payload. In many cases, worm detection can be similar to scan detection, and many researchers use the same approach for both threats. The approach adopted by Wagner *et al.* [154], for example, can naturally be extended to worms, as well as the ones of Gao *et al.* [56] and Zhao

et al. [159] (Sections 3.1 and 3.2).

Dübendorfer *et al.* [38] and Wagner *et al.* [39] attempt to characterize the host behavior on the basis of incoming and outgoing connections. The proposed algorithm assigns the hosts of a network to a set of predefined classes: the *traffic class*, the *connector class* and the *responder class*. The traffic class includes hosts that send more traffic than what they receive. Hosts that show an unusual high number of outgoing connections are part of the connector class. Finally, hosts involved in many bidirectional connections belong to the responder class. The definition of these classes is such that only suspicious hosts will belong to them. In the proposed model, a host can also belong to one or more classes. Figure 3.4 describes the three classes (sets) and their possible intersections. The method periodically checks the status of the hosts of an entire network. Massive changes in the cardinality of one or more classes are an indication of a worm outbreak. The authors validate their approach on fast spreading worms such as Witty and Blaster.

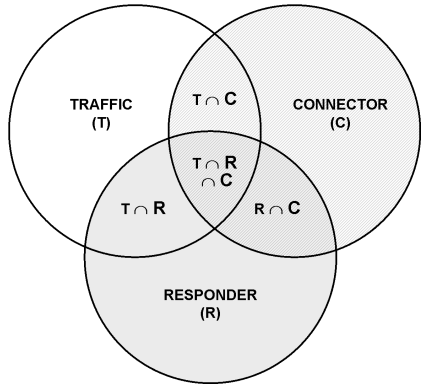
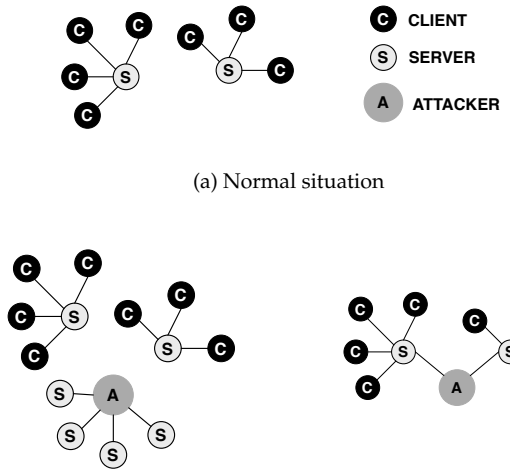


Figure 3.4: Host classes ad their intersections [38, 39].

A different approach is taken by Dressler *et al.* [37]. The authors exploit the correlation between flows and honeypots logs. In this case, the need for a *ground truth*, i.e., a trusted source of information for the system validation, made the authors rely on a honeypot. In this way, deploying at the same time a honeypot, a flow monitor and a data collection database, it is possible to carefully identify *worm flow-signatures*, i.e., a sequence of connections and flow-related information about the scanning and transmitting behavior of a worm.

Finally, Collins *et al.* [24] propose a solution to the problem of *hit-list worms*



(a) Normal situation

(b) Increased vertex cardinality

(c) Enlarged connected component

Figure 3.5: Example of graph based hit-list worm spreading analysis [24].

detection. A hit-list worm is a worm that bases its scanning strategy on the sequential probing of a predefined, usually human-compiled, list of vulnerable hosts that are likely to be always online. This technique is used because worms usually have a slow initial spreading phase, and the use of a hit-list consistently increases the initial infection speed. Since hit-lists are commonly used to start infections, detecting them as soon as possible is a useful containment technique. Collins *et al.* develop a graph-based algorithm that slices the network according to a monitored protocol, focusing on protocols such as HTTP, FTP, SMTP, and on Oracle traffic. They argue that the number of hosts normally using a certain protocol, as well as the pattern of communication between hosts, is regular over time. This regularity is disturbed only when a new host starts to scan the network following a hit-list. The authors rely on a graph theoretical approach: they map the hosts on the vertexes of a graph, and describe the communication patterns as connected components in the graph (connected subgraphs with a maximal number of vertexes). For example Figure (a) depicts the normal connection pattern in a network as a graph with 9 vertexes and 2

connected components, of cardinality 5 and 4 respectively. In this framework, a hit-list worm outbreak can perturbed the graph in two ways: (i) the worm will contact hosts that are not normally contacted, therefore increasing the number of vertexes in the network graph; this situation is Figure (b), where we now observe a graph with 14 vertexes, (ii) the worm will contact servers that in normal condition would not have any communication exchange; the attacker will act as a vertex that allows to merge different connected components; in Figure 3.5(c), the attacker merged the two original connected components into a single one with cardinality 10.

3.4 Botnets

As explained in Section 2.2, Botnets consist of infected hosts (bots) controlled by a central entity, known as master (or bot-master). As these networks tend to be spread over multiple administrative zones, complete identification of a Botnet is a difficult problem. Since bots are no longer harmful once the master is isolated, a straightforward mitigation approach is to identify the master. Nevertheless, as Zhu *et al.* [160] pointed out in their survey on Botnet research, the defense against Botnets is not yet efficient and the research in this field is still in its infancy.

As a matter of fact, many Botnets are controlled through Internet Relay Chat (IRC) channels, which can be identified at flow level. Karasaridis *et al.* [77] propose a model of IRC traffic that does not rely on specific port numbers. The authors address two main points. First, they propose a multistage procedure for detecting Botnets controllers. Starting from reports of malicious activity obtained from diverse sources (e.g., scan logs, SPAM logs, and viruses), the authors identify groups of flows involved in suspicious communications (*candidate controller conversations*). These communications may happen between a host and a candidate server (*controller*) that uses either an IRC port (e.g., 6667, 6668 or 7000) or that hides the control traffic using a different protocol. In the second case, the candidate conversation is checked against the flow model. The second aim of Karasaridis *et al.* is, once the controllers have been identified, to group the suspected bots into behavioral groups, i.e., clusters of bots that show the same activity pattern. For this purpose, they suggested a hierarchical clustering procedure that groups the hosts based on their port activities. In [77], the authors also explain why Botnet detection slightly differs from scan or DoS detection. For scans and DoS, current research aims at real-time detection, with alerts that permit the network administrator to intervene as soon as possible.

In the case of Botnets, only long time observations can lead to the identification of the bots and the controller.

In a similar way, the work of Livadas *et al.* [96] and Strayer *et al.* [142] approach the problem by modeling the TCP flows of IRC chats. The authors present the first results of a study pertaining to the use of machine learning techniques for Botnet traffic identification. In particular, they structure their approach in order to answer two research questions: is it possible to distinguish between (i) IRC and non-IRC traffic; (ii) Botnet IRC traffic and normal IRC traffic. In the paper, the effectiveness of machine learning methods, such as Naive Bayes classifiers, Bayesian networks and classification trees, is tested. The input is an enriched version of flows (including additional information, such as variance of the bytes per packet in the flow, or the number of packets for which the PUSH flag is set). The work shows that automatic identification of Botnet IRC traffic seems possible.

A different approach is proposed by Gu *et al.* [60]. They developed a Botnet detector, *BotMiner*, which is independent of Botnet Command and Control protocols and structures. Gu *et al.* developed a detection framework that aims to characterize a Botnet as a "*coordinated group* of *malware* instances that are *controlled* via Command and Control channels".

BotMiner sniffs the traffic at the observation point and conducts two parallel analyses. On one side, it relies on flows for detecting groups of hosts with similar communication patterns. On the other side, it inspects packet payloads (via Snort) in order to detect anomalous activities. These activities are then clustered in order to detect groups of hosts that have similar malicious behavior. In both steps, unsupervised clustering techniques have been used, such as the X-means algorithm [29]. As the authors describe, both step are necessary in order to properly identify possible bots, and a cross-correlation phase is performed in order to merge the results of the previous analysis and to extract meaningful groups of malicious host that could form a Botnet. The approach, which has already been implemented in a working prototype, shows good detection results. Moreover, it clearly shows that the problem of Botnet detection is more complex than the general problem of attack detection. A misbehaving host, indeed, is not sufficient to indicate the presence of a Botnet. More sophisticated intra-host communication analysis is needed to characterize the group nature of Botnets.

Even though Gu *et al.* [60] and Karasaridis *et al.* [77] present better results than Livadas *et al.* [96] and Strayer *et al.* [142], all the contributions clearly show that the problem of Botnet detection still remains to a large extent unsolved. This is mainly due to the subtle and highly dynamic evolution of Botnets. Since

the research on Botnet identification is still in its infancy, a strong research effort is needed to develop effective detection procedures.

3.5 Solutions classification

We now classify the state-of-the-art solutions presented in the previous sections according to the detection taxonomy in Section 2.3.

We must first note, however, that not all the categories included in the works of Debar *et al.* [32, 33] and Axelsson [5] are relevant in the case of flow-based intrusion detection. In the first place, we are interested only in network data, therefore we do not considered further the *audit source location* category. The *detection paradigm* (state/transition-based) is applicable mainly to host-based solutions, and for this reason has also not been addressed. We also have not considered Axelsson's *security* class, since only one of the contributions explicitly addresses the problem of attack resilience [56]. Finally, it is important to note that the studied approach seems focused on developing flow-based detection engines, and less effort is put on problems like *interoperability* between different instances of the IDS, or between the IDS and other network components (firewalls, routers). Among the considered contributions, only a few specifically address this subject, such as Li *et al.* [92] and Gao *et al.* [56].

Table 3.1, which presents our classification, gives some insight in the current research trends in flow-based intrusion detection. As for payload-based solution, the *anomaly-/misuse-based* classes play an important role: we can see contributions in both fields. Some researchers, such as Münz *et al.* [110], Dübendorfer *et al.* [38] and Wagner *et al.* [39], developed *compound* methods. This is due to the interest in combining the strengths of both anomaly and misuse-based approaches, as well as to the increasing interest in multi-purpose platforms that offer a shared base for different detection modules. On some occasions [142, 159, 96], however, the detection approach is unclear or not specified. It is also sometimes the case that the contributions address more general problems than detection and refer to security only as a possible application. An example of this is the work of Zhao *et al.* [159], treating the general problem of *super-sources/destinations*, i.e., hosts that for reasons also other that attacks have a large number of connections. In other cases, the authors simply do not provide enough details to permit a clear classification, as in the case of Strayer *et al.* [142] and Livadas *et al.* [96].

By considering the behavior on detection, the focus seems to be on passive solutions, completing their task with an alert to the network administrator.

The majority of the solutions, therefore, rely on human intervention for attack mitigation and blocking. At the same time, nevertheless, Table 3.1 shows that there is a clear preference for *real-time solutions*, which clearly indicate the need for fast responses in flow-based intrusion detection.

All of the contributions rely on *centralized* data processing. Flows are a powerful approach to data reduction, as already discussed in Chapter 1. Flow data are particularly suitable to be exported towards remote collection points, making it easy to develop a system based on distributed *data collection* points. The majority of the solutions that we studied assume a single (centralized) collection point for the ease of analysis, but the authors do not explicitly exclude the possibility of distributed collection.

3.6 IDS trends

The analysis of the state-of-the-art in the previous sections highlights emerging trends in flow-based intrusion detection. The recent spread of 1-10 Gbps technologies, and the day by day increasing network usage and load, have clearly indicated that *scalability* is a key problem. In this context, the use of flow-based solutions for monitoring and detection help to solve the issue. They achieve, indeed, *data and processing time reduction*, opening the way to high-speed detection on large infrastructures. For the University of Twente, for example, we estimate that the ratio between packets exported by NetFlow (containing the flow records) and the packets on the network is, on average, equal to 0.1%.

The complete absence of payload, however, is in some cases still perceived as the main drawback of flow-based approaches. For example, the use of flow-based techniques makes it very difficult to detect so-called *semantic attacks*, i.e., attacks for which the disruptive power is in the payload, and which do not create visible flow variations (bytes, number of packets or number of flows). However, as we can see in the case of Botnet detection, researchers propose to analyze flows to discover behavioral pattern, i.e., automatically group hosts with a similar suspicious behavior in a certain period of time. For this class of approaches, flow correlation is clearly a strong point. In our opinion, and as argued in Chapter 1, however, flow-based intrusion detection is not meant to substitute payload-based solutions, but to complement them in situations where technological constraints make payload-based techniques infeasible.

Figure 3.6 shows, in a schematic time line, the evolution of payload-based intrusion detection, flow-based technologies and flow-based intrusion detection. Payload-based solutions have been the first effort in developing network-

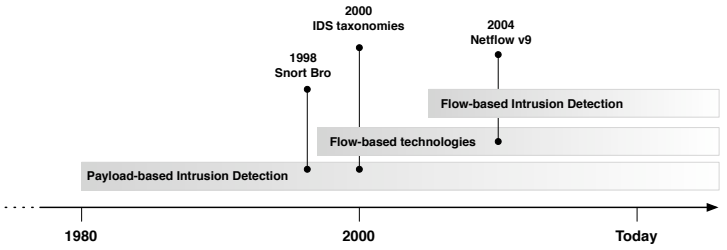


Figure 3.6: Time line of evolution of intrusion detection and flow-based technologies.

based intrusion detection. Nevertheless, they are still today a meaningful approach to security. Figure 3.6 also shows the rise of flow-based technologies, to which we referred in Chapter 2. Once flow-based monitoring became an established technology, we can see how flows became also a source of data for intrusion detection. Our analysis showed that the major efforts in flow-based detection concentrate on DoS, scan and worm detection, while Botnet detection appears to be a more recent research field.

3.7 Summary

In this chapter, we provided a survey of the state of the art in flow-based intrusion detection spanning the period 2002-2008.

Relatively to the attack classification presented in Chapter 2, flow-based solutions are usually addressing only a subset of attacks, i.e., attacks that can be detected without the need to monitor the payload content. Sections 3.1–3.4 summarized the main contributions in the area, grouped accordingly to the attack category they address: *Denial of Service*, *Scans*, *Worms* and *Botnets*.

In Section 3.5 we made use of the intrusion detection taxonomy previously introduced in Chapter 2 to classify the surveyed contributions. The results of our analysis showed that the research effort is at the moment focused on *passive* and *centralized* solutions with, primarily, *centralized data collection*. Moreover, we also noticed an evenly shared interested between *anomaly-based* and *misuse-based* system and a clear attention for *real-time* systems.

Finally, Section 3.6 outlined the evolution since the advent IDSs, of payload-based IDS, flow-based technologies and flow-based intrusion detection.

System	Detection Method	Behavior on detection	Usage Frequency	Data processing	Data collection
Li *et al.* [92] Gao *et al.* [56]	anomaly	active	real-time	centralised	distributed
Zhao *et al.* [159]	not spec	not spec	real-time	centralized	distributed
Kim *et al.* [78]	misuse	passive	real-time	centralized	distributed
Münz *et al.* [110]	compound	passive	real-time	centralized	distributed
Lakhina *et al.* [83, 85, 86, 84]	anomaly	passive	real-time	centralized	centralized
Wagner *et al.* [154]	anomaly	passive	real-time/batch	centralized	centralized
Gates *et al.* [58]	misuse	passive	batch	centralized	centralized
Stoecklin *et al.* [140]	anomaly	passive	batch	centralized	centralized
Dübendorfer *et al.* [38] [39]	compound	passive	real-time	centralized	centralized
Collins *et al.* [24]	anomaly	passive	real-time	centralized	centralized
Dressler *et al.* [37]	misuse	passive	real-time	centralized	centralized
Karasaridis *et al.* [77]	misuse	passive	real-time	centralized	centralized
Livadas *et al.* [142] [96]	not spec	passive	batch	centralized	centralized
Gu *et al.* [60]	anomaly	passive	real-time	centralized	centralized

Table 3.1: Categorization of the state-of-the-art flow-based solutions according to the taxonomy introduced in Chapter 2.

CHAPTER 4

Detecting attacks using flow data

Network attacks present a threat for both information integrity and network performance. In Chapter 2 and Chapter 3, we gave an overview of the most commonly observed attacks and the main flow-based approaches to detect them. In this chapter, we will show that *time series* are a useful starting point for intrusion detection. We will investigate how attacks can be characterized based on flow data only. Moreover, we will argue that the analysis of traffic at application level, namely performing a *traffic breakdown*, is beneficial to intrusion detection.

The present chapter is organized as follows:

- Section 4.1 describes our general approach.
- In Section 4.2, we describe the architecture used for collecting the flow traces on which we base our analysis. The data collection has been conducted on the University of Twente (UT) network and on SURFnet, the Dutch Research Network [143].
- Section 4.3 and Section 4.4 present examples of traffic characterization for two different application-layer protocols, namely Secure Shell Protocol (SSH) and Domain Name System (DNS). In particular, the SSH traffic will play an important role in the remainder of this thesis, since this protocol appeared to be a common target of attacks and will therefore be used as running examples.
- Finally, Section 4.5 presents our concluding remarks.

4.1 Analysis approach

The analysis we conduct in this chapter introduces two key-concepts of our approach to flow-based intrusion detection:

- We approach the problem of traffic characterization by mean of flow-based traffic *time series*. Since flows carry no payload, a single flow will in general not provide enough information to prove that an attack is ongoing. We believe, however, that attacks, or more generically, anomalies can be characterized looking at the evolution of flow traffic over time, as presented in flow-based traffic time series. Moreover, time series allow to analyze traffic in a streaming fashion, close to what would happen in a real time system. Flows offer diverse metrics for building time series. Some are directly derived from the definition of flow, such as the number of different accessed ports in a time bin; other can be derived metrics, such as the entropy of the IP space in a time bin, as in Wagner *et al.* [154]. We concentrate on the time series created based on the number of *flows*, *packets* and *bytes* per time bin. A time bin can have a duration from a millisecond to several minutes. In this chapter, we will investigate if flow, packet and byte time series are suitable for intrusion detection.
- We approach flow-based intrusion detection performing a *traffic breakdown* at application level. We argue that the separate analysis of diverse traffic slices is beneficial to intrusion detection since it discloses events that would otherwise remain hidden in the general traffic pattern.

4.2 Data collection

The traces on which we perform our analysis have been collected at two locations: the University of Twente and the Dutch Research Network, SURFnet. The traces cover a period of time of two working days, that is between Wednesday, August 1, 2007, 00:00 and Thursday, August 2, 2007, 23:59. Figure 4.1 presents the architecture used during the data collection, which we will describe in Sections 4.2.1 and 4.2.2.

After the data collection, we post-process the flow records to create time series. The time series describe the evolution over time of the total number of flows, packets and bytes per time bin and they can be created based on the time information contained in a flow record: start timestamp or end timestamp. In other contexts, time series may also be created on the basis of the export time

or the time the flow record is processed by the collector, as for example in the case of NfSen [64]. We based the time series on the flow start time. This means that, for each time bin in the time series, we consider all the flow records which start timestamp falls into the considered time bin. We then calculate, for the considered time bin, the total number of flow records and the sum of packets and bytes. In the following, we select a time bin of 600 seconds.

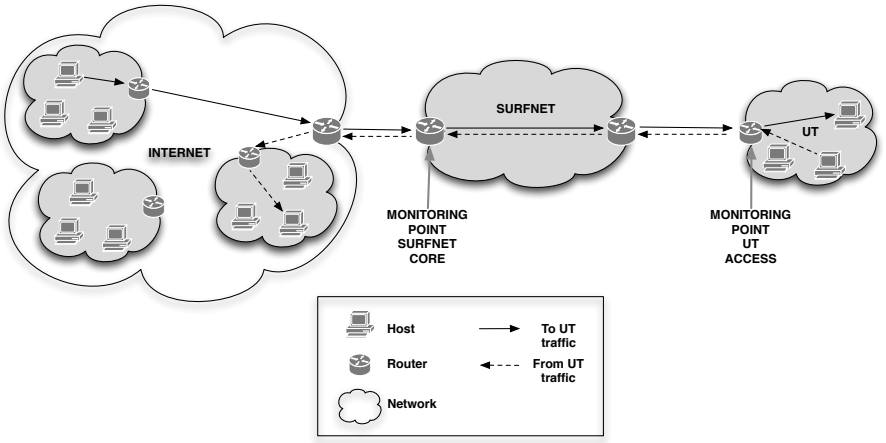


Figure 4.1: Data collection architecture.

4.2.1 The University of Twente access router

The UT network is a /16 network providing connectivity to the employees and the students in the university buildings on the campus. The UT is connected to the Internet via a 10 Gbps link and routes its traffic using a Netflow-enabled router, as indicated in Figure 4.1. The router continuously exports a feed of flow data, used by the network administrators for monitoring purposes. In the UT router setup, no sampling is applied. During the monitoring period, we analyzed the flows exported by the router.

Figure 4.2 presents the combined incoming/outgoing UT load during the data collection. The time series is based on a time bin of 600 seconds, for this analysis a good compromise between accuracy and number of samples. This means that each data point accounts for the total amount of bytes transferred over the UT network in the previous 600 seconds. The UT traffic shows a clear

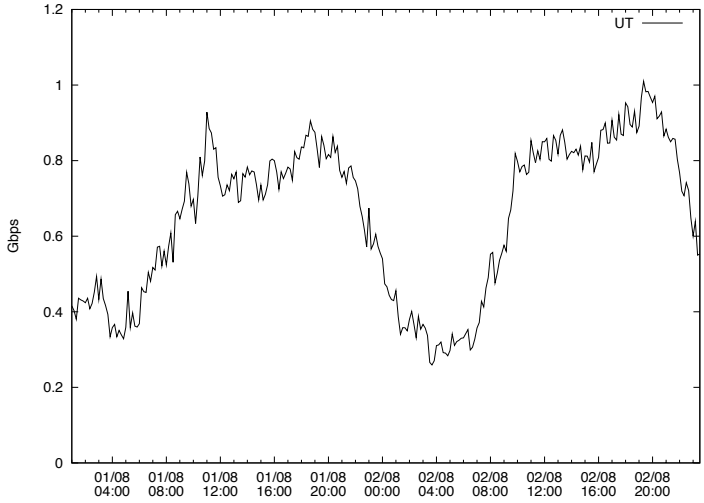


Figure 4.2: Byte time series, showing UT traffic.

day-night pattern, with peak of activity between 8:00 and 20:00 and with a minimum around 4:00. During the data collection, we measure an average load of 652 Mbps, with peaks up to 1.01 Gbps during daytime and minimum of 259 Mbps at night. The total amount of flow records during the observation period is of 982.7 million, which, once exported, corresponds to 0.2% of the total amount of transferred bytes. We calculate that we achieve a data reduction factor of 30 as ratio between the number of packets on the network and the number of exported flow records.

4.2.2 The SURFnet core routers

The SURFnet infrastructure has national coverage and offers high-speed connectivity to the main research and educational institutions in the Netherlands. SURFnet also functions as network service provider for these institutions, routing their traffic towards main backbone networks, such as, among others, Géant [59] and Internet2 [74]. The SURFnet topology consists of edge routers, providing connectivity to the users' domain access routers, and core routers, interfacing SURFnet to other backbones. Like UT, also SURFnet relies on Netflow for monitoring purposes. During the data collection, we analyzed the

flow data exported by the core routers, indicated in Figure 4.1. To reduce the load on routers, SURFnet applies systematic sampling with a ratio 1:100 during the flow creation (see Section 2.1.4). In our analysis, we estimate the real amount of traffic (packets and bytes) by scaling these metrics by a factor of 100.

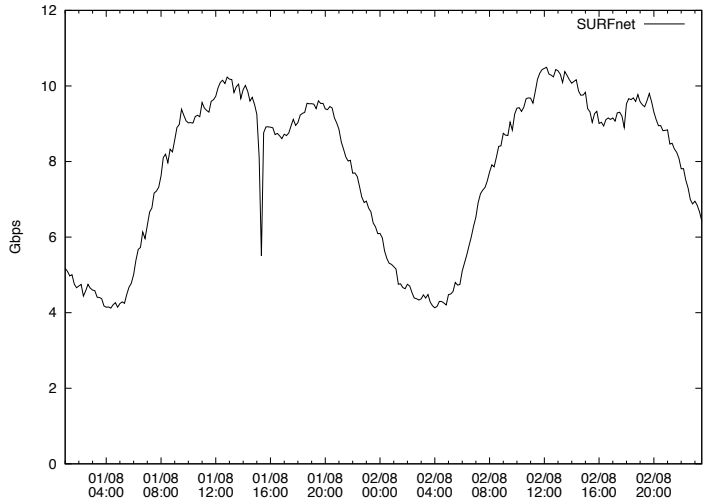


Figure 4.3: Byte time series, SURFnet traffic.

Figure 4.3 shows the SURFnet traffic load during the data collection. Also in this case the time series are based on time bins of 600 seconds. Like UT, SURFnet traffic presents a clear day-night pattern, with peak of activity between 8:00 and 18:00 and with a minimum around 4:00. Around 16:00, on August 1, 2008, the load on SURFnet drops abruptly. Since no error has been detected in our measuring setup, we suspect the down-peak to be caused by a flow creation and exporting failure in the SURFnet infrastructure, or, less likely, to a network hardware failure. However, this event is not affecting our analysis. During the monitoring period, SURFnet presented an average load of 7.73 Gbps with peaks of 10.5 Gbps. The minimum load has been of 4 Gbps. SURFnet also exported 523.7 million flows, which correspond to 0.01% of the total amount of transferred bytes. We have therefore a data reduction factor of 400 as ratio between the number of packets on the network and the number of exported flow records.

Note: It is interesting to notice that the number of flows measured in SURFnet is smaller than the number of flows generated by the UT network. This is due to the fact that, in the case of flows, it is not possible to compensate for the sampling by means of a scaling factor as for packets and bytes. To explain this, let us consider two extreme cases. First, let us suppose that each packet passing the monitoring point generates a flow and that the monitoring interface samples with 1:100 sampling rate. Then, we can assume that the number of exported flow records will correspond to $1/100$ of the real number of flows. However, let us now consider the opposite case, in which all the packets passing the monitoring point belong to the same flow. In this case, it would be incorrect to assume that we observed 100 different flows. In real traffic, it has been shown that a small number of flows account for the biggest percentage of the traffic, while the size of the majority of the flows is small [108, 51]. This situation is known as "the elephant and mice phenomenon". Fioreze [51] also shows that, since elephant flows generate a large amount of packets, there is a higher probability that a sampled packet belongs to one of these flows. In light of these considerations, we did not apply any scaling factor to the number of flows.

Figures 4.2 and 4.3 do not suggest the presence of network anomalies. However, during the monitoring, UT seemed to be subject of repeated and diverse attacks, even if without apparent real damage. The application breakdown of the traffic traces will make the attacks clearly visible. In the following sections, we focus on two examples: SSH and DNS traffic. The choice of these two specific applications is due to the fact that, quite surprisingly, the SSH service resulted to be one of the major attack targets, both in intensity and in number of attacks. By experience, we also noticed that DNS tends to produce a quite regular traffic volume. This characteristic makes it quite easy to detect suspicious variations in traffic intensity.

In the following, we will give a detailed description of the traffic anomalies concerning SSH and DNS traffic, in both the UT and SURFnet traces. In particular, we are interested in (i) assessing the effect of sampling on network anomalies, that is, checking if anomalies can still be visible also in the presence of sampling; (ii) validating our anomalies by checking if they appear consistently in both networks. In order to do this, in the following we analyze only

the UT traffic flowing through the SURFnet network, namely the flows in the SURFnet trace which source or destination host belongs to the UT network, as suggested in Figure 4.1. Packet and byte SURFnet time series are scaled by a factor of 100.

4.3 SSH traffic

SSH is one of the most common protocols to connect with remote hosts. SSH corresponds to 1% of packets and the 1.2% of bytes of the total incoming-outgoing UT traffic. This section will characterize the normal and anomalous SSH traffic as it can be observed at flow level. We analyze flow, packet and byte time series and highlight the changes due to network anomalies.

4.3.1 Traffic analysis

Figure 4.4 shows the SSH byte traffic time series in the observation time frame. In the same graph, we show both the UT and SURFnet traffic volumes. The SURFnet traffic has been scaled to remove the effect of sampling. The two measurements present the same trends. In general, the bytes trend in the observation period is quite irregular with sharp peaks and down-peaks. This huge variation in the byte time series can be explained considering the diverse usage of the SSH protocol: interactive traffic during a terminal session or file transfer by means of `scp` or `sftp`. However, the irregularities in the time series do not provide any evidence of an anomalous activity.

On the other side, looking at the packet time series (Figure 4.5), it is possible to note that in the morning of August 2, the UT network saw a massive increase of its SSH traffic. The time series is indeed characterized by sudden peaks during which the number of packets per time bin increases by several millions. In some cases, we observe a peak of up to almost 8 million packets, more than 6 times higher than the average number of packets per time bin. If we consider the flow time series, as in Figure 4.6, we can observe how the trend is confirmed also in this case. The time series presents peaks during which the number of flows per time bin raises from a few thousand to half a million. Again, the number of flows per time bin in SURFnet follows the behavior of the UT trace, despite the use of sampling. This phenomenon is particularly visible during the massive peaks, namely in the early morning of August 1 and in the late morning of August 2.

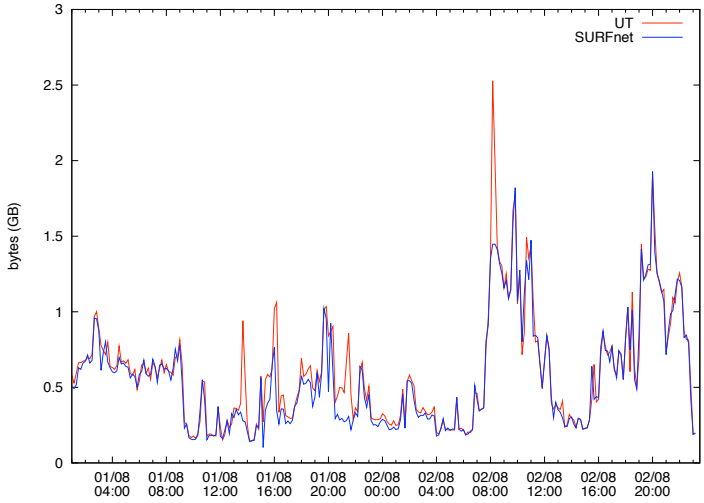


Figure 4.4: Byte time series, showing UT and SURFnet (estimated values) SSH traffic.

Summarizing, at the moments of major SSH activity, we observe a suspiciously high number of flows, created by a higher number of packets than usual. The effect of this activity on the bytes time series, however, is not sufficient to indicate the presence of an anomaly if we consider this metric by itself. The byte activity in the anomaly time frame is indeed similar to the activity we observe on August 2, between 16:00 and 23:00. This second time frame, however, is not correlated to any suspicious activities in the packet and flow time series. This suggests that the hosts involved are sending/receiving relatively small packets to/from many different hosts: this scenario suggests the possibility of a scan. A more detailed inspection of the trace shows indeed that few source hosts made the UT network object of massive SSH dictionary attacks , during which the attackers were scanning the UT network and performing user and password guessing on almost all the hosts.

It is important to underline that it is the simultaneous observation of all the three metrics, namely flows, packets and bytes, which permits to discriminate between normal and malicious traffic. For example, a peak in the packet time series can be caused by both a file transfer and a scan. However, if we would observe also the byte and flow time series, we could see that a scan would produce a peak in the flow time series, but not in the byte time series. The

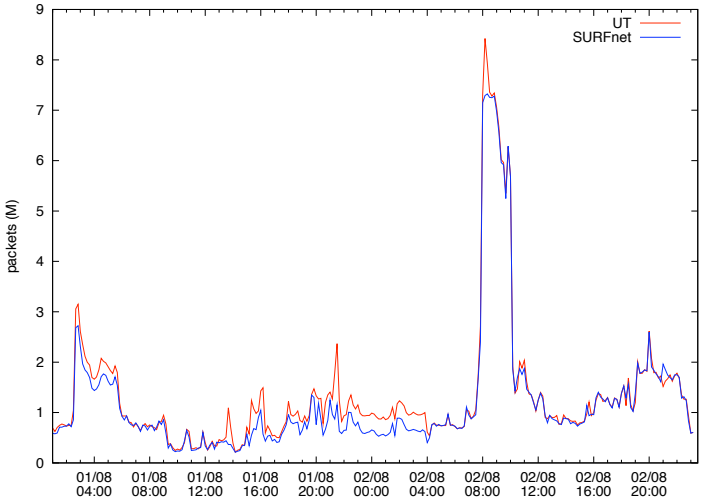


Figure 4.5: Packet time series, showing UT and SURFnet (estimated values) SSH traffic.

traffic characteristics during the peaks made the SSH traffic trace worthy of deeper analysis. In the following, we concentrate only on the peak in the time frame from 7:50 to 10:10 on August 2, when the number of flows per time bin rises up to a maximum of almost 600000 flows. We will refer to this time window as *SSH anomalous time frame*.

4.3.2 Normal *versus* anomalous traffic

The goal of the following analysis is to prove that indeed the previously identified peak is due to an attack. To characterize the network behavior during the anomaly, we need to compare it with a second observation time frame, which will provide us an overview of the network during a benign interval. The time window that we have chosen spans over a period of 2 hours, between 8:00 and 10:00 of August 1. During this time frame, we are not observing any fast variation of the flow frequency. Since we are interested in SSH scans and we are assuming that SSH scans produce variation in the flow frequency, we also assume the second time frame to be an example of *normal* network behavior. We refer to this time window as *normal time frame*.

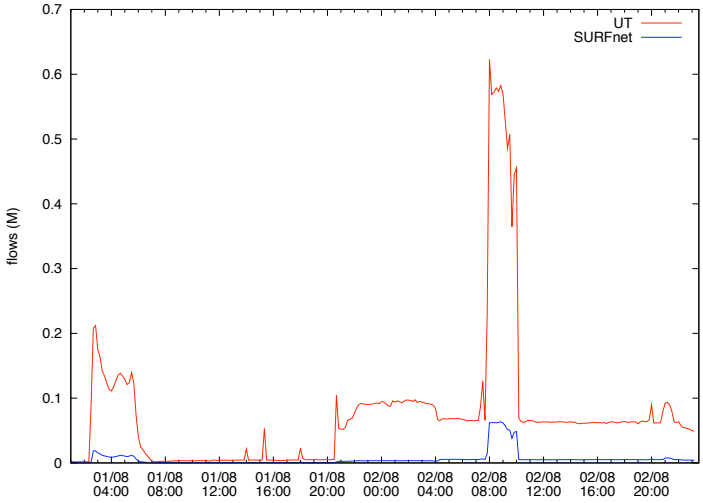


Figure 4.6: Flow time series, showing UT and SURFnet SSH traffic.

Looking at the number of active hosts in the anomalous and normal time frames, Table 4.1 shows that the normal time frame is characterized by a balanced number of sources and destinations, both in UT and SURFnet. On the contrary, in the anomalous time frame, we can observe an increased number of destinations, several times bigger than the number of sources. The number of destination hosts in the UT trace suggests that the scan covers the entire UT network (that is, as reported in Section 4.2, a /16 network, i.e., with up to 65534 hosts), while the increased number of source hosts is an effect of the scanning activity (some of the destination hosts react to the probes). A similar trend is visible in SURFnet, even though the number of different sources and destinations is in this case affected by the sampling. We therefore observe a smaller number of IP addresses in the case of SURFnet.

The study of the top active sources w.r.t. the number of originated flows shows that the anomalous time frame is dominated by the presence of three major senders, which caused the attack. Table 4.2 shows how the traffic, expressed in flows, packets and bytes, is distributed with respect to the sources during the anomalous time frame. Together, the three most active sources are responsible for 98 - 99% of the total amount of flows in both UT and SURFnet. All the three hosts were scanning the UT network. As already suspected dur-

	Anomalous time frame		Normal time frame	
	Sources	Destinations	Sources	Destinations
UT	2763	65342	629	647
SURFnet	597	3020	192	192

Table 4.1: Number of distinct source and destination addresses during the anomalous and normal time frames in the UT and SURFnet traces.

ing the time series analysis, also the number of packet is unbalanced towards the major senders (responsible for 70% circa of the packets in both UT and SURFnet). Finally, it is important to notice that the scan *does not* deeply affect the bytes distribution: 75% and the 69% of the bytes volumes, respectively, in UT and SURFnet is still due to normal traffic.

	Flow Percentage		Packets Percentage		Bytes Percentage	
	UT	SURFnet	UT	SURFnet	UT	SURFnet
Top 1	82.6%	89.5%	65.7%	71.2%	22.3%	28.1%
Top 2	13.5%	9.2%	6.7%	7.3%	2.3%	2.8%
Top 3	2.1%	0.3%	0.4%	0.3%	0.1%	0.1%
	98.2%	99%	72.8%	78.8%	24.7%	31%
Others	1.8%	1%	27.2%	21.2%	75.3%	69.0%

Table 4.2: Percentage of flows, packets and bytes for the attackers and the not suspicious hosts during the SSH anomalous time frame.

To give a visual representation of the network behavior during the anomalous and normal time frame, we present the scatter-plot in Figure 4.7. For each 600 seconds bin during the normal and anomalous time frame, we measure the number of packets, bytes and flows. We assign to each metric an axis in a 3D space and plot each time bin as a point in this space. Figure 4.7 shows a representation of the anomalous and normal time frame. In case of the anomalous time frame, also the projections on the planes are plotted. The graph allows us to see that points belonging to the normal time frame tend to group together in a part of the space characterized by relatively small number of packets and bytes. Moreover, the time bins in this group show a very low number of flows. On the contrary, the spatial disposition of the anomalous time frame describes a totally different behavior. Also in this case, the time bins during the anomaly tend to be spatially close. This is an indication of the fact that they share com-

mon features. In addition, as emphasized by the projections, points in this group present high values of the coordinates x (packets) and z (flows), while only few cases show a massive byte volume (y axis). Most importantly, the two groups are spatially distant, confirming that anomalous and normal time intervals show clearly detectable differences.

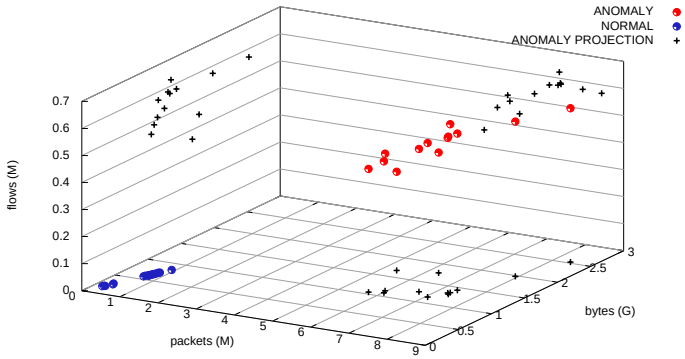


Figure 4.7: SSH anomalous and normal time frame space disposition (UT trace).

4.4 DNS traffic

DNS is the second trace we analyze. Commonly, DNS is responsible for less than 1% of the incoming-outgoing data volume at the UT network. In the following, we characterize the normal and anomalous DNS traffic and analyze flow, packet and byte time series.

4.4.1 Traffic analysis

In Section 4.3, SSH traffic seems to suggest that the analysis of the flow time series can easily indicate the presence of anomalies. Unfortunately, this hypothesis does not hold for DNS traffic. As it can be seen in Figure 4.8, the number of flows per time bin is almost constant during the entire observation period and nothing suggests the presence of an anomaly.

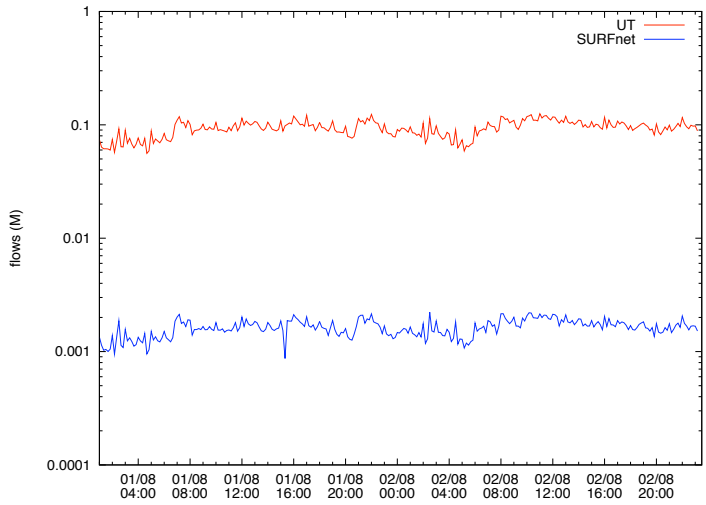


Figure 4.8: Flow time series, showing UT and SURFnet DNS traffic (in logarithmic scale).

The situation appears to be different if we do not observe the flow time series but instead the packet and byte time series. Figures 4.9 and 4.10, indeed, show that in the time window between 1:40 and 7:00 on August 1, the UT network saw a massive increase in the volume of DNS traffic, both in packets and in bytes. In particular, both measures rise abruptly from a few thousands to millions (between 10 to 28 million in a time bin). The SURFnet trace shows the same behavior, even in presence of sampling.

The just described anomaly will not be noticed if only the flow time series is considered. This observation is particularly relevant because it witnesses that the flow time series is not expressive enough to characterize DNS anomalies. By definition, DNS traffic produces quite small UDP packets during the query process and it relies on TCP only in case of databases updates. Since the analysis of the TCP and UDP percentage during the anomaly shows that the 99.7% of the flows are UDP and they are responsible for 99.9% of the bytes volume, we can exclude that the anomaly is caused by a database update. We proceed now for a more detailed analysis of the anomaly.

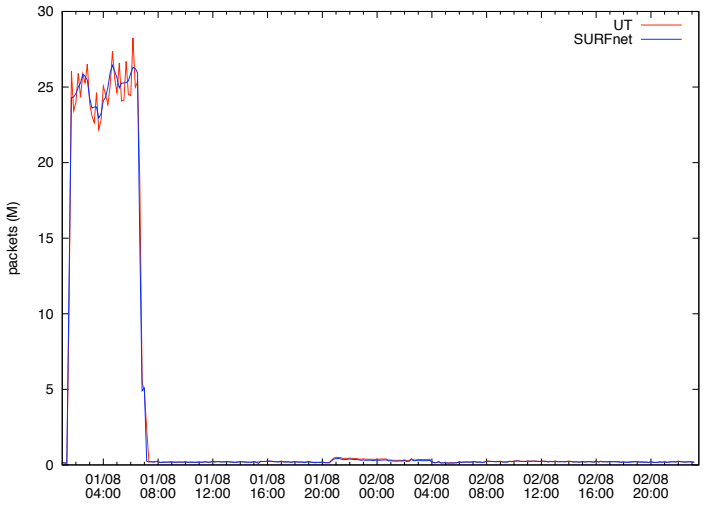


Figure 4.9: Packet time series, showing UT and SURFnet (estimated values) DNS traffic.

4.4.2 Normal *versus* anomalous traffic

As already for SSH, a non-anomalous interval has been chosen for comparison purposes. The *DNS normal time frame* spans between 12:00 and 17:00 of August 1. The large amount of bytes sent depicts a different scenario compared to the one presented in Section 4.3: the sharp variation in the byte and packets time series, together with the use of a large percentage of UDP packets suggests indeed the possibility of a DoS attack against a few number of destination hosts. The study of the anomalous time frame w.r.t the volume of byte sent clearly shows the prevalence of three attack sources. Differently from the scenario of the SSH anomaly, the three sources are creating on average less that 300 flows each, being in this way responsible for only the 0.003% of the total UT flows. On the other hand, each of these sources generates a packet volume almost 50 times bigger than all the other sources together. This ratio, measured in terms of bytes is 20. The SURFnet measurements show a similar ratio. As it can be seen in Table 4.3, the top sending hosts are responsible for more than 99% of the packets and 98% of bytes in both UT and SURFnet traces. A deeper analysis of the traces shows that the three major sources share a single destination,

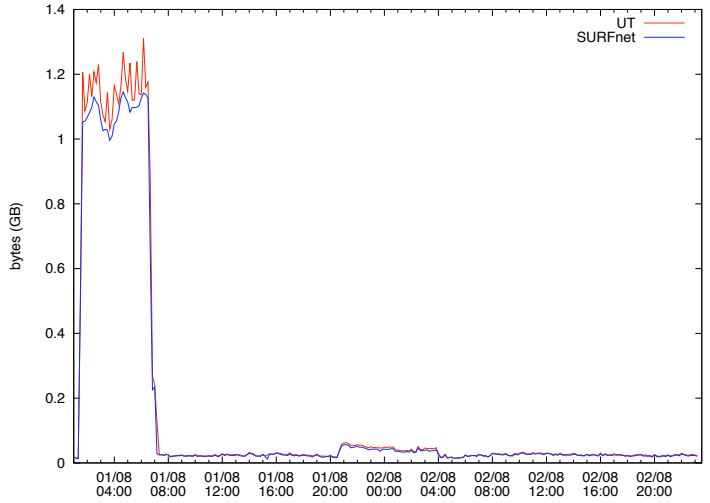


Figure 4.10: Byte time series, showing UT and SURFnet (estimated values) DNS traffic.

towards which 33GB of data have been sent during the entire anomalous time frame (with packets of constantly exactly 46B in size). This configuration supports the thesis that the destination host has been victim of a distributed DoS targeting the DNS service.

	Flow Percentage		**Packets Percentage**		**Bytes Percentage**	
	UT	SURFnet	UT	SURFnet	UT	SURFnet
Top 1	0.01%	0.14%	35.3%	35.3%	34.9%	34.8%
Top 2	0.01%	0.15%	32.6%	32.6%	32.3%	32.5%
Top 3	0.01%	0.14%	31.4%	31.4%	31%	31%
	0.03%	0.43%	99.3%	99.3%	98.2%	98.3%
Others	99.97%	99.57%	0.7%	0.7%	1.8%	1.7%

Table 4.3: Percentage of flows, packets and bytes for the attackers and the not suspicious hosts during the DNS anomalous time frame.

As done previously in Section 4.3, a 3D representation of the anomalous and normal time frames is now presented in Figure 4.11. Also in this case, the spatial disposition of the points in the two groups confirms the diversity between anomalous and normal time intervals. Points in the normal time frame

show a relative variability in the number of flows, but almost no changes in the number of packets and bytes. On the contrary, the points in the anomalous group are characterized by large x and y coordinates (packets and bytes). Only two time bins during the anomalies are distant from the majority: they show indeed a relatively small number of packets and bytes. Nevertheless, the projection of the anomaly on the packet-byte plane confirms that these points are in any case anomalies. All the points in the anomalous time frame, including the two just described, belong indeed to the same line. This is a consequence of the fact that the attackers were flooding the victim with fixed size packets. The points that appear to diverge from the general behavior are the result of the concluding phase of the attack, a fluctuation visible also in Figures 4.9 and 4.10. Last, in the graph it is possible to see that the number of flows during the anomalous and normal time frames do not differ enough to detect the ongoing attack, confirming the observation about the flow time series.

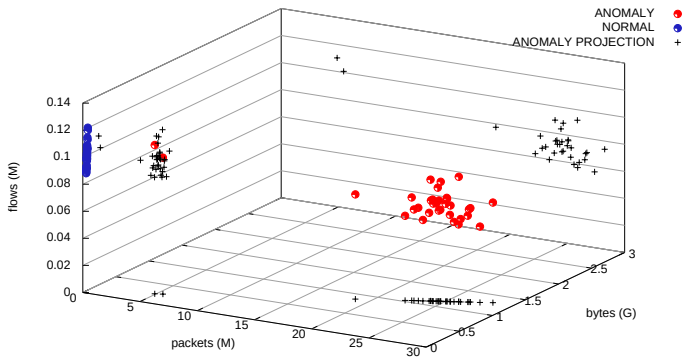


Figure 4.11: DNS anomalous and normal time frames space disposition (UT trace).

4.5 Concluding remarks

This chapter presented the analysis we conducted on flow-based traces collected on the University of Twente and on the Dutch National Research Network, SURFnet.

In Section 4.2, we described the measurement setup used to collect the traces, showing that the analysis of the network traffic as a whole is likely to hide ongoing attacks. Sections 4.3 and 4.4 describe in detail two flow anomalies, on the SSH and DNS traffic time series, respectively.

The analysis we performed on UT and SURFnet traces leads us to the following conclusions. First, our analysis showed that flow-based *time series* can be a meaningful approach to flow-based intrusion detection. Time series allow us to analyze data, keeping into account temporal relations between events. Flows are suitable for the creation of time series based on the number of flows, packets and bytes per time bin. In our analysis, we investigated whether the three metrics are all needed to identify intrusions, or whether they can be considered separately. The analysis of the SSH and DNS anomalies led to the conclusion that, to correctly identify suspicious traffic in general, all three metrics should be taken into consideration. More specifically, for certain classes of attacks, the choice to monitor only a single metric may still be sufficient. However, the decision of which metric is the most indicative of an intrusion has to be taken based on which application we are monitoring. This means that we can choose to monitor the number of flows in the case of SSH attacks, but that packets or bytes are preferable in the case of DNS traffic. Our study proved that this conclusion also holds in the presence of sampling. Sections 4.3 and 4.4 showed that the sampled traces closely approximate the non-sampled traces, which means that accurate anomaly detection is possible even in case of sampling. This observation suggests that the development of scalable, but still accurate intrusion detection solutions is possible.

Second, our analysis also showed that considering the network traffic as a whole would likely hide the presence of network anomalies. Such anomalies can instead be made evident by the *traffic breakdown* according to the originating application, as we do in the case of SSH and DNS. This approach entails that, at flow level, intrusion detection should take advantage of the data-reduction and anomaly-exposure induced by the traffic breakdown. This should lead to the design of modular intrusion detection systems targeting specific attacks.

Flow-based time series give us information about the amount of data transferred over the network. However, as pointed out in Section 1.1.1, flows can also provide information about network interactions. Recently, we observed a growing interested on how such information can be used for intrusion detection. By considering point-to-point communications, it is possible to identify sequences of correlated flows, or flow patterns, that define network services as well as attack fingerprinting. An example in this direction is the work of

Marinov *et al.* [99], that proposes an ad-hoc flow records query language able to describe causality dependencies between flows. The authors describe, as example, the fingerprinting of the Blaster worm. Point-to-point communications patterns can be exploited also in scenarios in which the network is *well-behaved*, meaning with this that we are dealing with a fixed number of devices, few protocols and regular communications. This is the case for example for intrusion detection in SCADA (Supervisory Control and Data Acquisition) networks, a topic outlined in the work of Barbosa *et al.* [7].

CHAPTER 5

Manual ground truth generation

Properly classifying objects into classes in an automatic manner is a problem that intersects many different research areas. Classification, or supervised learning, is a vastly studied topic in machine learning [23] and it finds application in fields such as, for example, computer vision [130], text classification [129], computer aided diagnosis and medical imaging [61], bioinformatics [12] and intrusion detection [87]. A common problem among all the classification approaches is the need for validation data, and more specifically, data for which the *correct* classification is known. Such data sets are often known as *ground truth* . Ground-truth data are difficult to acquire, since they require deep domain knowledge, they may be privacy sensitive and, not less important, they need in most cases to be manually created, a task that requires considerable effort. Intrusion detection, that aims to classify traffic into benign and malicious instances, is in this respect no exception.

In Chapter 6 we will approach the problem of automatic creation of ground-truth data sets. In this chapter, instead, we investigate how to manually create a flow-based data set of intrusion-relevant events. In particular, we will give answer the following research questions:

- *Which requirements should a flow-based labeled data set for intrusion detection meet?*
- *Which infrastructure is suitable for the data set collection?*
- *How can we label the collected data?*

This chapter is organized as follows:

- Section 5.1 describes the ground-truth problem and specifies which requirements a labeled data set should meet.

- Section 5.2 discusses possible choices for the data collection setup, such as monitoring at *network level*, *subnetwork level* or *host level*. The same section also describes the setup best suited for our research, which is a honeypot.
- Section 5.3 presents the procedure followed to label the data set.
- Section 5.4 provides insights into the resulting data set, describing its main characteristics.
- Finally, Section 5.5 presents the lesson learned on the topic of manual flow labeling.

5.1 Motivation and objectives

When proposing a new IDS, researchers usually evaluate it by testing it on labeled (or annotated) traffic traces, i.e., traffic traces with known and marked anomalies and incidents [101]. We refer to these traces as *ground truth*. In Section 5.1.1, we give an overview of the existing ground-truth data sets for intrusion detection, while in Section 5.1.2 we clarify why there is a strong need of this type of traces. Finally, Section 5.1.3 describes which requirements a data set should meet.

5.1.1 Existing traces

Even though trace sharing is a long-lasting goal of the network monitoring community, with repositories like Caida DATA [147], DatCat [30], Crawdad [25], Simpleweb [132] and the MOME project [103], these traces are usually not labeled and not specifically targeted to intrusion detection. Only few payload-based labeled data sets for evaluation of IDSs exist and are publicly available:

- The DARPA 1998 and DARPA 1999 data sets, developed by the MIT Lincoln Labs and sponsored by the US Defense Advanced Research Projects Agency. The DARPA data sets [93, 94, 65] consist of artificial background traffic, which simulates the normal network usage of an air force base, combined with malicious attack traffic.
- The KDD99 data set [141] and the NSL-KDD data set [146]. The KDD99 data set is build upon the traffic in the DARPA 1998 data set, but uses an extended labeling. The NSL-KDD data set [146], on the other hand, is a

reduced version of KDD99 that aims to avoid record redundancy in the data set. Both data sets have been in use in the data mining and machine learning community, but researchers in the area of intrusion detection still refer directly to the DARPA data sets.

- Considering the constant evolution of network threats and the consequent aging of the DARPA traces, a recent attempt in addressing the problem of shared labeled data sets comes from the US Military Academy West Point, New York [124]. The authors suggest that network cyber-defense competitions, such as the Defcon's Capture The Flag (CTF) competition and Cyber Defense Exercise (CDX), can be used for creating labeled data sets. The authors propose a general approach for data capturing during such competitions and illustrate their approach by proposing a first data set. The data set is publicly available, but we are not aware of any IDSs that have been evaluated using this data set.

- A recent attempt to propose a database of labeled traffic for IDSs comparison and evaluation is the work of Owezarski [113]. The database consists of a collection of packet traces from the METROSEC experimental platform [102]. The collected traces include benign background traffic, attack traffic and, more generally, network anomalies. Given its recent publication, we are currently not aware of contributions that make use of this database. Moreover, at the moment the database cannot be publicly released due to privacy concerns.

Considering the current situation, research on IDS generally suffers from a lack of shared data sets for benchmarking and evaluation. Several difficulties prevent the research community to create and publish such traces, in the first place the problem of balancing between privacy and realism. It is natural that the most realistic traces are those collected "in the wild", for example at Internet service providers or in corporate networks. Unfortunately, these traces would reveal privacy sensitive information about the involved entities, hence, they are rarely published. On the other hand, artificial traces, i.e., traces that have not been collected but artificially generated, can avoid the problem of privacy but they usually require deeper domain knowledge to achieve a realistic result. Therefore, most publications use non-public traffic traces for evaluation purposes. Moreover, we have no knowledge of any publicly available labeled flow-based traffic trace.

5.1.2 Why ground truth is important

Knowing the ground truth for any of the classification problems in intrusion detection, bioinformatics, or computer vision can be thought as "knowing the right answer". Let us think of the following situation: a friend asks you as a personal favor to help him correcting an exam of the first year medicine students in his university. It is a true/false test, therefore the task, even though time consuming, does not appear to be too difficult. However, your friend forgets to provide you with the right answers, the ground truth for the exam. Since you are most probably not an expert on the topic, it is clear that even the task of correcting a true/false test has become impossible.

In intrusion detection, ground truth is useful, for example, in the following situations:

- *Repeatability of experiments*: a desirable quality of a scientific approach is that the results can be reproduced and the experiments repeated later in time. In intrusion detection, this goal is in some case difficult to achieve. An example is the spam detection approach that we proposed in Sperotto *et al.* [138]. The paper, which followed from the master thesis of Vliek [153], proposes a set of rules to identify possible spamming host at flow level. In the validation phase, since there was no information about the content of any SMTP connection, it was crucial to find a reliable source that could confirm that a host was indeed spamming. We relied therefore on DNS blacklists, Internet services that publish lists of offending IP addresses. However, DNS blacklists are by definition volatile, since a host that sent spam in the past could, in a later stage, be rehabilitated. Since DNS blacklists can be queried but not easily downloaded, their volatility makes repeatability of experiments a complex task.

- *Validation of new approaches*: researchers continuously propose new approaches to the intrusion detection problem. The aims are both to deal with specific attacks and to improve existing detection techniques. However, every new approach needs to be validated. An example among many is the work of Bolzoni *et al.* [14], in which the authors show the effectiveness of their approach by benchmarking it against the DARPA 1999 data set, considered the *de facto* standard payload-based intrusion detection data set until the half of the 2000s.

- *Comparison of different approaches*: intrusion detection approaches are not only validated. It is often the case that the gain in performance is quan-

tified with respect to other state-of-the-art solutions. Examples are the works of Lippman *et al.* [93], proposing the DARPA 1998 data set and comparing the detection performance of several IDSs tested on this data set, or the work of Lazarevic *et al.* [88], that focuses on the performance of anomaly-based systems. When it is needed to compare different approaches, it is fundamental to benchmark the systems against the same data set.

- *Training and parameter tuning*: as described in Chapter 2, anomaly-based systems rely on a model of normal behavior. However, to properly approximate the benign status of the system under protection, it is necessary to properly tune the model parameters, usually by means of a training phase. Anomaly based systems rely on ground-truth traces for the training process, that is, to compute the model parameters that certify a correct distinction between malicious and benign traffic. Examples are the work of Zanero [158] and Bolzoni *et al.* [14], where the training is conducted on an attack free part of the DARPA 1999 data set.

There are strong motivations for creating and sharing ground-truth data sets. However, even if it is a required step for development and evaluation of IDSs, ground truth can hide several pitfalls. First, information included in the data set might not be representative of real traffic. The DARPA data sets have been the object of such criticism in the work of McHugh [100] and Mahoney *et al.* [97]: the provided background traffic has not been proved to be similar to a realistic scenario; the ratio between malicious and background traffic is not representative of a real situation; the capturing architecture is too simple and artifacts in the traffic might be used for achieving a high detection rate. However, as McHugh clearly states, despite the criticism, the DARPA data sets are "the only work of this kind worthy the critical effort". A second pitfall is to simplify the data sets for the need of research fields other than intrusion detection. The example in this case is the NSL-KDD data sets: a subset of the DARPA'98, derived from the KDD'98 data set and specifically created for testing new machine learning algorithms. The data set has been cleaned, simplified and made smaller, to avoid that specific traffic characteristics, such as for example the presence of duplicated records, might bias a learning algorithm. Clearly, such data set fits the requirements of the machine learning community, but is not suitable anymore for intrusion detection. Finally, a last pitfall concerns the risk of optimizing the detection for a known problem. Achieving good detection results on a single ground-truth data set does not necessary imply that the detector will perform at the same way in a different scenario.

5.1.3 Data set requirements

Despite the importance of ground truth for validating and evaluating IDSs, such data sets are generally time consuming to build and need a structured creation process. In this subsection, we will describe the requirements a data set should meet and that will lead us during the data collection process. The requirements are:

- *Realism*: a data set is realistic if it describes a daily situation on the monitoring point, such as for example the daily traffic on a network. We therefore prefer to focus our data collection on traces collected "in the wild".
- *Completeness in labeling*: our aim is to provide a labeled data set. More specifically, we want all the security accidents to be labeled. This also means that together with the raw data that will form our data set, we need to provide evidence of the malicious or benign content of the traffic.
- *Correctness in labeling*: besides being complete, we want the data set to be correct. This means that our knowledge of the security events in the data set has to be certain.
- *Sufficient trace size*: we require our data set to have a sufficiently large size, meaning that we aim for collecting a sufficient number of security events.
- *Achievable in reasonable labeling time*: finally, we aim for a solution that allows us to build a labeled data set in a reasonable time.

5.2 Infrastructure for data collection

While dealing with the creation of a flow-based data set, the choice of a proper data collection setup is critical, since it has impact on the size of the resulting data set and, moreover, on its quality.

In this section, we will discuss two key-topics that led our decisions on the final data collection set-up: *measurement scales* and *flow collection location*. Sections 5.2.1 and 5.2.2 will describe the impact of the measurement scale and the flow collection location on the data set requirements that we outlined in Section 5.1. In Section 5.2.3, we discuss the results of our analysis. Finally, Section 5.2.4 will describe our experimental setup.

5.2.1 Measurement scale

Flows can be collected at different measurement scales. With measurement scale, we mean the number of hosts involved in the measurement, and, as a consequence, the quantity of traffic that we should analyze and label. In our case, we outline three measurement scales: *network level*, *subnetwork level* and *host level*.

The *network level* corresponds to monitoring traffic from the entire University of Twente (UT) network. As described in Chapter 4, the UT network has a /16 address space and it is connected to the Internet by means of a 10 Gbps optical connection with an average load of around 650 Mbps and peaks of around 1.1 Gbps. Several hundred million flow records are exported per day [136]. Traces collected on this network are for sure realistic, since they describe the daily UT network usage. However, we cannot accomplish the goals of completeness in labeling within reasonable time, due to the quantity of data to be analyzed. Labeling network-wide traces therefore suffers from scalability issues.

Let us now consider the *subnetwork level*. We analyze a small subnetwork that is primarily used by our team for research purposes. Due to the limited number of users, we assumed it would be easy to distinguish trusted IP addresses from unknown ones, leaving out only a small fraction of suspicious traffic to be further analyzed. However, our findings show that more than 60% of the connections cannot easily be categorized as malicious or benign. Collecting on a small subnetwork ensures us to have a realistic data set, but, as in the case of the *network level*, it is neither complete nor achievable in a reasonable time.

A different setup, and the one that we finally chose, is based on monitoring at host level. We consider a single host with *enhanced logging* specifically tuned to track malicious activities, e.g., a *honeypot*. A honeypot can be defined as an "environment where vulnerabilities have been deliberately introduced to observe attacks and intrusions" [117]. In this case, the trace size is smaller and, consequently, the labeling is time limited. An everyday service setup ensures the traffic to be realistic. Most importantly, the access to the logs ensures us enough additional information to achieve both completeness and correctness in labeling. In the following, we will refer to the honeypot as the *monitoring point*.

5.2.2 Flow collection location

We have different options for the flow collection location. A possibility is to collect the flows generated by a Netflow-enabled router, in our case the UT one. However, decoupling the flow creation from the log location introduces errors in measurements. Examples are non-uniform timing delays and split TCP sessions. We address these issues below.

A timing delay is the time skew between the moment a flow record is created in the router cache and the moment the software on the honeypot will create a log entry for the suspicious traffic. In an ideal situation, the timestamp of the flow record and the one of the log are identical. However, a timing delay is introduced due to network synchronization issues or the time the honeypot needs to process the traffic and to determine that an intrusion occurred. If the honeypot relies on complex software, such as for example Nepenthes [6], we experienced non-uniform timing delays that made the correlation between flows and logs impossible.

A split TCP session occurs when a TCP connection has been divided in several flow records, due to the active timeout on the Netflow router. In this situation, the first step in order to associate a security event to the flows that cause it is to group flow records into TCP sessions, on the basis of the 5-tuple flow definition (as in Section 2.1.1). However, a 5-tuple does not necessarily specify a TCP connection in a unique manner. It might happen, indeed, that the same 5-tuple can be reused in different TCP sessions, even within a short time interval between different sessions. The problem is even more troublesome in a security context, in which an attacker can deliberately bind its traffic to a certain 5-tuple by forging the attack packets. The issue that arises from this observation is how we can regroup TCP sessions in a reliable manner. A first possibility appears in Fioreze [51]. The author proposes to distinguish different TCP sessions based on the time gap between the end time of a flow record and the start time of the next one. If two flow records are sufficiently distant in time, we can assume that a new TCP session has started. However, this method does not ensure that we can recombine the TCP sessions without introducing errors, namely without merging unrelated TCP sessions. A second possibility would be to base the TCP session merging on the TCP flag information. A flow record containing a FIN flag would in this case signal the end of a session. However, this information, although useful, is not always exported, as it happens to be the case for our Netflow router. We conclude therefore that errors in the flow records merging would affect directly on the quality of the data set, since we could not ensure a proper matching between the logs and the flows.

A data set built in presence of timing delays and merged TCP sessions would have a serious lack in completeness and correctness. Another possibility is therefore to dump the traffic reaching the monitoring point and to create the flows off-line after the data collection is completed. This decision allows to have complete control over the flow-creation process, overcoming the problem of the session splitting and minimizing the delay between the flow creation and the event logging.

In this subsection, we analyzed several methods for collecting the flow data we need for a labeled data set. We pointed out, in particular, the disadvantages of decoupling the flow creation from the log location and we concluded that, in our case, it is advisable to dump the traffic to and from the monitoring point and to create the flows off-line.

5.2.3 Discussion

We will now summarize the methods we studied for the data set collection in light of the data set requirements (Section 5.1.3).

	Requirements				
	Realistic	Labeling completeness	Labeling Correctness	Trace size	Labeling time
University Network	✓			✓	
University Subnet	✓			✓	
Honeypot router flows	✓			✓	✓
Honeypot off-line flows	✓	✓	✓	✓	✓

Table 5.2: Data set collection.

Table 5.2 provides an overview of the presented approaches and their capability of satisfying the requirements in Section 5.1. Please note that, since monitoring the university network or a subnetwork does not scale, we did not explore further the options of router/off-line flow creation. All our approaches ensure the trace to be realistic. Monitoring the UT network or a subnetwork would also provide sufficiently large traces. After we measured the traffic

reaching the monitoring point, we can say that also in this case this requirement is met. Regarding completeness and correctness, large network traces reduce the trust we can have on the labeled data set. The honeypot approach is more reliable since it offers additional logging information, but in this case, the flow collection/creation is crucial. As previously discussed in this section, relying on external flow sources can introduce measurement errors. Creating the flows off-line, on the other hand, allows to have flows that better match the needs of a labeled data set. Finally, large infrastructures suffer from scalability in labeling time, while the honeypot setup overcomes this problem.

From this section, we can conclude that monitoring a single host with enhanced logging capabilities is a promising setup for the creation of flow-based data sets.

5.2.4 Experimental setup

The honeypot was installed on a virtual machine running on Citrix XenServer 5 [20]. The decision to run a virtualized host is due to the flexibility to install, configure and recover the virtual machine in case it is compromised. In addition, a compromised virtual machine can be saved for further analysis. Diverse scenarios are possible, in terms of number of virtual machines, operating systems and software. Our experimental setup consisted of a single virtual machine, on which we installed a Linux distribution (Debian Etch 4.0). In order to keep the setup simple, controllable and realistic, we decided not to rely on honeypot software, but to configure the host ourselves. The monitoring point behaves, therefore, as a simple Linux server. In particular, the following services have been installed:

- SSH: Beside the traditional service logs, the OpenSSH service [112], running on Debian, has been patched in order to log sessions: for each login, the *transcript* (user typed commands) and the timing of the session have been recorded. This patch is particularly important to track active hacking activities.

- Apache web server: a simple webpage with a login form has been deployed. We relied on the service logging capabilities for checking the content of the incoming HTTP connections.

- FTP: the chosen service was `proftp` [118]. As for HTTP, we relied on the FTP logs for monitoring attempted and successful connections.

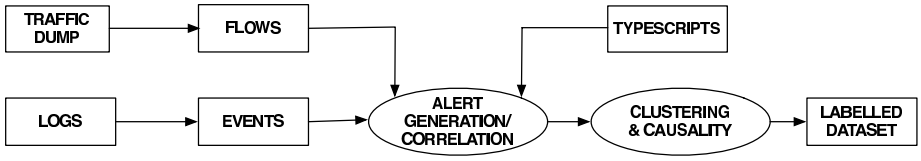


Figure 5.1: From raw data (packets and logs) to the labeled data set.

Along with the service logs, we decided to dump all the traffic that reached the honeypot during our observation period. The virtual machine ran for 6 days, from Tuesday, September 23, 2008, 12:40 to Monday, September 19, 2008, 22:40. The monitoring window consists of both working days and weekend days. The monitoring point was hosted in the UT network and directly connected to the Internet. The data collection resulted in a 24 GB dump file containing 155.2 million packets.

5.3 Data processing and labeling

The labeling process enriches the data trace with information about (i) the type and structure of the malicious traffic, (ii) dependencies among different isolated malicious activities. The latter is particularly important for a flow-based data set where, by design, no further detail on the content of the communication is available to the end user. In this section, we research how it is possible to formalize and structure the information collected at the monitoring point.

Figure 5.1 gives an overview on the data processing and labeling. As a first step, shown in the left part of the figure, the collected traffic is converted into flows. In addition, the data extracted from the diverse log files are converted into a common format, called "events", that simplifies the processing steps that will follow. The resulting flow records and events feed the alert generation/correlation process. At this stage, we rely also on information coming from the typescript of the SSH sessions on the monitoring point, that is, the recording of all the command the hackers executed during the observation period. Finally, a post-processing step generates additional information, namely the so-called cluster alerts and the causality information. This step groups alerts in order to describe more complex security events. The different steps are explained in the following Sections 5.3.1 through 5.3.4.

5.3.1 From packets to flows

The first step is the creation of flows from the traffic trace. In our data set, a flow closely follows the Netflow version 5 [19] definition and has the following form:

$$F = (IP_{src}, IP_{dst}, P_{src}, P_{dst}, Pckts, Octs, T_{start}, T_{end}, Flags, Prot)$$

, where the unidirectional communication is defined by the source and destination IP addresses IP_{src} and IP_{dst}, the employed ports P_{src} and P_{dst} (in case of UDP/TCP traffic), and the level 3 protocol type $Prot$. The fields $Pckts$ and $Octs$ give the total number of packets and octets, respectively, in the data exchange; the TCP header flags are stored as a binary `OR` of the flags in all the packets of the flow (field $Flags$); the start and end time of the flow are given in T_{start}, respectively, T_{end} in millisecond resolution. The flow creation has been performed using `softflowd` [133].

5.3.2 From log files to log events

Information about attacks against the monitoring point can be extracted from the log files of the services we were monitoring. In order to simplify the alert generation process, the relevant data found in the various log files are converted into log events. A log event consists of the following information:

$$L = (T, IP_{src}, IP_{dst}, P_{src}, P_{dst}, Descr, Auto, Succ, Corr)$$

, where T gives the timestamp of the attack (as found in the logs), IP_{src} and P_{src} give the IP address and used port (if available) of the attacker, and IP_{dst} and P_{dst} give the attacked IP address and port number. This distinction is needed because during the observation period, the monitoring point has been both the target and the source of attacks. In addition, a deeper manual analysis of the log files reveals whether an attack was automated or manual and succeeded or failed (flags $Auto$ and $Succ$, respectively). The field $Descr$ allows us to enrich the data set with additional information retrieved from the log files. An example is the payload content of an HTTP request:

```
"GET /cacti/include/config_settings.php?".
```

The field $Corr$ is a utility field and later used by the alert generation process to indicate if the log event has been already processed.

5.3.3 Alert generation and correlation

Goal of this step is to generate so-called alerts and to correlate each generated alert to one or more flows. An alert describes a security incident and is represented by the following tuple:

$$A = (T, Descr, Auto, Succ, Serv, Type).$$

The fields T, $Descr$, $Auto$, $Succ$ are defined as for the log events (see Section 5.3.2). The field $Serv$ gives the service addressed by the security incident, for example SSH or HTTP. The $Type$ field describes the type of the incident, that is if the event is a simple connection or represents a group of connections. Note that, in this context, we refer to any packet exchange to and from the monitoring point as a "connection", without implying that an actual TCP connection has been established. The alert generation process consists of three steps that are explained in the following.

Alerts from log events

For attacks toward the monitoring point, alerts can be directly generated from the log events. The fields T, $Descr$, $Auto$, $Succ$ of an alert record are set to the values of the corresponding fields of the log event. The $Serv$ field is set accordingly to the destination port field of the log event, for example $Serv =$ SSH if $P_{dst} = 22$. In this phase of the labeling, the $Type$ field is always set to the value `CONN`, indicating that the alert describes a malicious connection attempt.

To correlate an alert with a flow, we have to find the flow that corresponds to the log event from which the alert has been generated. As it will be explained below, this is not a trivial task since the timing information extracted from a log file may not be aligned with the flow one. In addition, we would like to correlate not only the incoming flow (as seen from the monitoring point) to the alert but also the response flow of the honeypot.

We use the flows as starting point for the alert generation and correlation. This avoids that a flow is correlated to more than one alert. The resulting procedure for a service s is shown in Algorithm 1. As a first step, a best matching response flow is selected for each flow of the considered service (lines 3-7). The matching is made based on the flow attributes. If there are more than one candidate flows, the closest in (future) time is chosen. It is possible, nevertheless, that such a flow does not exist, for example in the case in which the target was a closed destination port. Since the flow tuple source/destination address/port

may appear multiple times in the data set, as explained in Section 5.2.2, the parameter δ ensures that an incoming flow is not correlated with a response too late in time. We found that values of δ from 1 to 10 seconds are possible.

Algorithm 1 Correlation procedure.

1: **procedure** ProcessFlowsForService (s : service)
2: **for all** Incoming flows F_1 for the service s **do**
3: Retrieve matching response Flow F_2 such as
4: $F_2.IP_{src} = F_1.IP_{dst} \wedge F_2.IP_{dst} = F_1.IP_{src} \wedge$
5: $F_2.P_{src} = F_1.P_{dst} \wedge F_2.P_{dst} = F_1.P_{src} \wedge$
6: $F_1.T_{start} \leq F_2.T_{start} \leq F_1.T_{start} + \delta$
7: with smallest $F_2.T_{start} - F_1.T_{start}$;
8: Retrieve a matching log event L such as
9: $L.IP_{src} = F_1.IP_{src} \wedge L.IP_{dst} = F_1.IP_{dst} \wedge$
10: $L.P_{src} = F_1.P_{dst} \wedge L.P_{dst} = F_1.P_{src} \wedge$
11: $F_1.T_{start} \leq L.T \leq F_1.T_{end} \wedge$ **not** $L.Corr$
12: with smallest $L.T - F_1.T_{start}$;
13: **if** L exists **then**
14: Create alert $A = (L.T, L.Descr, L.Auto, L.Succ, s, \texttt{CONN})$.
15: Correlate F_1 to A ;
16: **if** F_2 exists **then**
17: Correlate F_2 to A ; $L.Corr \leftarrow \mathbf{true}$;
18: **end if**
19: **end if**
20: **end for**

After searching for a response flow, the algorithm proceeds with retrieving the best matching log event (lines 8-12). The log event must match the flow characteristics. The timing constraint in this case forces the log event to be in the interval between the beginning and the end of the flow. Moreover, since log files do not provide us with millisecond precision and multiple alerts can be generated by the same host in the same second, we require the matching log event to not have been consumed before (line 11). If the matching event exists, the algorithm will create the alert and correlate it with the flow and, if possible, with the response flow (lines 13-18). Finally, the log event will be marked as consumed (line 17).

Alerts for outgoing attacks

Some of the incoming attacks against the SSH service were successful. Consequently, the attacker sometimes used the monitoring point itself to launch SSH scans and dictionary attacks against other machines. To generate alerts for these outgoing attacks, the analysis of the SSH sessions typescripts allows us to reconstruct the times and the destinations of the different attacks launched from the honeypot. Similarly to the previous step, this information is used to find the corresponding flows and to correlate them to alerts of type `CONN`.

Alerts for side effects

Several attacks to and from the monitoring point have caused non-malicious network traffic that we consider as "side effects". Most notably, SSH and FTP connection attempts caused ICMP traffic and traffic to the `auth/ident` service (port 113) of the attacker. Furthermore, one attacker installed an IRC proxy on the honeypot. For these flows, we have created alerts of type `SIDE_EFFECT` with $Serv =$ `ICMP`, respectively, $Serv =$ `auth` or $Serv =$ `IRC`.

5.3.4 Cluster alerts and causality information

The alerts described in the previous section represent single security incidents and are directly correlated with one or more flows. However, we argue that this is not the only valuable information that a flow based data set can include. For example, the current alert-flow mapping does not provide information about the relations between sets of alerts. To overcome this issue, we generate so-called cluster alerts and causality information to describe relationships between alerts. Figure 5.2 shows an example of how such information is introduced in the data set. The example can represent, for example, the relation between a successful SSH session on the monitoring point and an SSH scan launched during such session. The successful SSH connection causes an SSH scan to be launched. The scan can be seen as a group of SSH connections towards target hosts.

Cluster alerts are used to label logical groups of alerts. For example, in the case of an SSH scan consisting of a certain number of connection attempts, we create basic alerts of type `CONN` for each connection, plus *one* cluster alert of type `SCAN` for the entire scan operation. More formally, a basic alert A belongs to a scan alert C, if (i) the alert A has the same source IP and type as the other alerts in the cluster, and (ii) the alert is not later than γ seconds after the latest

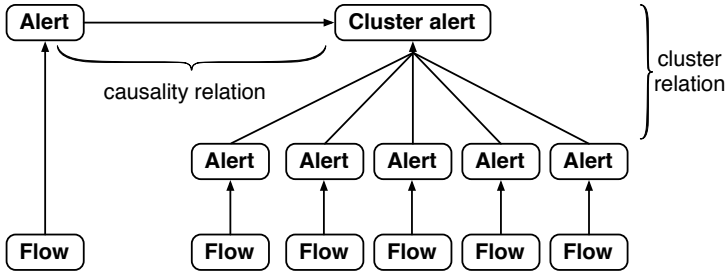


Figure 5.2: Clustering and causality information.

alert in the cluster. In our case we set $\gamma = 5$ seconds, but depending on the nature of the attack other values might be also suitable.

As a final step, we manually add causality information. In the current setup, that means that we have created (i) links between the alert representing an attacker's successful log-in attempt into the honeypot via SSH and all alerts raised by the attacker during that SSH session, and (ii) links between the alerts of the ICMP and `auth/ident` flows and the alerts of the SSH and FTP flows that caused them.

Our data set has been implemented in a MySQL database. The structure of the database reflects the alert and flow structure and the relations between these categories. Figure 5.3 shows the used database schema.

5.3.5 Semi-automatic labeling and manual steps

The correlation procedure described in the previous subsections entails both semi-automatic and manual steps.

Flows are created from the traffic dump in an automatic manner using `softflowd`. The processing of the log files, on the other hand, is not as straightforward as the flow creation. We process the log files using shell scripts to extract the information we are interested in. However, such scripts needs to be specifically written according to the format of the log files, the information we want to retrieve and the process we are monitoring. Moreover, deciding if an attack was manual or automated is an entirely manual process. We proposed in Section 5.3.3 an algorithm to correlate flows and security events. Such an algorithm allows us to process the majority of the events in the data sets, and it is suitable for the categories of attacks we observed. However, the algorithm would need to be manually tuned to correlate other types of attacks. An ex-

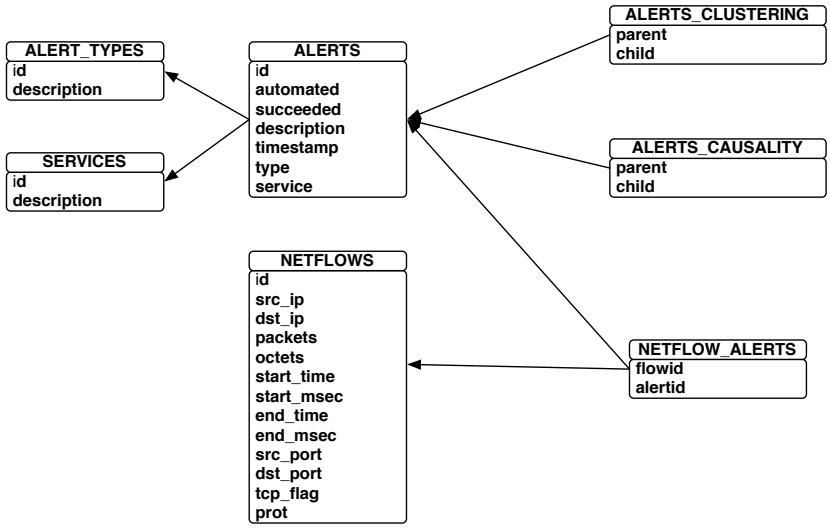


Figure 5.3: Database structure.

ample is the labeling of vertical scans, which target several ports on the same victim. These attacks cannot be handled in the current version since the algorithm is parameterized on a specific service. The SSH typescripts have been analyzed in a completely manual manner, since attackers try to obfuscate their actions using different commands and command names. Finally, clustering and causality information has been introduced manually.

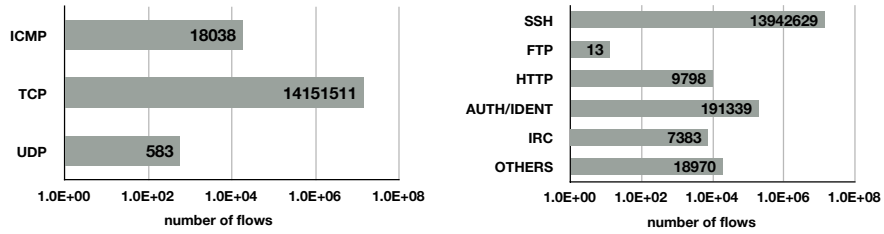
5.4 The labeled data set

The processing of the dumped data and logs, collected over a period of 6 days, resulted in 14.2M flows and 7.6M alerts. Section 5.4.1 will describe the data set breakdown at flows and alerts level, and in Section 5.4.2 we will discuss the main characteristic of the data set traffic.

5.4.1 Flow and alert breakdown

Figure 5.4(a) and 5.4(b) present a breakdown of the flows according to the level 3 protocols and the active services, respectively.

At network level, the collected data set presents a subdivision in only three IP protocols: ICMP, TCP and UDP. The majority of the flows have protocol TCP (almost 99.9% of the flows). A second slice of the data set consists of ICMP traffic (0.1% of the flows). Finally, only a negligible fraction is due to UDP traffic. The level 3 protocol breakdown is consistent with the active services breakdown, shown in Figure 5.4(b). The most active service in the data set has been SSH, followed on the distance by `auth/ident`.



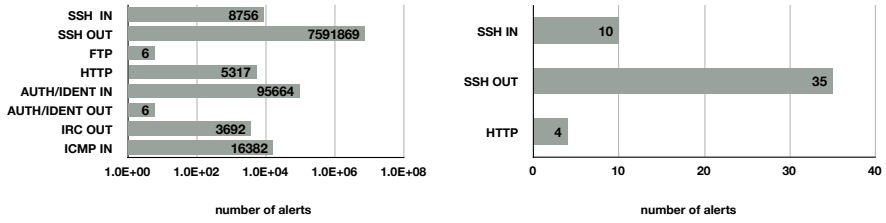
(a) (b)

Figure 5.4: Flow breakdown: the level 3 protocol (a) and the service traffic (b).

Figures 5.5(a) and 5.5(b) show the alert breakdown according to malicious connections and scans on the most active services. SSH and `auth/ident` are the only services for which the monitoring point has been both a target and a source. The host received several thousand SSH connections and it has been responsible of millions of outgoing ones. It has been the target of extensive `auth/ident` requests, triggered by the scanning activity. The monitoring point itself produced only a negligible number of outgoing connections to port 113, due to the FTP server. As shown in Figure 5.5(b), the SSH alerts can be grouped into 45 scans, 10 incoming and 35 targeting remote destinations. We also have been the target of 4 HTTP scans. None of the FTP, `auth/ident` or `irc` alerts can be clustered into scans.

5.4.2 Discussion on the data set

This section discusses the results of the correlation process, pointing out the characteristic of both the labeled and unlabeled traffic. Our correlation process



(a) (b)

Figure 5.5: Alert repartition for basic (a) and cluster (b) types.

succeeded in labeling more than 98.5% of the flows and almost all of the alerts (99.99%).

Malicious traffic

All the labeled traffic is related to the monitored services. Since we did not interfere in any way with the data collection process, i.e., we avoided any form of attack injection and we did not advertise our infrastructure on hacker chats, we can assume that the attacks present in the data set reflect the situation on real networks.

The majority of the attacks targeted the SSH service and they can be divided into two categories: the automated and the manual ones. The first ones are well-known automated dictionary scans, where a program enumerates usernames and passwords from large dictionary files. This attack is particularly easy to observe at flow level, since it generates a new flow for each connection. Attacks of the second type are manual connection attempts, in which the attacker himself logs into the system and not a script. There are 28 of these in our trace and 20 of them succeeded.

The HTTP alerts labeled in our data set are automated attacks that try to compromise the service by performing a scripted series of connections. These scans are executed using tools like Nikto and Whisker, both easily available online. Differently from the SSH connections, no manual HTTP attacks are present, probably due to the very simple HTTP setup chosen.

Regarding the FTP traffic, the data set contains only 6 connections to this

service on our host, during which a FTP session has been opened and immediately closed. Even if the connections did not have a malicious content, this behavior could be part of a reconnaissance performed by an attacker gathering information about the system.

Side-effect traffic

Part of the traffic in our data set is the side effect of attacks, but it cannot be considered by itself malicious. `auth/ident`, ICMP and `irc` traffic fall into this category. The scanning activities have been responsible for the majority of the flows to and from port 113. The service is supposed, indeed, to retrieve additional information about the source of a TCP connection. Regarding the ICMP traffic, more than 120,000 incoming packets have type *time exceeded* and *destination unreachable* and come from networks that the monitoring point scanned.

The analysis of our host showed that a hacker installed an IRC proxy that received chat messages from several channels. It appears that no message has been sent from the monitoring point and that the channels have not been used for any malicious activity.

Unknown traffic and uncorrelated alerts

For a small fraction of the data set, we cannot establish the malicious/benign nature of the traffic. This traffic consists of three main components: (i) SSH connections for which it has not been possible to find a matching alert, (ii) heterogeneous traffic, containing all the flows having as destination a closed port on our host, and (iii) some non-malicious connections to the HTTP and `vnc` services. The `vnc` service was accessible because of the XEN virtualization software running on the monitoring point. Strangely, no attacker identified this service and tried to compromise it by using known vulnerabilities or performing a brute-force dictionary attack on the password. The `vnc` flows in the data set are originating from two hosts which contacted the service but did not complete the connection.

Regarding the alerts, for few of them (0.01%) no matching traffic has been found. This makes us aware that during the collection phase some packets that reached the monitoring point may not have been recorded. A possible explanation is that the OS kernel dropped some of the packets during the moment of higher network activity, such as the scanning activities.

5.5 Lesson learned and conclusions

This chapter dealt with the topic of manually creating a flow-based labeled data set. A labeled data set provides the ground-truth knowledge for validation and evaluation of IDSs. In the specific case of flow-based IDSs, as far as we know, such a labeled data set did not exist prior to our research.

In Section 5.1, we motivated the need and the importance of ground truth and describe the requirements we put on the data set. In Section 5.2, we reported on our operational experience in collecting meaningful data as basis for our data set: in particular, we described the decision process that lead to a honeypot-based setup in relation to both measurement scale and flow collection location. After the data collection, Section 5.3 described the labeling process. We proposed a semi-automatic labeling procedure that allows to process the majority of the collected flows and correlate them with the relevant security event. Finally, Section 5.4 presented the final data set.

The main contribution of this chapter is to present the first labeled data set for flow-based intrusion detection. The data set is available in anonymized form at the address: *http://traces.simpleweb.org*.

While approaching the problem of creating a labeled data set, we learned that the choice of the proper data collection infrastructure is a crucial point. It has indeed impact not only on the feasibility of the labeling, but also on the trustworthiness of the result. We studied several data collection infrastructures, enlightening strengths and drawbacks. We conclude that, in the case of flow-based data sets, the most promising measurement setup is monitoring a single host with enhanced logging capabilities. In the specific context of this experiment, the host was a honeypot. The information collected permitted us to create a database of flows and security events (alerts).

However, we are aware of the limitation that our approach entails. In particular, the presented trace mainly consists of malicious traffic. For IDS evaluation, this means that our data set allows to detect false negatives (missed events) but not false positives (benign traffic classified as malicious). A possible extension is therefore a monitor setup that would allow to capture also benign traffic. An example would be a server that is daily accessed by benign users. Monitoring such a server would result in a trace with a more balanced content of malicious and normal traffic. We believe that an approach similar to the one we presented in this chapter will be useful also in this scenario.

We also described a semi-automated correlation process that matches flows with security events and, in addition, reflects the causality relations between the security events themselves. The results of the correlation process show that

we have been able to label more than 98% of the flows in the trace. However, the correlation process proves that, although we limited our measurements to a single host, labeling remains a complex task that requires human intervention. Only some of the steps in the correlation process can be partially automated and deep domain knowledge is needed. In Chapter 6 we will approach the ground-truth problem from a modeling prospective, proposing a procedure for creating labeled data set in an automatic manner.

CHAPTER 6

Automatic ground truth generation

In Chapter 5 we showed that manual creation of a flow-based data set for which the ground truth is known is a time and resource-consuming task. Moreover, such a data set is the result of the trade-off between realism on one side, and correctness and completeness on the other. Therefore, the data set we proposed, even though it is a promising beginning, still presents limitations, the main one being the fact that it mainly consists of malicious traffic.

To overcome these problems, the goal of this chapter is to develop models that can be used to generate ground truth in an automatic manner. We present our general approach in Section 6.1. To develop these models, we will first analyze the traffic flowing over the network in Section 6.2. This analysis will allow us to develop specific models in Section 6.3. The outcome of our models will then be validated against real traffic in Section 6.4. In Section 6.5 we present a few remarks on the generality of the proposed models. Finally, in Section 6.6 we summarize our findings.

6.1 General approach

As stated above, the goal of the research presented in this chapter is to automatically generate ground truth for flow-based intrusion detection. In Chapter 4, we showed that time series derived by flow information can be a powerful tool for characterizing attacks. Our proposal is therefore to enrich such time series with ground-truth information. More specifically, we refine our goal by aiming to *synthetically generate* time series of flows, packets and bytes for which the ground truth is known.

Our approach is based on the following cornerstones:

- We advocate the use of models to describe the significant characteristics of flow, packet and byte time series and their evolution over time.

In order to do that, we need of a modeling framework that (i) explicitly takes into account time, and (ii) can be used in a generative manner. From the literature, a promising approach is provided by Hidden Markov Models (HMMs). HMMs are effective in modeling sequential data [17]. Introduced in the early 1970s [10], they have been successfully applied to different scientific fields. Examples are biological sequence analysis [42], speech recognition [122] and pattern recognition [50]. HMMs can be trained on real data and their main characteristic is the ability to capture the temporal behavior of the observed processes. Moreover, they can be used for generation purposes. We describe the background on HMMs in Appendix A.

- Once we are able to generate time series, we need to enrich them with ground-truth information. To achieve this, we need to know, at each instant of time in our time series, if an attack is ongoing or if the network is safe. We propose, therefore, to model malicious and benign traffic separately. This allows us to study and model in detail malicious and normal traffic behavior. Moreover, we assume the time series for an entire network to be the combined result of benign and malicious network activities. We then generate malicious and benign SSH traffic separately, and combine them at a later stage into a meaningful time series for which the ground truth is known.

In this chapter, we focus on SSH traffic, which will be in the remaining part of this thesis our running example. As pointed out in Chapter 4, the SSH protocol is regularly under attack, and therefore it allows us to collect a suitable amount of attack instances on which we can base our modeling. However, we will argue later that our approach can be adapted to other type of traffic, in Section 6.5.

6.2 Flow-based characterization of SSH traffic

Our approach aims to model time series for SSH attack and normal traffic. To do that, the first step is to gain knowledge about what such a traffic looks like "in the wild". In this section, we therefore analyze and characterize SSH time series derived by real flow measurements. First, we describe how an SSH dictionary attack looks like if only flow information is available. Second, we provide an example of a network time series, including both malicious and benign traffic. Here, and in Chapter 7, we define benign traffic relatively to the

attack traffic. In doing so, we defined as *normal* all the traffic that has not been directly generated or received by an attacker.

6.2.1 SSH dictionary attack

SSH dictionary attack is one of the most common threats in cyber space [125]. The attack behavior that we will describe is typical for such attack. While monitoring the University of Twente (UT) network, we observe on average two SSH dictionary attacks with these characteristics per day. In this section, we present, as an example of such class of attacks, the traffic generated by a host known to have performed a SSH dictionary attack against the UT network. The attack took place in the early afternoon of July 16, 2008 and lasted approximately 40 minutes. During this interval, approximately 8300 distinct university hosts have been scanned. The attack generated a volume of traffic of approximately 32400 flows, 279000 packets and 30.5MB.

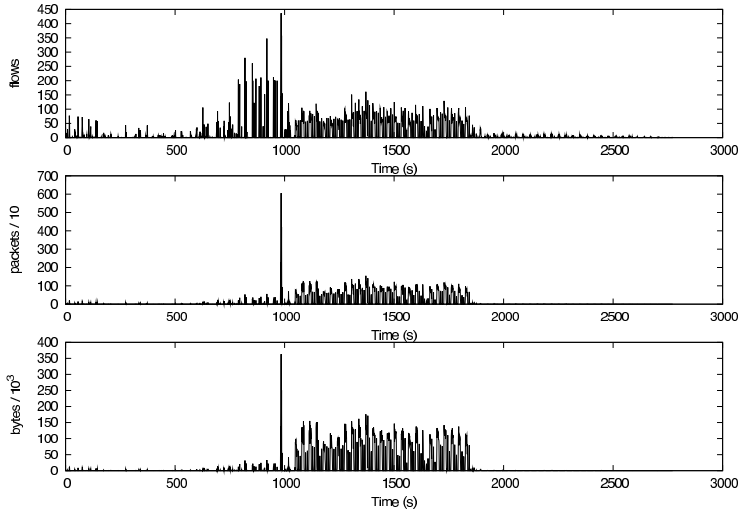


Figure 6.1: Overview of an SSH dictionary attack at flow level .

As explained in Chapter 4, we study the evolution of the attack by considering the time series of flows, packets as well as bytes that it generates. In Chapter 4, we presented such time series at a time resolution of 600 seconds. For the analysis in Chapter 4, this resolution provided a good overview of the

overall traffic behavior. However, for the aim of modeling, such a coarse time resolution can potentially introduce artifacts, due to the data aggregation into the time bins. In this chapter, we propose to study traffic time series at a finer granularity, which provide us with detailed information on the traffic evolution. We therefore present time series at 1 second resolution.

Figure 6.1 shows, for the attacker, the evolution over time of (i) the number of flows created per second, (ii) the number of packets transferred per second, and (iii) the number of bytes transferred per second. Each value in the time series accounts for both the traffic generated by the attacker and the traffic that he receives from the victims. During the attack, the intensity of the flow, packet and byte time series vary. In the flow time series, we can see that in about the first 1000 seconds the attack intensity grows, reaching a peak of 450 flows/s. After that, the number of flows per second drops abruptly and roughly stabilizes around 100 flows/s. Finally, the attack activity slowly fades off in the last 500 seconds.

The packet and byte time series both show a similar trend, with little traffic exchanged during the first 1000 seconds of the attack, then a high peak followed by a roughly stable traffic intensity of around 100 packets/s and 150 bytes/s. Finally, the attack slowly dies. Moreover, after the first 1000 seconds, the shown trend closely resembles the one of the flow time series. This behavior suggests that during an SSH dictionary attack, the flow, packet and byte time series are mutually correlated. In particular, the time series of packets and bytes appear to be strongly correlated during the entire attack, while the flow time series only partially follows the packet and byte trend.

A different view on the attacks is given by Figure 6.2. Each mark in the graph represents either a malicious SSH connection from the attacker to a victim or the answering connection from the victim back to the attacker. The y-axis gives the 65,535 possible destination addresses in the university network. We identify *three attack phases*. During the *scanning phase* (first 1000 seconds), the attacker performs a sequential SSH scan spanning over the entire network address space. In this phase, the attacker gathers information on which hosts vulnerable SSH services run. Only few victims respond to the attack. Once this phase is completed, the attacker initiates a brute-force user/password guessing attack based on the information stored in the attacker's dictionary file (*brute-force phase*). In this phase, only a small subset of the hosts in the network is involved, those who responded to the SSH connection attempt. This phase corresponds to the second block of 1000 seconds and it is characterized by a high interaction between the attacker and the victims. Finally, after about 2000 seconds since the beginning of the attack, the brute-force phase ends. Neverthe-

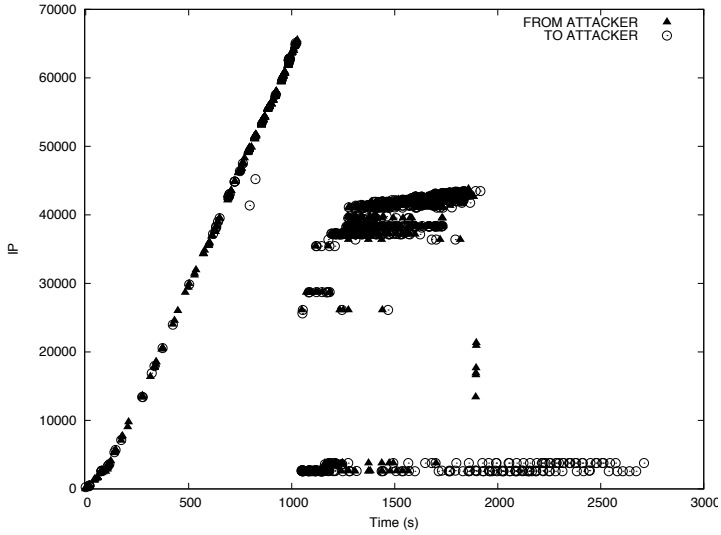


Figure 6.2: Connection pattern between an attacker and the victims .

less, the time series in Figure 6.1 show that after this moment in time there is still traffic. Form Figure 6.2 it is now evident that the residual traffic is due to a small number of compromised hosts that communicate with the attacker. We refer to this final phase as the *die-off phase*.

Although the three attack phases are clearly visible in Figure 6.2, they are not so clearly identifiable from the flow, packet and byte time series shown in Figure 6.1. However, the fact that the three time series are correlated allows us to derive a more suitable measure. Figure 6.3 shows the evolution over time of the *packets per flow*, again at 1 second resolution. Using this measure, the three phases are clearly visible. The scanning phase is characterized by only a few packets per flow, on average between 1 and 1.5. These values are consistent with a scenario in which several three-way handshakes are initiated but only few are completed. When the brute-force phase starts, the number of packets per flow has a sharp rise: from 1.5 to an average of about 11. During this phase, several user/password combinations are tested against the same victim. This explains why the attacker produces a higher number of packets per flow. Finally, the die-off phase sees again only few packets per flow. In the majority of the cases, we observe only one packet per flow. Such traffic may be due

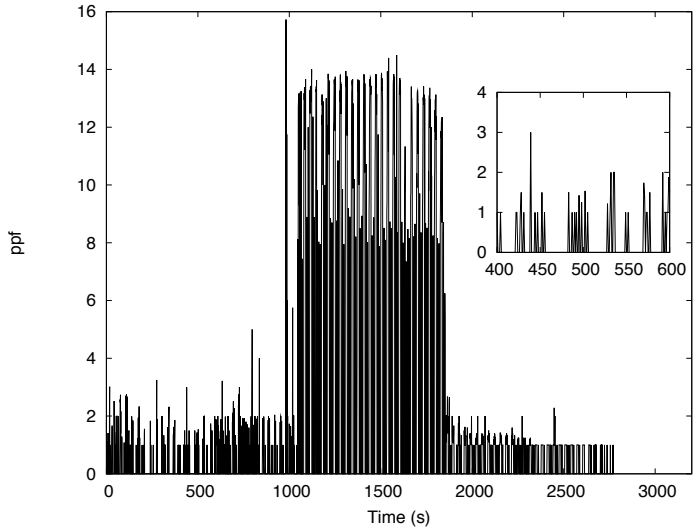


Figure 6.3: Packets per flow time series during a scan

to SSH keepalive server options, and it also tell us that the attacker does not properly terminate the SSH sessions (we do not observe any traffic from the attacker side). The variation of packets per flow over time therefore seems to be a key characteristic of the behavior of an SSH dictionary attack. It moreover shows that the flow, packet and byte time series still carry enough information to characterize the attack. Finally, in Figure 6.3, we also show that, at a deeper analysis, the time series show that the activity pattern is not constant in time: each second of activity is often followed by one or more seconds of inactivity. An example of this phenomenon, which will become important when modeling the attack, is shown in the subplot in Figure 6.3.

6.2.2 An SSH time series

We now consider a more general case, namely an SSH time series, consisting of both benign and malicious traffic. Figure 6.4 presents the flow, packet and byte time series at 1 second resolution for 24 hours of SSH traffic. The trace has been collected on the UT network, and it contains one SSH attack. The time series show both normal and malicious SSH traffic. The time window during

the attack is depicted in black.

The normal traffic flow time series appears as a baseline of modest intensity, ranging from a maximum of approximately 50 flows/s in the night to a maximum of approximately 75 flows/s around midday. At a closer look, as in the case of the SSH dictionary attack, also the normal time series appears as a sequence of active and inactive time bins. The time series also shows a moderate day/night pattern, with the day period between 8:00 and 16:00. The packet and byte time series appear strongly correlated and present several peaks during the observation period. These peaks are usually caused by file transfers over SSH and do not correspond to any considerable variation in the flow time series. Packets and bytes show therefore only a modest correlation with the flow time series. In the case of the normal SSH traffic time series, we cannot conclude that there exists a key metrics for describing the traffic evolution, opposed to the packets per flow time series for the SSH attack traffic (cf. Figure 6.3). However, in order to generate meaningful time series, we should properly model the correlations between flows, packets and bytes. We will discuss the correlation values in Section 6.4.

Between 8:00 and 9:00, the University of Twente network has been the target of a SSH dictionary attack, clearly noticeable as a peak in the flow time series. The attack lasts 26 minutes and reaches an intensity of 360 flows/sec. Considering the time window in which the attack took place, we once again confirm the observation in Chapter 4: we observe that an SSH dictionary attack has only a minimal impact on the packet and byte time series. The amount of traffic in both packet and byte time series is indeed comparable to a network that is not under attack. In this specific case, therefore, the number of flows per second is a suitable metric for detecting SSH dictionary attacks.

In the following section, we will describe how the flow-based characterization that we presented in this section can be used to model SSH time series.

6.3 The traffic models

The aim of our models is to generate meaningful synthetic flow, packet and byte time series, both in terms of SSH attack and normal traffic. In this section, we describe the SSH traffic models we propose. We based our modeling choices on the observations in Section 6.2. We aim to map key characteristics of the attack and normal traffic, such as for example the phases of an SSH dictionary attack, directly in our models. Sections 6.3.1 and 6.3.2 will introduce the attack model and the normal traffic model, respectively.

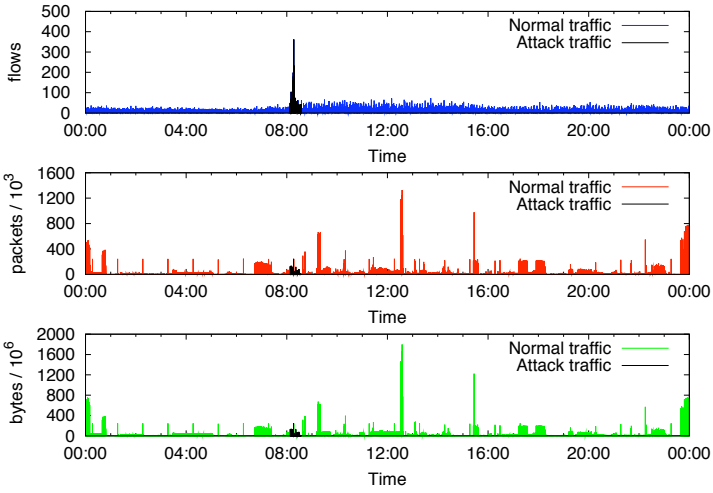


Figure 6.4: Example of an SSH network time series.

6.3.1 SSH dictionary attack traffic

This section introduces the Markov chain and the output probabilities for a model describing the SSH dictionary attack that we described in Section 6.2.

The Markov chain

The SSH dictionary attack introduced in Section 6.2 consists of three phases: a scanning phase, a brute-force phase and a die-off phase. Moreover, as shown in the subplot of Figure 6.3, the time series appear to be constituted of interleaved sequences of active and inactive time bins. We make use of these characteristics to define a Discrete Time Markov Chain (DTMC) for our model. Our model consists of the following seven states, which are shown in Figure 6.5, together with the possible state transitions:

- the states S_i, $i = 1, 2, 3$. In these states, the attacker is *active* and causes network traffic;
- the states I_i, $i = 1, 2, 3$. In these states, the attacker is *temporary inactive*;

- the end state End.

The state S_1 is the starting state, which denotes the beginning of the scan activity. The states S_1 and I_1 model the scanning phase of the attack. For the specific class of attacks introduced in Section 6.2, once the attack moves from the scanning phase to the brute-force phase (states S_2 and I_2), it will not return to the scanning phase. This models the fact that, for the analyzed attack type, the scan will be performed only once per attack. The die-off phase (states S_3 and I_3), on the other hand, partially overlaps with the brute-force phase, as shown by the fact that the states $\{S_2, I_2, S_3\}$ form a fully connected chain. This decision is due to the fact that the die-off phase mainly consists of packets flowing from the compromised hosts to the attacker. Since the die off phase may be due to SSH keepalive packets, they can occur also before the brute-force phase is completed. However, as it will be shown in Section 6.4.2, the transition probabilities learned during the training will favor transitions in the same phase. This is because it is only after the brute-force phase is concluded that the die-off phase becomes manifest. Finally, attacks will have different durations. We therefore model them by introducing the state End, that indicated the end of an attack. Transitions to the End states have been observed from each active state S_i, thus reflecting the fact that some attacks stop after the scan phase or that the die-off phase may not exist.

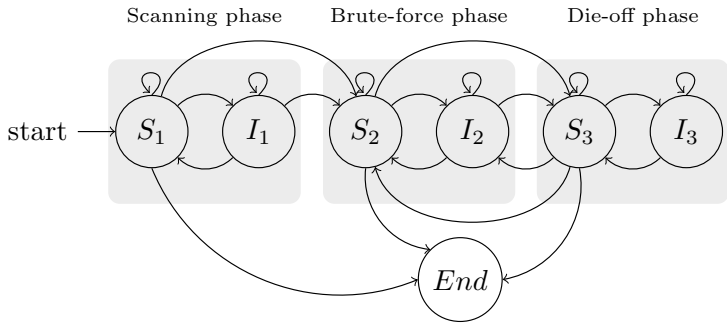


Figure 6.5: DTMC for SSH dictionary attack traffic.

The output probabilities

As noted previously in this section, we aim to reproduce meaningful flow, packet and byte time series. Hence, at each transition, our model should out-

put a triple, that we indicate with (F, P, B), with the values of flows, packets and bytes for the latest time bin of the time series.

However, it is important to keep in mind that these three values are not independent, as shown in Section 6.2. A possibility to generate correctly correlated values for the three time series would be to rely, for each state of the model, on a joint output probability distribution $\mathbb{P}_{F,P,B}$. However, this solution proved to be not efficient once implemented, due to the overhead in handling a triple-joint empirical distribution. In the following, we will present an alternative approach that approximates the triple-joint probability distribution $\mathbb{P}_{F,P,B}$ by means of random variables indicating the number of flows F, the number of packets per flow PPF and the number of bytes per packet BPP. To each active state S_1, S_2 and S_3 in our model, we assign the following empirical distributions:

- an empirical probability distribution of the number of flows per time bin, $\mathbb{P}_F$;
- an empirical joint probability distribution of packets per flow (PPF) and bytes per packet (BPP) per time bin, $\mathbb{P}_{PPF,BPP}$.

At each transition, values of F, PPF and BPP are generated according to the empirical distributions associated to the current destination state. The joint probability distribution $\mathbb{P}_{PPF,BPP}$ ensures the strong correlation between packets and bytes that we have observed in the data. Note that in the states I_1, I_2 and I_3 the attacker is by definition temporarily inactive. Therefore the probability distributions are such that $\mathbb{P}(F = 0) = 1$ and $\mathbb{P}(PPF = 0, BPP = 0) = 1$ and the triple $(F = 0, PPF = 0, BPP = 0)$ is the only allowed output.

6.3.2 Normal SSH traffic

We will now describe the Markov chain and the output probabilities that define the normal SSH traffic model. The normal SSH traffic model is more general than the one we presented in Section 6.3.1 for the SSH dictionary attack. The SSH attack model describes the traffic generated by a single attacker and its victims. The normal traffic model, on the other hand, describes the legitimate SSH traffic present on a network as a whole. This allows us to describe the normal SSH traffic by means of a simple two-state Markov chain, which we will describe in the following.

The Markov chain

As for the SSH dictionary traffic, also a normal traffic time series consists of a sequence of active and inactive time bins. We model this behavior directly in the state chain, as shown in Figure 6.6. The DTMC consists of two states:

- the state A, in which SSH traffic is present on the network (*activity*);
- the state I, in which no SSH traffic is present on the network (*inactivity*).

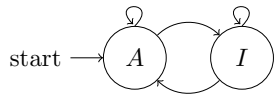


Figure 6.6: DTMC for normal SSH traffic.

The stochastic process starts in state A, that models the presence of SSH traffic activity. Since we consider a network as a process that is persistent in time, no End state has been introduced in the model.

The output probabilities

Similarly to what has been done for the SSH dictionary attack model, also in the case of the normal output probabilities, we propose an approach alternative to the triple-joint distribution $\mathbb{P}_{F,P,B}$. We rely also in this case on intermediate measures such as the packets per flow PPF and the bytes per packet BPP. However, differently from the attack model, we now ensure a correct correlation between flow, packet and byte time series by means of two joint distributions. The following distributions are therefore assigned to the state A:

- an empirical joint probability distribution of flows (F) and packets per flow (PPF) per time bin, $\mathbb{P}_{F,PPF}$;
- an empirical joint probability distribution of packets per flow and bytes per packet (BPP) per time bin, $\mathbb{P}_{PPF,BPP}$.

At each transition, a realization of the random variables F, PPF and BPP is generated. In state I, the model outputs only the triple $(F = 0, PPF = 0, BPP = 0)$.

6.4 Model validation

We validate the proposed models by verifying that the synthetic time series that we generate maintain the statistical characteristics of the original data sets. This section describes our testing methodology and presents the experimental results we obtained.

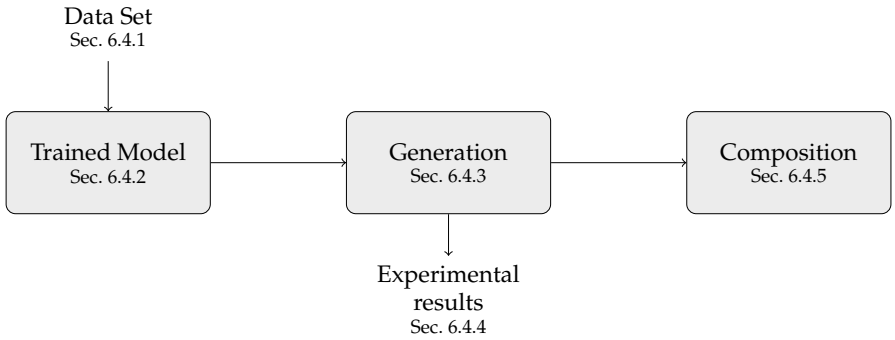


Figure 6.7: Model validation structure.

Figure 6.7 provides a schematic representation of how this section is organized. In Section 6.4.1 we introduce the data set we use for validation. Each data set, as we will explain in Section 6.4.1, contains SSH normal and attack traffic. The validation proceeds accordingly to the following steps, as described in Figure 6.7:

1. for each data set, we train an attack or normal HMM. We present the trained models in Section 6.4.2;

2. each trained model is used to generate a set of *synthetic time series*. As explained in Section 6.3, we aim to generate time series for flows, packets and bytes. The generation procedure is described in Section 6.4.3;

3. the statistical properties of the synthetic time series are analyzed to check if the models are able to approximate the original traffic. We describe our testing methodology and provide our experimental results in Section 6.4.4;

4. finally, we provide an example of synthetic network time series, including both normal and attack traffic. We describe how we create such a time series in Section 6.4.5.

The aim of the validation is to verify whether the models introduce errors in creating synthetic traffic. Such a situation could happen due to the stochastic nature of our models. We therefore train and test each of our models separately. Another reason for having separate training and testing is that several months have passed between the collections of our data sets. We argue therefore that it is possible, if not likely, that some of the statistical properties we consider have changed over time.

6.4.1 Data sets

We evaluate our models on two data sets of SSH traffic collected at the University of Twente network. We refer to these data sets as *Set 1* and *Set 2*. Each data set spans over 8 consecutive days and contains benign traffic as well as several SSH dictionary attacks of the type described in Section 6.2. The first data set has been collected in the period July 13–20, 2008 and contains 13 attacks; the second one in the period April 19–26, 2009 and contains 17 attacks. Table 6.1 presents an overview of the attacks in the studied data sets in terms of the average values of flows, packets and bytes per second. This initial analysis shows that the average values of flows, packets and bytes over time have changed in time. In *Set 2*, the attackers appear to produce on average more than twice the amount of packets and bytes compared to *Set 1*. This suggests that, while the attack mechanism stays the same, the attack intensities are likely to vary in the course of time. A possible explanation can be found in the dictionary used to perform the brute-force phase. By analyzing attack scripts on compromised machines and honeypots, we observe that the dictionary changes over time, including a longer list of user/password pairs. Similarly, Table 6.2 summaries the main characteristics of the normal SSH traffic in the data sets. Differently from the malicious traffic, the statistical characteristics of the normal traffic are subdued to a smaller variation over time.

Data Set	Attacks	F/s	P/s	B/s
Set 1 (13-20 July 2008)	17	11.06	66.91	7337.33
Set 2 (19-26 April 2009)	13	15.80	150.52	19016.00

Table 6.1: Statistical characteristics of the SSH attack traffic in the collected data sets.

Data Set	F/s	P/s	B/s
Set 1 (13-20 July 2008)	3.28	2318.99	2031952.31
Set 2 (19-26 April 2009)	3.14	1909.16	1564403.71

Table 6.2: Statistical characteristics of the SSH normal traffic in the collected data sets.

6.4.2 The trained models

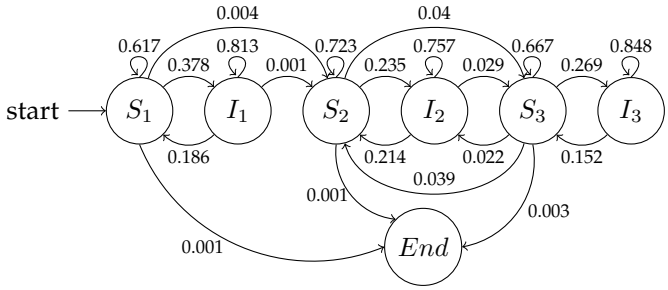
We present here the Markov chain for the attack traffic and the normal traffic models as result of the training (see Section A.2). We also provide details relatively to the empirical output probability distributions.

Figures 6.8(a) and 6.8(b) show the Markov chains for the attack traffic, after the training on *Set 1* and *Set 2*, respectively. Both models favor transitions within the same phase, as we see by looking at the transition probabilities. However, note that there are some differences between the two chains. In particular, in *Set 2* there is no transition between S_1 and End, as well as I_1 and S_2. This means that, in *Set 2*, such transition never occurs. Finally, for completeness, we present additional information on the empirical distributions learned from the attack training sets in Table 6.3. In Table 6.3, we indicate the scanning, brute-force and die-off phases with the subscript $\cdot_1$, $\cdot_2$, $\cdot_3$, respectively. The table confirms that the intensity of the attacks is likely to change over time, especially during the scanning phase (phase 1). The maximum number of flows in this phase has indeed almost doubled from *Set 1* to *Set 2*. However, we can also observe that the values for PPF and BPP have a large similarity between the two training sets. This confirms our intuition that, while the intensity of the attack may change, the attack method remains the same.

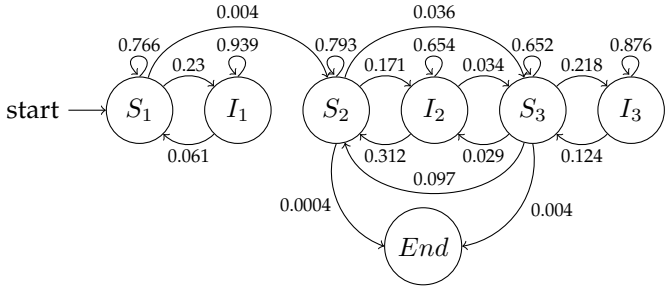
Figure 6.9 shows the Markov chains for the normal traffic. In this case the training on *Set 1* and *Set 2* results in the same chain, with similar transition probabilities. This suggests that, even though several months have passed between the creation of the two data sets, the normal traffic characteristics are likely to change only slightly over time. Such conclusion is supported also by the measurements in Table 6.2 and in Table 6.4, where we presents the details of the empirical distributions learned from the training sets in the case of normal SSH traffic.

6.4.3 Time series generation

We define a synthetic time series as a sequence of observations that the models output when a random path is taken. The path begins in the starting state of



(a) *Set 1*



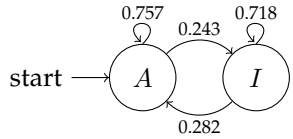
(b) *Set 2*

Figure 6.8: Markov chain for the attack traffic models.

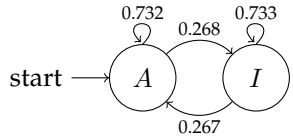
the models (S_1 or A). The generation process can be summarized as follows. Let us assume the model to be in state s_i:

1. at time $t \in \mathbb{N}$, the model jumps from the current state s_i to the next state s_j according to the transition probabilities a_{ij}, $j = 1, \ldots n$;

2. if s_j is the End state, in the case of the attack model, or if we reach a specified number of transitions, in the case of the normal model, the path will be concluded and the trace ends. Regarding this step, it is important



(a) *Set 1*



(b) *Set 2*

Figure 6.9: Markov chain for the normal traffic models.

	Set 1				**Set 2**			
	Min	**Max**	**Avg**	**Std**	**Min**	**Max**	**Avg**	**Std**
F_1	1	789	42.26	76.54	1	3825	26.87	101.33
F_2	1	519	50.45	54.63	1	860	58.03	67.48
F_3	1	227	5.40	10.24	1	250	6.77	12.86
PPF_1	1	26.48	1.96	2.03	1	27	2.28	2.65
PPF_2	1	16.50	10.63	2.50	1	17	11.01	2.00
PPF_3	1	5	1.32	0.65	1	5	1.5	0.90
BPP_1	40	156.42	58.48	6.70	40	225.71	62.15	16.69
BPP_2	50.88	267.27	125.67	13.66	52	319.42	132.33	13.31
BPP_3	40	836	79.19	78.31	46	1148	73.07	66.20

Table 6.3: Empirical distribution details for F, PPF and BPP in each attack phase for the training data sets.

to keep in mind that an attack time series is relative to a single attacker and its victims, while a normal SSH time series describes the traffic of an entire network;

	Set 1				Set 2			
	Min	**Max**	**Avg**	**Std**	**Min**	**Max**	**Avg**	**Std**
F	1	1026	6.32	6.37	1	1074	6.40	10.02
PPF	1	1094587	965.44	6519	1	766816.5	608.11	5472.20
BPP	40	1500	250.87	290.07	40	1500	251.07	276.18

Table 6.4: Empirical distribution details for F, PPF and BPP for the normal traffic in the training data sets.

3. once s_j has been selected, the model randomly generates values of F, PPF and BPP, by inverting the empirical cumulative distribution functions derived by the output distributions associated to the state;
4. the model outputs the triple (F, P, B), calculated based on the random values drawn in the previous step:

$$\begin{aligned} P &= PPF \cdot F, \\ B &= BPP \cdot PPF \cdot F; \end{aligned}$$

5. once the observations have been emitted, the process iterates from Step 1.

At each iteration, the model generates a triple (F, P, B). The generation is independent from the previous outputs and is controlled only by the (joint) empirical probability distributions of F, PPF and BPP associated with the current state.

The presented models have different mechanism for deciding the length of a generated time series, as indicated in Step 2 of the generation procedure. The attack model controls autonomously the duration of a trace, since a trace ends only when a transition to the End state is randomly selected. However, this is not the case for the normal traffic model, where we simulate an ergodic process. In this second case, we manually set the length of a trace. In our generation process, we set the length of a sequence of normal traffic to $n = 50000$ transitions. Each transition corresponds to a time bin of one second in the synthetic time series.

6.4.4 Testing methodology and experimental results

Our testing methodology aims to measure the statistical characteristics of a set of synthetic traces and compare them to the ones of the original data sets *Set 1*

and *Set 2*. Each statistical metric is calculated for flows, packets and bytes. We are interested in three types of statistical measures:

- the mean and standard deviation of flows (μ_F, σ_F), packets (μ_P, σ_P) and bytes (μ_B, σ_B) over time. These measures describe the *overall behavior* of flows, packets and bytes independently of each other in a trace;
- the correlation coefficients between, respectively, flows and packets ρ_{FP}, flows and bytes ρ_{FB} and packets and bytes ρ_{PB}. These measures describe the *dependence* between flows, packets and bytes in the same trace;
- autocorrelation of lag 1, therefore between consecutive time bins, of flows (R_F), packets (R_P) and bytes (R_B). The autocorrelation captures the *evolution* of a trace over time, measuring the inter-relation of the trace with itself in different moments in time.

In our experimental results, we calculate the average values of each measure for both the original data sets and the synthetic ones. For the synthetic trace, we also calculate the 95% confidence intervals. Each synthetic data set consists of 300 traces. Finally, we evaluate how well the synthetic traces approximate the original ones. To do so, we calculate for each measure m the relative error, express in percentage, between the original traces and the synthetic ones, as follows:

$$Err = \frac{|m_{orig} - m_{syn}|}{m_{orig}} \cdot 100\%.$$

This section describes our experimental results, summarized in Tables 6.5 and 6.6, as follows. The columns *Set 1* and *Set 2* show the statistical metrics for the original data sets. Similarly, the columns *Synthetic 1* and *Synthetic 2* present the statistical values we calculate for our synthetic time series and the respective 95% confidence intervals. Note that the data set *Synthetic 1* has been generated by a model trained on the original data set *Set 1*, and *Synthetic 2* by a model trained on *Set 2*. Finally, the column *Err* gives the relative error between the original values and the synthetic ones, expressed as percentage.

For *Set 1* and *Set 2*, both the attack model and the normal traffic model correctly approximate the mean, standard deviation and correlation coefficients. The relative errors are indeed, in almost all cases, smaller than 10% for the synthetic attack traffic and smaller than 1.5% for the synthetic normal traffic. In all cases, the confidence intervals are close to the average measure values.

Both models, however, fail to fully capture the autocorrelation of the time series. This measure is in all cases lower than the original. This means that the original time series have a higher regularity, while the models introduce additional randomness in the synthetic time series. This phenomenon is related to the Markov property and to the output probabilities. At each transition, indeed, the model chooses the next state only based on the current one, without memory of any past transition. Moreover, at each transition, the model emits an output that is independent from the previous ones. As a consequence, for example, the SSH dictionary attack model imitates a less regular brute-force phase, while the normal traffic model, for example, does not include the moderate night/day pattern shown in the original flow time series.

6.4.5 Time series composition

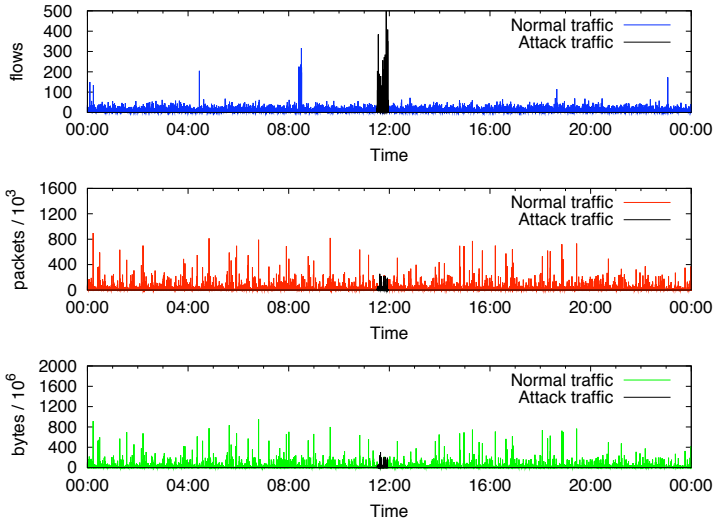


Figure 6.10: Example of synthetic network time series.

Our network traffic generation is based on the independent generation of synthetic time series for normal and malicious traffic. We proceed as follows:

- a normal traffic time series of length n is generated. For this experiment we choose $n = 86400$, corresponding to a time series of one day;

	Set 1	Synthetic 1	Err (%)
μ_F	11.06	12.27 $\pm$ 0.33	10.9
μ_P	66.91	66.66 $\pm$ 3.67	4.6
μ_B	7337.33	7524.73 $\pm$ 523.11	2.5
σ_F	36.45	38.33 $\pm$ 1.12	5.1
σ_P	324.29	243.43 $\pm$ 10.91	24.9
σ_B	28510.35	28345.60 $\pm$1616.63	0.5
ρ_{FP}	0.79	0.79 $\pm$ 0.012	0.1
ρ_{FB}	0.76	0.74 $\pm$ 0.016	2.3
ρ_{PB}	0.94	0.98 $\pm$ 0.002	4.7
R_F	0.46	0.23 $\pm$ 0.009	49.8
R_P	0.56	0.25 $\pm$ 0.012	54.7
R_B	0.58	0.26 $\pm$ 0.012	54.9

(a) Set 1

	Set 2	Synthetic 2	Err (%)
μ_F	15.80	15.15 $\pm$ 0.65	4.1
μ_P	150.52	138.85 $\pm$ 8.5	7.7
μ_B	19016.00	18107.88 $\pm$ 1153.53	4.7
σ_F	40.0	47.01 $\pm$ 1.87	0.174
σ_P	379.38	419.16 $\pm$ 16.55	10.4
σ_B	47060.07	55378.58 $\pm$ 2239.91	17.6
ρ_{FP}	0.83	0.86 $\pm$ 0.01	3.9
ρ_{FB}	0.79	0.81 $\pm$ 0.01	2.4
ρ_{PB}	0.98	0.98 $\pm$ 0.001	0.1
R_F	0.64	0.26 $\pm$ 0.01	59.3
R_P	0.71	0.30 $\pm$ 0.009	57.7
R_B	0.75	0.30 $\pm$ 0.009	59.2

(b) Set 2

Table 6.5: Numerical results for the SSH dictionary attack.

	Set 1	**Synthetic 1**	**Err (%)**
μ_F	3.28	3.28 $\pm$ 0.003	0.04
μ_P	2318.99	2326.84 $\pm$ 10.61	0.33
μ_B	2031952.31	2028120.93 $\pm$ 11141.46	0.19
σ_F	5.57	5.57 $\pm$ 0.06	0.01
σ_P	19895.84	19942.63 $\pm$ 157.65	0.23
σ_B	20828807.55	20885405.66 $\pm$ 192057.48	0.27
ρ_{FP}	0.15	0.15 $\pm$ 0.002	1.19
ρ_{FB}	0.12	0.12 $\pm$ 0.001	0.09
ρ_{PB}	0.97	0.96 $\pm$ 0.0005	0.12
R_F	0.53	0.36 $\pm$ 0.004	32.0
R_P	0.016	0.02 $\pm$ 0.0005	13.3
R_B	0.008	0.01 $\pm$ 0.0005	46.5

(a) Set 1

	Set 2	**Synthetic 2**	**Err (%)**
μ_F	3.14	3.14 $\pm$ 0.004	0.11
μ_P	1909.16	1912.80 $\pm$ 10.52	0.19
μ_B	1564403.71	1565983.35 $\pm$ 10199.39	0.1
σ_F	7.13	7.15 $\pm$ 0.078	0.2
σ_P	20930.8	20846.08 $\pm$ 192.16	0.4
σ_B	19526211.3	19588363.33 $\pm$ 150151.74	0.3
ρ_{FP}	0.14	0.14 $\pm$ 0.001	1.42
ρ_{FB}	0.12	0.12 $\pm$ 0.001	0.33
ρ_{PB}	0.95	0.95 $\pm$ 0.002	0.33
R_F	0.53	0.23 $\pm$ 0.0036	56.1
R_P	0.03	0.01 $\pm$ 0.0006	65.2
R_B	0.01	0.008 $\pm$ 0.0006	48.0

(b) Set 2

Table 6.6: Numerical results for the normal SSH traffic model.

- a random number of attack time series is generated. We assume the network to be under attack one or two times per day;
- the starting time of each attack is randomly positioned in the time range of the normal traffic time series;
- we super-impose the attack time series to the normal traffic time series, that is, for each time bin, we calculate the total number of flows, packets and bytes as the sum of the normal traffic component and the attack component.

Figure 6.10 displays an example of synthetic network time series that we compare with the traffic in Figure 6.4. The synthetic network time series spans over one day and contains one synthetic attack. The attack lasts 20 minutes and presents a peak of 499 flows/s. Both the original attack and the synthetic one are comparable in duration and intensity. However, a comparison of Figure 6.4 and Figure 6.10 shows that the original time series appears more structured than the synthetic ones. In the original flow time series we see a moderate night/day pattern, already described in Section 6.2, that is not captured by the synthetic time series. Regarding the packet and byte time series, in the original traffic we observe that the moments of higher activity are grouped in bursts, while in the synthetic time series the traffic appears equally noisy over time. This confirms the observation in Section 6.4.4, clearly indicating that the synthetic time series has a higher random component, and thus a different autocorrelation, compared to the original one.

6.5 Discussion on the proposed models

In this chapter, we use SSH dictionary attacks and normal SSH traffic as our running example. In particular, Section 6.3 presented the models for such classes of traffic. A question that might arise while studying the proposed models concerns the generality of the approach. In this section, we present some remarks regarding this topic.

In Section 6.1, we argued that one of the cornerstones on which our approach is based is the separate modeling of malicious and benign traffic. The model for the malicious traffic would therefore describe a specific attack. This requires the researcher to have a clear knowledge of how the attack works and what it looks like at flow level. Moreover, the training of the model is subject to the availability of flow traces containing the attack. However, once a specific

attack has been modeled based on representative samples, i.e., once the hidden chain has been defined, the HMM framework is flexible enough to fully describe several variations in behavior. In the case of the SSH dictionary attack, for example, the time spent in each phase might change over time: in this case, the transition probabilities would need to be recalculated to capture the new behavior. Another change in behavior would be relative to the number of measured flows, packets and bytes per seconds. This is for example what we observed when comparing *Set 1* and *Set 2* for our SSH dictionary attack. In this case, the output distributions would need to be adjusted.

Normal traffic is defined in contrast to the attack traffic. We might therefore say that we consider the normal traffic as background. Consequently, we need a less detailed model for describing the normal traffic. In our example, we chose a two-state model. Models that are more complex are possible, depending on the traffic characteristics we are interested in. For example, an extended model could mimic the day-night pattern observed in the normal SSH traffic. Such a model could be based, for example, on two or more (A, I) pairs, i.e., a situation in which for each active state, there is an inactive state associated to it. Each pair can be trained on part of the day-night pattern. However, since the transitions between pairs would of course be governed by probabilities, it can happen that the model cannot approximate the day-night pattern with sufficient precision. To force the model to mimic the day-night pattern in a more accurate way, we should introduce memory of when we last transit into the current (A, I) pair. Considering types of traffic other than SSH, normal traffic for other protocols might be described by using the same model we propose for SSH traffic. Of course, the transition probabilities and the output probabilities would then be set according to the considered protocol.

Besides on the specific protocol, the models depend also on the type of data we use for the training, i.e., on flow-based time series. Flow-based time series appear to consist of sequences of active and inactive time bins, due to the information used to create the time series (see. Section 4.2). The models we propose are consequently based on (A, I) pairs. Since an (A, I) pair describe a basic characteristic of the time series we are working with, we believe that the (A, I) pairs can be considered as the basic "building block" for flow-based HMMs, at least at lower time resolution. When increasing the granularity of the time series, from one second to several seconds, for example, it would be less likely to observe inactivity bins, until the state I would not result necessary anymore.

6.6 Summary

In this chapter, we presented an approach to generate synthetic time series, enriched with ground-truth information. Our approach is based on Hidden Markov Models. We showed that our models provide a compact representation of the traffic, where only a few states are needed to fully describe the traffic evolution.

In Section 6.3 we provided models for both malicious and normal SSH traffic. Our models are inferred using only flow information; the training was based on real traffic traces captured at the University of Twente network. In Section 6.4 we then proved that our models can be used to generate meaningful flow-based time series that emulate the behavior of SSH attackers and users. The attacker model approximates the mean, standard deviation and correlation of the original trace with a 10% relative error. The normal traffic model shows a relative error up to only 1.5%. The drawback of using HMMs is that the proposed models fail in approximating the autocorrelation. This phenomenon is due to the higher degree of randomness that the models introduce in the synthetic time series and it is implicitly related to the Markov property. In Section 6.4.5, we showed an example of a synthetic time series including a dictionary attack. The synthetic time series clearly resembles the original one, as presented in Section 6.2. Finally, in Section 6.5 we briefly discuss the generality of our approach.

Modeling flow-based time series using an HMM-based approach gives promising results. In the future, we aim to further enhance our models by improving the autocorrelation. Possibilities in this direction are the use Semi-Markov processes [126] or the refinement of the current models by adding more states. Moreover, related work that explicitly accounts for autocorrelation can be found in the field of traffic modeling for performance evaluation, as for example [62, 16, 127]. However, these approaches are usually focusing on the autocorrelation of a single metrics, such as for example the number of packets arriving in a certain interval. Differently, we are working with several traffic metrics, namely the number of flows, packets and bytes. The challenge that we face, in this case, would consist in extending these and similar approaches to model both the autocorrelation of a certain measure and, at the same time, the correlation with other measure.

The research presented in this chapter proves that not only we can capture the main statistical characteristics of the original time series, but also that we can qualitatively reproduce time series that resemble real-world traffic. Moreover, as far as ground truth is concerned, we are now able to create labeled data

sets in an automatic manner, solving the problem of collecting and labeling real data for the specific case of intrusion detection based on flow time series.

CHAPTER 7

Tuning intrusion detection systems

Chapters 5 and 6 dealt with the creation of labeled data sets for intrusion detection, in terms of manually and automatically generated ground truth, respectively. This chapter moves now the focus of research to the second issue that we have identified in Section 1.2, i.e., the optimization of an IDS based on high-level policies.

To address this problem, the goal of this chapter is to present an optimization procedure that aims to tune the parameters of an IDS to perform as specified in high-level policies. We provide details regarding our approach in Section 7.1. As introduction to our procedure, we first present in general terms the IDS we want to optimize in Section 7.2, the type of attack we aim to detect in Section 7.3, and the performance measures used to validate our findings in Section 7.4. Our optimization procedure, that we present in Section 7.5, is supported by an extensive validation step. We present our results in Section 7.6. Moreover, our research opens the way to further developments: we take a first step towards adaptability, which can be seen as a form of optimization over the long term, in Section 7.7. We suggest extensions of the basic assumption for IDSs validation in Section 7.8. Finally, Section 7.9 summaries our findings.

7.1 General approach

An IDS aims to discriminate between malicious and benign activities. Since attackers are often trying to conceal their activities among normal traffic, performing such a task is not easy. It is therefore commonly accepted that, while monitoring, the IDS might make some errors, such as missing intruders (false negatives) or erroneously reporting benign activities (false positives). However, while tuning a system, we implicitly aim to achieve the best compromise, or *optimal solution*, that is keeping the false positives as low as possible while

trying to detect as many real attacks as possible. Note that, in many cases, decreasing the rate of false positives means that the false negative rate increases, as we have schematically indicated in Figure 1.3. Therefore, there exists a natural trade-off between detecting all anomalies (at the expense of raising alarms too often), and missing anomalies (but not issuing any false alarms).

Moreover, we argue that there might exist several optimal strategies, depending on the situation. For example, if the IDS reports to another system, such as, for example, an alert-management module, one might want to be more permissive with respect to the number of raised false positives, since they will be filtered by the alert-management module at a later stage. However, if the IDS reports directly to a human system administrator, one might not want the system to raise alerts too often. In addition, the type of attack being monitored, and its impact on the monitored infrastructure, can influence the choice of how dangerous it is to miss an attack. Therefore, while seeking the optimal solution, high-level policies should be taken into account.

The goal of this chapter is to address the optimality question, usually left to the care of the system developer or the IT personnel, in an autonomic manner. We propose a systematic framework that, by keeping into account high-level policies, enables tuning the parameter of an IDS to ensure optimal detection (w.r.t. the trade-off described above). To reach our goal, our approach is based on the following assumptions:

- The data aggregation in network flows implies that less data need to be monitored and analyzed for intrusions. However, the coarser data granularity and the lack of payload information make flow-based intrusion detection a challenge. To compensate for the lack of information carried by a single flow, we introduced, in Chapter 4, *time series* of flows, packets and bytes per second. These time series allowed us to reveal trends in the traffic that could be exploited for intrusion detection. In this chapter, we propose an example of a time series-based, anomaly-based IDS. The system relies on a model of normality capturing the safe behavior of the monitored network. In the specific context of this thesis, the normality model is a Hidden Markov Model (HMM).

- We propose a systematic framework for optimizing the parameters of our flow-based, time series-based IDS. We approach the parameter tuning by solving a non-linear optimization problem that addresses the trade-off between true positives and false positives in a probabilistic manner. Moreover, our solution explicitly keeps into account the fact that the op-

timal solution might depend on high-level policies, following by the specific situation in which the IDS should operate.

- The system that we propose relies on a model of normality. However, as often is the case for network traffic, normality can be subject to changes, due to, for example, seasonal trends. A desirable characteristic in an anomaly-based system is therefore to be resilient to changes in normality, meaning that the system is either able to keep functioning even if the normality model needs to be re-adjusted, or it is able to re-tune its parameters to include the new situation. We refer to the latter as *adaptability*. In our opinion, adaptability constitutes a field of research in itself. In this chapter, we take a first step towards adaptability, showing how such characteristic could be included in the proposed method.

7.2 Detection system principles

The general assumption underlying network anomaly detection is that malicious activities (statistically) deviate from the network's normal behavior [33]. In this section, we present in general terms an anomaly-based system that builds upon the techniques developed in Chapter 6. We then point out the challenges faced when aiming to automatically tune and optimize the system parameters.

Figure 7.1 presents a high-level overview of the proposed detection system. We assume the system to be based on a probabilistic model of normal behavior, indicated by λ. The system observes a stream of network measurements $\{o_1, o_2, \ldots\}$ and analyzes a window (of length $w \in \mathbb{N}$) of past measurements, to which we refer as *observation sequence*. Observation sequences have been introduced because, since flows do not carry information on the content of a communication, in many cases a single flow or a single time bin can hardly be indicative of the presence of an attack. At time $t \in \mathbb{N}$, our detection procedure evaluates the likelihood of observing the sequence $O_t^w = \{o_{t-w+1}, \ldots, o_t\}$, with respect to our base model λ; we denote this likelihood by $\Phi(O_t^w \mid \lambda)$. Informally, the value $\Phi(O_t^w \mid \lambda)$ is an indication of how likely it is to observe O_t^w according to what we consider normality. Evidently, unlikely sequences will yield relatively low likelihoods, legitimate sequences relatively high likelihoods. The challenge is to discriminate between these two scenarios.

For the implementation of our system, we defined the base model λ as an HMM. As argued in Chapter 6, HMMs are particularly suitable for time-series

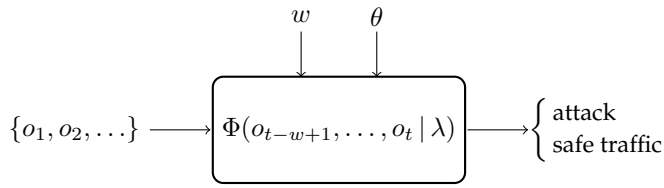


Figure 7.1: Detection system.

anomaly detection because they allow a compact representation of the studied system, in this case network traffic, while encoding its evolution in time. Moreover, HMMs offer efficient ways of calculating the probability that a specific sequence of observations occurs.

The selection of the parameters θ and w does play a crucial role in the effectiveness of the detection system. If θ is too low, it is likely that the system would be prone to raise false alerts (false positives, in the sequel denoted by FP), by raising alerts also in case of legitimate observation sequences; if θ is too high, on the other hand, the rate of alerts being generated would decrease, but the system would become less reactive to malicious activities (false negatives; FN).

The impact of w can be understood as follows. If w is small, a single observation may have significant impact on the statistic $\Phi(O_t^w \mid \lambda)$, possibly causing "too quick" false alerts (FP). On the other hand, bearing in mind laws of large numbers, large values of w may smooth out the effect of occasional unlikely events, with the risk of missing anomalies (FN).

The values of θ and w define therefore interesting trade-offs that determine the performance of the detection system. The goal of this chapter is to present a systematic framework yielding optimal values for the parameters θ and w. It should be realized, though, that, as argued in Section 7.1, this selection very much depends on the specific goals pursued in the situation at hand: is the primary objective to allow false alerts but to be sure to flag all the anomalies, or vice versa, or perhaps some intermediate scenario? The selection procedure that we propose here explicitly takes into account this preference.

7.3 SSH case study

In the following sections, we will illustrate the performance of the proposed detection system by means of the case study that accompanied us along this

thesis: the SSH dictionary attack and the SSH normal traffic.

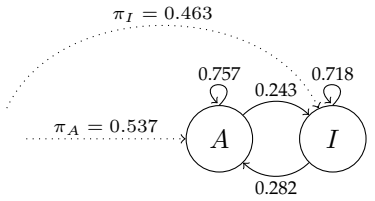


Figure 7.2: Normal SSH traffic model.

As indicated before, we should first define a base model λ describing the normal behavior, i.e., the traffic that was not generated by an SSH dictionary attack. We do so by relying on the HMM of Figure 7.2, which was developed and validated in Chapter 6. The Discrete Time Markov Chain (DTMC) consists of two states: a state A, in which SSH traffic is present on the network (*activity*), and a state I, in which no SSH traffic is present on the network (*inactivity*). The model parameters are estimated by training the model on traces of legitimate SSH traffic collected on the UT network (cf. Section 6.4.1). We let the starting distribution $\pi = [\pi_A, \pi_I]$ be equal the DTMC's steady state distribution. Each state is augmented with an output probability distribution that in this case is the empirical distribution function of flows per second. Empirical distributions of packets per second and bytes per seconds are also possible. However, as described in Chapter 4, flows per second is the discriminant measure between malicious and benign activities for SSH dictionary attack, therefore we concentrate on that distribution.

We indicate with $O_t^w = \{o_{t-w+1}, \ldots, o_t\}$ the observation sequence in the time-frame $t-w+1, \ldots, t$ in our training set. Note that, according to such definition, each observation sequence in the training set is unique, independently from the values o_i that constitute it. In the HMM framework, we refer to the probability of an observation sequence O_t^w as:

$$\Phi(O_t^w|\lambda) = \sum_{\forall Q} \pi_{q_{t-w+1}} b_{q_{t-w+1}}(o_{t-w+1}) \prod_{i=t-w+2}^{t} a_{q_{i-1}q_i} b_{q_i}(o_i), \qquad (7.1)$$

where we assume that the observation sequence O_t^w could have been generated by several state sequences $Q = \{q_{t-w+1}, \ldots, q_t\}$ and we indicate, with $a_{q_j q_i}$ the transition probability from state q_j to state q_i, and with $b_{q_i}(o_i)$ the output

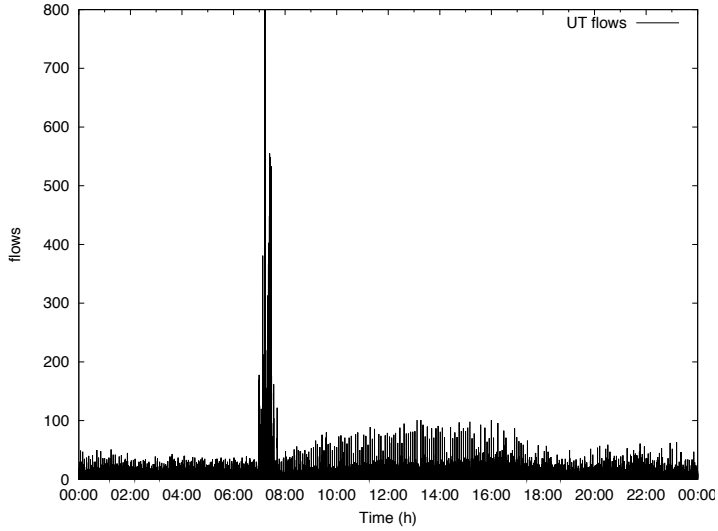


Figure 7.3: Time series for a day of SSH traffic (flows/second).

probability of symbol o_i in state q_i. $\Phi(O_t^w|\lambda)$ tends to be small when w grows, and therefore it is more convenient to work with

$$\ell_t^w := -\log \Phi(O_t^w|\lambda), \tag{7.2}$$

where the minus sign allows us, in our example, to work in $\mathbb{R}^+$.

> **Note:** There are similarities between using an HMM for generation purposes, as done in Chapter 6, and for calculating the probability of an observation sequence. In both cases, indeed, we are progressively building a path on the Markov chain. However, in the case of the generative process (see Section 6.4.3), each transition and output is randomly chosen at each instant of time t. When we are calculating the probability of an observation sequence, on the contrary, we should take into account that the same observation sequence could have been generated by several different state sequences. We are therefore considering the joint probability over all the possible state paths in the chain that could have generated the observation sequence.

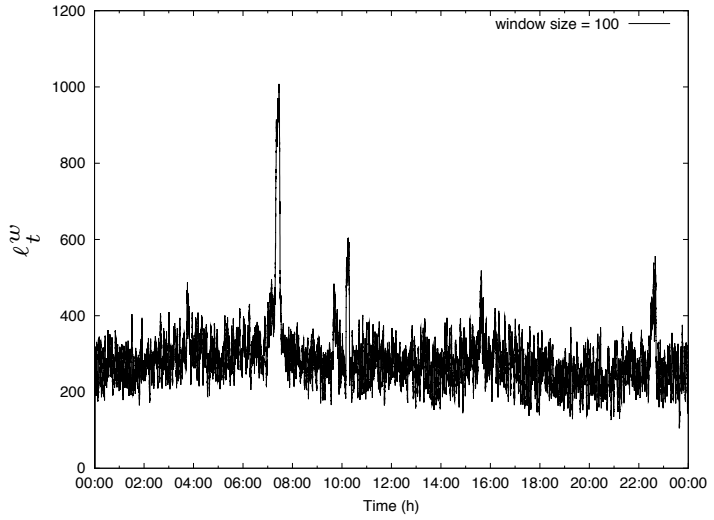


Figure 7.4: Log-likelihood of the time series of flows per second in Figure 7.3.

Since the HMM in Figure 7.2 is trained on benign SSH traffic, the model is supposed to assign relatively high (low) ℓ_t^w values to sequences of malicious (legitimate) traffic. Figure 7.3 shows the time series of flows per second for the SSH traffic measured at the University of Twente on July 16, 2008. Between 7:00 and 7:40, the university network has been the target of an SSH dictionary attack. For the detail of that type of attack, we refer to Section 6.2. In Figure 7.4 we present an example of how ℓ_t^w raises in case of network anomalies, based on the data of Figure 7.3, taking $w = 100$ sec. The window of observations slides over the time series with a step size of 1 second. Figure 7.4 shows that ℓ_t^w attains values roughly between 200 and 400 during the normal phases, whereas during the attack the value of ℓ_t^w abruptly raises to 1000. This graph suggests that ℓ_t^w is a suitable measure for discriminating between normal and anomalous traffic. However, we observe that also other observation sequences provoke a raise of ℓ_t^w, for example around 10 AM, 4 PM, and 10 PM. In these cases, applying a threshold θ of 500, a network anomaly would have been reported even though there was no indication of malicious activity; $\theta = 800$ would be a more appropriate choice here. This underscores that procedure to soundly select θ is of crucial importance.

7.4 Attacks, observations and detection

In this section, we will define the performance measures we will later use to evaluate and optimize our prototype detection system. In Section 7.4.1, we describe our system in terms of a binary classifier. This characterization allows us to define the performance metrics we are interested in, in Section 7.4.2.

7.4.1 Binary classifier

A detection system is traditionally regarded as a *binary classifier* [47]. A *binary classifier* analyzes a set I of instances of a problem and map these to two *prediction classes*. The prediction classes in the case of intrusion detection are usually indicated as:

- P, or *positive*: $P \subseteq I$, such that $i \in P$ if i has been labeled as an attack;
- N, or *negative*: $N \subseteq I$, such that $i \in N$ if i has been labeled as legitimate.

We assume $P \cap N = \emptyset$ and $P \cup N = I$. The performance of a classifier is evaluated based on how precise the prediction classes are compared to the actual classes of the instances (the ground truth). We indicate the actual classes as:

- M, or *malicious*: $M \subseteq I$, such that $i \in M$ if i is an attack;
- B, or *benign*: $B \subseteq I$, such that $i \in B$ if i is legitimate.

Also in this case we assume $M \cap B = \emptyset$ and $M \cup B = I$.

Actual and prediction classes

The definition of the sets B, M, P and N depends on the specific characteristics of the IDS. In certain cases, the definition is straightforward. For instance, for a stateless payload based IDS, M would be the set of packets generated by an attacker, B the set of legitimate packets and packets would be assigned to P and N according to the output of the detection engine. In other words, for this type of payload-based IDS, a packet is an instance of the classification problem.

In the case of flow-based intrusion detection, such a characterization is not straightforward. As described in Section 7.2, we propose to approach flow-based intrusion detection by considering a window of past observations O_t^w. This approach implies that O_t^w is the traffic instance that we aim to classify. We therefore define B, M, P and N as sets of observations sequences.

In the text below, we will show that B, M, P and N depends on the observation sequence length w. We therefore stress this dependence by introducing the superscript B^w, M^w, P^w and N^w.

Given an observation sequence $O_t^w = \{o_{t-w+1}, \ldots, o_t\}_t$, each observation o_i can be either malicious, if there was an attacker active at time i, or benign, if no attacker was active at time i. Each observation sequence can therefore contain several malicious observations. Due to the sliding window mechanism, the malicious observation will be added on the right side of the observation sequence and leave the observation sequence on the left side. In other words, we define an attack to begin when the first malicious observation appears in the rightmost (youngest) position in the observation sequence; similarly, an attack ends when a benign observation appears in the rightmost position in the observation sequence.

We derive therefore the following definitions:

$$M^w = \{O_t^w \mid o_t \text{ is malicious}\}, \tag{7.3}$$

$$B^w = \{O_t^w \mid o_t \text{ is benign}\}. \tag{7.4}$$

With this definition of M^w and B^w, it is important to note that the discriminant condition between malicious and benign observation sequences is given by the youngest observation. The consequence of this is that an observation sequence will be considered malicious (benign) even if all but the last observation are benign (malicious). In other words, we can say that our definitions focus on the current situation and not on the past observations. Note, however, that other definitions of M^w and B^w are possible.

Finally, the prediction classes are such that an observation sequence is malicious if it deviates from the base model of traffic λ. We therefore define:

$$P^w = \{O_t^w \mid \ell_t^w > \theta\}, \tag{7.5}$$

$$N^w = \{O_t^w \mid \ell_t^w \leq \theta\}. \tag{7.6}$$

7.4.2 Performance measures

This subsection introduces the performance measures of interest for our study. We propose three groups of performance metrics. The first two, the *confusion matrix* and the *detection rate* are traditionally used in evaluating IDSs [13]. The third group, the *detection* and *normalization lags*, has been introduced in this thesis to quantify the delays the detection system incurs in detecting an attack and in recognizing the re-establishment of a normal situation.

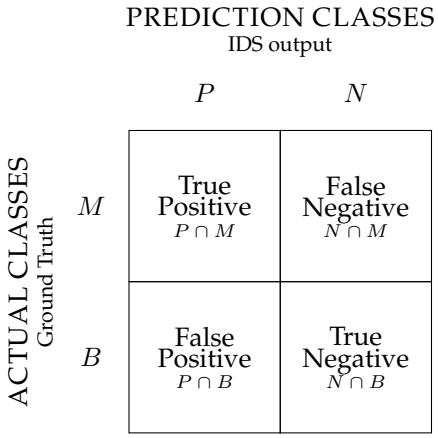


Figure 7.5: Confusion matrix [47].

Confusion matrix

The *confusion matrix* [47] describes both the corrected classified instances and the *errors* between the classes. Based on the confusion matrix, the following performance metrics can be defined:

- the *true positive* $\text{TP} = \#\{P \cap M\}$; and the *false negative* $\text{FN} = \#\{N \cap M\}$;
- the *true negative* $\text{TN} = \#\{N \cap B\}$; and the *false positive* $\text{FP} = \#\{P \cap B\}$.

Figure 7.5 gives a graphical representation of the confusion matrix and the derived performance metrics [47]. Such measures are often expressed in terms of *rates*, that is $\text{TP-rate} = \text{TP}/\#\{M\}$, $\text{FP-rate} = \text{FP}/\#\{B\}$, and so on. In the sequel, we will use the notation TN, TP, FN, FP as rates, such that $\text{TN}+\text{FP} = \text{TP}+\text{FN} = 1$.

Detection rate

Traditionally, intrusion detection measures the system performance in terms of TN, TP, FN, FP, based on the sets M, B, P and N. In our case, the element of such sets are observation sequences O_t^w. Most probably, an attack will consist of a set of observation sequences. We indicate the starting and ending time of an attack $\mathbf{A}$ as t_s and t_e, respectively. Following Bolzoni [13], we define $\mathbf{A}$ to be

detected if there exists at least one observation sequence O_t^w, $t_s \leq t \leq t_e$ such that $O_t^w \in P \cap M$, i.e., at least one observation sequence part of the attack is flagged as positive.

Besides the rate of correctly classified observation sequences, we can therefore also measure, for a set of attacks in our data sets, how many attacks have been detected. We define this performance measure as the *detection rate* per attack:

$$DR = \frac{\#\{\text{detected attacks}\}}{\#\{\text{attacks}\}}.$$

Detection and normalization lags

A detection system aims to identify the presence of an attack. A likely situation is that classification errors would be observed at the beginning and at the end of the attack. While the traffic is entering the observation window, some observation sequences will be classified as negative, since only few observation are malicious. Conversely, while the window is leaving the attack, some observation sequences will be flagged as positive, since only their youngest observations are benign. However, it is highly preferable that an attack is detected fast. Moreover, it is desirable for a detection system to also recognize as fast as possible that an anomalous situation is over.

Since in our setup we rely on a sliding window mechanism with steps of 1 second, we can measure how promptly the system reacts by analyzing the classification errors at the beginning and at the end of an attack. It is therefore desirable to have a low rate of false negatives in the beginning of an attack and a low rate of false positives when an attack is over.

Given an attack $\mathbf{A}$, and its starting and ending time, t_s and t_e, we then define the following measures:

- the *detection lag*, i.e., the delay before an attack is detected:

$$D_{\mathbf{A}}^w := \min\{t \geq t_s |\, O_t^w \in M \cap P\} - t_s;$$

- the *normalization lag*, i.e., the delay in recognizing that an attack is over. Since we aim to investigate the impact of the history on the analysis of normal traffic, we measure the normalization lag within w observations from the end of the attack:

$$E_{\mathbf{A}}^w := \min\{t_e \leq t < t_e + w |\, O_t^w \in B \cap N\} - t_e.$$

Finally, we can calculate the average, over the attacks present in our data sets, of the detection and normalization lags.

7.5 The optimization procedure

In Section 7.1, we argued that tuning an IDS entails searching for an optimal solution w.r.t. the trade-off between detecting as many attack as possible and having a low rate of false alarms. More generally, let us consider an IDS with parameters $p_1, \ldots, p_n$ and a high-level policy $\mathcal{P}$ describing the required behavior of the IDS in terms of some performance metrics $\mathcal{M}_1, \ldots, \mathcal{M}_m$, such as for example TP and TN. To automatically tune the system parameters to satisfy $\mathcal{P}$, the following steps should be taken. First, it is necessary to define a mathematical relation *Goal* of the performance metric that we aim to optimize, that keeps into account the policy $\mathcal{P}$:

$$\underset{p_1,\ldots,p_n}{\text{maximize}} \quad \text{Goal}(\mathcal{M}_1, \ldots, \mathcal{M}_m, \mathcal{P}).$$

This relation describes the goal, such as for example maximizing the correct classification. However, in the present formulation, the optimization problem requires that we are able to measure the performance metrics $\mathcal{M}_1, \ldots, \mathcal{M}_m$. This operation can be computationally intensive. The second step we propose is, when the situation allows it, to analytically express the relation between the performance measures and the system parameters. Once we have expressed a mathematical relation between performance measures and the system parameters, the last step is to analytically solve the optimization problem. In case the second step cannot be undertaken, it should be evaluated if it is still possible to solve the optimization problem by using empirical data. However, as said, this operation may be costly. Table 7.2 summarizes the steps described above and shows how we will address these in the following.

7.5.1 The optimization problem

This section focuses on formalizing the parameter tuning as an optimization problem (step 1 of the procedure in Table 7.2). Our aim is to determine suitable values for the parameters w and θ. The performance of an IDS is usually presented by means of a so-called Receiver Operating Characteristics (ROC) [47]. The ROC curve plots the combinations of FP and TP, by varying θ. ROC curves are a technique for visualizing the performances of a classifier, since they show

General procedure	Setup used in this thesis
IDS parameters $p_1, \ldots, p_n$ Performance metrics $\mathcal{M}_1, \ldots, \mathcal{M}_m$ Policy $\mathcal{P}$	IDS parameters θ, w Performance metrics TN, TP Policy (α, β)
step 1: define the optimization problem $\underset{p_1,\ldots,p_n}{\text{maximize}} \quad \text{Goal}(\mathcal{M}_1, \ldots, \mathcal{M}_m, \mathcal{P})$.	**step 1:** $\underset{\theta,w}{\text{maximize}} \quad \alpha \cdot \text{TN} + \beta \cdot \text{TP}$
step 2: express an analytical relation between $p_1, \ldots, p_n$ and $\mathcal{M}_1, \ldots, \mathcal{M}_m$.	**step 2:** $\text{TN} = \mathbb{P}(X^w \leq \theta), \text{TP} = 1 - \mathbb{P}(Y^w \leq \theta)$ $X^w, Y^w \overset{\text{d}}{=} \mathbb{G}\text{m}(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma})$
step 3: analytically solve the optimization problem.	**step 3:** solve Problem (7.7)

Table 7.2: The optimization procedure.

the "relative trade-offs between benefits (TP) and costs (FP)" [47]. Intuitively, the steeper the ROC curve grows, the better is the performance of the classifier.

Figure 7.6 shows examples of ROC curves obtained by testing the HMM-based detection system on a synthetic data set of SSH traffic for different window sizes. As mentioned above, each point on the ROC curves is determined by a different value of the threshold θ. The curves show a steep growth of TP for low values of FP, while the curves grow slower for higher values of FP. The ROC curves indicate that the proposed detection system is able to discriminate between malicious and benign observation sequences with high TP and low FP. However, Figure 7.6 also shows that:

- The ROC curves growth depends on the observation sequence length w; we plotted curves for three values w. They indicate that the performance of the procedure can be improved by choosing w optimally. For instance, as shown in the subplot of Figure 7.6, to obtain $\text{TP} = 0.98$, $w = 100$ would provide the lowest FP. Moreover, the optimal value of w will change with the desired TP and FP.

- Since θ controls in which point of a ROC curve we operate, the choice of

the threshold value θ depends on the desired performance. For example, for $w = 100$ we can obtain $\text{TP} = 0.8$ with $\text{FP} = 0.02$, but also $\text{TP} = 0.93$ with $\text{FP} = 0.1$.

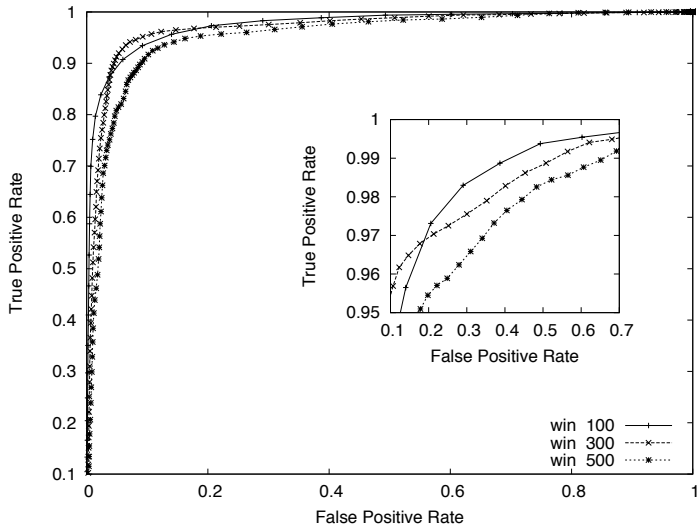


Figure 7.6: Receiver Operating Characteristics, for varying window size.

The above reasoning indicates that one can optimize the parameters w and θ, and the optimum critically depends on the relative importance the user assigns to TP and FP. Making use of the relation $\text{TN} = 1 - \text{FP}$, we formulate the optimization problem as follows:

$$\underset{\theta,w}{\text{maximize}} \quad \alpha \cdot \text{TN} + \beta \cdot \text{TP}. \tag{7.7}$$

Here the parameters α and β embody the relative importance the operator assigns to TN and TP. If he chooses α substantially larger than β, the optimization would lead to (w, θ) that makes the system behave conservatively, flagging anomalies only if there are strong evidences of an attack, at the expense of relatively many false negatives. On the other hand, if he picks α considerably smaller than β the system would become more aware of malicious traffic, favoring in this way TP, but by increasing FP.

7.5.2 Probabilistic interpretation of the classification problem

We concentrate now on formalizing the relation between the system parameters and the performance metrics (step 2 of the procedure in Table 7.2). For each observation sequence O_t^w, the value ℓ_t^w is an indication of the sequence being malicious or benign. We now define the following random variables: for an arbitrary observation sequence of length w,

$$X^w := -\log \Phi(O_t^w|\lambda) \quad \text{for } O_t^w \in B^w$$

for a benign observation sequence, and

$$Y^w := -\log \Phi(O_t^w|\lambda) \quad \text{for } O_t^w \in M^w$$

for a malicious observation sequence. Note that in defining the random variables X^w and Y^w we make use of the logarithm notation introduced in Equation (7.2). For any given w, we can estimate the density functions $f_X^w(\cdot)$ and $f_Y^w(\cdot)$, in the form of normalized frequency histograms. An example of these are shown in Figure 7.7 for $w = 100$, based on the analysis of normal and malicious traffic. In the figure we also show an example value for θ, which splits the sample spaces in regions. The subspace to the left of the threshold corresponds to negatively-flagged inputs; in the same way, the subspace to the right of the threshold corresponds to positively-flagged inputs. The intersection between such subspaces and the probability distributions results in regions that we can map directly on the confusion matrix of our binary classifier. In particular, according to this formulation, we have

$$\text{TN} = \mathbb{P}(X^w \leq \theta), \quad \text{FP} = 1 - \mathbb{P}(X^w \leq \theta), \tag{7.8}$$

$$\text{FN} = \mathbb{P}(Y^w \leq \theta), \quad \text{TP} = 1 - \mathbb{P}(Y^w \leq \theta). \tag{7.9}$$

We can now rewrite Problem (7.7) as follows:

$$\underset{\theta,w}{\text{maximize}} \quad \alpha \cdot \mathbb{P}(X^w \leq \theta) + \beta \cdot (1 - \mathbb{P}(Y^w \leq \theta)). \tag{7.10}$$

In Figures 7.8(a) and 7.8(b) we report separately, for readability, the density functions and the confusion matrix measures for X^w and Y^w, respectively. Figure 7.7 also shows that the distributions of X^w and Y^w overlap, and θ cannot be chosen such that it perfectly discriminates between malicious and benign. The presence of an overlapping region, and its size, can change from data set to data set, and this imposes a sort of theoretical limit to the classification performance of the detection procedure.

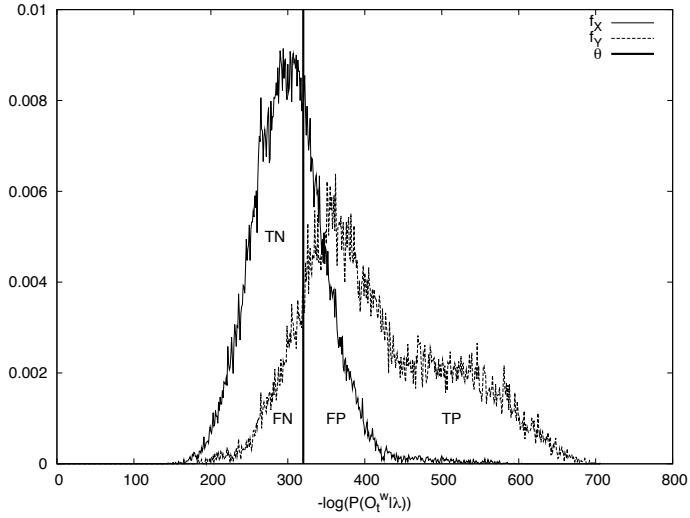
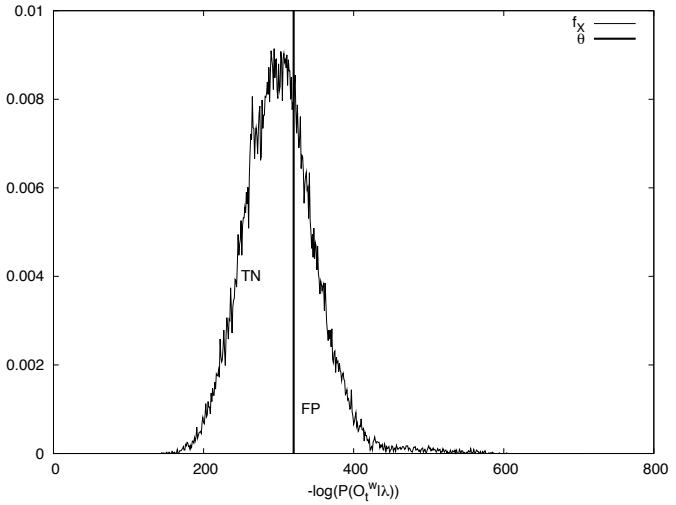
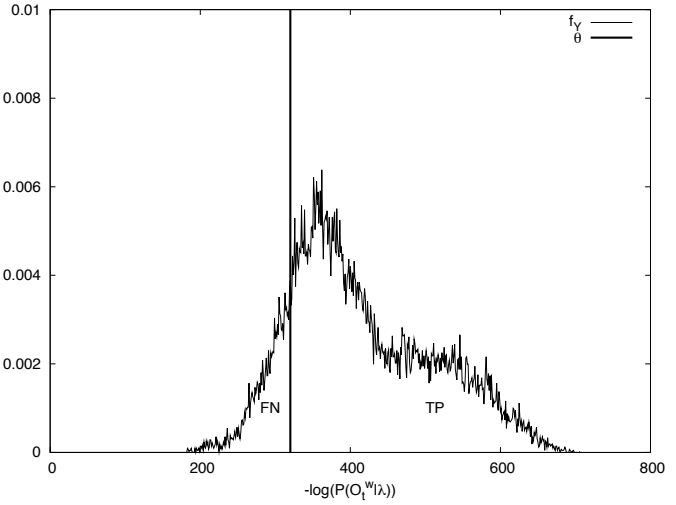


Figure 7.7: Performance measures and empirical distribution functions.

In (7.10), TN and TP are expressed in terms of the cumulative distribution functions of X^w and Y^w. These cumulative distribution functions can be determined empirically, in the same way as we did for the densities in Figure 7.7. However, since our goal is to perform the optimization in (7.10), which is over θ and w, we need the distribution functions $\mathbb{P}(X^w \leq \cdot)$ and $\mathbb{P}(X^w \leq \cdot)$ for a broad range of values of w. Evidently, estimating these for many and especially for large values of w will make the optimization procedure slow: the complexity of computing ℓ_t^w is linear in w, and such measure is computed for each observation sequence, i.e., each time the window slides over the time series. Therefore, especially for a large amount of training data, such a procedure is costly. To reduce the complexity, we propose to (i) approximate the empirical distribution densities by means of fitted distributions for some (relatively small) set of w, and to (ii) find a relation between the fitted parameters and w (for example by approximating them with simple polynomial functions). This approach yields approximations of the distribution function for any value of w. Our analysis of the empirical density function for X^w and Y^w for variable values of w shows that a *Gaussian mixture distribution* (denoted by $\mathbb{G}\mathrm{m}$) offers a suitable fit for the considered distributions. Following the convention to write



(a)



(b)

Figure 7.8: Performance measures and empirical distribution functions (details).

vectors in bold, we say that a random variable Z is $\mathbb{G}\mathrm{m}(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ of order $n \in \mathbb{N}$, with $p_i \geq 0$ adding up to 1, if its probability density function is in the form

$$f_Z(x) = \sum_{i=1}^{n} p_i f_{N_i}(x),$$

where the components $N_i = \mathcal{N}(\mu_i, \sigma_i)$, for $i = 1, \ldots, n$, are independent normally distributed random variables.

Finally, to finalize our procedure, step 3 can be completed by optimizing (7.10). Since the empirical density function for X^w and Y^w are fitted with Gaussian mixture distributions, (7.10) can also be rewritten as the sum of the cumulative distribution functions of the Gaussian component, i.e., as a sum of cumulative distribution function of normal distributions. We indicate the cumulative distribution function of $\mathcal{N}_i(\mu_i, \sigma_i)$ with

$$F(\theta, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\theta} e^{-(x-\mu)^2/2\sigma^2} dx. \tag{7.11}$$

Then, (7.10) can be rewritten as

$$\begin{aligned} \underset{\theta, w}{\text{maximize}} \quad & \alpha \cdot \sum_{i=1}^{n} p_{iX^w} F(\theta, \mu_{iX^w}, \sigma_{iX^w}) \\ & + \beta \cdot (1 - \sum_{i=1}^{m} p_{iY^w} F(\theta, \mu_{iY^w}, \sigma_{iY^w})), \end{aligned} \tag{7.12}$$

where n and m are the number of Gaussian components for X^w and Y^w, respectively, p_{iX^w}, μ_{iX^w} and σ_{iX^w} the parameters of the Gaussian Mixture for X^w, and p_{iY^w}, μ_{iY^w} and σ_{iY^w} the parameters of the Gaussian Mixture for Y^w. However, the integral in 7.11 has no analytical solution. Therefore, (7.12) has to be numerically solved.

7.6 Validation

The aim of this section is twofold. After presenting the data sets used for testing (Section 7.6.1), we validate our optimization procedure (Section 7.6.2). Then, we study how varying α and β affects the performance of the system (Section 7.6.3).

7.6.1 Data sets

We test our procedure using four data sets of SSH time series of flows per seconds, each containing both legitimate and malicious traffic. Two data sets, *Synthetic 1* and *Synthetic 2*, are synthetically generated as described in Chapter 6. The other two data sets, *Original 1* and *Original 2*, consist instead of time series created directly from real traffic collected at the UT network. The synthetic data sets *Synthetic 1* and *Synthetic 2* consist of 20 time series of 5400 seconds (time slots) each, each one of them containing one attack. The data sets *Original 1* and *Original 2* are parts of data sets *Set 1* and *Set 2*, presented in Section 6.4.1. More specifically, they consist of the second half of *Set 1* and *Set 2*. Data set *Original 1* contains 7 attacks, while data set *Original 2* contains 6 attacks. Attacks have been identified based on the reports of a quarantine system deployed at UT, and have been manually labeled.

The HMMs λ have been trained using the first half of *Set 1* and *Set 2*, respectively. Note that, by splitting *Set 1* and *Set 2*, we made sure that we train the models and test our procedure on different data.

Gaussian fits

In Section 7.5, we propose Gaussian Mixture distributions as suitable fits for the empirical density functions X^w and Y^w. However, the number of components of the Gaussian mixtures can be data set specific. In our setup, we observed that

- $n = 2$ yields an excellent fit for the data sets *Synthetic 1* and *Synthetic 2*. We therefore define:

  $$X^w \stackrel{\mathrm{d}}{=} \mathbb{G}\mathrm{m}(\boldsymbol{p}_B^w, \boldsymbol{\mu}_B^w, \boldsymbol{\sigma}_B^w)$$

  and

  $$Y^w \stackrel{\mathrm{d}}{=} \mathbb{G}\mathrm{m}(\boldsymbol{p}_M^w, \boldsymbol{\mu}_M^w, \boldsymbol{\sigma}_M^w);$$

- for the data sets *Original 1* and *Original 2*, we choose $n = 1$ for X^w and $n = 2$ for Y^w:

  $$X^w \stackrel{\mathrm{d}}{=} \mathcal{N}(\mu_B^w, \sigma_B^w)$$

  and

  $$Y^w \stackrel{\mathrm{d}}{=} \mathbb{G}\mathrm{m}(\boldsymbol{p}_M^w, \boldsymbol{\mu}_M^w, \boldsymbol{\sigma}_M^w).$$

Figure 7.9 presents an example of the fits for the benign and malicious distributions in *Synthetic 1*, obtained for $w = 300$. Finally, the relation between the $\mathbb{G}\mathrm{m}$ parameters $\boldsymbol{p}^w$, $\boldsymbol{\mu}^w$ and $\boldsymbol{\sigma}^w$ and the window size w, for both the benign and malicious distributions, has been expressed by means of polynomial fits. Our findings show that $\boldsymbol{\mu}^w$ is a linear function of w, $\boldsymbol{\sigma}^w$ can be approximated by a polynomial of degree 3 and $\boldsymbol{p}^w$ by a polynomial of degree 2. For an example of such polynomial functions, we refer to Appendix B.

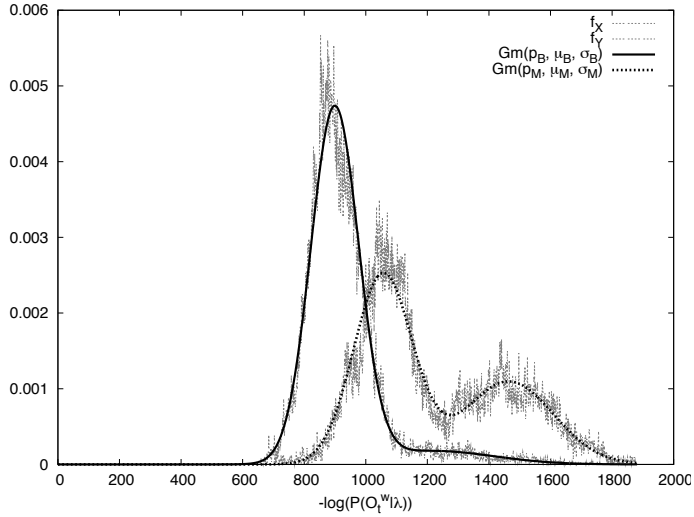


Figure 7.9: Example of distribution fits, $w = 300$.

7.6.2 Optimization procedure validation

In Section 7.5, we proposed an optimization procedure build upon Gaussian Mixture distributions. In this section, we now validate such a procedure, by measuring how precisely we can approximate the empirical optimal results. We therefore compare the results of the optimization problem (i) calculated on empirical data and (ii) analytically calculated by means of Gaussian fits. For this test, we set $\alpha = \beta = 1$. The empirical data are obtained by directly measuring TN and TP on the empirical distribution of ℓ_t^w, similar to the one shown in Figure 7.7.

Tables 7.3 presents the optimization errors for the data set *Synthetic 1*. The table show, for various values of the window length w, (i) the optimal empirical detection rates and threshold; (ii) the optimal analytical detection rates and threshold as obtained through our procedure by means of Gaussian fits; and (iii) the relative error between these, expressed in $\%$.

The results in Tables 7.3 show that the approach, based on fitted $\mathbb{G}\mathrm{m}$ distributions, very well approximates the results one would have found using the empirical distributions. For the data set *Synthetic 1*, the relative error for TN, TP and θ is indeed for all the considered cases lower than 5%.

We repeated this validation step also for the data sets *Synthetic 2*, *Original 1* and *Original 2*. For *Synthetic 2*, the error is lower than 4%, while in the case of *Original 1*, it is up to 10%. For the data set *Original 2*, on the other hand, we observe a higher error in TP and TN, in certain cases up to 31%. Such higher errors are due to the polynomial fits used to approximate the parameters of X^w and Y^w as functions of w. In this case, the polynomial fitting does not yield to the same excellent fits as for the other data sets. The error in approximating θ remains, however, in any case lower than 10%. For the results relative to these data sets, we refer to Appendix B.

Our findings confirm that the Gaussian mixtures are a suitable fit for the malicious and benign ℓ_t^w distributions and they can be therefore used to compute the optimal parameter θ. Also in the cases in which the estimation of TP and TN is not subject to a higher error, we are indeed able to closely approximate θ.

	Empirical			**Analytical**			**Error (%)**		
w	TN	TP	θ	TN	TP	θ	TN	TP	θ
100	0.94	0.90	321	0.98	0.95	320.94	3.79	4.72	0.02
200	0.95	0.93	626	0.97	0.96	624.14	1.60	2.75	0.30
300	0.94	0.94	920	0.95	0.94	918.13	1.42	0.28	0.20
400	0.91	0.93	1197	0.93	0.93	1202.64	1.76	0.79	0.47
500	0.90	0.92	1497	0.91	0.92	1479.77	0.85	0.75	1.15
600	0.88	0.91	1777	0.89	0.92	1752.37	0.79	2.01	1.39
700	0.87	0.89	2102	0.87	0.93	2023.13	0.74	4.44	3.75
800	0.85	0.89	2366	0.85	0.93	2294.53	0.04	4.21	3.02
900	0.82	0.90	2613	0.83	0.93	2568.68	1.04	3.02	1.70
1000	0.80	0.90	2869	0.82	0.93	2847.47	2.53	2.65	0.75

Table 7.3: Empirical vs. analytical results for the optimization problem (*Synthetic 1*).

7.6.3 Performance measures

The aim of this section is to study the impact of the parameters α and β on the performance of our simple detection system. We therefore report here the performance measure for varying ratio β/α. Note that it is not needed to know the absolute values of α and β, as only their ratio affects the outcome of the optimization procedure. As explained in Section 7.4.2, the performance are expressed in terms of: (i) the confusion matrix and (ii) the detection rate, as commonly done in intrusion detection; (iii) the detection and normalization lags in recognizing the presence of an attack or the re-establishment of a normal situation.

Confusion matrix

			Confusion matrix		**Detection and normalization lags**			
β/α	w_{opt}	θ_{opt}	TP	TN	$\overline{D}_{\mathbf{A}}$	$std(D_{\mathbf{A}})$	$\overline{E}_{\mathbf{A}}$	$std(E_{\mathbf{A}})$
0.1	110	381.59	0.82	0.99	61.85	38.60	32.42	24.26
0.2	120	404.24	0.87	0.98	53.75	21.76	43.21	25.54
0.3	130	430.12	0.89	0.97	53.50	19.57	52.95	28.89
0.4	130	426.11	0.90	0.97	52.25	19.36	56.37	28.62
0.5	130	423.07	0.91	0.97	51.55	19.65	59.47	28.92
0.6	140	451.3	0.92	0.96	51.45	21.26	68.37	29.93
0.7	140	449.13	0.92	0.96	50.80	21.06	69.11	30.07
0.8	140	447.26	0.92	0.96	50.45	21.01	71.16	30.01
0.9	140	445.63	0.92	0.96	46.95	19.92	72.37	28.52
1	140	444.16	0.92	0.95	46.75	19.83	74.11	28.14
1.5	140	438.54	0.93	0.94	42.45	21.73	76.95	27.09
2	150	464.26	0.94	0.93	43.55	22.67	82.22	28.94
4	150	454.01	0.96	0.90	36.35	22.38	89.82	29.23

Table 7.4: Performance measures for *Synthetic 1*.

Tables 7.4, 7.5, 7.6 and 7.7 present the confusion matrix and the detection and normalization lags, as achieved by the system for the data sets *Synthetic 1*, *Synthetic 2*, *Original 1* and *Original 2*, respectively. The rows in the tables are obtained as follows. First, for a given β/α, we compute the optimal w and θ, based on the learned HMM λ and the Gaussian fits for the considered data set. Then the data set is analyzed by calculating the test statistic ℓ_t^w for each observation sequence. An observation sequence is then classified as positive

			Confusion matrix		Detection and normalization lags			
β/α	w_{opt}	θ_{opt}	TP	TN	$\overline{D}_{\mathbf{A}}$	$std(D_{\mathbf{A}})$	$\overline{E}_{\mathbf{A}}$	$std(E_{\mathbf{A}})$
0.1	100	418.39	0.41	0.98	258.15	176.79	33.44	23.84
0.2	100	397.30	0.49	0.97	188.25	161.70	40.94	25.45
0.3	100	386.29	0.53	0.96	153.00	144.85	44.06	24.63
0.4	100	378.47	0.57	0.94	115.00	132.34	46.39	25.02
0.5	220	765.59	0.74	0.89	153.65	135.76	153.35	29.43
0.6	250	856.51	0.78	0.87	130.60	139.30	180.25	29.58
0.7	270	915.30	0.81	0.85	140.40	139.89	201.13	31.69
0.8	290	974.55	0.83	0.84	128.25	143.27	222.19	32.73
0.9	300	1001.75	0.85	0.83	129.20	147.28	233.75	33.22
1	320	1061.59	0.85	0.82	118.30	123.74	252.47	34.13
1.5	340	1106.62	0.88	0.77	98.90	119.67	275.93	31.01
2	350	1124.13	0.91	0.74	61.15	75.72	292.00	32.32
4	340	1055.90	0.95	0.62	38.80	72.76	297.85	32.94

Table 7.5: Performance measures for *Synthetic 2*.

			Confusion matrix		Detection and normalization lags			
β/α	w_{opt}	θ_{opt}	TP	TN	$\overline{D}_{\mathbf{A}}$	$std(D_{\mathbf{A}})$	$\overline{E}_{\mathbf{A}}$	$std(E_{\mathbf{A}})$
0.1	940	3555.26	0.06	1.00	762.00	0	56.00	-
0.2	1000	3651.20	0.06	0.99	755.00	0	149.00	-
0.3	1000	3577.90	0.06	0.99	733.00	0	182.00	-
0.4	330	1245.08	0.10	0.99	1057.00	1461.53	60.75	82.21
0.5	1000	2817.50	0.22	0.75	774.00	1379.88	560.00	409.26
0.6	1000	2808.64	0.22	0.74	773.20	1379.66	561.00	409.68
0.7	1000	2801.65	0.23	0.74	772.00	1379.05	561.75	409.84
0.8	1000	2795.88	0.23	0.74	771.40	1378.82	563.00	409.94
0.9	1000	2790.95	0.23	0.73	770.80	1378.49	565.75	410.67
1	1000	2786.64	0.23	0.73	770.60	1378.60	569.50	410.52
1.5	1000	2770.58	0.26	0.72	766.40	1380.26	573.25	411.75
2	1000	2759.21	0.28	0.72	746.20	1391.33	575.50	412.86
4	1000	2727.14	0.32	0.70	735.20	1394.97	590.75	417.67

Table 7.6: Performance measures for *Original 1*.

or negative. Finally, TP and TN are calculated. Note that the tables also include results for the detection and normalization lags. We will discuss such results later in this section. We first focus on the effect of varying α/β on the metrics

			Confusion matrix		Detection and normalization lags			
β/α	w_{opt}	θ_{opt}	TP	TN	$\overline{D}_{\mathbf{A}}$	$std(D_{\mathbf{A}})$	$\overline{E}_{\mathbf{A}}$	$std(E_{\mathbf{A}})$
0.1	1000	3923.12	0.22	0.99	640.80	523.94	0	-
0.2	1000	3854.63	0.23	0.98	623.20	531.67	0	-
0.3	1000	3803.11	0.23	0.97	607.00	530.02	0	-
0.4	1000	3752.98	0.24	0.95	597.60	526.69	0	-
0.5	1000	3691.12	0.24	0.93	583.80	526.55	0	-
0.6	1000	3565.83	0.29	0.88	565.80	516.73	0	-
0.7	1000	3351.39	0.45	0.76	467.50	464.88	178.00	398.02
0.8	1000	3247.08	0.63	0.71	442.17	454.67	208.20	408.37
0.9	1000	3179.27	0.70	0.67	416.50	436.71	29.75	59.50
1	1000	3128.34	0.74	0.62	395.67	425.35	33.25	66.50
1.5	1000	2975.06	0.90	0.46	310.17	383.29	146.50	179.09
2	1000	2888.05	0.93	0.36	250.83	349.33	140.67	243.64
4	310	795.16	0.98	0.12	28.00	68.10	∞	∞

Table 7.7: Performance measures for *Original 2*.

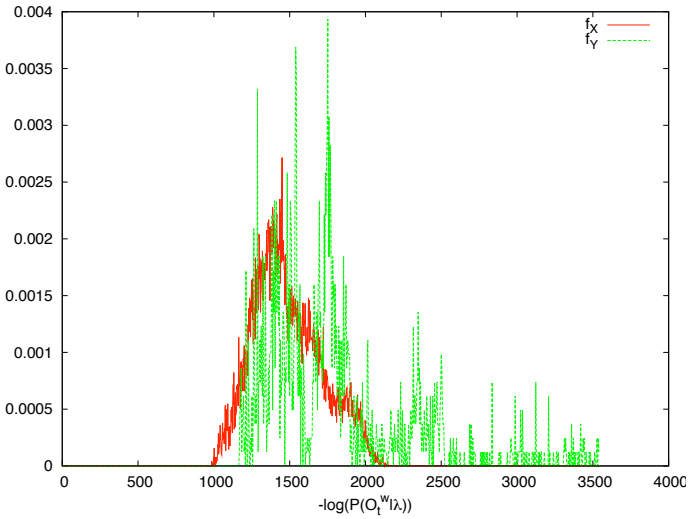


Figure 7.10: Distribution for data set *Original 2*, $w = 500$.

TP and TN. Considering the results, we observe that:

- the optimal choice of the design parameters w and θ is data set specific. This observation entails that w and θ have to be determined for the network under consideration; as could be expected, there are no universally suitable values;
- the optimal choice of the design parameters w and θ is (β/α)-specific. This means that, when increasing β/α, the system slowly shifts from favoring TN to favoring TP, as desired. This is visible by observing that for increasing values of β/α, TN shows a decreasing trend, while TP progressively increases.

Comparing the synthetic and original data sets, we observed that the measured detection rates are sometimes lower than expected. In particular, we observe that there is a decrease in performance for the original data sets with respect to the synthetic ones. More specifically, TP in the original data set shows just a mildly increasing trend, whereas TN is decreasing considerably faster than in the synthetic case. The explanation for this phenomenon is to be found in the distributions of malicious and benign observation sequences, that, as we pointed out in Section 7.5.2, determine the limitations to the system performance. In the case of original traces, we observed, compared to the synthetic case, a larger overlap of the benign and malicious distributions, i.e., those of X^w and Y^w. We give an example of such distributions in Figure 7.10. The distributions in Figure 7.10 are relative to the data set *Original 2* and calculated for $w = 500$. The consequence of such overlapping distributions is that, despite the fact that the chosen parameters are optimal, no better detection rates are possible in the current setup. Several causes are behind such a limited performance. First, the definitions of M^w and B^w in Section 7.4.1 are such that partially benign and partially malicious observation sequences would lead to similar ℓ_t^w values. We address this topic in Section 7.8. Second, the prototype used for our analysis, presented in Section 7.2, is very simple. Such a prototype was a good starting point for describing our optimization procedure, but several extensions would be needed before it could be used as an IDS. We discuss a possible extension in Appendix C. Finally, such results provide us with indirect information about the quality of the synthetic time series generated as in Chapter 6. In particular, in Section 6.4.4 we showed that our models are able to capture the main statistical characteristics of the original data sets, with the exception of the autocorrelation. We suspect that the higher random component in the synthetic time series causes the smaller overlap of the considered

distributions. Improving the autocorrelation is therefore a key topic for future extensions.

Detection and normalization lags

Tables 7.4, 7.5, 7.6 and 7.7 also report the average detection lag $\overline{D}_{\mathbf{A}}$ and the average normalization lag $\overline{E}_{\mathbf{A}}$ and their respective standard deviations. In calculating the detection and normalization lags, we adhere to the following conventions: $D_{\mathbf{A}} = \infty$ if an attack is undetected; $E_{\mathbf{A}} = \infty$ if the system does not recognize the normal situation within a w slots from the end of the attack. This choice is due to the fact that we are interested in the effect the history has in recognizing that an attack is over. $\overline{D}_{\mathbf{A}}$, $\overline{E}_{\mathbf{A}}$ and their respective standard deviations are calculated only on the finite values of $D_{\mathbf{A}}$ and $E_{\mathbf{A}}$.

In all considered data sets, we observe that the detection lags decrease for increasing values of β/α, that is, if the system favors TP. However, at the same time we also observe that, for increasing β/α, the normalization lag tends to increase. This situation describes well how varying the relative importance of TN and TP does affect not only the confusion matrix, but indirectly also how timely we are able to detect an attack or recover from it. Note that, in Table 7.6, for $\beta/\alpha < 0.4$, the standard deviation of both the detection and the normalization lags is undefined. This is due to the fact that, for this β/α ratio, the system detects only one attack. In Table 7.7, we observe that $\overline{E}_{\mathbf{A}}$ and $std(E_{\mathbf{A}})$ can be undefined or ∞. For $\beta/\alpha < 0.6$, i.e., for ratios that favor the TN rate, the normalization lag is equal to 0. This value, together with the low TP rate, indicates that the tail of the attack is generally missed and flagged as negative. For $\beta/\alpha = 4$, i.e., for a ratio that clearly favors the TP rate, the normalization lag raises to ∞. To understand this result it is necessary to keep in mind that β/α describes the relative importance of TN and TP. For such a high β/α ratio, the trace is almost entirely flagged as positive. Therefore, normality is never recognized within w slots after the end of the attack, leading to $\overline{E_{\mathbf{A}}} = \infty$.

Detection rates

We investigate the *detection rate* per attack. We conducted our analysis for varying β/α and considering the optimal threshold and observation sequence length, as presented in Tables 7.4, 7.5, 7.6 and 7.7. Table 7.8 presents the detection rate for the four considered data sets. In the case of the synthetic data sets *Synthetic 1* and *Synthetic 2*, the system correctly detect all the attacks, showing $DR = 100\%$. In the case of the original data sets *Original 1* and *Original 2*, we

observe that the detection rate is not constant, but changes with varying ratio β/α. As expected, the detection rate increases for increasing β/α, i.e., when the system favors the TP rate.

	DR (%)			
β/α	*Synthetic 1*	*Synthetic 2*	*Original 1*	*Original 2*
0.1	100	100	14	83
0.2	100	100	14	83
0.3	100	100	14	83
0.4	100	100	57	83
0.5	100	100	71	83
0.6	100	100	71	83
0.7	100	100	71	100
0.8	100	100	71	100
0.9	100	100	71	100
1.0	100	100	71	100
1.5	100	100	71	100
2.0	100	100	71	100
4.0	100	100	71	100

Table 7.8: Detection rates.

7.7 First approach towards adaptability

In the proposed system, the model λ describes the normal situation of the network. However, "normality" itself might be subject to changes. Network data, and derived data as for example the flow-based time series, might change over time. Holidays can influence the number of users on the network; periodic backups can influence the network usage pattern; finally, exceptional data transfer can bias our knowledge.

A challenging line of research is to make a detection system resilient to changes in the modeled reality, or in other words, introducing *adaptability* into the system. Adaptability can be seen as a form of system optimization over the long term. The aim of adaptability is indeed to maintain high performance also in case of the traffic features changing over time. With respect to this thesis, we consider adaptability the natural follow up of the presented research, and a new topic of research in itself. In this section, we take a first step into this

new field. The section should therefore be seen as a proof of concept, showing possible ways to introduce adaptability in a system to improve its long term performance.

7.7.1 Adaptability in the normal network model

A direct approach to deal with outdated normality models would be to retrain them. However, to retrain, one needs new ground truth data sets, which, as discussed in Chapters 5 and 6, can be difficult to obtain. Some authors [149] propose to use the traffic that an IDS flags as benign to retrain the IDS itself. Others propose to base the IDS on models that are resilient to noisy data, performing therefore on-line learning without the use of a ground truth data set [80]. However, the risk in this solution is that the IDS may slowly learn classification errors, since we might include malicious traffic in the pool of training data. In this section, we investigate if it is possible to develop an alternative approach to adaptability that avoids the retraining of the model. To do this, we will first discuss the various forms in which adaptability can be introduced in a system. For that purpose, we concentrate once again on our HMM-based "demo system" (Section 7.2).

Adaptability could be introduced in the system in Section 7.2 in any component of our HMMs that has been learned on the training set. We identify the following situations:

- **Markov chain**: the Markov chain in Figure 7.2 describes the behavior of the network w.r.t. interleaving active and inactive time slots. The transition probabilities are learned based on the training set. If the traffic pattern changes over time, as for example because we observe a higher number of inactive (or active) time bins, it is possible that the transition probabilities need to be adjusted.

- **Output distributions**: in our models, the output distributions are empirically learned on the training set. However, as previously described in this section, flow measurements can be subject to change relatively to the number of users on the network and the usage pattern. In this case, the transition probabilities might not change, but we observe a higher number of flows/s. The output distributions spaces are therefore likely to also change over time.

In the case of the data sets used in this thesis, we observed that the normal network behavior captured in the Markov chains is subject only to slight

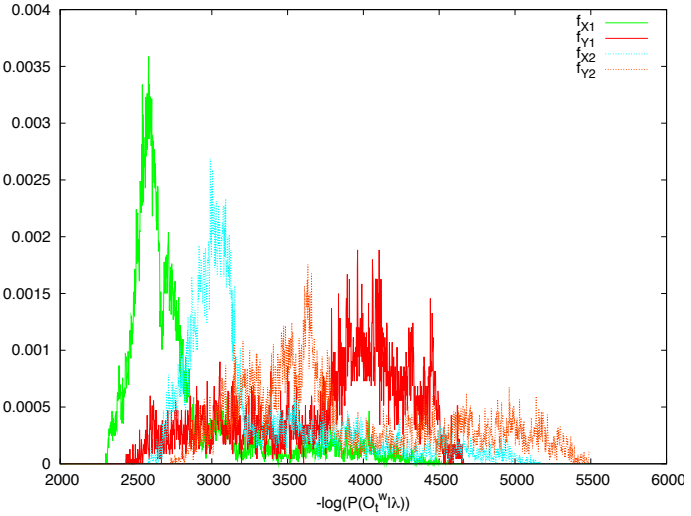


Figure 7.11: Changes in the ℓ_t^w distributions in *Set 1* and*Set 2*: empirical density functions .

changes (see Figure 6.6). Despite several months have passed between the collection of the data sets *Set 1* and *Set 2*, we measure an absolute difference lower than 0.025 in the transition probabilities of the two considered data sets.

The output distributions of the training part of *Original 1* and *Original 2*, however, show significant differences in the empirical distributions boundaries. Figures 7.11 and 7.12 show the empirical distribution function for *Original 1* and *Original 2*, $w = 1000$, and the fitted Gaussian Mixture distributions, respectively. Note that in these figures we indicate with f_{X1} and f_{Y1} the density functions for, respectively, the benign and malicious observation sequences for the data set *Original 1*. Similarly f_{X2} and f_{Y2} indicate the benign and malicious observation sequences for the data set *Original 2*. We can observe the following:

- considering the lower bound of the probability domain, the empirical distributions are subject to a *shift*. Such a shift can be due, for example, to a different number of users on the network, and, consequently a higher network usage. This observation is consistent with the fact that *Original 1* has been created in summer time, while *Original 2* during a normal

working period;

- considering the upper bound of the probability domain, we observe that the empirical distributions are, in this case, subject to a *shift* and a *stretching factor*. This phenomenon can be due to the increased attack intensities measured in *Original 2* compared to *Original 1* (see Table 6.1).

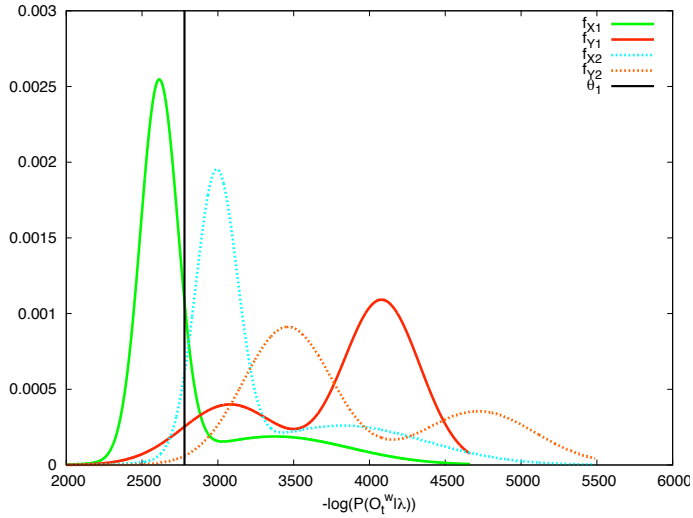


Figure 7.12: Changes in the ℓ_t^w distributions in *Set 1* and*Set 2*: fitted distributions .

In Figure 7.12, we have indicated the optimal threshold θ_1 for data set *Original 1*, for the parameters $w = 1000$, $\alpha = \beta = 1$. From Table B.4, $\theta_1 = 2786$. The above observations and the situation in Figure 7.12 show how θ_1 yields to suboptimal results if it is applied to observation sequences from data set *Original 2*. The optimal threshold for *Original 2* would have a higher value than θ_1. From the data in Table B.4, we have $\theta_2 = 3128$. A possibility to introduce adaptability in the system would be to parametrize the Gaussian mixture fits not only on the window size, but also on a measure of traffic that is indicative of a change in the traffic characteristics. An example for such measure can be the average throughput in a certain observation period. However, this solution would introduce an additional approximation in the Gaussian fit, besides the polynomial approximation of the parameters $\boldsymbol{p}^w$, $\boldsymbol{\mu}^w$ and $\boldsymbol{\sigma}^w$ (Section 7.6.1).

A simpler solution would be to adapt the threshold according to an observed range of ℓ_t^w values. We present details about this approach in the following.

A geometrical approach to adaptability

We propose a "geometric method" to adapt the threshold θ. Let us suppose that the range of ℓ_t^w values for the data used to train the system is

$$r_{old} = [\min_{old}, \max_{old}].$$

The system is now analyzing new data for which the range of ℓ_t^w values is

$$r_{new} = [\min_{new}, \max_{new}].$$

We assume that $|\min_{new} - \min_{old}| \geq 0$ (there might be a shift) and $||r_{new}| - |r_{old}|| \geq 0$, where with $|r|$ we indicate the length of the interval r (there might be a stretch).

We define the lower bound shift Δ_1, the upper bound shift Δ_2 and the stretch s:

$$\Delta_1 = \min_{new} - \min_{old}, \tag{7.13}$$

$$\Delta_2 = \max_{new} - \max_{old}, \tag{7.14}$$

$$s = \Delta_2 - \Delta_1. \tag{7.15}$$

Figure 7.13 summarizes the introduced measures. Given a threshold θ_{old} com-

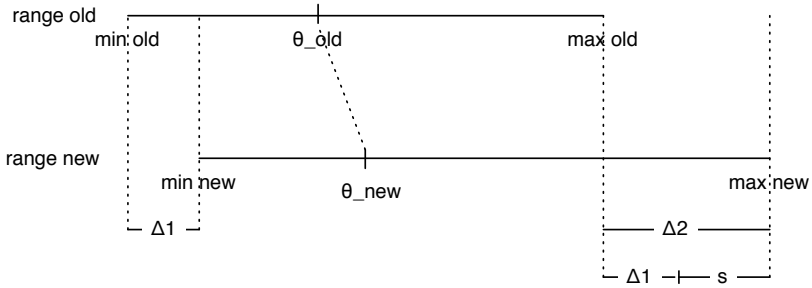


Figure 7.13: Adaptability schema.

puted accordingly to the optimization procedure, we now want to calculate

the value of the new threshold θ_{new}, keeping into account that (i) θ_{new} is the result of the combined action of shift and stretch factors and (ii) the stretch is not constant. We propose the following formula

$$\theta_{new} = \theta_{old} + \Delta_1 + s \cdot \rho^n, \tag{7.16}$$

where $\rho = \frac{\theta_{old} - min_{old}}{max_{old} - min_{old}}$ quantifies the relative position of θ_{old} in the interval r_{old}. The exponent n is needed to express the fact that, according to our measurements, the stretch is usually not linear. In our data, in the experiments that follow we use $n = 3$.

Experimental results

The experimental results presented in this section aim to verify that the geometric threshold scaling can help in maintaining performance levels similar to the optimal ones in cases in which the model of normality is outdated. In other words, we verify that we can use the threshold scaling method instead of the more costly operation of retraining. To validate the threshold scaling method, we proceed as follows:

1. we train the model of normality λ on the training data set *Set 1* (see Section 7.6.1);

2. we measure the performance of the system on the data set *Synthetic 2*. This data set contains data that are *foreign* to the model λ. The time series in *Synthetic 2*, indeed, mimic the behavior of the training set *Set 2*. We refer to this operation as *cross testing*;

3. finally, we measure the performance of the system on the data set *Synthetic 2* when threshold scaling is applied.

Table 7.9 summarizes the performance results for the test data set *Synthetic 2*. Table 7.9 is to be read as follows. For each β/α, the system solves the optimization problem and computes, at the best of its knowledge, the optimal threshold, θ_{opt}. Table 7.9 then reports the optimal TP and TN rates for the *Synthetic 2* when tested against a model trained on *Set 2*. We indicate this model as λ_2. These results are the same of the one in Table 7.5 and are reported here for completeness. Table 7.9 then show the performance in the case in which the model is trained on *Set 1*. We indicate this model as λ_1. We calculate the optimal TP and TN rates measured when testing λ_1 on *Synthetic 2* without applying threshold

	λ_2-*Synthetic 2*			λ_1-*Synthetic 2* (cross test)			λ_1-*Synthetic 2* (scaling)		
β/α	θ_{opt}	TP	TN	θ_{cr}	TP_{cr}	TN_{cr}	θ_{sc}	TP_{sc}	TN_{sc}
0.1	418.39	0.41	0.98	381.59	0.78	0.81	409.13	0.64	0.91
0.2	397.30	0.49	0.97	404.24	0.83	0.75	432.08	0.7	0.88
0.3	386.29	0.53	0.96	430.12	0.86	0.71	459.57	0.74	0.86
0.4	378.47	0.57	0.94	426.11	0.87	0.69	454.74	0.76	0.84
0.5	765.59	0.74	0.89	423.07	0.88	0.67	451.09	0.78	0.82
0.6	856.51	0.78	0.87	451.3	0.89	0.64	481.87	0.79	0.81
0.7	915.30	0.81	0.85	449.13	0.9	0.63	479.27	0.8	0.8
0.8	974.55	0.83	0.84	447.26	0.9	0.62	477.03	0.81	0.79
0.9	1001.75	0.85	0.83	445.63	0.91	0.61	475.06	0.82	0.78
1	1061.59	0.85	0.82	444.16	0.91	0.6	473.31	0.83	0.77
1.5	1106.62	0.88	0.77	438.54	0.92	0.56	466.6	0.85	0.74
2	1124.13	0.91	0.74	464.26	0.93	0.52	498.81	0.86	0.74
4	1055.90	0.95	0.62	454.01	0.95	0.45	486.32	0.89	0.67

Table 7.9: Performance measures for *Synthetic 2* (i) tested against the model of normality λ_2, (ii) in case of cross testing and (iii) when the threshold scaling method is applied.

β/α	$\text{TP}_{\text{cr}} - \text{TP}$	$\text{TP}_{\text{sc}} - \text{TP}$	$\text{TN}_{\text{cr}} - \text{TN}$	$\text{TN}_{\text{sc}} - \text{TN}$
0.1	0.37	0.24	-0.18	-0.07
0.2	0.34	0.22	-0.22	-0.09
0.3	0.32	0.21	-0.24	-0.10
0.4	0.30	0.19	-0.26	-0.10
0.5	0.14	0.04	-0.22	-0.06
0.6	0.11	0.01	-0.22	-0.05
0.7	0.09	-0.01	-0.22	-0.05
0.8	0.07	-0.02	-0.22	-0.05
0.9	0.06	-0.03	-0.22	-0.04
1	0.06	-0.03	-0.22	-0.04
1.5	0.04	-0.03	-0.21	-0.03
2	0.03	-0.05	-0.22	0
4	0	-0.06	-0.17	-0.05

Table 7.10: Differences in performance measures between (i) *Synthetic 2* and the cross testing case and (ii) *Synthetic 2* and the threshold scaling case.

scaling (cross testing). Finally, Table 7.9 reports the performance of λ_1 when threshold scaling is applied.

Considering the cross testing case, in which no threshold scaling is applied, from Table 7.9 we observe that the system has good performance w.r.t. the TP rate, but the TN rate decreases rapidly towards 50%, for increasing β/α. By comparison with the case in which we use a model trained on *Set 2* on the data set *Synthetic 2*, we know that the performance for the cross testing is not optimal. This situation can be understood considering the example in Figure 7.11. The benign and malicious distribution for *Set 2* are shifted towards higher values of ℓ_t^w, compared to the ones for *Set 1*. Therefore, the optimal threshold for *Set 1* underestimates the optimal threshold for *Set 2*, resulting in a lower TN rate and in a higher TP rate. The threshold scaling compensates the shift by mapping each θ_{opt} to a higher value.

Table 7.10 shows the changes in the performance of our system, expressed as the difference between the optimal rates (TP and TN), the rates in case of cross testing (TP_{cr} and TN_{cr}) and the rates when we apply the threshold scaling, (TP_{sc} and TN_{sc}). Considering the TP rate, we observe that both the cross testing and the threshold scaling approximates well the original TP values. However, the threshold scaling has a better approximation for $\beta/\alpha < 0.6$. In the case of the TN rate, however, the threshold scaling yields to a better approximation of the optimal performance for all the considered values of β/α, with an error lower than 10%.

7.8 Is a binary classifier enough?

Until now, we described an IDS as a binary classifier. This approach, where the IDS signals either that an attack is ongoing, or that there is no attacks, is common in IDS research [14, 13, 158]. However, we can argue if this approach is optimal given the type of problem instance we want to classify. Each observation sequence O_t^w is a set of observations. Each observation is in itself either malicious or benign, as explained in Section 7.4.1. As a consequence, there might exist several possible definitions of the actual classes M and B. For example, we could define an observation sequence as benign only if all its observations are benign; or, if at least half of its observations are benign. The definition we proposed in Section 7.4.1 discriminates between the malicious and benign classes by considering only the youngest observation o_t. However, the drawback of such a definition lies in the fact that the detection system might wrongly classify observation sequences at the beginning and at the end of an attack, as discussed in Section 7.4.2. Almost completely benign (malicious) sequences for which o_t is malicious (benign) are likely to be misclassified.

This observation suggests that there exists a subset of O_t^w for which we do not have enough knowledge to perform a correct classification, i.e., a third class U of *undecided* instances. In this context, a binary classifier like the one described in Section 7.4.1, would not be sufficient anymore. To picture it in a metaphorical manner, think to a traffic light. The green and red lights give you definite indication of your situation (you are allowed or prohibited to proceed): these are the classes of malicious and benign observation sequences; the yellow light warns you that you should expect a change (you should stop soon): this corresponds to the *undecided* class.

We present here a possible way to rewrite our classification problem such as to include *undecided* instances. Note, however, that this should be considered just one among many possible extended definitions:

$$\begin{aligned} M^w &= \{O_t^w \mid \forall i, t-w+1 \leq i \leq t, o_i \text{ is malicious}\}, \\ B^w &= \{O_t^w \mid \forall i, t-w+1 \leq i \leq t, o_i \text{ is benign}\}, \\ U^w &= \{O_t^w \mid \exists i, j, t-w+1 \leq i, j \leq t, j \neq i, o_i \text{ is benign}, o_j \text{ is malicious}\}. \end{aligned}$$

In the new definition, an observation sequence is benign if *all* the observation in it are benign. Similarly, an observation sequence is malicious if *all* the observation in it are malicious. The class U^w describes the *undecided* instances, containing both malicious and benign observations. It is now necessary, however, to define the behavior of the system while handling observation sequences belonging to U^w. A possibility in this sense is to model the statistical properties of the observation sequences in U^w, as it has been done for M^w and B^w. We could therefore define a random variable

$$V^w := -\log \Phi(O_t^w \mid \lambda) \quad \text{for } O_t^w \in U^w.$$

For each $O_t^w \in U^w$, the measure

$$\mathbb{P}(V^w \leq -\log \Phi(O_t^w \mid \lambda))$$

could be the starting point for describing how close the considered observation sequence is to a benign or malicious sample.

A third actual class leads to the definition of a multi-threshold system. Similarly to what we did in (7.7), we could now find the optimal threshold θ_1 between the X^w (benign) and V^w (undefined) distributions, as well as the optimal threshold θ_2 between the V^w (undefined) and Y^w (malicious) distributions. The system would then report:

- that the current observation sequence is a negative sample if $\ell_t^w < \theta_1$;

- that the current observation sequence is a positive sample if $\ell_t^w \geq \theta_2$;
- the degree of danger for the current observation sequence if $\theta_1 \leq \ell_t^w < \theta_2$.

More research is needed to assess if a multi-threshold system like the one sketched here is suitable for detection.

7.9 Summary

In this chapter, we presented a probabilistic approach, in terms of the optimization procedure in Table 7.2, to tune the parameters of a simple anomaly-based IDS that entirely relies on flow information. The optimization procedure we proposed allows to maximize the correct detection and minimize the errors. Moreover, the procedure takes explicitly into account high-level policies, in the form of the relative importance of detecting all the attacks versus keeping the false alarm rate low. The system parameters we are interested in are the observation sequence length w and the probability threshold θ. We formalized the optimization problem as:

$$\underset{\theta,w}{\text{maximize}} \quad \alpha \cdot \text{TN} + \beta \cdot \text{TP},$$

where α and β express the relative importance of TN and TP.

By varying the ratio β/α, we were able to fine tune the system to favor either the detection of all the anomalies (high TP rate) or the detection of attacks only when they are certain (high TN rate). Our findings also show that β/α has impact on the detection rate and on how timely the system is able to detect an attack or recover from it. We believe therefore that, when expressing a usage policy in terms of the relative importance we put on TN and TP, it should be taken into account that such a policy affects the system performance in multiple ways. The choice of which metric is most important, and therefore of which is a suitable policy, is left to the user.

In this chapter, we also took a first step towards adaptability, with the aim of making the system resilient to changes in the normal situations of the monitored network. Adaptability can be seen as a form of system optimization over the long term, coping with changes of the traffic features over time. We proposed an approach to adaptability that avoids the retraining of the system, operation that can be costly due to the need of ground truth data sets. Our findings showed that it is possible to geometrically scale the threshold θ to ensure results close to optimal.

Finally, some directions for future work emerged from our analysis. In particular, we investigated how the performance of our demo system could be improved by introducing a third classification class. Our analysis showed that there might exist cases in which there is not enough information for taking a definitive decision on the nature of the traffic. We argue therefore that the traditional concept of a detection system as a binary classifier could not be enough. In Section 7.8, we envision that a three-class classifier could enrich the detection system and improve its performance.

CHAPTER 8

Conclusions

This chapter presents the conclusions to the research in this thesis and suggests some directions for future work.

The chapter is organized as follows:

- Section 8.1 presents the overall conclusions to the research in this thesis. In this section, we also provide answers to the Research Questions that we have identified in Section 1.3.
- Section 8.2 concludes the thesis by pointing out possible research extensions and directions for further work.

8.1 Overall conclusion

As suggested by the survey of the state of the art in Chapter 3, the research in flow-based intrusion detection is flourishing with solutions to different aspects of the detection problem. Researchers focus their attention on diverse attack types, as well as on different IDS types. However, as often happens, what goes unsaid is sometimes more interesting than what is in plain light. Two aspects of the intrusion detection research have therefore caught our attention, mainly for their absence.

First, despite the variety of solutions for intrusion detection, a common effort towards IDS evaluation is still missing. As we argued in Chapters 5 and 6, all the intrusion detection solutions have in common the need for public *ground-truth* data sets. Ground-truth data sets are time-consuming to create, require deep domain-knowledge and are rarely shared since they often are privacy sensitive. In this thesis, we overcame the lack of ground-truth data sets by creating a flow-based public data set (see Chapter 5) and by proposing a method to create labeled synthetic time series (see Chapter 6). We believe that

the community would benefit from a larger number of shared ground-truth data sets, for both validation and results comparison purposes.

Second, once a new IDS has been proposed, its tuning is often left to the attention and expertise of IT specialists, who have the duty to make operational decisions. However, nothing but experience, obtained through trial-and-error, ensures the specialist that he is using the IDS at its optimal levels and that no better detection results can be achieved. In light of the research conducted in this thesis, we believe that parameter tuning for intrusion detection should be addressed in a systematic manner to ensure optimal detection. Note that optimality is to be intended in the context of the trade-off between false positive and false negative, as presented in Figure 1.3. We also believe that optimality can be situation-specific and high-level policies addressing optimality conditions should be taken into account while tuning the system parameters.

As stated in Section 1.3, the goal of our research was to propose a structured approach to detect anomalies using flow data and time series. We believe that such an approach should bring into focus topics as *validation* and *tuning* of IDSs. Our general conclusion is therefore that the research attention, focused mainly on intrusion detection *systems*, should be enlarged to include issues that can be considered as a basis of intrusion detection: publicly available ground-truth data sets and optimal parameter tuning.

Our research goal has been previously refined into four Research Questions (see Section 1.3), to which we now provide the following answers.

Research Question 1: What is the state of the art in the field of flow-based intrusion detection?

In Chapter 3, we presented a survey of the state of the art in flow-based intrusion detection. Flow-based intrusion detection is a relatively recent field of research, whose first contributions date back to 2002. Since then, several approaches to the problem of detection have been proposed. By analyzing and categorizing them, we identified the major trends in the field (Section 3.6). We concluded that the research efforts are at the moment focused on passive and centralized solutions with, primarily, centralized data collection. Moreover, we also noticed an evenly shared interest between anomaly-based and misuse-based systems and a clear attention for real-time systems.

Research Question 2: How can traffic anomalies be characterized in time series derived from flow data?

In our research, we focused on anomaly characterization in time series. Chapter 4 presented an extensive data analysis on flow-data from the University of Twente and SURFnet, the Dutch national research and education network. Our analysis leads us to the following conclusions. First, *time series* of flows, packets and bytes can be a suitable approach to flow-based intrusion detection, since they allow data analysis by keeping into account temporal relations between events, in this case the amount of traffic. Second, to more clearly identify and characterize anomalies, we suggest to perform an application-based *traffic breakdown*. With this we mean that anomalies that are not noticeable by looking at the whole traffic can be identified by looking at, for example, only SSH or DNS traffic. Examples of such anomalies are provided in Sections 4.3 and 4.4. The traffic breakdown can therefore empower flow-based intrusion detection by reducing the amount of data to be analyzed and by facilitating anomalies exposure. This observation suggests that a feasible approach to flow-based intrusion detection should encompass the design of modular intrusion detection systems targeting specific applications. Moreover, the combined analysis of flow, packet and byte time series can strengthen the certainty of the presence of an attack. However, for certain classes of attacks, the choice to monitor only some of the aforementioned metrics can be sufficient.

Research Question 3: How can we determine ground-truth information for flow-based intrusion detection?

We cover two approaches aiming to determine ground-truth information for flow-based intrusion detection.

First, we investigated the creation of a flow-based data set of security-relevant events, where each of the attacks has been manually labeled. Building this type of data set can be challenging. Our research covered several aspects of the problem, namely which requirements should such a data set meet (Section 5.1.3), which infrastructure is suitable for data collection (Section 5.2) and, finally, how the collected data can be labeled (Section 5.3). Our findings show that the most promising measurement setup among the analyzed ones is monitoring a single host with enhanced logging capabilities. The information collected permitted us to create a database of both flows and security events (derived by the logs). However, we are aware of the limitations that our approach entails. In particular, the fact that the collected trace mainly consists of ma-

licious traffic. As results of the research in Chapter 5, we built and publicly released a flow-based labeled data set. At the best of our knowledge, our effort constitutes the first publicly available labeled flow-based traffic trace. The data set is available in anonymized form at the address: *http://traces.simpleweb.org*. Despite this favorable outcome, the lesson learned in Chapter 5 is that, although we limited our experiments to a single host, labeling remains a complex task that requires human intervention.

In Chapter 6, we investigated the possibility to generate ground-truth information in an automated manner. We proposed a modeling approach for flow-based traffic time series based on Hidden Markov Models (HMMs). We showed that the models that we developed provide a compact representation of the traffic, where only few states are needed to fully describe the traffic evolution (Section 6.3). Moreover, HMMs can be used for generative purposes, allowing us to create time series for which the ground truth is known. In Chapter 6, we focused on SSH traffic, that became our running example through the remainder of the thesis. The models we proposed are inferred by studying real SSH traffic time series for both attack and normal traffic, captured at the University of Twente. The research presented in Chapter 6 showed that: (i) our HMM-based approach can capture the main statistical characteristics of the original time series (as quantitatively shown in Section 6.4.4); (ii) by qualitative investigation, our approach can reproduce time series that resemble the real traffic (see Section 6.4.5); (iii) as far as ground truth is concerned, we are now able to create labeled data sets in an automated manner.

Research Question 4: How can we tune the parameters of a flow-based IDS based on high-level policies?

The performance of an IDS is governed by the trade-off between false positives and false negatives. In Chapter 7, we proposed a mathematical framework, in terms of an optimization procedure, to treat such trade-off in a systematic manner. We tested our procedure on a *history-based*, *probabilistic* and *anomaly-based* detection system that we introduced in Section 7.2. Such a simple model became our demo system. We then proposed, in Section 7.5, a probabilistic optimization procedure to tune parameters of the detection system such as the history length and the alert threshold with the aim of maximizing the correct detection (*true negative*) and minimizing the errors (*false positive*). A key characteristic of the proposed solution is that it regards optimality according to the high-level policies, since the choice of which side of the aforementioned trade-off to favor is case-specific.

Our extensive validation on synthetic and original ground-truth data sets, in Section 7.6, showed that several optimal solutions are possible. However, it should also to be noted that, by describing malicious and benign traffic in terms of probability distributions, as in Figure 7.7, we shed light to the limitation of the proposed system. The optimal true positive and false positive rates are limited by the overlapping region between the malicious and benign traffic distributions. The presence of such overlapping region is partly due to the definitions of the malicious and benign observation sequences (see Section 7.4.1). We argue in Section 7.8 that the definitions we chose, although motivated by the presence of an attack in the analyzed history, might be too strict and can cause a decrease of the system performance. We therefore suggest that the current IDS practice, based on a two-class comparison between the ground truth (benign and malicious traffic instances) and the system output (positive and negative traffic instance), might not be the most suitable one for history-based, flow-based systems.

8.2 Future research direction

We present here some suggestions for further research:

- The focus of this thesis was on modeling and analysis of real data traces. Therefore, a real-time prototype has not yet been implemented. Building such a prototype would pose interesting challenges, such as, for example, the efficient on-the-fly creation of time series. Even though our approach is based on a Netflow v5 type of information, we believe that a running prototype would benefit from using newer technologies, such as, for example, the extended timeout management of Flexible Netflow. Moreover, a prototype would compel us to investigate the usability of the system, similarly to what has been discussed in Appendix C.

- The running example through this thesis has been SSH traffic. There has been no fundamental decision in choosing SSH over other protocols, beside the fact that we found it to be among the most frequent types of attacks. Nothing in our approach is SSH-specific. This means that our HMM-based approach could be extended in multiple directions, namely, attacks other than dictionary attacks, and applications other than SSH.

- The research conducted in Chapter 5 concluded with a publicly available flow-based ground-truth data set. However, we are aware that this

promising result has left some unanswered questions. We believe that a compelling one concerns how to extend the current data collection architecture such that we can capture not only malicious traffic, but also benign traffic. Of course, while doing that, data set requirements such as described in Section 5.1.3 will still be a "must", which poses us the challenge of finding a new approach.

- In this thesis, and in the literature in general, an IDS can be seen as a binary classifier, which' output classes are "positive" (attack) or "negative" (normal traffic). However, in Chapter 7 we argue that this convention might not be totally satisfactory for probabilistic history-based detection systems. It would be interesting to investigate if the current evaluation paradigm can be extended, for example to a three-class classifier, where the third class would now include instances for which we do not have enough information to take a decision. Moreover, it would be interesting to measure the effect of such extension on the overall performance of the detection system.

APPENDIX A

Hidden Markov Models

Hidden Markov Models (HMMs) are a class of statistical models able to describe sequences of data resulting from the interaction of several random processes.

HMMs are effective in modeling sequential data [17]. Introduced in the early 1970's [10], they have been successfully applied to different scientific fields. Examples are biological sequence analysis [42], speech recognition [122] and pattern recognition [50]. HMM can be trained on real data and their main characteristic is the ability to capture the temporal behavior of the observed processes.

In the field of networking, several contributions rely on the framework of HMMs. The work of [157] proposed to formalize traffic exchange in terms of "HMM profiles", a stochastic structure suited for sequence alignment. The same models have been successfully used while searching for similar protein structure in large protein databases [43]. The results in [157] showed that the models are able to classify traffic sequences at application level. In [28, 27], the authors proposed a packet-level model of traffic sources based on HMMs. The authors aim to model packet inter-arrival time and packet size and the model proved to be effective in application classification. This application is similar to the traditional use of HMMs in speech recognition: a new sample is tested against a pool of models, each one encoding a spoken word or a specific application behavior. The model that more closely approximate the sample, i.e., the model that has the higher probability to encode the sample, defines the class of the sample. In this way, a word or an application can be recognized. Moreover, a second fruitful application of the models in [28, 27] was in traffic prediction, namely to forecast short-term future traffic behavior.

A.1 Formal description

Originally introduced as "probabilistic functions of finite state Markov chains" in the work of Baum *et al.* [9], HMMs have been later made popular in the computer science community by the extensive tutorial by Rabiner [122].

Formally, an HMM is a Discrete Time Markov Chain (DTMC) where each state is augmented with a probability distribution over a finite set of output symbols. Given a sequence of states $Q = q_1 q_2 \dots$ with associated output symbols $O = o_1 o_2 \dots$ we say Q forms the *hidden sequence* and O forms the *observation sequence*. In this section we introduce the ideas at the basis of HMMs by means of a running example, in which we show how a basic intrusion detection system could be modeled as an HMM.

> **Example:** *The aim of an anomaly-based network intrusion detection system is to detect if any malicious activity is taking place on the monitored network. We therefore regard an IDS as a black box. At each instant of time, the black box observes network measurements, for example the number of flows per second, and provides as output a summary of the safety state of the network:* $\mathbb{A}$ *if an attack in ongoing and* $\mathbb{S}$ *if the network is safe.*

Following Rabiner [122], an HMM is characterized by the elements below:

N: the number of states in the model. The individual states are denoted as $S = \{s_1, \dots, s_N\}$. The state at time t is indicated with q_t. A model can be defined by the parameter N or by assigning physical significance to each state. The latter will occur in the models we propose.

M: the number of distinct observation symbols per state. The individual symbols are denoted as $V = \{v_1, \dots, v_M\}$. The observation symbols are the output of the model. The output at time t is defined as o_t.

A: a state transition probability matrix $A = \{a_{ij}\}$, where

$$a_{ij} = \mathbb{P}(q_{t+1} = s_j \mid q_t = s_i), \quad 1 \leq i, j \leq N.$$

B: an observation symbol probability distribution $B = \{b_j(v)\}$, where

$$b_j(v) = \mathbb{P}(o_t = v \mid q_t = s_j), \quad 1 \leq j \leq N.$$

In other words, $b_j(\cdot)$ is the probability distribution of the output symbols in the state s_j. Each state in the model will describe its own output symbols distribution.

π: the initial state distribution $\pi = \{\pi_i\}$, where

$$\pi_i = \mathbb{P}\{q_1 = s_i\}.$$

A model is usually referred to with the compact notation of $\lambda = (A, B, \pi)$, thereby leaving N and M implicit.

Example:

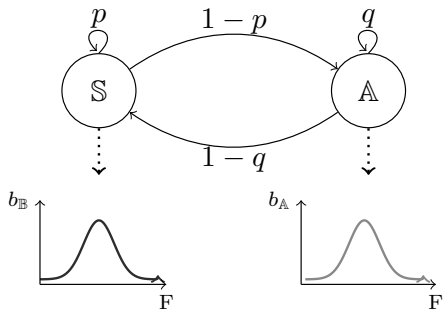


Figure A.1: Assessing the safety status of a network.

Let us now have a look into the box. We can model the proposed flow-based detection system with an HMM as follows:

- $N = 2$. *We say that the model is in state* $\mathbb{S}$ *if the network is safe and in state* $\mathbb{A}$ *if the network is under attack;*
- $M = K + 1$, *where we assume* K *to be the maximum number of flows/s that we can observe on the network. Therefore* $V = \{0 \dots K\}$.
- *The detection system transits from the state* S *to the state* A *when a malicious activity is detected, and from the state* A *to the state* S *when the malicious activity is over. In this example, we assume* $p = 0.9$ *and* $q = 0.9$.
- *since our model is anomaly-based, we abide to the general assumption that malicious activities deviate from the normal behavior of the observed system, in our case, the network. We indicate with* $b_{\mathbb{S}}(\cdot)$ *the density function associated to state* $\mathbb{S}$, *and with* $b_{\mathbb{A}}(\cdot)$ *the density function associated to state* $\mathbb{A}$. *In this example, we assume the output symbols in state* $\mathbb{S}$ *to be distributed as the binomial distribution*

$\mathcal{B}(K+1, 0.5)$. *Similarly, we assume the output symbols in state* $\mathbb{A}$ *to be distributed as* $\mathcal{B}(K+1, 0.9)$.

- *when the detection system becomes operational, the network might be safe or currently under attack.* π *indicates the probability that each one of these events happen in the moment we start monitoring the network. In our example, we assume* $\pi_{\mathbb{S}} = 0.8$ *and* $\pi_{\mathbb{A}} = 0.2$.

A.2 The three basic problems for HMMs

The literature refers to three basics problems that need to be addressed to make HMMs an effective tool with practical applications [122]. The basic problems are:

Problem 1 : given an observation sequence $O = o_1 o_2 \dots o_T$ and a model $\lambda = (A, B, \pi)$, we want to efficiently compute $\mathbb{P}(O|\lambda)$, that is the probability of observing the sequence O given the model.

Problem 2 : given an observation sequence $O = o_1 o_2 \dots o_T$ and a model $\lambda = (A, B, \pi)$, we want to choose the state sequence $Q = q_1 q_2 \dots q_T$ that best explains the observation sequence, that is, the most probable sequence Q.

Problem 3 : given a set of observation sequences, $\mathcal{X} = \{(o_{1,k} o_{2,k} \dots o_{T,k})\}_k$, that we call *training set*, we want to infer the model parameters that maximize the probability of generating $\mathcal{X}$.

The three basic problems have been long investigated. The rationale behind Problem 1 is that there might be more that one hidden sequence Q that will lead to output O, and enumerating all of them is a costly operation. Baum *et al.* [11] developed an iterative solution, known as *Forward-Backward procedure*, capable of efficiently compute the probability of an observation sequence.

> **Example:** *Let us consider again the flow-based detection system. Let us consider to have observed the following sequences of flows/s:*
>
> $$\begin{aligned} O_1 &= \{100, 102, 100, 100, 90, 100, 95, 100, 98, 100\}, \\ O_2 &= \{100, 102, 100, 100, 90, 90, \mathbf{200}, \mathbf{200}, 80, 100\}. \end{aligned}$$
>
> O_1 *has been structured such that it mimics a likely observation sequence. The component of* O_1*, indeed, are likely to be all generated by the output*

distribution of $\mathbb{S}$*. On the contrary,* O_2 *mimics the happening of an unlikely event that raises the number of flows/s, such as an attack. For such observations sequences, we have, respectively,* $\log(\mathbb{P}(O_1|\lambda)) = -32.2$*, and* $\log(\mathbb{P}(O_2|\lambda)) = -63.7$*, indicating that* O_1 *has a higher probability than* O_2*. Note that the logarithmic notation is introduced since* $\mathbb{P}(O = (o_1 \dots o_n)|\lambda) \to 0$ *for* $n \to \infty$*.*

Note that, if, for a given observation sequence, the hidden sequence is known, the calculation of the observing the sequence can be simplified as follow. Given an observation sequence $O = o_1 o_2 \dots o_t$ and the state sequence $S = s_1 s_2 \dots s_t$ that generated it, we can calculate

$$\mathbb{P}(O|\lambda) = \pi_{s_1} b_{s_1}(o_1) \prod_{i=2}^{t} a_{s_{i-1} s_i} b_{s_i}(o_i).$$

Also for Problem 2 the assumption holds that more than one hidden sequence Q might lead to the observation sequence O. However, not all the hidden sequences will be equally likely. The *Viterbi algorithm* [152, 53] efficiently calculates the most likely hidden sequence, that is the hidden sequence Q that maximizes $\mathbb{P}(Q, O|\lambda)$.

Example: *Considering the observation sequences* O_1 *and* O_2 *introduced in the previous example, the most probable hidden sequences are:*

$$\begin{aligned} Q_1 &= \{\mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}\}, \\ Q_2 &= \{\mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{S}, \mathbb{A}, \mathbb{A}, \mathbb{S}, \mathbb{S}\}. \end{aligned}$$

Finally, Problem 3 defines the case in which the model parameters are learned from a set of observation sequences. Solving Problem 3 is fundamental to tune an HMM to a specific problem and a specific training set. In particular, it is fundamental to estimate the correct transition probabilities. Parameter estimation for HMMs is a well-studied field, and several approaches have been proposed, such as the Baum-Welch algorithm [10] or the simulated annealing method of [3]. However, if the hidden sequence is known, as for applications in which there exists additional domain knowledge, it is possible to train the models in a simpler way, *i.e.*, by estimating the transition probabilities as:

$$a_{ij} = \frac{|\{\text{transitions from } s_i \text{ to } s_j\}|}{|\{\text{transitions from } s_i\}|}.$$

APPENDIX B

Addendum on optimization procedure validation

This appendix is meant to support Chapter 7. It reports extra results on (i) the relations between the Gaussian Mixtures parameters and the window size, in Section B.1 and (ii) the optimization procedure, in Section B.2.

B.1 Gaussian fits and window size

In Section 7.5.2, we indicated that a possibility to generalize the relation between the Gaussian Mixture parameters $\boldsymbol{p}^w$, $\boldsymbol{\mu}^w$ and $\boldsymbol{\sigma}^w$ and the window size w is by means of polynomial functions. When validating our approach, our findings show that $\boldsymbol{\mu}^w$ is a linear function of w, $\boldsymbol{\sigma}^w$ can be approximated by a polynomial of degree 3 and $\boldsymbol{p}^w$ by a polynomial of degree 2.

In this section, we present an example of polynomial fits for the Gaussian Mixture parameters in the case of the data set *Synthetic 2*. For this data set, Gaussian Mixtures with 2 components yield excellent fits for both the benign and malicious distributions. In Figures B.1, B.2 and B.3, we show how the Gaussian Mixture parameters can be approximated by polynomial curves, in the case of the means $\boldsymbol{\mu}^w$, the standard deviations $\boldsymbol{\sigma}^w$ and the weights of Gaussian mixture components, respectively. In the figures, we present the values of the parameters, computed at sampled window sizes, and the polynomial curves. We indicated with $(\cdot)_{B1}$ and $(\cdot)_{B2}$ the parameters relatively to the first and second benign Gaussian component, respectively. Similarly, $(\cdot)_{M1}$ and $(\cdot)_{M2}$ indicate the parameters of the malicious Gaussian components. As suggested by the results in Figures B.1, B.2 and B.3, we can closely approximate the Gaussian Mixtures parameters with polynomial curves. In Table B.1, we presents the average relative errors between the values of the Gaussian Mix-

ture parameters and the values provided by the fitting polynomial, measured at the sampled window sizes. In the case of the means $\boldsymbol{\mu}^w$, we achieve an average relative error lower than 2%. The error for the standard deviations $\boldsymbol{\sigma}^w$ is in overall lower than 2.3%. Finally, the components weights $\boldsymbol{p}^w$ show an average relative errors lower than 5%.

Par.	Avg. rel. error (%)	Par.	Avg. rel. error (%)	Par.	Avg. rel. error (%)
μ_{X1}	1.9	σ_{X1}	0.50	p_{X1}	4.9
μ_{X2}	0.11	σ_{X2}	0.50	p_{X2}	0.64
μ_{Y1}	0.85	σ_{Y1}	1.32	p_{Y1}	3.50
μ_{Y2}	0.69	σ_{Y2}	2.30	p_{Y2}	1.95

Table B.1: Average relative error (%) between the Gaussian Mixture parameters and the fitting polynomial functions (data set *Synthetic 2*).

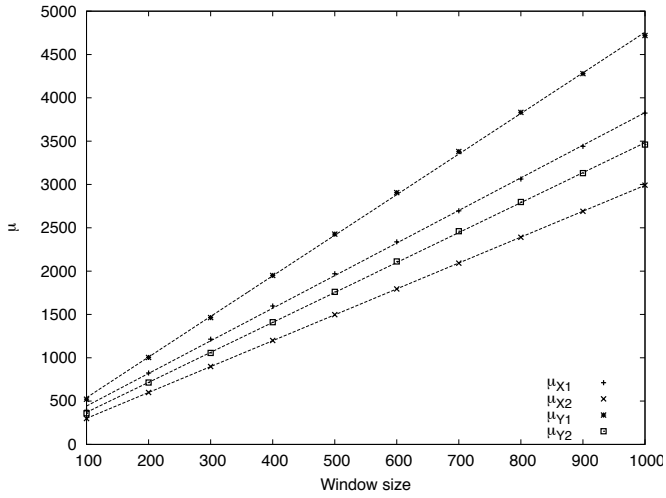


Figure B.1: Gaussian Mixture means for the data set *Synthetic 2* and polynomial fits.

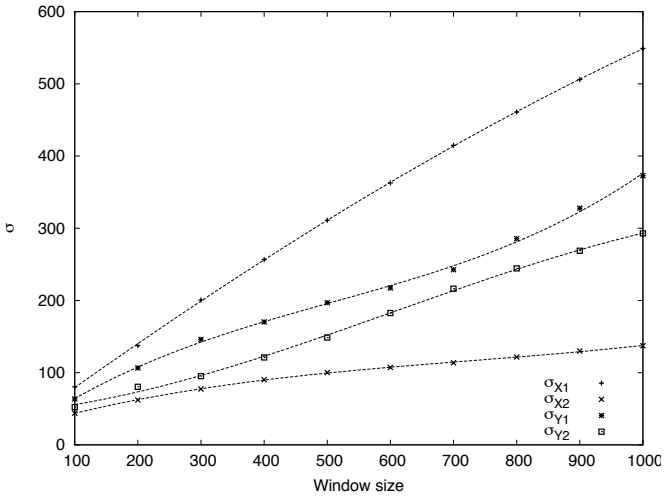


Figure B.2: Gaussian Mixture standard deviations for the data set *Synthetic 2* and polynomial fits.

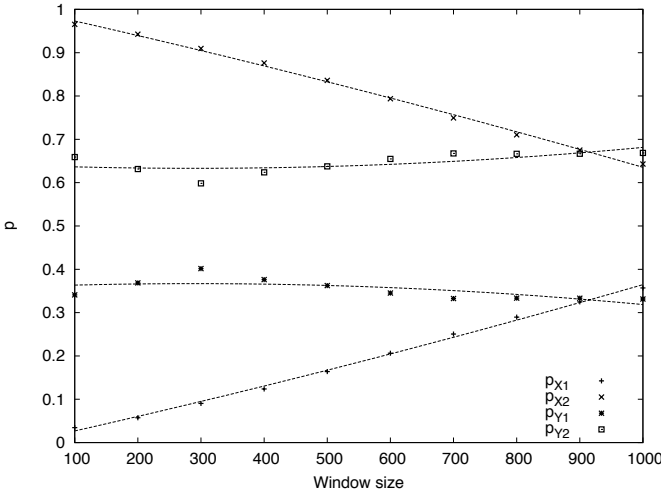


Figure B.3: Weight of the Gaussian Mixture components for the data set *Synthetic 2* and polynomial fits.

B.2 Optimization errors

This section reports results that validate the goodness of the Gaussian Mixture fits on which the optimization procedure in Chapter 7 is based. For details about our validation approach, see Section 7.6.2.

	Empirical			Analytical			Error (%)		
w	TN	TP	θ	TN	TP	θ	TN	TP	θ
100	0.84	0.73	347	0.86	0.76	352.10	2.20	2.92	1.47
200	0.82	0.82	669	0.84	0.81	673.86	2.54	0.94	0.73
300	0.85	0.83	1010	0.83	0.84	997.04	2.39	1.00	1.28
400	0.79	0.88	1307	0.82	0.84	1319.21	3.62	3.53	0.93
500	0.77	0.88	1626	0.80	0.85	1639.53	3.45	3.90	0.83
600	0.77	0.88	1959	0.78	0.85	1957.86	0.59	2.33	0.06
700	0.76	0.86	2288	0.75	0.86	2274.46	0.89	0.58	0.59
800	0.70	0.88	2570	0.72	0.87	2589.83	2.70	2.17	0.77
900	0.69	0.86	2902	0.69	0.87	2904.65	0.06	0.76	0.09
1000	0.65	0.87	3189	0.65	0.88	3219.65	1.02	0.95	0.96

Table B.2: Empirical vs. theoretical results for the optimization problem (*Synthetic 2)*.

The results in Tables B.2, B.3 and B.4 present the optimization errors for the additional data sets: *Synthetic 2*, *Original 1* and *Original 2*, respectively.

	Empirical			Analytical			Error (%)		
w	TN	TP	θ	TN	TP	θ	TN	TP	θ
100	0.53	0.81	263	0.47	0.80	259.35	10.53	0.48	1.39
200	0.56	0.83	532	0.51	0.85	529.23	8.70	2.59	0.52
300	0.59	0.83	804	0.55	0.86	805.74	6.17	3.16	0.22
400	0.60	0.84	1077	0.57	0.86	1084.93	3.92	2.44	0.74
500	0.62	0.85	1358	0.59	0.86	1365.92	3.57	1.94	0.58
600	0.62	0.86	1633	0.61	0.87	1648.31	1.50	1.34	0.94
700	0.63	0.86	1916	0.62	0.87	1931.79	1.09	1.71	0.82
800	0.62	0.86	2181	0.63	0.87	2216.15	1.96	1.26	1.61
900	0.66	0.84	2501	0.64	0.88	2501.17	2.97	5.28	0.01
1000	0.65	0.86	2758	0.65	0.89	2786.64	0.40	2.69	1.04

Table B.3: Empirical vs. theoretical results for the optimization problem (*Original 1*)

The results confirm the findings of Section 7.6.2, i.e., that the approach, based on fitted $\mathbb{G}\mathrm{m}$ random variables, closely approximates the results one

	Empirical			Analytical			Error (%)		
w	TN	TP	θ	TN	TP	θ	TN	TP	θ
100	0.84	0.73	347	0.65	0.55	318.42	22.42	25.55	8.24
200	0.82	0.82	669	0.58	0.67	611.21	29.49	18.52	8.64
300	0.85	0.83	1010	0.58	0.69	916.26	31.09	16.94	9.28
400	0.79	0.88	1307	0.60	0.69	1225.44	23.52	21.57	6.24
500	0.77	0.88	1626	0.62	0.68	1537.65	19.52	23.08	5.43
600	0.77	0.88	1959	0.64	0.67	1852.39	17.05	23.18	5.44
700	0.76	0.86	2288	0.66	0.66	2169.26	12.98	23.23	5.19
800	0.70	0.88	2570	0.68	0.66	2487.84	3.64	25.82	3.20
900	0.69	0.86	2902	0.69	0.65	2807.68	0.68	24.92	3.25
1000	0.65	0.87	3189	0.71	0.64	3128.34	9.41	25.99	1.90

Table B.4: Empirical vs. theoretical results for the optimization problem (*Original 2*).

would have found using empirical distributions. For the data set *Synthetic 2*, the relative error for TN, TP and θ is for all the considered cases lower than 4%. In the case of the original data sets, for *Original 1*, the relative error for TP and TN is lower than 4% for $w \geq 400$. For $w < 400$, we observe an error lower than 10%. For the data set *Original 2*, on the other hand, we observe a higher error in TP and TN; that for $w = 300$ is up to 30%. The error in approximating θ remains lower than 10%. Such higher errors are due to the polynomial fits used to approximate the parameters of X^w and Y^w, as functions of w.

APPENDIX C

Can we improve the performance of the detection system?

The focus of Chapter 7 has been on the optimization of detection system parameters. We therefore do not propose a deployed intrusion detection system, but we just introduce a simple system that would allow us to explain our ideas regarding optimization. The question may arise, however, whether such a system could be used in practice. In this appendix, we propose a possible extension to the system to improve its performance in case it would report directly to a human operator.

The limitation of the detection system is the high $\text{FP} = 1 - \text{TN}$ rate. For example, for $\alpha = \beta = 1$ and the data set *Synthetic 1*, we measured $\text{FP} = 0.05$. Since the system relies on a sliding window mechanism with a step of 1 second, the system checks the condition for malicious traffic $\ell_t^w \leq \theta$ with a rate of 1 time per second. Consequently, on the long term, the system will report 3 false positives per minute. From a user perspective, such a rate of false positive is likely to be too high.

A possibility to decrease the FP rate can be found in extending the current setup with an *alert management module*. The aim of such a module would be to filter spurious alerts and to notify the user only when the system gains sufficient certainty about the presence of an attack. The rationale behind such a module is that situations other than an attack might influence ℓ_t^w. An example is presented in Figure 7.4, where ℓ_t^w appears to increase regularly at 6 hours intervals. A first step towards an alert management system would be to investigate how the observation sequences labeled as P are distributed over time. We expect an attack to create a steady sequence of consecutive P observations, and a temporary anomaly to soon return to normality. If this condition is met, it could pave the way to a simple alert management system that would aggregate sequences of alerts into a single notification to the user. The proposed

solution is likely to decrease the rate of false positive perceived by the final user. However, alert filtering and grouping might have negative consequences as well. For example, it could introduce a delay in the notification of an attack, to be added to the already existing detection lag measured in Section 7.6.3. As often in the case of anomaly based intrusion detection, decisions regarding the desired performance constitute a trade-off. The relative importance of the performance measures is therefore to be decided according to the specific case study and the desired performance.

Bibliography

[1] M. Almgren, E. L. Barse, and E. Jonsson. Consolidation and evaluation of IDS taxonomies. In *Proc. of 8th Nordic Workshop on Secure IT systems (NordSec '03)*, 2003.

[2] N. Alon, N. Duffield, C. Lund, and M. Thorup. Estimating arbitrary subset sums with few probes. In *Proc. of the 24th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '05)*, pages 317–325, 2005.

[3] C. Andrieu and A. Doucet. Simulated Annealing for Maximum A Posteriori Parameter Estimation of Hidden Markov Models. *IEEE Transactions on Information Theory*, 46(3):994–1004, 2000.

[4] A.Wagner, T. Dübendorfer, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *Proc. of 2003 ACM workshop on Rapid Malcode (WORM '03)*, pages 34–41, 2003.

[5] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., 2000.

[6] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. In *Proc. of the 9th Int. Symp. on Recent Advances in Intrusion Detection (RAID '06)*, pages 165–184, 2006.

[7] R. Barbosa and A.Pras. Intrusion Detection in SCADA Networks. In *Proc. of 4th Int. Conf. on Autonomous Infrastructure, Management and Security (AIMS '10)*, pages 163–166, 2010.

[8] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW '01)*, pages 69–73, 2001.

[9] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[10] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[11] L. E. Baum and G. R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.

[12] H. Bhaskar, D. C. Hoyle, and S. Singh. Machine learning in bioinformatics: A brief survey and recommendations for practitioners. *Computers in Biology and Medicine*, 36(10):1104–1125, 2006.

[13] D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. PhD thesis, University of Twente, Enschede, June 2009.

[14] D. Bolzoni, E. Zambon, S. Etalle, and P. H. Hartel. Poseidon: a 2-tier anomaly-based network intrusion detection system. In *4th IEEE Int. Information Assurance Workshop (IWIA '06)*, pages 144–156, 2006.

[15] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proc. of the 6th ACM SIGCOMM Conf. on Internet Measurement (IMC '06)*, pages 159–164, 2006.

[16] P. Buchholz. An EM-Algorithm for MAP fitting from real traffic data. In *Computer Performance - Modelling Techniques and Tools*, pages 218–236, 2003.

[17] F. Camastra and A. Vinciarelli. *Markovian Models for Sequential Data*. Springer London, 2008.

[18] Cisco.com. Cisco IOS Flexible NetFlow White Paper. http://www.cisco.com, Sept. 2010.

[19] Cisco.com. Cisco IOS NetFlow Configuration Guide, Release 12.4. http://www.cisco.com, Sept. 2010.

[20] Citrix XenServer 5. http://www.citrix.com/, Sept. 2010.

[21] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), July 2008.

[22] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), July 2008.

[23] B. Clarke, E. Fokoué, and H. H. Zhang. Supervised learning: Partition methods. In *Principles and Theory for Data Mining and Machine Learning*. Springer New York, 2009.

[24] M. Collins and M. Reiter. Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *Proc. of 10th Int. Symposium on Recent Advances in Intrusion Detection (RAID '07)*, pages 276–295, 2007.

[25] CRAWDAD: Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.cs.dartmouth.edu/, Sept. 2010.

[26] D. Dagon, G. Gu, and C. Lee. A taxonomy of botnet structures. In *Botnet Detection*. Springer US, 2007.

[27] A. Dainotti, W. de Donato, A. Pescape, and P. Rossi. Classification of Network Traffic via Packet-Level Hidden Markov Models. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM 2008)*, pages 1–5, 2008.

[28] A. Dainotti, A. Pescapé, P. S. Rossi, F. Palmieri, and G. Ventre. Internet traffic modeling by means of Hidden Markov Models. *Computer Networks*, 52(14):2645–2662, 2008.

[29] A. M. Dan Pelleg. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proc. of the 17th Int. Conf. on Machine Learning*, pages 727–734, 2000.

[30] DatCat: Internet Measurement Data Catalog. http://imdc.datcat.org, Sept. 2010.

[31] W. de Bruijn, A. Slowinska, K. van Reeuwijk, T. Hruby, L. Xu, and H. Bos. SafeCard: A Gigabit IPS on the Network Card. In *Proc. of the 9th Int. Symp. on Recent Advances in Intrusion Detection (RAID '06)*, pages 311–330, 2006.

[32] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(9):805–822, 1999.

[33] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion detection systems. *Annales des Telecommunications*, 55(7–8):361–378, 2000.

[34] H. Debar and J. Viinikka. Intrusion detection: Introduction to intrusion detection and security information management. In *Foundations of Security Analysis and Design III*, pages 207–236, 2005.

[35] Diadem Firewall European Project. http://www.diadem-firewall.org, Sept. 2010.

[36] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proc. of the 11th ACM Conf. on Computer and Communications Security (CCS '04)*, pages 2–11, 2004.

[37] F. Dressler, W. Jaegers, and R. German. Flow-based Worm Detection using Correlated Honeypot Logs. In *Proc. of 15th GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS '07)*, pages 181–186, 2007.

[38] T. Dubendorfer and B. Plattner. Host behaviour based early detection of worm outbreaks in internet backbones. In *Proc. of the 14th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE '05)*, pages 166–171, 2005.

[39] T. Dübendorfer, A. Wagner, and B. Plattner. A framework for real-time worm attack detection and backbone monitoring. In *Proc.of 1st IEEE Int. Workshop on Critical Infrastructure Protection (IWCIP' 05)*, pages 3–12, Nov. 2005.

[40] N. Duffield, C. Lund, and M. Thorup. Flow sampling under hard resource constraints. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):85–96, 2004.

[41] 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.

[42] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[43] S. R. Eddy. Profile hidden markov models. *Bioinformatics Review*, 14(9):755–763, 1998.

[44] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, 21(3):270–313, 2003.

[45] J. Estévez-Tapiador, P. Garcia-Teodoro, and J. E. Díaz-Verdejo. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, 27(16):1569–1584, 2004.

[46] Z. Fadlullah, T. Taleb, N. Ansari, K. Hashimoto, Y. Y. Miyake, Y. Nemoto, and N.Kato. Combating Against Attacks on Encrypted Protocols. In *IEEE Int. Conf. on Communications (ICC '07).*, pages 1211–1216, 2007.

[47] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[48] A. Feldmann, J. Rexford, and R. Cáceres. Efficient policies for carrying web traffic over flow-switched networks. *IEEE/ACM Transactions on Networking*, 6(6):673–685, 1998.

[49] Fiber to the Home Council Europe. http://www.ftthcouncil.eu/, July 2010.

[50] G. A. Fink. *Markov Models for Pattern Recognition: From Theory to Applications*. Springer-Verlag New York, Inc., 2008.

[51] T. Fioreze. *Self-management of hybrid optical and packet switching networks*. PhD thesis, University of Twente, Feb. 2010.

[52] T. Fioreze, M. O. Wolbers, R. van de Meent, and A. Pras. Finding elephant flows for optical networks. In *Proc. of 10th IFIP/IEEE Int. Symposium on Integrated Network Management (IM '07)*, pages 627–640, 2007.

[53] J. Forney, G.D. The viterbi algorithm. *Proc. of the IEEE*, 61(3):268–278, March 1973.

[54] M. Fullmer. Flow-tools. http://www.splintered.net/sw/flow-tools/, Sept. 2010.

[55] M. Gao, K. Zhang, and J. Lu. Efficient packet matching for gigabit network intrusion detection using TCAMs. In *Proc. of 20th Int. Conf. on Advanced Information Networking and Applications (AINA'06)*, pages 249–254, 2006.

[56] Y. Gao, Z. Li, and Y. Chen. A DoS Resilient Flow-level Intrusion Detection Approach for High-speed Networks. In *Proc. of 26th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'06)*, page 39, 2006.

[57] M. Garuba, C. Liu, and D. Fraites. Intrusion techniques: Comparative study of network intrusion detection systems. In *Proc. of 5th Int. Conf. on Information Technology: New Generations (ITNG '08)*, pages 592–598, 2008.

[58] C. Gates, J. McNutt, J. Kadane, and M. Kellner. Scan Detection on Very Large Networks Using Logistic Regression Modeling. In *Proc. of 11th IEEE Symposium on Computers and Communications (ISCC '06)*, pages 402–408, 2006.

[59] GÉANT. Géant. http://www.geant.net, Sept. 2010.

[60] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of 17th USENIX Security Symposium (USENIX Security '08)*, pages 139–154, 2008.

[61] M. Gurcan, L. Boucheron, A. Can, A. Madabhushi, N. Rajpoot, and B. Yener. Histopathological image analysis: A review. *IEEE Reviews in Biomedical Engineering*, 2:147 –171, 2009.

[62] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE Journal on Selected Areas in Communications*, 9(2):203 –211, 1991.

[63] P. Haag. Nfdump. http://nfdump.sourceforge.net/, Sept. 2010.

[64] P. Haag. Nfsen: Netflow sensor. http://nfsen.sourceforge.net, Sept. 2010.

[65] J. Haines, R. Lippmann, D. Fried, M. Zissman, E. Tran, and S. Boswell. 1999 DARPA Intrusion Detection Evaluation: Design and Procedures. Technical Report TR 1062, MIT Lincoln Laboratory, 2001.

[66] L. R. Halme and R. K. Bauer. AINT misbehaving – A taxonomy of anti-intrusion techniques. In *Proc. of 18th NIST-NCSC National Information Systems Security Conference*, pages 163–172, 1995.

[67] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[68] G. He and J. C. Hou. An in-depth, analytical study of sampling techniques for self-similar internet traffic. In *Proc. of the 25th IEEE Int. Conf. on Distributed Computing Systems (ICDCS '05)*, pages 404–413, 2005.

[69] P. Horn. Autonomic computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com, 2001.

[70] J. D. Howard. *An analysis of security incidents on the Internet 1989-1995*. PhD thesis, Carnegie Mellon University - UMI Order No. GAX98-02539, 1998.

[71] V. Igure and R. Williams. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys & Tutorials*, 10(1):6–19, 2008.

[72] InMon Corporation. sflowtrend. http://www.inmon.com, Sept. 2010.

[73] International Telecommunication Union. Ict statistics. http://www.itu.int/ITU-D/ict/, Sept. 2010.

[74] Internet 2. Internet 2 research network. http://www.internet2.edu/, Sept. 2010.

[75] IsarNet Software Solutions. Isarflow. http://isarflow.com/, Sept. 2010.

[76] E. Izkue and E. Magaña. Sampling time-dependent parameters in high-speed network monitoring. In *Proc. of the ACM Int. workshop on Performance monitoring, measurement, and evaluation of heterogeneous wireless and wired networks (PM2HW2N '06)*, pages 13–17, 2006.

[77] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. of the 1st Conf. on First Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, pages 1–8, 2007.

[78] M.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. Hong. A flow-based method for abnormal network traffic detection. In *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS'04)*, pages 599–612, 2004.

[79] J. Kinable. Detection of network scan attacks using flow data. In *Proc. of the 8th Twente Student Conference on IT*, 2008.

[80] R. Koch and G. Dreo. Fast learning neural network intrusion detection system. In *Proc. of 3rd Int. Conf. on Autonomous Infrastructure, Management and Security (AIMS '09)*, pages 187–190, June 2009.

[81] C. Kruegel, F. Valeur, and G. Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*. Springer-Verlag Telos, 2004.

[82] H. Lai, S. Cai, H. Huang, J. Xie, and H. Li. A parallel intrusion detection system for high-speed networks. In *Proc. of the 2nd Int. Conf. Applied Cryptography and Network Security (ACNS'04)*, pages 439–451, May 2004.

[83] A. Lakhina, M. Crovella, and C. Diot. Characterization of network-wide anomalies in traffic flows. In *Proc. of 4th ACM SIGCOMM Conf. on Internet measurement (IMC '04)*, pages 201–206, 2004.

[84] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. of the Conf. on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04)*, pages 219–230, 2004.

[85] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *ACM SIGCOMM Computer Communication Review*, 35(4):217–228, 2005.

[86] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *Proc. of the joint Int. Conf. on Measurement aand modeling of computer systems (SIGMETRICS '04/Performance '04)*, pages 61–72, 2004.

[87] A. Lazarevic, V. Kumar, and J. Srivastava. Intrusion detection: A survey. In *Managing Cyber Threats*. Springer US, 2005.

[88] A. Lazarevic, A. Ozgur, L. Ertoz, J. Srivastava, and V. Kumar. A comparative study of anomaly detection schemes in network intrusion detection. In *Proc. of the 3rd SIAM Int. Conf. on Data Mining*, pages 1–14, 2003.

[89] M. Lee, T. Shon, K. Cho, M. Chung, J. Seo, and J. Moon. An Approach for Classifying Internet Worms Based on Temporal Behaviors and Packet Flows. In *Proc. of 3rd Int. Conf. on Intelligent Computing (ICIC '07)*, pages 646–655, 2007.

[90] W. Lee, C. Wang, and D. Dagon. *Botnet Detection. Countering the Largest Security Threat*. Springer US, 2008.

[91] P. Li, M. Salour, and X. Su. A survey of internet worm detection and containment. *IEEE Communications Surveys & Tutorials*, 10(1):20–35, 2008.

[92] Z. Li, Y. Gao, and Y. Chen. Towards a high-speed router-based anomaly/intrusion detection system. http://conferences.sigcomm.org/sigcomm/2005/poster-121.pdf, 2005.

[93] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S.E.Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *Proc. of the DARPA Information Survivability Conf. and Exposition (DISCEX '00)*, pages 12–26, 2000.

[94] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):597–595, 2000.

[95] W. J. Liu and J. Gong. Double sampling for flow measurement on high speed links. *Computer Networks*, 52(11):2221–2226, 2008.

[96] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer. Using machine learning techniques to identify botnet traffic. In *Proc. of the 31st IEEE Conf. on Local Computer Networks (LCN'06)*, pages 967–974, 2006.

[97] M. Mahoney and P. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *Proc. of the 6th Int. Symposium on Recent Advances in Intrusion Detection (RAID '03)*, pages 220–237, 2003.

[98] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is sampled data sufficient for anomaly detection? In *Proc. of the 6th ACM SIGCOMM Conf. on Internet Measurement (IMC '06)*, pages 165–176, 2006.

[99] V. Marinov and J. Schönwälder. Design of a stream-based IP flow record query language. In *Proc. of 20th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM '09)*, pages 15–28, 2009.

[100] J. McHugh. Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[101] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman. An overview of issues in testing intrusion detection systems. Technical Report NIST IR 7007, National Insititute of Standards and Technology, 2003.

[102] METROSEC. Metrology for security and quality of service. http://spiderman-2.laas.fr/METROSEC/, Sept. 2010.

[103] MOME project. http://www.ist-mome.org, Sept. 2010.

[104] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[105] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring Internet denial-of-service activity. *ACM Transactions on Computer Systems*, 24(2):115–139, 2006.

[106] C. Morariu, P. Racz, and B. Stiller. Design and Implementation of a Distributed Platform for Sharing IP Flow Records. In *Proc. of the 20th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM '09)*, 2009.

[107] C. Morariu and B. Stiller. DiCAP: Distributed Packet Capturing architecture for high-speed network links. In *Proc. of the 33rd IEEE Conf. on Local Computer Networks (LCN '08)*, Oct. 2008.

[108] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying Elephant Flows Through Periodically Sampled Packets. In *Proc. of the 4th ACM SIGCOMM Conf. on Internet Measurement (IMC '04)*, pages 115–120, 2004.

[109] B. Morin and L. Mé. Intrusion detection and virology: an analysis of differences, similarities and complementariness. *Journal in Computer Virology*, 3(1):39–49, 2007.

[110] G. Münz and G. Carle. Real-time Analysis of Flow Data for Network Attack Detection. In *Proc. of 10th IFIP/IEEE Int. Symposium on Integrated Network Management (IM'07)*, pages 100–108, 2007.

[111] T. Oetiker. MRTG - The Multi Router Traffic Grapher. http://oss.oetiker.ch/mrtg/, Sept. 2010.

[112] OpenSSH. http://www.openssh.com/, Sept. 2010.

[113] P. Owezarski. A database of anomalous traffic for assessing profile based IDS. In *Proc. of the 2nd Int. Workshop on Traffic Monitoring and Analysis (TMA '10)*, pages 59–72, 2010.

[114] P. Owezarski, J. Mazel, and Y. Labit. 0day anomaly detection made possible thanks to machine learning. In *Proc. of the 8th International Conference (WWIC '10)*, pages 327–338, May 2010.

[115] A. Patcha and J.-M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

[116] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[117] F. Pouget and M. Dacier. Honeypot-based forensics. In *Asia Pacific Information technology Security Conference (AusCERT '04)*, May 2004.

[118] proftp. http://www.proftpd.org/, Sept. 2010.

[119] PSAMP. Packet Sampling (PSAMP) working group. http://www.ietf.org/html.charters/psamp-charter.html, Sept. 2010.

[120] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer. Information Model for IP Flow Information Export. RFC 5102 (Proposed Standard), Jan. 2008.

[121] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational).

[122] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, 1989.

[123] M. Roesch. Snort, intrusion detection system. http://www.snort.org, Sept. 2010.

[124] B. Sangster, T. O'Connor, T. Cook, R. Fanelli, E. Dean, W. J. Adams, C. Morrell, and G. Conti. Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets. In *Proc. of the 2nd Workshop on Cyber Security Experimentation and Test (CSET '09)*, 2009.

[125] SANS Institute. Top-20 2007 Security Risks (2007 Annual Update). www.sans.org, Sept. 2010.

[126] J. Sansom and P. Thomson. Fitting hidden semi-Markov models to breakpoint rainfall data. *Journal of Applied Probability*, 38A:142–157, 2001.

[127] A. Scherrer, N. Larrieu, P. Owezarski, P. Borgnat, and P. Abry. Non-gaussian and long memory statistical characterizations for internet traffic with anomalies. *IEEE Transactions on Dependable and Secure Computing*, 4(1):56 –70, 2007.

[128] R. Schweller, L. Zhichun, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M.-Y. Kao, and G. Memik. Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications. In *Proc. of the 25th IEEE Int. Conf. on Computer Communications (INFOCOMM '06)*, pages 1–12, 2006.

[129] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[130] N. Sebe, I. Cohen, A. Garg, and T. S. Huang. *Machine Learning in Computer Vision (Computational Imaging and Vision)*. Springer-Verlag New York, Inc., 2005.

[131] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[132] Simpleweb trace repository. http://traces.simpleweb.org, Sept. 2010.

[133] Softflowd. http://www.mindrot.org/projects/softflowd/, Sept. 2010.

[134] Spamhaus.org. Effective spam filtering. http://www.spamhaus.org/whitepapers/effective_filtering.html, Sept. 2010.

[135] S. M. Specht and R. B. Lee. Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. In *Proc. of the ISCA 17th Int. Conf. on Parallel and Distributed Computing Systems (ISCA PDCS '04)*, pages 543–550, 2004.

[136] A. Sperotto, R. Sadre, and A. Pras. Anomaly Characterization in Flow-Based Traffic Time Series. In *Proc. of the 8th IEEE Int. Workshop on IP Operations and Management (IPOM '08)*, pages 15–27, 2008.

[137] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of ip flow-based intrusion detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[138] A. Sperotto, G. Vliek, R. Sadre, and A. Pras. Detecting spam at the network level. In *Proc. of the 15th Open European Summer School and IFIP TC6.6 Workshop (EUNICE '09)*, pages 208–216, 2009.

[139] Sprint.net. http://www.sprint.net, Sept. 2010.

[140] M. Stoecklin, J.-Y. L. Boudec, and A. Kind. A Two-Layered Anomaly Detection Technique Based on Multi-modal Flow Behavior Models. In *Proc. of 9th Int. Conf. on Passive and Active Measurement (PAM '08)*, pages 212–221, 2008.

[141] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan. Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. In *Proc. of the 2000 DARPA Information Survivability Conference and Exposition*, pages 130–144, 2000.

[142] W. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In *Botnet Detection: Countering the Largest Security Threat*. Springer New York, 2008.

[143] SURFnet. www.surfnet.nl, Sept. 2010.

[144] Symantec.com. The state of spam, a monthly report - june 2010. http://www.symantec.com/, Sept. 2010.

[145] T. Taleb, Z. M. Fadlullah, K. Hashimoto, Y. Nemoto, and N. Kato. Tracing back attacks against encrypted protocols. In *Proc. of the 2007 Int. Conf. on Wireless communications and mobile computing (IWCMC '07)*, pages 121–126, 2007.

[146] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proc. of the 2nd IEEE Int. Conf. on Computational intelligence for security and defense applications (CISDA '09)*, pages 53–58, 2009.

[147] The Cooperative Association for Internet Data Analysis. CAIDA DATA. http://www.caida.org/data, Sept. 2010.

[148] B. Trammell and E. Boschi. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103 (Proposed Standard), 2008.

[149] D. F. van Vliet. Turnover Poseidon: Incremental Learning in Clustering Methods for Anomaly based Intrusion Detection. In *Proc. of the 4th Twente Student Conference on IT*, 2006.

[150] W. van Wanrooij and A. Pras. Filtering Spam from Bad Neighborhoods. *International Journal of Network Management (accepted for publication)*, 2010.

[151] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis. Gnort: High Performance Network Intrusion Detection Using Graphics Processors. In *Proc. of the 11th Int. Symp. on Recent Advances in Intrusion Detection (RAID '08)*, pages 116–134, 2008.

[152] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[153] G. Vliek. Detecting spam at the network level. Master's thesis, Master Thesis in Computer Science, University of Twente, Feb. 2009.

[154] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast ip networks. In *14th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE '05)*, pages 172–177, June 2005.

[155] H. Wang, Y. Lin, Y. Jin, and S. Cheng. Easily-Implemented Adaptive Packet Sampling for High Speed Networks Flow Measurement. In *Proc. of 6th Int. Conf. on Computational Science (ICCS'06)*, pages 128–135, 2006.

[156] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proc. of 2003 ACM workshop on Rapid malcode (WORM'03)*, pages 11–18, 2003.

[157] C. V. Wright, F. Monrose, and G. M. Masson. HMM Profiles for Network Traffic Classification. In *Proc. of the Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC '04)*, pages 9–15, 2004.

[158] S. Zanero. Analyzing TCP traffic patterns using self organizing maps. In *Proc. of 13th Int. Conf. Image Analysis and Processing (ICIAP '05)*, pages 83–90, 2005.

[159] Q. Zhao, J. Xu, and A. Kumar. Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation. *IEEE Journal on Selected Areas in Communications*, 24(10):1840–1852, 2006.

[160] Z. Zhu, G. Lu, Y. Chen, Z. Fu, P. Roberts, and K. Han. Botnet research survey. In *32nd Annual IEEE Int. Computer Software and Applications (COMPSAC '08)*, pages 967–972, 2008.

[161] C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proc. of 17th USENIX Security Symposium (USENIX Security '08)*, pages 138–147, 2002.

[162] T. Zseby, T. Hirsch, and B. Claise. Packet sampling for flow accounting: Challenges and limitations. In *Proc. of 9th Int. Conf. on Passive and Active Measurement (PAM '08)*, pages 61–71, 2008.

List of Acronyms

FN False negative rate.
FP False positive rate.
TN True negative rate.
TP True positive rate.

ACI Autonomic Computing Initiative.

DNS Domain Name System.
DoS Denial of Service.
DTMC Discrete Time Markov Chain.

HMM Hidden Markov Model.

ICMP Internet Control Message Protocol.
IDS Intrusion Detection System.
IETF Internet Engineering Task Force.
IPFIX IP Flow Information eXport.
IRC Internet Relay Chat.
ITU International Telecommunication Union.

MRTG Multi Router Traffic Grapher.

NfSen Netflow Sensor.

PSAMP Packet Sampling.

ROC Receiver Operating Characteristics.

SSH Secure Shell Protocol.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.
UT University of Twente.

Index

About the author

Anna Sperotto was born in Belluno, Italy, on November 3rd, 1982. She studied Computer Science at the Ca' Foscari University, Venice, Italy, where she graduated in 2004 (Bachelor of Science) and in 2006 (Master of Science), both cum laude. Close to the conclusion of her studies, she spent 5 months in Manchester, UK, as part of the Erasmus program. In 2006, she enrolled as a PhD candidate in the Design and Analysis of Communication Systems (DACS) group at the University of Twente. There she conducted her research under the umbrella of the Centre for Telematics and Information Technology (CTIT), as part of the Next Generation Protection and Security of Content (PROSECCO) Project, and the European Network of Excellence on Management Solutions for Next Generation Networks (EMANICS). Her main topics of interest are Intrusion Detection, Self-Learning and Graph Theory.

A list of her publications in reverse chronological order:

- Sperotto, A. and Schaffrath, G. and Sadre, R. and Morariu, C. and Pras, A. and Stiller, B.*An Overview of IP Flow-based Intrusion Detection* IEEE Communications Surveys & Tutorials, 12 (3). pp. 343-356.
- Hofstede, R.J. and Sperotto, A. and Fioreze, T. and Pras, A. *The Network Data Handling War: MySQL vs. NfDump*. In: 16th EUNICE/IFIP WG 6.6 Workshop on Networked Services and Applications - Engineering, Control and Management, 20-30 June 2010, Trondheim, Norway. pp. 167-176. Lecture Notes in Computer Science 6164. Springer Verlag.
- Fioreze, T., Granville, L., Pras, A., Sperotto, A. and Sadre, R. *Self-Management of Hybrid Networks: Can We Trust NetFlow Data?* In: 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009), 1-5 Jun 2009, Long Island, New York, USA. pp. 577-584. IEEE Computer Society Press.
- Pras, A., Sadre, R., Sperotto, A., Fioreze, T., Hausheer, D. and Schoenwaelder, J. *Using NetFlow/IPFIX for Network Management*. Journal of Network and Systems Management, 17 (4).
- Sperotto, A., Sadre, R., de Boer, P.T. and Pras, A. *Hidden Markov Model modeling of SSH brute-force attacks*. In: Integrated Management of Systems, Services, Processes and People in IT, Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2009, Octo-

ber 27-28, 2009, Venice, Italy. pp. 164-176. Lecture Notes in Computer Science 5841/2009. Springer Verlag.

- Sperotto, A., Sadre, R., van Vliet, D.F. and Pras, A. *A Labeled Data Set For Flow-based Intrusion Detection*. In: IP Operations and Management, Proceedings of the 9th IEEE Intenational Workshop IPOM 2009, October 29-30, 2009, Venice, Italy. pp. 39-50. Lecture Notes in Computer Science 5843/2009. Springer Verlag.
- Sperotto, A., Vliek, G., Sadre, R. and Pras, A. *Detecting Spam at the Network Level*. In: Proceedings of the 15th Open European Summer School and IFIP TC6.6 Workshop, EUNICE 2009, 7-9 Sep 2009, Barcelona. pp. 208-216. Lecture Notes in Computer Science 5733. Springer Verlag.
- Sperotto, A., Sadre, R. and Pras, A. *Anomaly Characterization in Flow-Based Traffic Time Series* In: 8th IEEE International Workshop on IP Operations and Management, IPOM 2008, 22-26 September 2008, Samos, Greece. pp. 15-27. Lecture Notes in Computer Science 5275/2008. Springer Verlag.
- Sperotto, A. and van de Meent, R. *A Survey of the High-Speed Self-Learning Intrusion Detection Research Area* In: First International Conference on Autonomous Infrastructure, Management and Security, 21-22 Jun 2007, Oslo, Norway. pp. 196-199. Lecture Notes in Computer Science 4543. Springer Verlag.
- Sperotto, A. and Pelillo, M. *Szemeredi's Regularity Lemma and Its Applications to Pairwise Clustering and Segmentation* In: Energy Minimization Methods in Computer Vision and Pattern Recognition Energy Minimization Methods in Computer Vision and Pattern Recognition, 6th International Conference, EMMCVPR 2007, 27-29 Aug 2007, EZhou, Hubei, China. pp. 13-27. Lecture Notes in Computer Science 4679. Springer Verlag.

Acknowledgments

A colleague once asked me why I find it nice to hike in the mountains. Well, you might be under the sun, going uphill, with a rucksack on your shoulders, heavy shoes and sore legs, but when you get there, in a refuge enclosed by the Dolomites, the view is breathtaking and highly rewarding.

I believe that taking a PhD moves along the same lines: four years ago I chose a path and I've steadily progressed till where I'm standing now.

However, as for in the mountains, where you must never go alone so that there is always a companion to push you forward when things get hard, so in this PhD there have been many persons to walk with me.

In my first week of work, Tiago told me: "Aiko is not just your supervisor, he's your mentor". I kept this in mind during these years, and I think it is really true. Aiko, I want to thank you for the care you put in guiding me and in preparing me not just for the scientific aspects of this job, but also for all the rest. If I have to choose something to take along with me, it would be your question: "What did you learn?". This may seem easy to answer, but experience taught me that, often, it is not. Thus when you do find the answer you really gained some insight. And this works, not just for science.

Many thanks also to Boudewijn, especially in the final phase of the work. I witnessed an amazing team-work of you and Aiko to make all this happen.

Ramin, I think you deserve at least as many thanks as the number of times I walked into your office, and we both know these have been many. You have taken part in all the phases of my PhD, the more as well as the less enjoyable ones, and you have supported my work also when things where so fuzzy that I was not really sure if I was solving a problem or making one. My general feeling is that many crucial steps-forward happened when I was perched on the cabinet in your office. :)

I'm convinced research is a team-work: what I do not know, somebody else probably knows. This explain why I'm often on the search for collaborations. I want therefore to thank all the people I've collaborated with in these years. Among them, special thanks go to Pieter-Tjerk, who showed me how to look at a problem from all the different angles, and to Michel, who really took the time to familiarize with my research and to come up with new, inspiring, research directions.

I could not have done this work, not to say, gone through the dark dutch winters, if I would not have been surrounded by a group of great colleagues to keep my mood up. Many of them are my friends also outside the office, but my first thanks go to Marijn,

Desi and Yimeng, who shared the office with me for four years. Despite the fact that we have probably been the most chatty room at DACS, we are now all dealing with our theses, and more: we are on the move. It has been great to spend my time with you and see all of you moving forward to various stages of your lives. I truly think that one of our best ideas was to rearrange the office so that we could face each other, and I believe we will be in touch for much longer than our studies.

Many thanks also to Giovane, Rafael, Idilio ("Hey Apples!"), Anja, Tiago, Laura, Luiz Olavo, Edu, Anne, Stephan, Martijn and all the other persons that occasionally join our lively after-lunch coffees; and to Ivan, Fei, Damiano, Emmanuele, Stefano, Lilya, Vanessa, Frank, Rick, Lianne, Andreas, Jelena, David and to all the other nice persons I met in these years. A big thank certainly goes to Andrea, that after keeping me company in Skype for years, finally decided to come to the Netherlands (good move! ;-). Thanks again to Marijn and Karen, who accepted to be my paranymphs.

Un grazie speciale, che va ben oltre quello che posso scrivere qui, va alla mia famiglia: Mamma, Papà, Gabriella, Francesca e Nonna, per cominciare. A voi è capitato il compito più difficile, che è stato quello di lasciarmi andare. E anche se un po' qui mi prendono in giro perchè parlo ancora tutti i giorni in Skype con Papà e al telefono con tutti gli altri, non cambierei questa abitudine per nulla al mondo. Anche se sono un po' orso e non ve lo faccio sempre capire, ricordatevi che vi voglio bene e mi mancate più di quanto posso dire. Non avrei potuto volare cosí lontano se non sapessi che ho sempre un nido a cui tornare. E poi grazie a tutti gli altri: Pieranna, Andrea, Veronica, Gabriele, Maria e la nostra mascotte Chiara, Sara, Fabio, Elena, Marco, Riccardo, Silvia, zia Luisa e zio Vittorio e zia Anna Maria.

En hier, heel veel dank aan Chris, Ronnie, Maarten, Gusta, Niels, Joris, Tessa en Bart: jullie hebben mij met open armen ontvangen, met veel liefde en gezelligheid.

And finally, Wouter. For you, thanks for all. You stand by me since I met you, you know all the ups-and-downs that there have been over these years of research: you always put me back on my feet and encouraged me to go on. But life is not just work and I can only say that I'm so proud of what we are building together.