VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE ÚTOKŮ V SÍŤOVÉM PROVOZU

INTRUSION DETECTION IN NETWORK TRAFFIC

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. IVAN HOMOLIAK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. PETR HANÁČEK

BRNO 2016

Abstrakt

Tato práce se zabývá problematikou anomální detekce síťových útoků s využitím technik strojového učení. Nejdříve jsou prezentovány state-of-the-art datové kolekce určené pro ověření funkčnosti systémů detekce útoků a také práce, které používají statistickou analýzu a techniky strojového učení pro nalezení síťových útoků. V další části práce je prezentován návrh vlastní kolekce metrik nazývaných Advanced Security Network Metrics (ASNM), který je součástí konceptuálního automatického systému pro detekci průniků (AIPS). Dále jsou navrženy a diskutovány dva různé přístupy k obfuskaci – tunelování a modifikace síťových charakteristik – sloužících pro úpravu provádění útoků. Experimenty ukazují, že použité obfuskace jsou schopny předejít odhalení útoků pomocí klasifikátoru využívajícího metriky ASNM. Na druhé straně zahrnutí těchto obfuskací do trénovacího procesu klasifikátoru může zlepšit jeho detekční schopnosti. Práce také prezentuje alternativní pohled na obfuskační techniky modifikující síťové charakteristiky a demonstruje jejich použití jako aproximaci síťového normalizéru založenou na vhodných trénovacích datech.

Abstract

The thesis deals with anomaly based network intrusion detection which utilize machine learning approaches. First, state-of-the-art datasets intended for evaluation of intrusion detection systems are described as well as the related works employing statistical analysis and machine learning techniques for network intrusion detection. In the next part, original feature set, Advanced Security Network Metrics (ASNM) is presented, which is part of conceptual automated network intrusion detection system, AIPS. Then, tunneling obfuscation techniques as well as non-payload-based ones are proposed to apply as modifications of network attack execution. Experiments reveal that utilized obfuscations are able to avoid attack detection by supervised classifier using ASNM features, and their utilization can strengthen the detection performance of the classifier by including them into the training process of the classifier. The work also presents an alternative view on the non-payload-based obfuscation techniques, and demonstrates how they may be employed as a training data driven approximation of network traffic normalizer.

Klíčová slova

Behaviorální analýza síťových anomálií, buffer overflow útoky, tunelovací obfuskace, obfuskace modifikující síťové charakteristiky, strojové učení s učitelem.

Keywords

Network behavioral anomaly detection, buffer overflow attacks, tunneling obfuscations, non-payload-based obfuscations, supervised machine learning.

Citace

Ivan Homoliak: Intrusion Detection in Network Traffic, disertační práce, Brno, FIT VUT v Brně, 2016

Detekce Útoků v Síťovém Provozu

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Petra Hanáčka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Ing. Ivan Homoliak
August 12, 2016

Poděkování

Chci poděkovat Doc. Dr. Ing. Petrovi Hanáčkovi za vedení této práce, odbornou pomoc a konzultace s ní spojené. Obdobně bych chtěl poděkovat také svému školiteli specialistovi Mgr. Kamilovi Malinkovi, Ph.D. za některé užitečné rady. Dále bych chtěl poděkovat:

- Matěji Grégrovi za vstřícnost a pomoc při získávání anonymizovaných vzorků reálného síťového provozu,
- Danielu Ovšonkovi za spolupráci při experimentech s tunelovací obfuskací,
- Martinu Teknösovi za spolupráci při experimentech s modifikací síťových charakteristik,
- Marošovi Barabasovi za užitečné rady a recenze některých částí práce,
- Petru Veigendovi za užitečné rady a také pomoc s jazykovou stránkou práce,
- Dominiku Breitenbacherovi za pomoc s jazykovou a typografickou stránkou práce,
- Stanislavu Kováči za pomoc s jazykovou stránkou práce,
- Zdeněku Vašíčkovi za pomoc s typografickou stránkou práce,
- týmu VirusTotal za bezplatné poskytnutí přístupu k placenému API,
- a také své rodině a některým přátelům za cennou psychickou podporu.

Ne vše je v skutku takové, jak se na první pohled může zdáti.
Not everything is in fact as it appears at the first glance.

– Autor neznámý / Unknown author

Contents

Chapter 1

Introduction

These days, network security is one of the most critical things that the corporations and organizations have to handle. More and more attacks are executed into the professionally secured corporate networks or into private networks and stations. New threats are emerging everyday and lots of old threats remain current. The impact of a successfully performed network attack can be very crucial, either in commercial or personal network environments. Network attack can cause huge financial and economical loss in the most of cases.

Many of actual network attacks detection systems are based on well known signatures of known network attack types. Signature based systems are very fast and they can provide very low false positive rate which favors them to be widely used. On the other hand, these systems cannot detect zero-day attacks and they may also have high false negative rate – representing successful evasion of target machines. Signature based systems are often bypassed by application layer obfustation techniques like substitutions, code transpositions, code compression, inserting of meaningless instructions, etc.

If the signature based system is deployed as a network device analyzing network traffic, then it is often facing the fact that useful content of the network flow is encrypted, and therefore it cannot detect potential threats. Because of this, there is a considerable interest in development of a novel detection methods. These methods are based on new features for description of network flow with intention to early identify emerging security incidents, rapidly detect infections within internal networks and instantaneously prevent forming attacks. Discussed methods are part of Network Behavioral Anomaly Detection (NBAD) intended for intrusion detection, which may be also referred to as Anomaly Detection Systems (ADS). NBAD tries to approach detection of network attacks by utilizing packets' headers and communication behavior, not the content of packets. NBAD has lot of advantages, but on the other hand, some of these systems based on flow feature extraction do not use enough quality and quantity features to perform classification decision. The decisions made by these systems are in many cases coarse-grained, and thus often not correct. Therefore, further research is needed to improve the detection capabilities of the NBAD systems.

1.1 Motivation

The most of the previous and current research on the field of network attacks detection and traffic classification utilizing machine learning techniques presents various methods pursued in their performance in contrast of lacking operational deployment of such systems [172].

These methods employ diverse machine learning algorithms and data mining techniques to examine and analyze some dataset without concerning how such dataset was collected and network traffic simulation performed. There are several reasons why such systems are not deployed in the real traffic environment (over-learned classifiers, polymorphism and metamorphism of the network malware, different characteristics of the zero-day attacks, etc.). All such systems can suffer from occurrence of some misclassified data which has very high costs. A legitimate communication may be classified as an attack in false positive case, and therefore a denial of legitimate service usage occurs. A malicious communication may be classified as legitimate one in false negative case, and thus an evasion occurs. Both cases are very critical. Therefore, general guidelines prefer to consider NBAD alerts with only some level of uncertainty, and rather to inspect detected threats by human expert.

Basic principles of NBAD open possibilities of an attacker to evade NBAD detection by using various obfuscation techniques. The motivation of my work is to learn the NBAD detection engine of being aware of various obfuscated network attacks' behavior, and thus moving forward the machine learning based network anomaly intrusion detection to its maximal potential use.

Aforementioned thoughts and facts bring me to consideration of making malicious traffic simulation as divergent as possible. I suppose to use diverse obfuscation techniques of network attacks, causing classifiers trained without knowledge of these modifications, be unable to provide acceptable response. Then, providing of supposed data modifications to a training phase of classifiers can strengthen their detection capabilities. In the other words, I want to prepare NBAD classifiers for occurrence of various obfuscation techniques which an attacker may exploit for evasion of NBAD.

1.2 Goal of the Thesis

The main goal of my thesis is to evaluate and improve properties of previously designed NBAD system called Automated Intrusion Prevention System (AIPS) and its detection features, respectively. The method of evaluation will not only include experiments with designed network features and optimization of machine learning techniques, but also, will take advantage of using various network layer based obfuscation techniques. The latter method approaches improvement of AIPS performance by aiming on divergence of training data, which is accomplished by obfuscation techniques. These obfuscation techniques are utilized in order to strengthen precision and recall of detection models of AIPS. The obfuscation techniques which I use in my research are based on non-payload-based network layer modifications of connection-oriented communications as well as tunneling network traffic through different protocol.

The next goal of the thesis is to provide an alternative view on the utilization of non-payload-based obfuscations. I will show how network traffic normalizer, which is usually prone to state holding and CPU overload attacks, can be omitted in considered NBAD system thanks to enough training data divergence achieved by proposed obfuscations. Thus, non-payload-based network traffic obfuscation techniques will be utilized as technique denoted as training data driven approximation of a network traffic normalizer, resulting into eliminating of current issues inherent to network traffic normalization.

1.3 Structure of the Thesis

Chapter 2 will discuss traditional and actual principles of network attacks detection and will present the taxonomy of intrusion detection methods. Chapter 3 will formally define IDS prediction task and then state definitions of prediction approaches employed through the thesis. It will also define performance measures utilized for binominal and multi-class prediction as well as the proposed way of their estimation in cross validation method. Chapter 4 will concern machine learning based intrusion detection systems as well as network traffic classification ones. It will also describe datasets utilized for evaluation of Intrusion Detection Systems (IDS), and finally provide overview of obfuscation and evasion approaches in ADS and IDS, supplemented by several prevention techniques against obfuscations and evasions. Chapter 5 will describe full concept of statistical and behavioral based system for anomaly intrusion detection – AIPS – which was developed under a project of Faculty of Information Technology in Brno. Also, my particular contributions, primarily residing in the design and the definition of the system's features denoted as Advanced Security Network Metrics (ASNM), will be discussed. The chapter will describe extraction process of ASNM and also formally define mathematical methods utilized for computation of the ASNM features. Chapter 6 will aim at the performance evaluation of the ASNM features, which is also result of my contributions to the project. Independently of mentioned project, the chapter will propose performance improvement of the detection by ASNM features and a supervised classifier, resulting into two categories of proposed obfuscation techniques. The first category of obfuscation techniques – called tunneling obfuscation – will be described and evaluated in Chapter 7, while the second category – called non-payload-based obfuscations – will be closely specified and evaluated in Chaper 8. The last part – Chapter 9 – will conclude the thesis and summarize the research contributions.

Chapter 2

Taxonomy of Network Intrusion Detection

According to [12], there are two basic types of network attacks detection. The first type represents signature matching and the second type is based on anomaly detection. Another newer kind of network attack detection are honeypot systems. The principles of each category together with their categorization are discussed in the current chapter. The most of further information about signature and anomaly based systems is gained from the technical report [12] and from the paper [103].

2.1 Signature Based Detection

The decision is performed according to a model of malicious process and attack's trace on a compromised system in the area of signature based detection. Signature based detectors are trying to detect the presence of malicious activity, regardless of the normal communication in the background. This fact puts strict requirements on the model of intrusion detection [12], which has to contain very precisely defined details of signatures. All the systems in this category are explicitly programmed and there is no learning at run time. These systems are divided into several categories: *state model*, *expert*, *chain-based* and *rule-based*.

Systems with State Model

These systems represent an attack as a set of states, where each of them must occur to declare an attack occurence. These states are represented by time series models.

This category can be further divided into two subcategories – state transition based and systems based on Petri Nets [120]. The states in the state transition systems have to form a sequence, which when executed, attack happens. The second subclass – Petri Nets systems can contain general structure of the state transition graph.

Expert Systems

Expert systems utilize predefined rules which represents attacks signatures. These systems very often need to meet a set of rules in order to declare attack detection. The problem with these systems is a long execution time of analysis due to the frequent usage of executive mechanisms [12].

String Matching Systems

Systems based on string matching operate by very simple substrings comparison in the text being transmitted between systems. This method is not flexible but easy to understand and can use the existing very effective algorithms for text strings searching.

Rule Based Systems

Rule based systems are similar to some simple expert systems, but they are faster. Rules have declarative character. Often it is necessary to optimize the various rules, for example transformation of complex expression into a tree representation, which is effectively evaluated with elimination of certain subterms [64].

2.2 Anomaly Based Detection

Another term which describes network anomaly based system is Network Behavior Anomaly Detection (NBAD) [103]. NBAD is an integral part of network behavior analysis (NBA), which offers security in addition to that provided by traditional anti-threat applications such as firewalls, intrusion detection systems, anti-virus software and spyware-detection software. NBAD often requires several sensors to create a good snapshot of a network and requires benchmarking and baselining to determine the nominal amount of a segment's traffic [198].

Anomaly detection does not monitor known characteristics of attacks which are signs of their occurrence, but it observes abnormalities in network traffic. Construction of such a detector needs to form a conclusion – what is the normal behavior of the observed body and the abnormal one. The paper [103] suggest classification of anomaly detection approaches based on employed methods into two groups: *learning-based* methods and *specification-based* methods. The technical report [12] proposes classification of anomaly detection systems into two categories: *self-learning systems* and *programmable systems*. Both of categorizations will be closely described in the following text.

2.2.1 Taxonomy by Axelsson

This taxonomy of anomaly based intrusion detection systems was proposed in 2000 by Stefan Axelsson in technical report "Intrusion detection systems: A survey and taxonomy" [12]. It is based on the way, how intrusion detection systems obtain information neccessary to perform detection or decision.

Self-learning Systems

The task of these systems is to learn from long-term observation of network traffic and to build a model of the operation and ongoing processes in the network traffic. We divide these systems according to their approach to time repeatability. They can either take it into account or omit it. The first group (repeatable) uses techniques like Markov decision processes [111] or neural networks [175], which can more trustworthy represent real communication [12].

The second group (unrepeatible) uses stochastic models. They can operate in the mode when they follow the normal network traffic and create model rules of normal communication or they can operate in detection mode, which monitor the compliance with these rules. The system will be alarmed if there occur weak consensus evaluated by weighting. Another

way they operate is in collecting of descriptive statistics of network traffic into profiles and building vectors of distances for examined traffic and profile. The system will be alarmed when the distance is large enough.

Programmable Systems

This variant require the programmer to teach the detection system certain anomaly events. Thus the user provides expert knowledge to the system. These systems are divided into those based on descriptive statistics and those implicitly denial. The systems based on descriptive statistics collect parameters and system information like the number of failed login attempts, the number of network connections, the number of commands with an error return code. The main idea of implicitly denial system is to put the system's state circumstances into harmless mode and consider all deviations from this mode as attacks. There is possible to see the correspondence with the implicitly deny security policy [12].

2.2.2 Taxonomy by Lim

This taxonomy of anomaly based intrusion detection systems was proposed in 2008 by Shu Yun Lim et al. in article "Network Anomaly Detection System: The State of Art of Network Behavioral Analysis" [103]. The authors mention anomaly detection algorithms are quite diverse in nature, and thus may fit into more than one category. Their classification attempts to find the most suitable category for all existing anomaly detection algorithms up to 2008. They also expressed a sentiment that their taxonomy is expected to continuously evolve before achieving a solid maturity for its implementation. All the approaches that concern the behavioral model construction are presented in Figure 2.1.

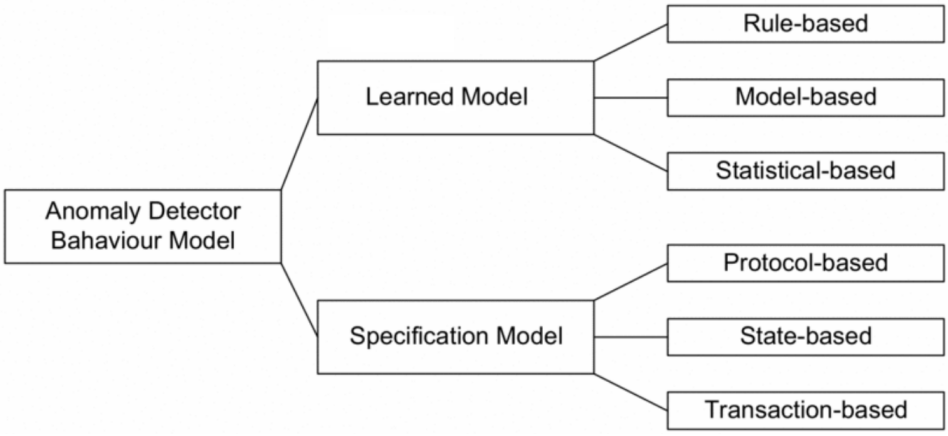

Figure 2.1: Taxonomy of anomaly detector's behavioral model [103]

2.2.2.1 Learned Model

In this category, an anomaly detection model must be trained on the specific network or host which is monitored. The training of the model utilizes observed behavior of the particular system and then, machine learning techniques are used to create a profile of such normal behaviors. In the stage of creating an anomaly detection model, **rule-based, model-based, and statistical-based** approaches have been adopted to create the baseline profiles.

Rule-based

Rule-based systems which are used in anomaly detection characterize the normal behavior of users, networks and/or computer systems by a set of rules. These predefined rules typically look for the high-level state change patterns observed in the audit data compared to predefined state change of penetration scenarios. An example of this subcategory is an expert system which represents the system's state by knowledge base (consisting of a fact base) and a rule base.

Model-based

Model-based behavioral intrusion detection attempts to model intrusions at a higher level of abstraction alike rule-based ones which simply attempts to bind audit records to expert rules. A model-based anomaly detector tries to restrict execution to a pre-computed model of expected behavior.

Researchers often use different types of models to characterize the normal behavior of the monitored system. In the model-based approaches, anomalies are detected as deviations from the model that represents the normal behavior. Examples of this sub-category can be considered data mining, neural networks, pattern matching, etc., utilized to build predictive models.

Statistical-based

Statistical-based anomaly detection was for the first time discussed by Denning and Neumann in 1985 [51]. The anomaly detector observes the activity of subjects and generates profiles representing their behavior. These profiles are designed to use a little memory to store their internal state, and to be efficient in updating because every profile may potentially be updated for every audit record. With processing of audit records, the system periodically generates a quantitative measure of the normal profile.

The well-known techniques of statistics can often be applied for this category as well. For example, data points that lie beyond a multiple of the standard deviation on either side of the mean might be considered anomalous. Another example can be the integral value of the absolute difference of two functions over time which might also be used as an indicator of the deviation of one function with respect to the other. There are more statistical techniques like Bayesian statistics, Co-variance matrices and Chi-square statistics [200] which might be used for the profiling of an anomaly detection. Nevertheless, statistical approaches have their disadvantages as statistical measures are insensitive to the order of occurrence of events. It is also difficult to determine a threshold above which an anomaly should be considered intrusive. Imprecise thresholds usually lead to either false positive or false negative cases.

2.2.2.2 Specification Model

The specification approach depends more on human observation and expertise than on mathematics. It was for the first time utilized by C. Ko et al. [91] in 1997. It uses a logic-based description of expected behavior to construct a base model. This specification-based anomaly detection system monitors multiple system elements, ranging from application to network traffic. The current category contains three approaches to specification of anomaly detection model: **protocol-based, state-based** and **transaction-based specifications**.

Protocol-based

Protocol anomaly detectors build models of TCP/IP protocols using their specifications [101]. Disadvantage of previously discussed statistical anomaly detection resides in inherent inability to create a model of normal network traffic statistics. Anomaly detection on the level of protocol is much easier because protocols are well defined and a model representing the normal behavior can be created with higher accuracy.

E. Lemonnier in his paper [101] uses protocol anomaly filter in order to analyze specific protocol and then model the normal usage of a specific protocol. This technique can be considered as a filter looking for protocol misuse. The protocols are usually described by rules which are specified in their official descriptions like RFCs or can be observed from practical usage of some proprietary protocols. Therefore, any use of a protocol outside of defined area can be considered as a protocol anomaly. In comparison with signature filters, the protocol-based filters do not need update of rules and has as long lifetime as the protocol they are modeling.

State-based

Many protocol anomaly detectors are built as state machines [165]. The reason of this fact is that all connection oriented protocols have a state, therefore, certain events must take place at certain time. Each state corresponds to a particular situation, e.g. a server waiting for a response from client. The transitions between the states describe the correct and expected changes between states.

As exemplar representative of this category is work of Z. Shan et al. [205] who use network state based model approach to describe intrusion. The work uses finite automata theory for state modeling and the authors are able to detect unknown attacks as well as known ones.

Transaction-based

Transactions are a well known concept which originates from the field of database management systems and now, are widely applied in other environments such as distributed systems. The definition of transaction is utilized to specify desired action and sequences of actions. The definition makes the transaction an integral part of security policies.

The first time when transactional based approach was used, authors R. Buschkes et al. [26] make detection of anomalies based on the definition of correct transactional behavior of normal traffic. In contrast to classical database and other transactional systems R. Buschkes et al. do not enforce the distinct transactions to be executed according to the ACID properties (Atomicity, Consistency, Isolation, Durability). Instead, they monitor the system only for any potential conflicts.

2.3 Honeypots Based Detection

The information value obtainable from the network traffic by IDS/IPS systems is still increasing. The number of used application layer protocols of TCP/IP[1] model, which use encryption, is increasing too. IDS and IPS systems also suffer by high false positive rate, which reduce the potential of their usage [12].

Honeypot is a resource, which should be attacked and compromised. The honeypot is monitored in detail. In other words, the value of the honeypot is in its unauthorized use, which can generate large amount of information. Monitoring data entering and leaving the honeypot allows us to obtain information that is not provided by IDS/IPS. For example, keystrokes during an interactive session can be monitored even with active encryption to protect network traffic. To detect malicious behavior of known attacks it is necessary to know its signatures for IDS/IPS and they often fail in detecting existing attacks that changed their behavior. Another advantage of honeypots is the fact, that they are able to detect the vulnerabilities that have not been discovered yet.

Since the honeypot has no production value, any attempt to establish a connection to it is considered suspicious. Forensic analysis of data collected by honeypots is less prone to false positive unlike IDS/IPS, because most of the data collected by these systems are directly collected during real attacks. Another advantage of honeypots is that they can run on any operating system and any services.

High-interactive honeypot represents a real system, that the attacker can interact with. In contrast of it stands low-interactive honeypot which simulates only some parts of the system (eg. network model). High-interactive honeypot can be compromised and may allow an attacker to gain full access to its system and use the honeypot system to perform other network attacks. Low-interactive honeypots simulate only services which cannot be used to gain full access to the system. Low-interactive honeypots are more limited, but their advantage represents obtaining information on the higher layer (eg. learn about network scanning activities or worms). They can also be used to analyze the behavior of spammers. Neither of these approaches is superior to another. Each has its advantages and disadvantages.

We distinguish between the physical and virtual honeypots. Physical honeypot is a real machine on the network that has its own IP address. Virtual honeypot is simulated using a different machine.

The important factor in obtaining information about network attacks and network scanning is the number of deployed honeypots, which affects the quality and quantity of collected data. A good example is monitoring of worms activities based on HTTP[2]. These worms can be identified only in the case of complete TCP 3-way handshake and transmission of attacks payload. Therefore the honeypot, which does not emulate vulnerable web service, cannot collect this information. The more honeypots are deployed, the more likely will be one of them attacked by worms [138]. The following information concerning the categorization of honeypots and their descriptions will be taken from [138].

2.3.1 High-interactive Honeypots

High-interactive honeypot is a conventional computer system, for example computer, router or switch. This system is not dedicated for any conventional use. It also does not have any regular active users. Therefore it has no unusual processes and does not generate

[1] URL: https://www.ietf.org/rfc/rfc793.txt.
[2] URL: http://tools.ietf.org/html/rfc2616.

any network traffic (except vulnerable services and the operations associated with them). These assumptions are taken into account during attacks detection. Every interaction with a honeypot is suspicious and potentially harmful. Therefore, all network traffic incoming to honeypots is logged to files for later analysis.

It is possible to connect several honeypots into the network called honeynet. Honeynet usually consists of several different types of honeypots, which have different platforms and operating systems. This distribution allows independent and simultaneous collection of data on different types of attacks. Therefore, it is usually possible to study deeply the information about the attacks and obtain qualitative results about the behavior of attackers. One of the key elements of honeynet is honeywall. Honeywall is a network device operating at the link layer model of TCP/IP model, which bridges honeywall from the rest of the network. This device reduces the risk to other stations in the network and allows to capture data for analysis. All inbound and outbound traffic must pass a *honeywall*. The information is collected using various methods like passive monitoring tools, IDS notifications, firewalls logs, etc. Attackers' activities are managed at the network layer of the TCP/IP model and all outgoing connections are filtered using IPS and connection limiters.

One of the problems of high-interactive honeypots are higher maintenance demands. Honeypot should be heavily monitored and the information about the activity inside should always be available. A full analysis of realized incidents can take hours or even days until the moment, we can fully understand what the attacker intended to achieve. High-interactive honeypots can be fully compromised because they run on real operating systems. The attacker can interact with real services and real system, which allows us to capture extensive information related to posed threats. It is possible to capture exploit content[3], monitor keystrokes, detect used tools and uncover attackers motives. One of disadvantages of high-interactive honeypots is the fact, an attacker can potentially control the entire operating system and through it can cause potential risk of production systems in the network. The main disadvantage of high-interactive honeypots are their demands on computing resources, problems with scaling and deploying in the real network conditions [12].

2.3.2 Low-interactive Honeypots

Low-interactive honeypots emulate services, network model or other aspects of the real system in contrast to the previous category. They allow an attacker limited interaction with the target system and they help to obtain quantitative information about the attacks. For example, an emulated HTTP server may implement only a subset of the HTTP specification. The interaction with the attacker is implemented at sufficient level to confuse the attacker or an automated tool performing attack, which may look for a specific file needed to compromise the system. The advantage of low-interactive honeypots is their simple maintenance. To start up operation of low-interactive honeypots is enough to run its application. Collecting of data is automatic. These data may represents information about network worm propagations or scans of open services. The installation of this type of honeypot is generally easier: we need only to install and configure the tool.

Low-interactive honeypots can be primarily used for the collection of statistical data and for gathering of high-level information about the patterns used in the attacks (signatures). They can be used as an IDS – as they provide early warning of the attacks presence. Even they can be used for decoying attackers from production machinery of a corporate network. Their further use is dedicated to detect worms and to learn about ongoing attacks. High-

[3]Code exploiting vulnerabilities of the service for malicious purposes.

interactive honeypots as well as low-interactive honeypots can be used to create a network called honeynet.

The attacker is unable to gain full control over the system, since he does not interact with the real service. Low-interactive honeypots provide a fully controllable and monitorable environment. Therefore, any risk associated with the possibility of breaking into a system where honeypot run is decreased [12].

2.3.3 Physical Honeypots

Another possible division of honeypots distinguishes between physical and virtual honeypots. A physical honeypot is running on a physical machine. Honeypot running on a physical machine often implies a high interaction and thus it allows complete compromise of target system. This category of honeypots is typically difficult to install and maintain. Deploying of large honeypots number in the corporate network is exhaustive to physical resources, therefore it is preferable to introduce virtual honeypots [12].

2.3.4 Virtual Honeypots

The main advantage of using virtual honeypot type is scalability and ease of maintenance. There can coexists several honeypots on the same physical machine. This approach is lightweighted in comparation with physical honeypots. Instead of physical machine deployment acting like a honeypot, multiple machines, that represent individual honeypots, can be deployed. This sollution leads to easier maintainability and lower physical demands. Usually, there can be used virtualization tools like VMware[4], USL[5], VirtualBox[6], QEMU[7] The main aspect which is necessary to take into account is the fact, that a virtual honeypot is simulated by another machine responsible for forwarding network traffic intended for honeypot and traffic generated by the honeypot [12].

2.3.5 Argos

Argos is a new tool that falls under the category of virtual high-interactive shadow honeypots [8]. It has been developed by researchers at the Vrije Universiteit Amsterdam in the Netherlands. This tool is capable of detecting zero-day attacks. It uses a technique called dynamic taint analysis [122] to monitor honeypots. At first, all data received over the network are marked as tainted. The use of these marked data is then tracked within the memory of honeypot's system. Once the flow of executing program is influenced by tainted data (eg. using instruction JUMP), Argos will detect it and generates an imprint of memory representing the current state of memory during attack.

Compared with other virtual honeypots, Argos represents a slightly different approach. Instead of pure implementation of the virtual machine, Argos detaily monitors it and tries to detect the moment in time when exploit has successfully compromised virtual machine.

Argos is designed over QEMU emulator which uses technique of dynamic translation. However, the performance is lower comparing to other virtualization tools. Argos adds the possibility of dynamic tainted data analysis to the QEMU emulator [12].

[4]URL: http://www.vmware.com/.
[5]URL: http://user-mode-linux.sourceforge.net/.
[6]URL: https://www.virtualbox.org/.
[7]URL: http://wiki.qemu.org/Main_Page.

Dynamic Analysis of Tainted Data

Dynamic analysis of tainted data is the core of Argos. This technique is based on the observation of the program's flow and its influence by attacker. This can be achieved by underrun or overflow of program's buffer. The attacker sends a specially-formed text string that overwrites sensitive memory locations and these data will affect the program's flow – for example a jump to memory location, where the attackers program is located. At that moment, dynamic analysis starts, which detects illegal use of tainted data. All external inputs of the program are marked as tainted and during the dynamic analysis are tainted variables monitored and controlled [12].

The main idea of dynamic tainted analysis method described in [204] is to examine manipulations with tainted values by malicious code. Tainted analysis uses tainting of sensitive data and return values from critical calls to track the propagation and see whether something malicious is happening. Each byte of returned value from system calls is given a unique label and all tainted memory is logged to keep track of assigning labels for memory location at a specific time.

Chapter 3

Data Mining in Intrusion Detection

Considering the taxonomy of network intrusion detection techniques described in Chapter 2, we decided to aim at specific category of network intrusion detection. According to taxonomy by Lim [103] (see Section 2.2.2), our research is relevant to *model-based* behavioral network intrusion detection and it utilizes data mining techniques. From the perspective of Axelsson's taxonomy [12] (see Section 2.2.1), our research belongs to group of *programmable systems* which require expert knowledge (ground truth) provided by the user or by another resource of expert knowledge. In the terms of data mining, this approach is called *supervised machine learning* which was for the first time described by Breiman and Ihaka in 1984 [24].

This chapter describes the principles and methods of data mining applicable in the area of intrusion detection. At first, prediction task of intrusion detection is defined. Then, we describe several prediction approaches based on machine learning, which will be utilized in our experiments performed in latter chapters as well as in some of state-of-the-art approaches. Later, we formally define performance measures of intrusion detection systems according to well known relations. Finally, we infer two ways of interpretation the multi-class confusion matrices and formally derive estimations of performance measures for k-fold cross validation method which will be employed in our further experiments.

3.1 IDS Prediction Task

Referencing to [92], let $X = V \times Y$ be the space of labeled samples,[1] where V represents the space of unlabeled samples and Y represents the space of possible labels (ground truth). Let $D_{tr} = \{x_1, x_2, \ldots, x_n\}$ be a training dataset consisting of n labeled samples, where $x_i = (v_i \in V,\ y_i \in Y)$. Consider classifier C which maps unlabeled sample $v \in V$ to a label $y \in Y$:

$$y = C(v), \tag{3.1}$$

and learning algorithm A which maps the given dataset D to a classifier C:

$$C = A(D). \tag{3.2}$$

The notation $y_{predict} = A(D_{tr}, v)$ denotes the label assigned to an unlabeled sample v by the classifier C build by learning algorithm A on the dataset D_{tr}. Now, all extracted features

[1] A sample refers to the vector of features extracted over a connection.

of the connection k can be used as input for the trained IDS classifier C which predicts the target label:

$$y_{predict} = A\big(D_{tr}, f(k)\big), \tag{3.3}$$

where

$$y_{predict} \in \{Intrusion, Legitimate\}, \tag{3.4}$$

for the case of binominal IDS classifier distinguishing only between intrusive and legitimate communications or

$$y_{predict} \in \left\{ \bigcup_{i=1}^{n} Intrusion_i \right\} \bigcup \left\{ \bigcup_{j=1}^{m} Legitimate_j \right\}, \tag{3.5}$$

for the case of multi-class IDS classifier distinguishing among subclasses of intrusive communications as well as legitimate ones.

3.2 Prediction Approaches

In the current section, we describe and define three representatives of prediction approaches in supervised machine learning, which can be employed for the purpose of intrusion detection having available ground truth about a training dataset. Namely, these approaches are posterior probability based Naive Bayes classifier, followed by maximum margin classifier of SVM, and finally hierarchical classifier – Decision Tree.

3.2.1 Naive Bayes Classifier

As the most of our experiments in the following chapters will utilize Naive Bayes classifier, now we thoroughly describe theory behind it, starting by derivation of probability rules, following by definition of Bayes theorem and maximum a-posteriori classifier utilizing it. Then, we describe naive approach of handling input variables for definition of naive Bayes model followed by construction of Naive Bayes classifier. Finally we describe handling of continuous data by univariate and multivariate Gaussian distributions. The following definitions are primarily taken from book [17].

Probability Rules and Bayes' Theorem

Firstly, we derive the rules of probability, considering two discrete random variables X and Y. We will suppose that X can take any of the values x_i where $i = \{1, \ldots, M\}$, and Y can take the values y_j where $j = \{1, \ldots, L\}$. Next, we consider a total of N trials in which we sample both of the variables X and Y, and let the number of such trials in which $X = x_i$ and $Y = y_j$ be n_{ij}. Also, let the number of trials in which X takes x_i be denoted by c_i, and similarly let the number of trials in which Y takes value y_j be denoted by r_j. Then, the probability that X will take the value x_i and Y will take the value y_j can be written as $p(X = x_i, Y = y_j)$ and is called the **joint probability** of $X = x_i$ and $Y = y_j$. It is given by the number of trials for which $X = x_i$ and $Y = y_j$, and hence

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}. \tag{3.6}$$

Here we implicitly consider the limit $N \to \infty$. The probability that X takes the value x_i irrespective of the value of Y can be written as $p(X = x_i)$ and is given by the fraction of the total number of trials whose values of X are equal to x_i, so that

$$p(X = x_i) = \frac{c_i}{N}. \tag{3.7}$$

Also, c_i can be formulated as $c_i = \sum_j n_{ij}$ and therefore, from 3.6 and 3.7 we have

$$p(X = x_i) = \sum_{j=1}^{L} p(X = x_i, Y = y_j), \tag{3.8}$$

which is the **sum rule of probability**. Note that $p(X = x_i)$ is sometimes called the **marginal probability**, because it is obtained by marginalizing – summing out the other variables (in this case Y).

If we consider only those instances for which $X = x_i$, then the fraction of such instances for which $Y = y_j$ is written as $p(Y = y_j|X = x_i)$ and is called the **conditional probability** of $Y = y_j$ with given $X = x_i$. The conditional probability is defined as

$$p(Y = y_j|X = x_i) = \frac{n_{ij}}{c_i}. \tag{3.9}$$

Then, from 3.6, 3.7 and 3.9, we can derive the following relationship

$$\begin{aligned} p(X = x_i, Y = y_j) &= \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p(Y = y_j|X = x_i)p(X = x_i), \end{aligned} \tag{3.10}$$

which is the **product rule of probability**. We can write fundamental rules of probability theory in compact notation:

$$\textbf{\textit{sum rule}} \qquad p(X) = \sum_Y p(X, Y), \tag{3.11}$$

$$\textbf{\textit{product rule}} \qquad p(X, Y) = p(Y|X)p(X). \tag{3.12}$$

Here $p(X, Y)$ is a joint probability, $p(Y|X)$ is a conditional probability and $p(X)$ is a marginal probability.

From the product rule, together with the symmetry property $p(X, Y) = p(Y, X)$, we obtain the following relationship between conditional probabilities

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}, \tag{3.13}$$

which is called **Bayes' theorem**.

Maximum A-posteriori Classifier

The Bayes' theorem can be utilized in both inferring and decision stages of the classification problem, whose purpose is to make decisions. Suppose we have an input vector $\boldsymbol{x}$ together with corresponding vector $\boldsymbol{t}$ of target class labels. The joint probability distribution $p(x, t)$ provides a complete summary of uncertainty associated with these variables. Determination of $p(x, t)$ from a set of training data is example of inference stage of the classification.

For simplicity consider two classes $C_k = \{C_1, C_2\}$ and unseen input vector $\boldsymbol{x}$ which needs to be classified. We are interested in the probabilities of the two classes, which are given by $p(C_k|\boldsymbol{x})$. Using Bayes' theorem, these probabilities can be expressed in the form

$$p(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{p(\boldsymbol{x})}. \tag{3.14}$$

Note that any of the quantities appearing in the Bayes' theorem can be obtained from the joint distribution $p(\boldsymbol{x}, C_k)$ by either marginalizing or conditioning with respect to the appropriate variables. We can now interpret $p(C_k)$ as the prior probability of class C_k, and $p(C_k|\boldsymbol{x})$ as the corresponding posterior probability. Our aim is to minimize the chance of assigning $\boldsymbol{x}$ to the wrong class, and thus intuitively we will choose the class having the higher posterior probability. Therefore, we will refer to this approach as Maximum A-posteriori Classifier (MAC). MAC minimizes the misclassification rate for each input $\boldsymbol{x}$.

We need a rule that assigns each value of $\boldsymbol{x}$ to one of the available classes. Such a rule will divide the input space into regions R_k called *decision regions*, one for each class, such that all points in R_k are assigned to class C_k. The boundaries between decision regions are called *decision boundaries* or *decision surfaces*. Each decision region needs to be contiguous but can comprise some number of disjoint regions. A mistake occurs when an input vector belonging into class C_1 is assigned to C_2 or vice versa. The probability of this happening is given by

$$\begin{aligned} p_{mistake} &= p(\boldsymbol{x} \in R_1, C_2) + p(\boldsymbol{x} \in R_2, C_1) \\ &= \int_{R_1} p(\boldsymbol{x}, C_2)dx + \int_{R_2} p(\boldsymbol{x}, C_1)dx. \end{aligned} \tag{3.15}$$

To minimize $p_{mistake}$ we should ensure that each $\boldsymbol{x}$ is assigned to class which has smaller value of integrand in 3.15. Thus, if $p(\boldsymbol{x}, C_1) > p(\boldsymbol{x}, C_2)$ for a given value of $\boldsymbol{x}$ then we should assign it to class C_1. From the product rule of probability we have $p(\boldsymbol{x}, C_k) = p(C_k|\boldsymbol{x})p(\boldsymbol{x})$. Because the factor $p(\boldsymbol{x})$ is common to both terms we can restate this result by saying that the minimum probability of making a mistake is obtained if each value of $\boldsymbol{x}$ is assigned to the class for which the posterior probability $p(C_k|\boldsymbol{x})$ is largest.

For the general case of K classes, it is easier to maximize the probability of being correct, which is given by

$$\begin{aligned} p_{correct} &= \sum_{k=1}^{K} p(\boldsymbol{x} \in R_k, C_k) \\ &= \sum_{k=1}^{K} \int_{R_k} p(\boldsymbol{x}, C_k)dx, \end{aligned} \tag{3.16}$$

which is maximized when the regions R_k are chosen such that each $\boldsymbol{x}$ is assigned to the class for which $p(\boldsymbol{x}, C_k)$ is largest. Again, using the product rule $p(\boldsymbol{x}, C_k) = p(C_k|\boldsymbol{x})p(\boldsymbol{x})$ and the fact that factor of $p(\boldsymbol{x})$ is common to all terms, we see that each $\boldsymbol{x}$ is assigned to the class having the largest posterior probability $p(C_k|\boldsymbol{x})$.

Combining Models – Naive Bayes Model

For more complex applications, we may wish to break the problem into a number of smaller subproblems each of which can be tackled by a separate module. Rather than combine

all heterogeneous input information into one huge input space, it may be more effective to built one system (model) per each input information (feature). All models give us posterior probabilities for the classes and we can combine the outputs systematically using the rules of probability. One simple way to do this is to assume that, for each class separately, the distributions of inputs for all input features are independent, so that

$$\begin{aligned} p(x_1, \ldots, x_n | C_k) &= p(x_1|C_k) \cdot p(x_2|C_k) \cdot \ldots \cdot p(x_n|C_k) \\ &= \prod_{i=1}^{n} p(x_i|C_k). \end{aligned} \tag{3.17}$$

This is an example of *conditional independence* property, because the independence holds when the distribution is conditioned on the class C_k. Then, posterior probability, considering all input features, is given by

$$\begin{aligned} p(C_k|x_1, \ldots, x_n) &\propto (C_k, x_1, \ldots, x_n) \\ &\propto p(x_1, \ldots, x_n|C_k)p(C_k) \\ &\propto p(C_k) \cdot p(x_1|C_k) \cdot p(x_2|C_k) \cdot \ldots \cdot p(x_n|C_k) \\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \\ &\propto \frac{1}{p(C_k)} \prod_{i=1}^{n} p(C_k|x_i). \end{aligned} \tag{3.18}$$

Thus we need the class prior probabilities $p(C_k)$, which can be estimated from fractions of data points in each class. Then, we need to normalize the resulting posterior probabilities so they sum to one. The particular conditional independence assumption (3.17) is an example of the *Naive Bayes model*.

Constructing Naive Bayes Classifier from a Probability Model

The Naive Bayes classifier combines the previous model with a decision rule. Common rule is to maximize probability of being correct, as we already described in this subsection. This rule is known as maximum a posteriori decision rule (MAP). The corresponding classifier – Naive Bayes classifier – is the function which assigns a class label $y = C_k$ as follows:

$$y = \underset{k \in \{1,\ldots,K\}}{argmax} \; p(C_k) \prod_{i=1}^{n} p(x_i|C_k). \tag{3.19}$$

Continuous Data

A class prior probability may be assumed by calculating an estimate for the class probability from the training set – i.e., (prior probability for a given class) = (number of samples in the class) / (total number of samples). To estimate the parameters of a feature's distribution, we have to assume a distribution, and thus consider parametric model, or generate non-parametric models for the features from the training set [85].

The typical example of parametric model is Gaussian Naive Bayes model. When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian (Normal) distribution. Suppose the training data contain a continuous attribute x. We first segment the data by the class, and

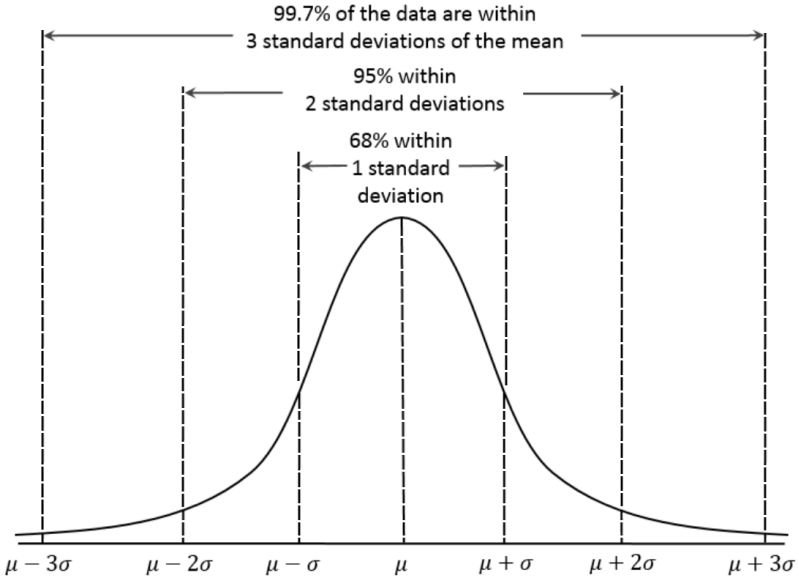

Figure 3.1: Univariate Gaussian distribution

then compute the mean and variance of x for each class. Let μ_c be the mean of the values in x associated with class c, and let ${\sigma_c}^2$ be the variance of the values in x associated with the class c. Then, probability distribution of x given a class c is defined by

$$\aleph(x|\mu, {\sigma_c}^2) = \frac{1}{\sqrt{2\pi{\sigma_c}^2}} e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}. \tag{3.20}$$

Note that square root of the variance, given by σ, is called the standard deviation. Figure 3.1 shows the plot of the univariate Gaussian distribution with mean μ and standard deviation σ. We can see that 95% of values fall into range $\langle -2\sigma, +2\sigma \rangle$ and 99.7% fall into interval sliced by $\pm 3\sigma$. From the form of (3.20) we can see that Gaussian distribution satisfies

$$\aleph(x|\mu, \sigma^2) > 0. \tag{3.21}$$

Also it is straightforward to show that the Gaussian distribution is normalized, so that

$$\int_{-\infty}^{\infty} \aleph(x|\mu, \sigma^2)\, dx \;=\; 1. \tag{3.22}$$

Therefore, (3.20) satisfies the two requirements for valid probability density. The estimations of parameters μ and σ^2 are performed per each class of input dataset, so that we consider only items of particular class $D_c \subset D$, where $T_c = |D_c|$:

$$\mu_c \;=\; \frac{1}{T_c} \sum_{i=1}^{T_c} x_i \;, \tag{3.23}$$

$$\sigma_c^2 \;=\; \frac{1}{T_c} \sum_{i=1}^{T_c} (x_i - \mu_c)^2. \tag{3.24}$$

In the following definitions we will not consider class of input dataset items. We are also interested in the Gaussian distribution defined over a D-dimensional vector $\boldsymbol{x}$ of continuous

variables, which is given by

$$\aleph(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}, \tag{3.25}$$

where the D-dimensional vector $\boldsymbol{\mu}$ is called the mean, the $D \times D$ matrix $\boldsymbol{\Sigma}$ is called the covariance, and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. Note that by bold notation we refer to vectors or matrices. Now suppose that we have a dataset of observations $D = \{\boldsymbol{x_i} \mid \boldsymbol{x_i} = (x_1, \ldots, x_N)^T\}$, where each item represents N observations of N scalar variables. We consider that data points are drawn independently from the same Gaussian distribution. We know that a joint probability of two independent events is given by the product of the marginal probabilities for each event separately. Therefore, we can write the probability of the dataset, given μ and σ^2, in the form

$$p(\boldsymbol{x}|\mu, \sigma^2) = \prod_{n=1}^{N} \aleph(x_n|\mu, \sigma^2). \tag{3.26}$$

When viewed as a function of μ and σ^2, we interpret it as likelihood function of approximation of input dataset by Gaussian distribution. Now we determine the parameters of Gaussian distribution by maximizing the likelihood function (3.26). Maximizing rewritten function (3.26) considering (3.20) with respect to μ, we obtain the maximum likelihood solution given by

$$\mu_{ML} = \frac{1}{N} \sum_{n=1}^{N} x_n, \tag{3.27}$$

which is the sample mean. Similarly maximizing that function with respect to σ^2, we obtain the maximum likelihood solution for the variance in the form

$$\sigma^2_{ML} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{ML})^2, \tag{3.28}$$

which is the sample variance.

Now we will consider class of input dataset items and will represent multivariate Gaussian distribution as Gaussian Mixture Model (GMM). The likelihood function of the GMM has following form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} p(\boldsymbol{x}|c_k)\ p(c_k) = \sum_{k=1}^{K} \aleph(\boldsymbol{x}|\boldsymbol{\mu_{c_k}}, \boldsymbol{\Sigma_{c_k}})\ p(c_k). \tag{3.29}$$

The maximum likelihood solutions of the function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are defined per each class c as follows:

$$\boldsymbol{\mu_c} = \frac{1}{T_c} \sum_{i=1}^{T_c} \boldsymbol{x_i^c}, \tag{3.30}$$

$$\boldsymbol{\Sigma_c} = \frac{1}{T_c} \sum_{i=1}^{T_c} (\boldsymbol{x_i^c} - \boldsymbol{\mu_c})(\boldsymbol{x_i^c} - \boldsymbol{\mu_c})^T, \tag{3.31}$$

where T_c represents size of the class.

Another common technique for handling continuous values is to use binning for discretization of the feature values. However, the discretization may throw away discriminative information [65].

3.2.2 Support Vector Machines

All the following definitions related to Support Vector Machines (SVM) were taken from [17]. SVM became popular in some years ago for solving problems in classification, regression as well as detection. An important property of SVM is that determination of the model parameters corresponds to a convex optimization problem, and thus any local solution is also a global optimum.

Kernel Function

First, we define *kernel function* which will be later utilized in other definitions. The kernel function is given by the relation

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x'}), \tag{3.32}$$

where $\boldsymbol{x}$ represents r-dimensional input feature space and the function $\phi(\boldsymbol{x})$ represents fixed nonlinear feature space mapping into the space with the different number of dimensions. From this definition, we can see that the kernel function is symmetric, and thus $k(\boldsymbol{x}, \boldsymbol{x'}) = k(\boldsymbol{x'}, \boldsymbol{x})$. The kernel concept was introduced into the field of pattern recognition by Aizerman et al. in 1964 [2]. Although neglected for many years, it was re-introduced into machine learning in the context of large margin classifiers by Boser et al. in 1992 [22], which gives rise to the technique of SVM.

Maximum Margin Classifier

We begin by discussion about two-class classification problem using linear models of the form

$$y(x) = \boldsymbol{w}^T \phi(x) + b, \tag{3.33}$$

where ϕ denotes a fixed feature-space transformation and parameter b denotes explicit bias. The training dataset comprises N input vectors $x_1, \ldots, x_N$, with corresponding target values $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$ for $n = 1, \ldots, N$ and new data points x are classified according to the sign of $y(x)$. Now, we will assume that the training dataset is linearly separable in feature space, so that by definition there exits at least one choice of parameters $\boldsymbol{w}$ and b such that a function of the form (3.33) satisfies $y(x_n) > 0$ for points having $t_n = +1$ and $y(x_n) < 0$ for points having $t_n = -1$, so that $t_n y(x_n) > 0$ for all training data points. SVM approaches the problem of separation of classes through the concept of *margin* which is defined to be the smallest distance between the decision boundary (hyperplane) and any of the samples, as illustrated in Figure 3.2. In SVM the decision boundary is chosen to be the one for which the margin is maximized. The perpendicular distance of a point x from a hyperplane defined by $y(x) = 0$ where $y(x)$ takes the form (3.33) is given by $|y(x)|/||\boldsymbol{w}||$. Furthermore, we are only interested in solutions for which all data points are correctly classified, so that $t_n y(x_n) > 0$ for all n. Thus, the distance of a point x_n to the decision surface is given by

$$\frac{t_n y(x_n)}{||w||} = \frac{t_n (w^T \phi(x_n) + b)}{||w||}. \tag{3.34}$$

The margin is given by the perpendicular distance to the closest point x_n from the dataset, and aim to optimize the parameters $\boldsymbol{w}$ and b in order to maximize this distance. Therefore,

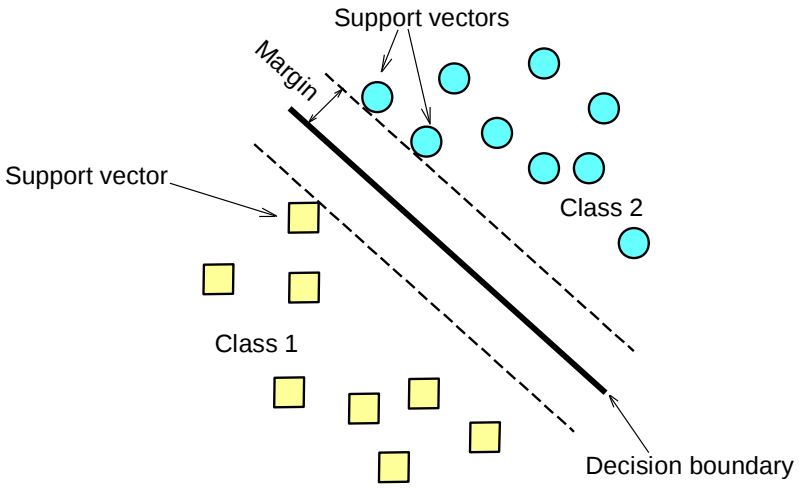

Figure 3.2: The concept of margin in SVM

the maximum margin solution is found by solving

$$\underset{\boldsymbol{w},b}{arg\ max} \left\{ \frac{1}{||w||} \underset{n}{min} \left[t_n(\boldsymbol{w}^T \phi(x_n) + b) \right] \right\}, \tag{3.35}$$

where we have taken factor $1/||w||$ outside the optimization over n because $\boldsymbol{w}$ does not depend on n. Direct solution of this problem would be very complex, and so we convert it into equivalent problem that is much more easier to solve. To this we note, that if we make the rescaling $\boldsymbol{w} \to \kappa \boldsymbol{w}$ and $b \to \kappa b$, then the distance from any point x_n to the decision surface, given by $t_n y(x_n)/||\boldsymbol{w}||$ is unchanged. We can use this freedom to set

$$t_n(\boldsymbol{w}^T \phi(x_n) + b) = 1 \tag{3.36}$$

for the point that is closest to the decision surface. Therefore, all data points will satisfy the constrains

$$t_n(\boldsymbol{w}^T \phi(x_n) + b) \geq 1, \qquad n = 1, \ldots, N. \tag{3.37}$$

This is known as the *canonical* representation of the hyperplane. Constraints are said to be *active* for the points which holds equality (3.36), and for the reminder are said to be *inactive*. By definition, there will always be at least one active constraint, because there will always be a closest point, and once the margin has been maximized there will be at least two active constraints. The optimization problem then simply requires that we maximize $||w||^{-1}$, which is equivalent to minimizing $||w||^2$, and thus we have to solve the optimization problem

$$\underset{\boldsymbol{w},b}{arg\ min}\ \frac{1}{2}||w||^2 \tag{3.38}$$

as subject to the constraints given by (3.37). This is an example of *quadratic programming* problem in which we are trying to minimize a quadratic function as subject to a set of linear inequality constrains. It appears that the bias parameter has disappeared from optimization, however, it is determined implicitly by the constraints (3.37). The constraints require that changes to $||w||$ are compensated by changes to b.

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geq 0$, with one multiplier a_n for each of the constraints in 3.37, giving the Lagrangian function

$$L(\boldsymbol{w}, b, \boldsymbol{a}) \;=\; \frac{1}{2}||w||^2 - \sum_{n=1}^{N} \left\{ t_n(\boldsymbol{w}^T \phi(x_n) + b) - 1 \right\}, \tag{3.39}$$

where $\boldsymbol{a} = (a_1, \ldots, a_N)^T$. Setting the derivatives of $L(\boldsymbol{w}, b, \boldsymbol{a})$ with respect to $\boldsymbol{w}$ and b equal to zero, we obtain the following two conditions

$$w \;=\; \sum_{n=1}^{N} a_n t_n \phi(x_n), \tag{3.40}$$

$$0 \;=\; \sum_{n=1}^{N} a_n t_n. \tag{3.41}$$

Eliminating $\boldsymbol{w}$ and b from $L(\boldsymbol{w}, b, \boldsymbol{a})$ using these conditions gives the *dual representation* of the maximum margin problem in which we maximize

$$\tilde{L}(\boldsymbol{a}) \;=\; \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \tag{3.42}$$

with respect of $\boldsymbol{a}$ as subject to the constraints

$$a_n \;\geq\; 0, \qquad n = 1, \ldots, N, \tag{3.43}$$

$$\sum_{n=1}^{N} a_n t_n \;=\; 0. \tag{3.44}$$

Here the kernel function is defined by $k(x, x') = \phi(x)^T \phi(x')$. Again, this takes the form of a quadratic programming problem in which we optimize a quadratic function of a as subject to a set of inequality constraints. The solution to a quadratic programming problem in M variables in general has computional complexity $O(M^3)$. Regarding dual formulation, we have turned the original optimization problem, which involved minimizing (3.38) over M variables into the dual problem (3.42), which has N variables. For a fixed set of basis functions whose number M is smaller than the number N of data points, the move to dual problem may appear disadvantageous. However, it allows the model to be reformulated using kernels, and therefore the maximum margin classifier can be applied efficiently to feature space whose dimensionality exceeds the number of data points, including infinite feature spaces. The kernel formulation also makes clear the role of the constraint that the Lagrangian function $\tilde{L}(\boldsymbol{a})$ is bounded below, giving the rise to a well defined optimization problem.

In order to classify new data points using the trained model, we evaluate the sign of $y(x)$ defined by (3.33). This can be expressed in terms of the parameters $\{a_n\}$ and the kernel function by substituting for $\boldsymbol{w}$ using (3.40) to give

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b. \tag{3.45}$$

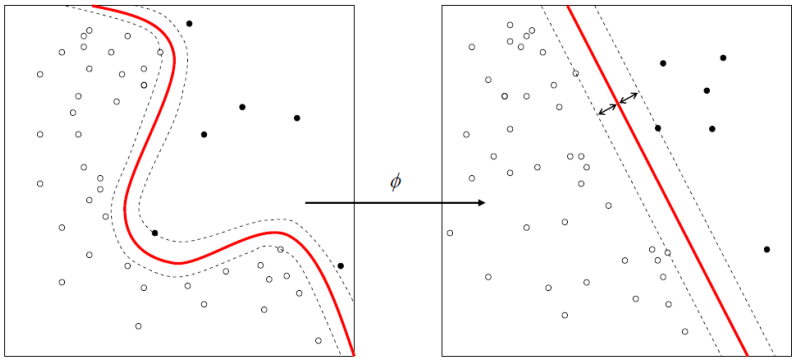

Figure 3.3: Support vectors with decision and margin boundaries

Constrained optimization of this form satisfies the *Karush-Kuhn-Tucker* (KKT) conditions, which in this case require the following three properties to be held:

$$a_n \geq 0, \tag{3.46}$$
$$t_n y(x_n) - 1 \geq 0, \tag{3.47}$$
$$a_n\{\, t_n y(x_n) - 1\} = 0. \tag{3.48}$$

Thus for every data point, either $a_n = 0$ or $t_n y(x_n) = 1$. Any data point for which $a_n = 0$ will not appear in the sum in (3.45) and hence plays no role in making predictions for new data points. The remaining data points are called *support vectors*, and because they satisfy $t_n y(x_n) = 1$, they correspond to point that lay on the maximum margin hyperplane in feature space, as illustrated in Figure 3.3 by data points laying on dashed lines.This property is central to practical applicability of SVM. Once, the model is trained, a significant amount of data points can be discarded and only support vectors retain. Having solved quadratic programming problem and found a value for $\boldsymbol{a}$, we can then determine the value of the threshold parameter b by noting that any support vector x_n satisfies $t_n y(x_n) = 1$. Using (3.45) get us with

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(x_n, x_m) + b \right) = 1, \tag{3.49}$$

where S denotes the set of indices of the support vectors. Although, we can solve this equation for b using arbitrarily chosen support vecotr x_n, a numerically more stable solution is obtained by first multiplying by t_n, making use of $t_n^2 = 1$. And then averaging these equations over all support vectors and solving for b to give

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(x_n, x_m) \right), \tag{3.50}$$

where N_S is the total number of support vectors.

For later comparison with alternative models, we can express the maximum margin classifier in terms of the minimization of an error function, with a simple quadratic regularizer, in the form

$$\sum_{n=1}^{N} E_\infty(y(x_n)t_n - 1) + \lambda ||w||^2, \tag{3.51}$$

where $E_\infty(z)$ is a function that is zero if $z \geq 0$ and ∞ otherwise. Also $E_\infty(z)$ ensures that the constraints (3.37) are satisfied. Note that as long as the regularization parameter satisfies $\lambda > 0$, its precise value plays no role.

The Figure 3.3 also shows the example of the classification by SVM. Although, the dataset is not linearly separable in the two-dimensional data space x, it is linearly separable in the nonlinear feature space defined implicitly by the nonlinear kernel function. Thus, the training data points are perfectly separated in the original data space.

Overlapping Class Distributions

So far, we have assumed that the training data points are linearly separable in the feature space $\phi(x)$. The resulting SVM will give exact separation of the training data in the original input space x, although the corresponding decision boundary will be nonlinear. However in practice, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization.

Therefore, we need a way of allowing the SVM to misclassify some of training data points. From (3.51) we see that in the case of separable classes, we implicitly used an error function that gave infinite error if a data point was misclassified and zero error if it was correctly classified. Then, we optimized the model parameters to maximize the margin. Now, we modify this approach so that data points are allowed to be on the 'wrong side' of the margin boundary, but with a penalty which increases with the distance from that boundary. For subsequent optimization problem it is convenient to make this penalty linear function of the distance. To do this, we introduce *slack variables*, $\xi_n \geq 0$ where $n = 1, \ldots, N$, with one slack variable for each training data point. These are defined by $\xi_n = 0$ for data points that are on or inside the correct margin boundary and $\xi_n = |t_n - y(x_n)|$ for other points. Thus, the data point that is on the decision boundary $y(x_n) = 0$ will have $\xi_n = 1$, and points with $\xi_n \geq 1$ will be misclassified. The exact classification constraints (3.37) are then replaced with

$$t_n y(x_n) \geq 1 - \xi_n, \qquad n = 1, \ldots, N \tag{3.52}$$

in which the slack variables are constraint to satisfy $\xi_n \geq 0$. Data point for which $\xi_n = 0$ are correctly classified and are either on the margin or on the correct of the margin. Points for which $0 < \xi_n \geq 1$ lie inside the margin, but on the correct side of the decision boundary, and those data points for which $\xi_n > 1$ lie on the wrong side of the decision boundary and are misclassified. This is sometimes described as relaxing the hard margin constraint to give a *soft margin* and allows some of the training data points to be misclassified. Note that while slack variables allow overlapping class distributions, this concept is still sensitive to outliers because the penalty for misclassification increases linearly with value of ξ.

Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. Therefore, we minimize

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||w||^2, \tag{3.53}$$

where the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. Because any point that is misclassified has $\xi_n > 1$, it follows that $\sum_n \xi_n$ is an upper bound on the number of misclassified points. The parameter C is therefore analogous to the inverse of regularization coefficient because it controls the trade-off between minimizing

training errors and controlling model complexity. Now, we minimize (3.53) as subject to the constraints (3.52) as well as $\xi_n \geq 0$. The corresponding Lagrangian function is given by

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||w||^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}a_n\{t_n y(x_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n, \tag{3.54}$$

where $\{a_n \geq 0\}$ and $\{\mu_n \geq 0\}$ are Lagrangian multipliers. The corresponding set of KKT conditions are given by

$$a_n \geq 0, \tag{3.55}$$
$$t_n y(x_n) - 1 + \xi_n \geq 0, \tag{3.56}$$
$$a_n(t_n y(x_n) - 1 + \xi_n) = 0, \tag{3.57}$$
$$\mu_n \geq 0, \tag{3.58}$$
$$\xi_n \geq 0, \tag{3.59}$$
$$\mu_n\xi_n = 0, \tag{3.60}$$

where $n = 1, \ldots, N$. Now, we need to optimize $\boldsymbol{w}$, b, and $\{\xi_n\}$ making use of the definition (3.33) of $y(x)$ to give

$$\frac{\partial L}{\partial w} = 0 \quad \Rightarrow \quad \boldsymbol{w} = \sum_{n=1}^{N}a_n t_n \phi(x_n), \tag{3.61}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad 0 = \sum_{n=1}^{N}a_n t_n, \tag{3.62}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad \Rightarrow \quad a_n = C - \mu_n. \tag{3.63}$$

Using optimized results to eliminate $\boldsymbol{w}, b$ and $\{\xi_n\}$ from the Lagrangian function, we obtain the dual Lagrangian function in the form

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N}a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}a_n a_m t_n t_m k(x_n, x_m), \tag{3.64}$$

which is identical to separable case, except the constrain are different. We note that $a_n \geq 0$ is required because these are Lagrangian multipliers. Furthermore, (3.63) and $\mu_n \geq 0$ implies $a_n \leq C$. Therefore, we minimize (3.64) with respect to the dual variables $\{a_n\}$ as subject to

$$0 \leq a_n \leq C, \tag{3.65}$$
$$\sum_{n=1}^{N}a_n t_n = 0 \tag{3.66}$$

for $n = 1, \ldots, N$, where (3.65) are known as *box constraints*. This represents a quadratic programming problem. As before, a subset of the data points may have $a_n = 0$, in which case they do not contribute to the predictive model (3.45). The remaining data points constitute the support vectors. The points have $a_n > 0$ and hence from (3.57) must satisfy

$$t_n y(x_n) = 1 - \xi_n. \tag{3.67}$$

If $a_n > C$, then (3.63) implies that $\mu_n > 0$, which from (3.60) requires $\xi_n = 0$ and thus such points lie on the margin. Points with $a_n = C$ can lie inside the margin and can either be correctly classified if $\xi_n \leq 1$ or misclassified if $\xi_n > 1$. To determine the parameter b in (3.33), we note that those support vectors for which $0 < a_n < C$ have $\xi_n = 0$ so that $t_n y(x_n) = 1$ and hence will satisfy

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(x_n, x_m) + b \right) = 1. \tag{3.68}$$

A numerically stable solution is obtained by averaging to give

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(x_n, x_m) \right), \tag{3.69}$$

where $\mathcal{M}$ denotes the set of indices of data points having $0 < a_n < C$.

Multiclass SVMs

The SVM is fundamentally a two-class classifier. However, in practice, we often have to deal with problems involving $K > 2$ classes. Therefore, various methods have been proposed for combining multiple two-class SVMs in order to build a multiclass classifier.

One commonly used approach is to construct K separate SVMs, in which the k-th model $y_k(x)$ is trained using the data from class C_k as the positive examples and the data from the remaining $K - 1$ classes as the negative examples, which is known as *one-versus-the-rest* approach [188]. However, using the decisions of the individual classifiers can lead to inconsistent results in which an input data point is assigned to multiple classes simultaneously. This problem is sometimes addressed by making predictions for new inputs $\boldsymbol{x}$ using

$$y(\boldsymbol{x}) = \max_k \; y_k(x). \tag{3.70}$$

Unfortunately, this heuristic approach suffers from the problem that different classifiers were trained on different training data, and thus there is no guarantee that real valued quantities $y_k(x)$ for different classifiers will have appropriate scales. Another problem of the one-versus-the-rest approach is that the training sets are imbalanced. A variant of one-versus-the-rest classifier was proposed by Lee et al. [100] who modify the target values so that the positive class has target $+1$ and the negative class has target $-1/(K-1)$.

Another approach is to train $K(K-1)/2$ different 2-class SVMs on all possible pair of classes, and then to classify test points according to which class has the highest number of 'votes'. This approach is sometimes called *one-versus-one*. Again, this can lead to ambiguities in the resulting classification. Also, for large K this approach requires significantly more training time than the one-versus-the-rest approach. Similarly, to evaluate test points, more computation is required.

The latter problem can be alleviated by organizing the pairwise classifiers into a directed acyclic graph, which is called DAGSVM [135]. For K classes, the DAGSVM has a total of $K(K-1)/2$ classifiers, and to classify a new test point only $K - 1$ pairwise classifiers need to be evaluated, with the particular classifiers used depending on which path through the graph is traversed. A different approach to multiclass classification, based on error-correcting output codes, was developed by Dietterich et al. in the paper [54] and applied

to SVMs by Allwein et al. [6]. It can be viewed as a generalization of the voting scheme of the one-versus-one approach in which more general partitions of the classes are used to train the individual classifiers. The K classes are represented as particular sets of responses from the two-class classifiers chosen, and together with a suitable decoding scheme, this gives robustness to errors and to ambiguity in the outputs of the individual classifiers. Although, the application of SVMs to multiclass classification problems remain an open issue, in practice the one-versus-the-rest approach is widely used in spite of its ad-hoc formulation and its practical limitations.

3.2.3 Decision Tree

All the following definitions related to Decision Tree are taken from the book [7]. Decision Tree is a data structure implementing the divide-and-conquer strategy. It is an nonparametric method which can be utilized for both classification and regression, however in the context of this work, we will consider only classification variant of Decision Tree. In nonparametric estimation, we divide the input space into local regions, which are defined by a distance measure like the Euclidean norm, and for each input, the corresponding local model is computed from the training data in that region.

More formally, Decision Tree is a hierarchical model for supervised machine learning where the local region is identified in a sequence of recursive splits in a small number of steps. Decision Tree is composed of internal *decision nodes* and *terminal leaves*. An example of Decision Tree together with corresponding dataset is depicted in Figure 3.4, where ovals denote decision nodes and squares denote leaves. Each decision node m implements a test function $f_m(x)$ with discrete output labeling the branches. Given an input at each node, a test is evaluated and one of the branches is taken depending on the output. This process starts at the root and is repeated recursively until a leaf node is hit. At this point, the value associated with the leaf constitutes the output.

Decision Tree is also a nonparametric model in the sense that we do not assume any parametric form of the class densities. Each $f_m(x)$ defines a discriminant in the d-dimensional input space dividing it into smaller regions which are further subdivided, as we take the path from the root down. $f_m(x)$ is a simple function and when written down as a tree, a complex function is broken down into a series of simple decisions. Different Decision Tree methods assume different models for $f_m(x)$, and the model class defines the shape of the discriminant and the shape of the regions. Each leaf node has associated output label, which in the case of classification is the class code (in the case of regression it is numeric value). A leaf node defines a localized region in the input space where instances falling in this region have the same labels in classification (or very similar numeric outputs in regression). The boundaries of the regions are defined by the discriminants which are coded into the internal nodes along the path from the root to the leaf node. The hierarchical placement of the decisions allows a fast localization of the region covering an input. For example, in the case if the decisions are binary, then in the best case each decision eliminates half of the cases. If there are b regions, then in the best case, the correct region can be found in $log_2\ b$ decisions.

Univariate Decision Tree

In each internal node of the Univariate Decision Tree, the test uses only one of the input dimensions. If an input dimension x_j is discrete, taking one of n possible values, then the decision node checks the value of x_j and takes the corresponding branch, which is implementation of a n-way split. The decision node has discrete branches, and thus a numeric

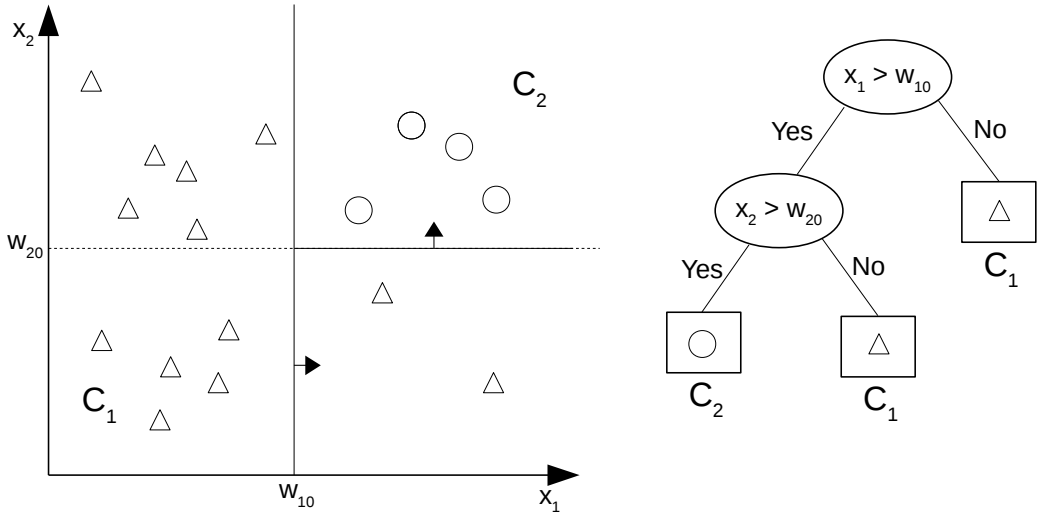

Figure 3.4: Example of dataset and corresponding Decision Tree

input should be discretized. If x_j is numeric ordinal (ordered), the best test is a comparison

$$f_m(x) : x_j > w_{m0}, \tag{3.71}$$

where w_{m0} is a suitably chosen threshold value. The decision node divides the input space into two regions: $L_m = \{x|x_j > w_{m0}\}$ and $R_m = \{x|x_j \leq w_{m0}\}$, which is called a *binary split*. Successive decision nodes on the path from the root to a leaf further divide these regions into two using other attributes, and thus generating splits orthogonal to each other. The leaf nodes defines hyperrectangles in the input space, which are depicted in Figure 3.4.

Tree induction is the construction of the tree given a training set of samples. For a given training set of samples, there exists many trees that code it with no error. We are interested in finding the smallest one among them, where tree size is measured as the number of nodes in the tree and the complexity of the decision nodes. Finding the smallest tree is NP-complete problem [141], and thus we are forced to use local search procedures based on heuristics which give reasonable trees in reasonable time.

Tree learning algorithms are greedy and, at each step, starting at the root with the complete training data, we look for the best split. This splits the training data into two or n parts, depending on whether the chosen attribute is numeric or discrete. We then continue splitting recursively with the corresponding subset until we do not need to split anymore, at which point a leaf node is created and labeled.

Classification Trees

Classification Tree represents application of Decision Tree for classification purpose. In this case, the goodness of the split is quantified by an *impurity measure*. A split is pure, if for all branches after the split, all the instances choosing the branch belong to the same class. Let us say for node m, N_m is the number of training instances reaching node m. It is N for the case of the root node. N_m^i of N_m belongs to class C_i, with $\sum_i N_m^i = N_m$. Given an instance reaching node m, the estimate of probability of class C_i is

$$P(C_i|x, m) \equiv p_m^i = \frac{N_m^i}{N_m}. \tag{3.72}$$

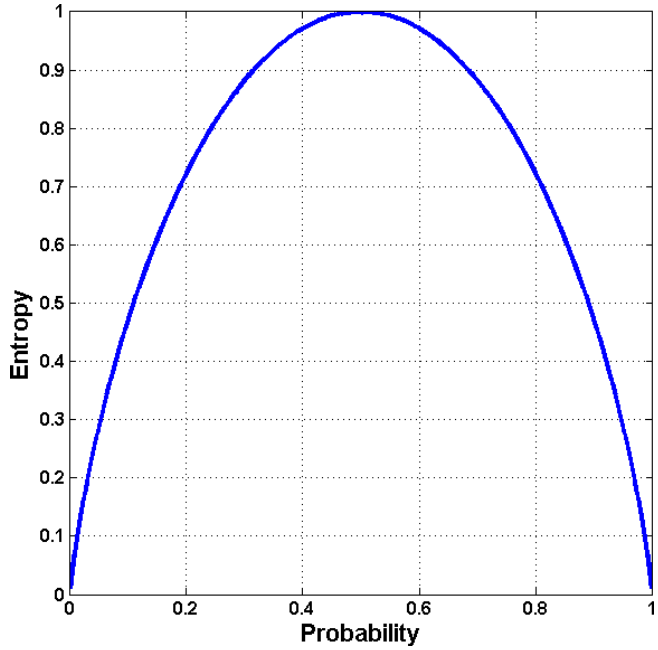

Figure 3.5: Entropy function for a two class problem

Node m is pure if p_m^i for all i are either 0 or 1. It is 0 if none of the instances reaching node m are of class C_i, and it is 1 if all such instances are of class C_i. If the slit is pure, we do not need to split anymore and can add a leaf node labeled with the class for which p_m^i is 1. One possible function for measuring impurity is *entropy* [141] (see Figure 3.5), which is defined as

$$\mathcal{I}_m = -\sum_{i=1}^{K} p_m^i \ log_2 p_m^i, \tag{3.73}$$

where $0 \ log_2 \ 0 \equiv 0$. However, the entropy is not the only possible measure. For a two-class problem where $p_1 = p$ and $p_2 = 1 - p$, $\phi(p, 1-p)$ is a nonnegative function measuring the impurity of a split if it satisfies the following properties [52]:

- $\phi\left(\frac{1}{2}, \frac{1}{2}\right) \geq \phi(p, 1-p)$, for any $p \in [0, 1]$,
- $\phi(0, 1) = \phi(1, 0) = 0$,
- $\phi(p, 1-p)$ is increasing in $p \in \left[0, \frac{1}{2}\right]$ and decreasing in $p \in \left[\frac{1}{2}, 1\right]$.

Examples of functions measuring impurity for two classes problem are following:

1. **Entropy**

$$\phi(p, 1-p) = -p \ log_2 \ p - (1-p) \ log_2(1-p). \tag{3.74}$$

   Note that Equation (3.73) is generalization for $K > 2$ classes.

2. **Gini index** [23]

$$\phi(p, 1-p) = 2p(1-p). \tag{3.75}$$

3. **Misclassification error**

$$\phi(p, 1-p) = 1 - max(p, 1-p). \tag{3.76}$$

These can be generalized to $K > 2$ classes, and the misclassification error can be generalized to minimum risk given a loss function. The results of the research showed that there is not a significant difference among these three measures [7].

If node m is not pure, then the instances should be split to decrease impurity. However, there are multiple possible attributes which can be used for splitting. For a numeric attribute, multiple split positions are possible. Among all, we are looking for the split that minimizes impurity after the split, as we want to generate the smallest tree. If the subsets after the split are closer to pure, then fewer splits will be needed afterward. However, we should note that this is local optimum and there is no guarantee that we will find the smallest Decision Tree.

Let us say at node m, N_{mj} of N_m take branch j; these are x_t for which the test $f_m(x_t)$ returns the outcome j. For a discrete attribute with n values, there are n outcomes, while for a numeric attribute there are two outcomes ($n = 2$), in either case satisfying $\sum_{j=1}^{n} N_{mj} = N_m$. N_{mj}^i of N_{mj} belong to class C_i: $\sum_{i=1}^{K} N_{mj}^i = N_{mj}$. Similarly, $\sum_{j=1}^{n} N_{mj}^i = N_m^i$. Given node m whose test returns the outcome j, the estimate for the probability of class C_i is following

$$P(C_i|x, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}} \tag{3.77}$$

and the total impurity after the split is given by

$$\mathcal{I}_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{mj}^i \; log_2 \; p_{mj}^i. \tag{3.78}$$

In the case of a numeric attribute, to be able to calculate p_{mj}^i using Equation (3.71), we also need to know w_{m0} for that node. Although, there are $N_m - 1$ possible w_{m0} between N_m data points, we do not need to calculate test for all points. It is enough to test, for example, at halfway between the points with the highest and the lowest value. Also, note that the best split is always between adjacent points belonging to different classes. Therefore, we try them and the best in terms of purity is taken for the purity of the attribute. In the case of the discrete attribute, no such iteration is necessary.

So for all attributes, discrete and numeric, and for a numeric attribute for all split positions, we calculate the impurity and choose the one which has the minimum entropy – measured e.g. by Equation (3.78). Then, tree construction continues recursively and in parallel for all the branches which are not pure, until are all pure. This is a basis for the *Classification and Regression Tree* (CART) algorithm [23], *ID3* algorithm [141] and its extension *C4.5* [142]. The pseudo code of the algorithm producing Classification Tree instances is depicted in Algorithm 1 and Algorithm 2. At each step of the tree construction, we choose the split which causes the largest decrease in impurity. It is represented by the difference between the impurity of data reaching node m – Equation (3.73) – and the total entropy of data reaching its branches after the split – Equation (3.78).

Algorithm 1: Function for splitting of attribute

```
1  Function SplitAttribute(𝒳) begin
2      minEntropy = MAX
3      for all attributes i = 1, ..., d do
4          if (x_i is discrete with n values) then
5              Split 𝒳 into 𝒳_1, ..., 𝒳_n by x_i
6              e = SplitEntropy(𝒳_1, ..., 𝒳_n) /*      Equation (3.78)      */
7              if (e < minEntropy) then
8                  minEntropy = e
9                  bestf = i
10             end if
11         else
               /*              x_i is numeric                              */
12             for all possible splits do
13                 Split 𝒳 into 𝒳_1, 𝒳_2 on x_i
14                 e = SplitEntropy(𝒳_1, 𝒳_2)
15                 if (e < minEntropy) then
16                     minEntropy = e
17                     bestf = i
18                 end if
19             end for
20         end if
21     end for
22 end
```

Algorithm 2: Construction of Classification Tree

```
1  Function GenerateTree(𝒳) begin
2      if (NodeEntropy(𝒳 < θ_I) then /*      Equation (3.73)      */
3          Create leaf labeled by majority class in 𝒳
4          Return
5      end if
6      i = SplitAttribute(𝒳)
7      for each branch of x_i do
8          Find 𝒳_i falling into branch
9          GenerateTree(𝒳_i)
10     end for
11 end
```

Multivariate Trees

In the case of univariate tree, only one input dimension is used at a split, however in the case of multivariate tree, all input dimensions can be used for splitting, and thus it is more general. In the context of this thesis, we will consider only univariate trees, as we will aim at analysis of single attributes primarily, where can help a lot univariate Decision Tree.

3.3 Performance Measures

The current section defines performance measures utilized in data mining which are adapted onto the domain of intrusion detection. There are described measures assuming prediction of input records into two classes – intrusive and legitimate – as well as prediction into more classes specifying particular subclass of intrusion or legitimate traffic. The interpretation of performance measures will be depicted by utilization of confusion matrices having specified format, which will be respected in data mining experiments performed in later chapters. Next, we formally define k-fold cross validation of a classifier which is employed as estimation of defined performance measures. The following definitions of performance measures are aimed at performance evaluation of IDSs regardless of whether they are based on signature or anomaly intrusion detection.

3.3.1 Binominal Prediction

Usually, true positive rate (TPR) and false positive rate (FPR) are used for performance measurement of binominal prediction, while positive representatives (P) denote intrusions and negative ones (N) denote legitimate traffic records. TPR – also known as detection rate (DR) or sensitivity or recall – is defined as the number of correctly predicted intrusion records divided by the total number of intrusion records (3.79). FPR – also known as fall-out or false alarm rate (FAR) – is defined as the number of normal records that are incorrectly predicted as intrusions divided by the total number of normal records (3.80). The true negative rate (TNR) – also known as specificity – is defined as the number of normal records that are correctly predicted divided by the total number of normal records (3.81). The positive predictive value (PPV) – also known as the precision – is defined as the number of correctly predicted intrusion records divided by the number of all records that are predicted as intrusions (3.82). The false discovery rate (FDR) is defined as the number of normal records that are incorrectly predicted as intrusions divided by the total number of records which are predicted as intrusions (3.83). The negative predictive value (NPV) is defined as the number of normal records that are correctly predicted divided by the number of records that are predicted as normal (3.84). The following equations depict these measures:

$$Recall = TPR = \frac{TP}{P} = \frac{TP}{TP+FN}, \tag{3.79}$$

$$Fall\text{-}out = FPR = \frac{FP}{N} = \frac{FP}{FP+TN}, \tag{3.80}$$

$$Specificity = TNR = \frac{TN}{N} = \frac{TN}{FP+TN} = 1 - FPR, \tag{3.81}$$

$$Precision = PPV = \frac{TP}{TP+FP}, \tag{3.82}$$

$$FDR = \frac{FP}{FP+TP} = 1 - PPV, \tag{3.83}$$

$$NPV = \frac{TN}{TN+FN}, \tag{3.84}$$

where TP, TN, FP, FN are the numbers of true positives, true negatives, false positives and false negatives, respectively. FPR and TNR measures have inverse relationship, therefore we will consider only TNR measure. The same fact is present in the case of FDR and

PPV, therefore we will consider only PPV measure. The accuracy of binominal prediction is defined as

$$Accuracy^B = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{P + N}. \tag{3.85}$$

Another commonly used measure is F-measure which is defined in the following equation [171]:

$$F\text{-}measure_\beta \sim F_\beta = \frac{(\beta^2 + 1) \times Recall \times Precision}{\beta^2 \times Precision + Recall}. \tag{3.86}$$

The F-measure is evenly balanced when $\beta = 1$. It favors recall when $\beta > 1$, and precision when $\beta < 1$. Because in intrusion detection problem are both cases – correct prediction of intrusions as well as legitimate traffic – equally important, we utilized $\beta = 1$ in our performance measurements, and therefore the F-measure is computed by formula

$$F_1 = \frac{2 \times Recall \times Precision}{Precision + Recall}. \tag{3.87}$$

Figure 3.6 shows 3D plot for F-measure for $\beta = 1$, while other two examples of F-measure for $\beta = 0.5$ and $\beta = 2$ are shown in Appendix A.

The definition of confusion matrix intended for displaying of binominal prediction results is depicted in Table 3.1. Note that we will refer both TNR and TPR as recall of associated class. Similarly, we will refer to NPV and PPV as precision of particular class.

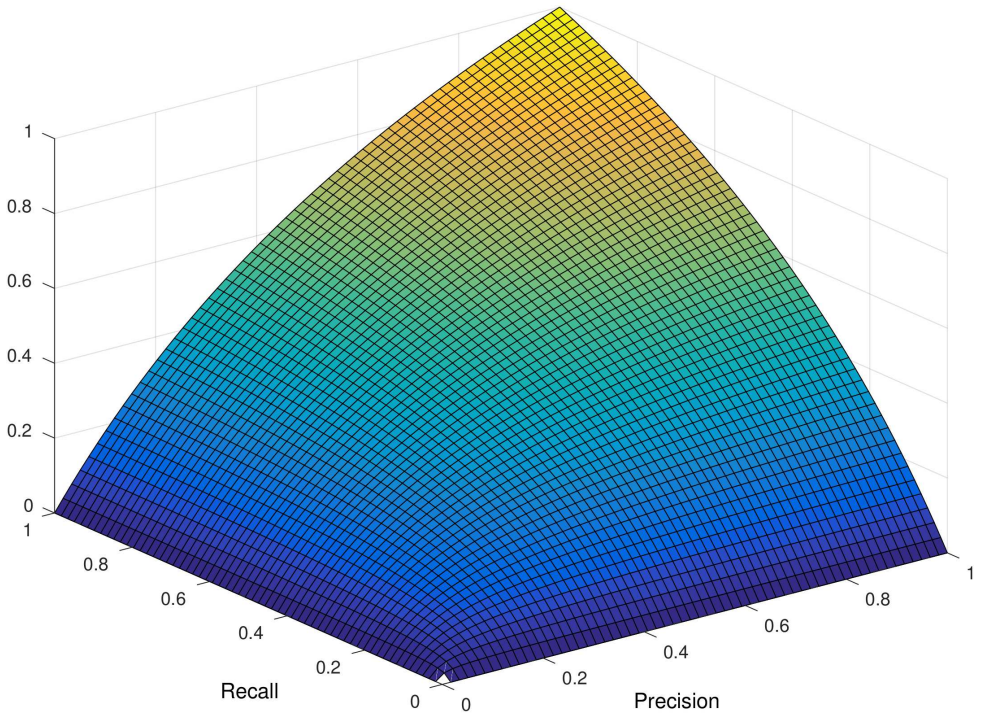

Figure 3.6: F-measure for $\beta = 1$

$Accuracy^B$		True Class		Precision
		Negative	**Positive**	
Predicted Class	**Negative**	TN	FN	NPV
	Positive	FP	TP	PPV
	Recall	TNR	TPR	F_1

Table 3.1: The definition of confusion matrix for binominal prediction

3.3.2 Multi-class Prediction

We have adapted performance measures of binominal prediction for the purpose of multi-class prediction which will be utilized in our experiments predicting subclass of intrusion as well as legitimate traffic. Therefore, following definitions will not distinguish between intrusion and legitimate classes, as it will be let on the interpretation of each class by human being.

Suppose having classes $C_0, \ldots, C_n$ where $n \in \aleph_1$. Then, we define the precision, recall and F-measure for class C_i as

$$Precision(C_i) \quad = \quad PPV(C_i) = \frac{TP(C_i)}{TP(C_i) + FP(C_i)}, \tag{3.88}$$

$$Recall(C_i) \quad = \quad TPR(C_i) = \frac{TP(C_i)}{|C_i|} = \frac{TP(C_i)}{TP(C_i) + FN(C_i)}, \tag{3.89}$$

$$F\text{-}measure(C_i) \quad \sim \quad F_1(C_i) = \frac{2 \times Recall(C_i) \times Precision(C_i)}{Precision(C_i) + Recall(C_i)}, \tag{3.90}$$

where class C_i represents positive records and classes $\left(\bigcup_{j=0}^{n} C_j\right) \setminus C_i$ represent negative ones. The accuracy of multi-class prediction is defined by the following formula:

$$\begin{aligned} Accuracy^P = & \frac{\sum_{i=0}^{n} TP(C_i)}{\sum_{i=0}^{n} TP(C_i) + \sum_{i=0}^{n} FN(C_i)} = \\ = & \frac{\sum_{i=0}^{n} TP(C_i)}{\sum_{i=0}^{n} (TP(C_i) + FN(C_i))}, \end{aligned} \tag{3.91}$$

where $n \in \aleph_1$ and the number of classes is equal to $n + 1$.

The confusion matrix for multi-class prediction is shown in Table 3.2 which considers definitions (3.88), (3.89), (3.90) and (3.91). We remark that depicted multi-class confusion matrix distinguish only between true positives and false negatives from the C_i perspective,

for $i \in (0, \ldots, n)$. True positives of all classes lay on the main diagonal of the confusion matrix. False negatives lay on all remaining positions and are interpreted vertically. Thus,

$$FN(C_i) = \sum_{j=0}^{n-1} FN^j(C_i). \tag{3.92}$$

$Accuracy^P$		**True Class**					
		$\boldsymbol{C_0}$	$\boldsymbol{C_1}$	$\cdots$	$\boldsymbol{C_n}$	$\boldsymbol{PPV(C_i)}$	$\boldsymbol{F_1(C_i)}$
Predicted Class	$\boldsymbol{C_0}$	$\boldsymbol{TP(C_0)}$	$FN^0(C_1)$	$\cdots$	$FN^0(C_n)$	$PPV(C_0)$	$F_1(C_0)$
	$\boldsymbol{C_1}$	$FN^0(C_0)$	$\boldsymbol{TP(C_1)}$	$\vdots$	$FN^1(C_n)$	$PPV(C_1)$	$F_1(C_1)$
	$\vdots$	$\vdots$	$\cdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	$\boldsymbol{C_n}$	$FN^{n-1}(C_0)$	$FN^{n-1}(C_1)$	$\cdots$	$\boldsymbol{TP(C_n)}$	$PPV(C_n)$	$F_1(C_n)$
$\boldsymbol{Recall(C_i)}$		$TPR(C_0)$	$TPR(C_1)$	$\cdots$	$TPR(C_n)$		

Table 3.2: The definition of confusion matrix for multi-class prediction

Another interpretation of multi-class confusion matrix, considering only C_k class as positive, says other entries than $TP(C_k)$ and $FN(C_k)$ are considered as $TN(C_k)$ in the case they are classified as non-C_k class and $FP(C_k)$ in the case they are incorrectly classified as C_k class, respectively. This interpretation is depicted in Table 3.3 which is equivalent to confusion matrix from Table 3.2.

$Accuracy^P$		**True Class**					
		$\boldsymbol{C_0}$	$\boldsymbol{C_k}$	$\cdots$	$\boldsymbol{C_n}$	$\boldsymbol{PPV(C_i)}$	$\boldsymbol{F_1(C_i)}$
Predicted Class	$\boldsymbol{C_0}$	$TN_0^0(C_k)$	$FN^0(C_k)$	$\cdots$	$TN_{n-1}^0(C_k)$	$PPV(C_0)$	$F_1(C_0)$
	$\boldsymbol{C_k}$	$FP_0(C_k)$	$\boldsymbol{TP(C_k)}$	$\vdots$	$FP_{n-1}(C_k)$	$PPV(C_k)$	$F_1(C_k)$
	$\vdots$	$\vdots$	$\cdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	$\boldsymbol{C_n}$	$TN_0^{n-1}(C_k)$	$FN^{n-1}(C_k)$	$\cdots$	$TN_{n-1}^{n-1}(C_k)$	$PPV(C_n)$	$F_1(C_n)$
$\boldsymbol{Recall(C_i)}$		$TPR(C_0)$	$TPR(C_k)$	$\cdots$	$TPR(C_n)$		

Table 3.3: The alternative definition of confusion matrix for multi-class prediction

Note that following sums depict some relations in the current multi-class confusion matrix:

$$TN(C_k) = \sum_{j=0}^{n-1}\sum_{h=0}^{n-1} TN_j^h(C_k), \tag{3.93}$$

$$FN(C_k) = \sum_{h=0}^{n-1} FN^h(C_k), \tag{3.94}$$

$$FP(C_k) = \sum_{j=0}^{n-1} FP_j(C_k), \tag{3.95}$$

where indexes h and j iterate trough rows and columns of confusion matrix, respectively.

3.3.3 K-fold Cross Validation of a Classifier

The k-fold cross validation [92] splits dataset D containing n samples into k mutually exclusive subsets (the folds) $D = \{D_1, D_2, \ldots, D_k\}$ having approximately equal sizes. The classifier is trained and tested k times, whereas each time $t \in \{1, 2, \ldots, k\}$ is trained on $D \setminus D_t$ and tested on D_t. Consider definitions (3.1), (3.2), (3.3) and (3.4) from Subsection 3.1 with D_{tr} symbol replaced by symbol D. The cross validation estimate of accuracy represents the total number of correct predictions, divided by the number of samples in the dataset – n. Consider D_t being the test set which includes samples $x_i = (v_i, y_i)$, then the k-cross validation estimate of accuracy and recall are defined as

$$Accuracy = \frac{1}{n} \sum_{t=1}^{k} \sum_{(v_i, y_i) \in D_t} \delta_2\Big(A(D \setminus D_t, v_i), y_i\Big), \tag{3.96}$$

$$Recall(y) = \frac{1}{n_y} \sum_{t=1}^{k} \sum_{(v_i, y_i) \in D_t} \delta_3\Big(A(D \setminus D_t, v_i), y_i, y\Big), \tag{3.97}$$

where $\delta_2(a, b) = 1$ if $a = b$ and 0 otherwise; $\delta_3(a, b, c) = 1$ if $a = b = c$ and 0 otherwise; n_y represents the number of samples in D labeled as y. Analogously, estimates of precision and F_1-measure are defined as

$$Precision(y) = \frac{1}{k} \sum_{t=1}^{k} \frac{\displaystyle\sum_{(v_i, y_i) \in D_t} \delta_3\Big(A(D \setminus D_t, v_i), y_i, y\Big)}{\displaystyle\sum_{(v_i, y_i) \in D_t} \Big[\delta_3\Big(A(D \setminus D_t, v_i), y_i, y\Big) + \widehat{\delta}_3\Big(A(D \setminus D_t, v_i), y_i, y\Big)\Big]}, \tag{3.98}$$

$$F_1\text{-}measure(y) = \frac{2 \times Recall(y) \times Precision(y)}{Precision(y) + Recall(y)}, \tag{3.99}$$

where $\widehat{\delta}_3(a, b, c) = 1$ if $a = c \wedge a \neq b$ (false positives) and 0 otherwise. These measures are dependent on the distribution into folds as well as on the number of samples in each class of a dataset.

Chapter 4

State of the Art

The related work discussed in this chapter is not limited only to supervised machine learning in network intrusion detection, however approaches to the unsupervised machine learning are described here, too. The problem of machine learning based network intrusion detection is supplemented by several network traffic classification approaches which we consider related as well. These two problems are very similar from the view of supervised prediction, which considers the same input features to perform traffic type classification or attack detection with potential attack type distinction.

At first, we describe data mining task in network intrusion detection and principles of buffer overflow attacks as the cause of network intrusions. Then, various datasets utilized for IDS evaluation purposes will be discussed and categorized. Next, classification of network anomaly intrusion detection approaches, according to input data they work with, is proposed. Later, relevant network anomaly intrusion detection and network traffic classification approaches are introduced. Finally, related work which is relevant to network attack obfuscations as well as evasion of IDS is discussed, whereby an emphasis is put on various obfuscation techniques.

4.1 Data Mining Task in Intrusion Detection

Data mining represents relatively new approach for the area of intrusion detection. Data mining was defined by Grossman et al. in the paper [62] as semi-automatic discovery of patterns, associations, changes, anomalies, rules, and statistically significant structures and events in a data. Today, many different types of data mining algorithms exist which include classification, link analysis, clustering, association, rule abduction, deviation analysis and sequence analysis. By using the data mining algorithms, it is possible to extract knowledge from large datasets, analyse them and integrate it into a trained intrusion detection model. This approach considers the intrusion detection as data analysis process, whereas the signature and anomaly based approaches represent knowledge engineering processes [134].

Subcategory of intrusion detection approaches utilizing machine learning, is network based one. Figure 4.1 depicts data mining approach of building network anomaly intrusion detection models [98].

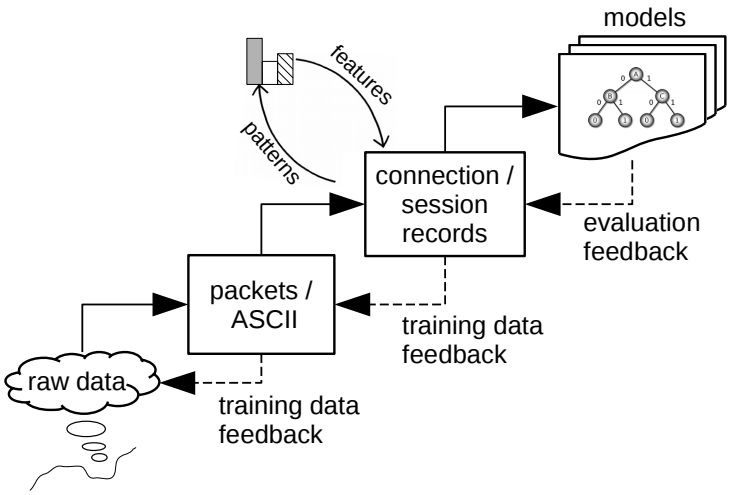

Figure 4.1: Data mining approach of building intrusion detection models [134]

First, raw data is converted into ASCII format of network packets information, which is next processed into connection level information. These connection level records contain features like service, duration, size etc. Data mining algorithms are applied to this data to create models of intrusion detectors. Many of the state-of-the-art approaches aim at optimization of models or feature set, and do not care how training dataset was collected. Optimization of feature set is represented by evaluation feedback in the figure. In our thesis, we primarily aim at different kind of optimization which resides in obtaining and providing specific training data for feature extraction, and thus for model training too. In Figure 4.1, this kind of optimization is represented by training data feedback.

4.2 Buffer Overflow Flaws as a Cause of Intrusions

Buffer overflow (BO) attacks exploiting the network vulnerabilities continue to be the one of the most dangerous threats in the domain of information security. Figure 4.2 shows increasing number of vulnerabilities prone to buffer errors and Figure 4.3 shows the ratio of buffer-error vulnerabilities to total vulnerabilities per year [125]. This class of attacks forms non-negligible portion of all security attacks simply because BO vulnerabilities are very easy to perform [25, 82, 126].

Buffer Overflow Attacks & Vulnerabilities

The following information related to BO vulnerabilities and attacks were primarily taken from [42]. BO attacks dominate in the class of remote penetration attacks because a BO vulnerability presents for the attackers exactly what they need: the ability to inject and execute an attack code. The injected attack code runs with the privileges of the vulnerable application and allows the attacker to bootstrap whatever other functionality in order to control the compromised PC. The goal of the BO attack is to subvert the function of a program in a way that the attacker can take control of that program, and if a program is running under user with sufficient privilege, then adjacent host can be controlled as well. In order to achieve this goal, the attacker must achieve two sub-goals:

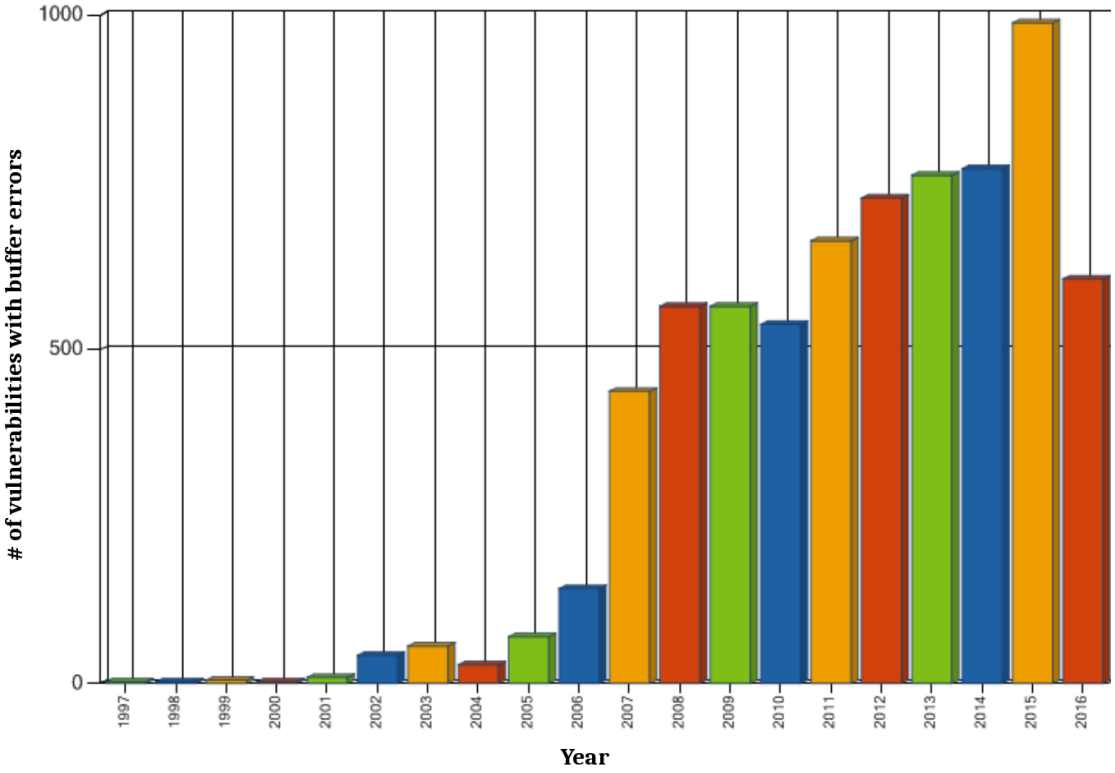

Figure 4.2: The number of vulnerabilities with buffer errors per year (generated at [125] on 21^{st} of June 2016)

1. arrange suitable code to be available in the program's address space,
2. force the program to jump to that code, with suitable parameters loaded into the registers and memory.

There are two ways of arranging the attack code to be in victim program's address space: *inject* it or *use suitable existing code* if it is present. In the case of injecting, the attacker provides the string as the input of the program, which stores it into a buffer. Actually, the string contains native CPU instructions of victim host's platform. The attacker does not need to overflow any buffer to do this. The buffer can be located anywhere on the stack, the heap or the static data area. In the case that suitable code is already in program's code, the attacker only need to parametrize this code by argument specifying e.g. shell command, and then ensure jumping of process control flow to this code.

Forcing the program to jump on attacker's code can be done by overflowing a buffer which has weak or non-existent boundary checking. By overflowing the buffer, the attacker can overwrite the program state with (almost) arbitrary sequence of bytes, resulting into bypassing program logic of the victim. This is referred to as corrupting the program state. The distinguishing factor among BO attacks is a kind of corrupted state as well as the location of corrupted state in the memory, which can be divided into three categories:

- **activation record** – whenever is a function called, it pushes activation record, which is also known as the *stack frame*. The stack frame contains, among other things, the

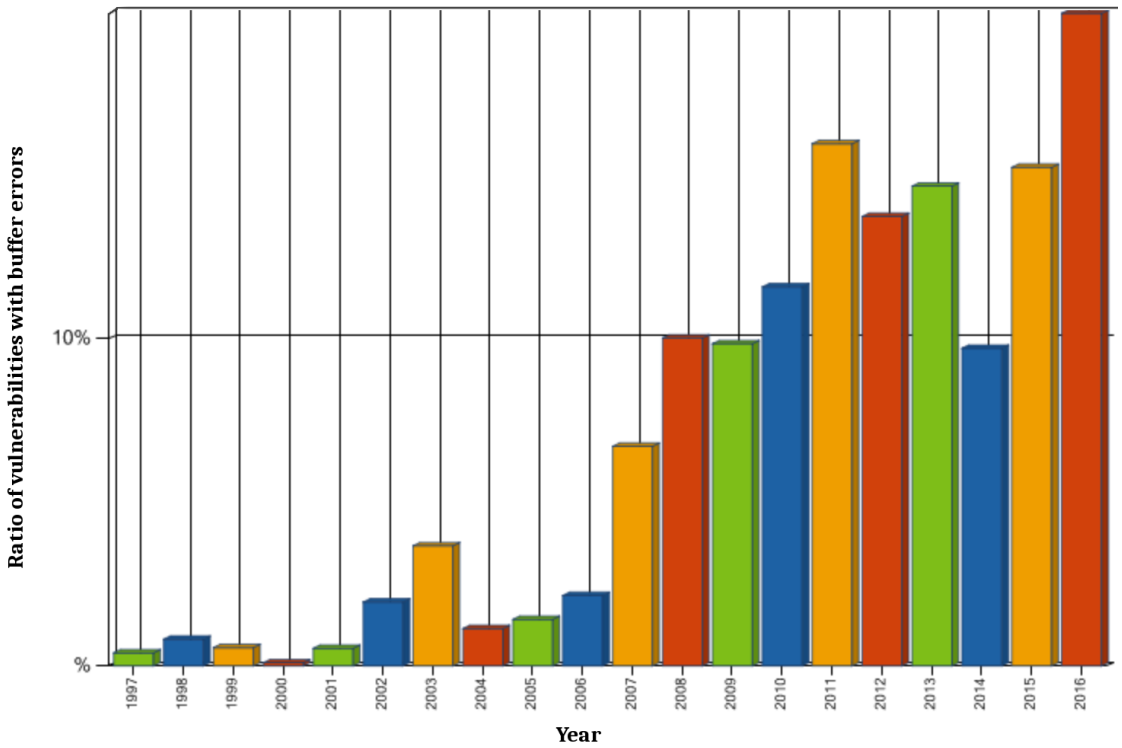

Figure 4.3: Ratio of vulnerabilities with buffer errors per year (generated at [125] on 21^{st} of June 2016)

return address of the function, which is loaded into instruction pointer register of the CPU when function finishes, and further program jumps onto that address. Attacks which corrupt activation record of a function, and thus return address, overflow local variables of a function. The attacker then forces the program to jump on attack code, when victim function returns. This kind of buffer overflow is called *stack smashing attack* [126, 41, 169],

- **pointer on a function** – it can be allowed anywhere on the stack, heap, area of the static data, so the attacker only need to find suitable vulnerable buffer near a function pointer and overflow it in order to change the value of a pointer. Later, when the program makes the call to this function pointer, it will instead jump into the attacker's desired location,

- **long jump buffer** – contains address of the code as well as state of the process and serves for recovering process state by calling *longjmp* function which returns process control on the address set by *setjmp* function, and recovers state of the process. Alike function pointers, long jump buffers can be allocated anywhere, so the attacker only needs to find adjacent vulnerable buffer and overflow its content by desired address. Note that this is the *C* language specific category, as *C* includes specific checkpoint/rollback system called *setjmp/longjmp*, which is the equivalent to the concept of exception handling mechanism [104] of newer programming languages than *C*.

Combining Code Injection and Control Flow Corruption

The simplest and the most common form of BO attack combines an injection technique with activation record corruption in single string. In this case, the attacker locates local vulnerable variable of a function, feeds it with large string containing executable code and simultaneously changes return address of the activation record [126].

The injection and corruption do not have to necessarily happen in one step. The attacker can inject code into one buffer without overflowing it, and overflow a different buffer to corrupt a code pointer. This often happens when vulnerable buffer has boundary checking but performs it in wrong way, so the buffer can be overflowed up to certain number of bytes.

The next combination of injection with corruption happens in the case when the attacker is trying to use present code, which is necessary to parametrize, instead of injecting his own code. An example is the function *exec(param)* in *libc*, where attacker need to modify parameter *param* in order to execute desired system's utility, e.g. */bin/sh*.

The Defense against Buffer Overflow

Several approaches for defending against BO vulnerabilities and attacks exist. The first one is to write correct code, which is often impossible due to human factor, and can be expensive and time consuming in the case of utilization of tools for static and dynamic analysis. The next option is to make storage areas of buffer non-executable, which prevents the attacker from injecting malicious code. But, as attacker does not need to necessarily insert his own code and may exploit existing one, this method has limitations which allow existence of vulnerabilities. Another option is to perform checking of array boundaries on each access to array, which could be accomplished by compiler, and thus it is called *direct compiler* approach. Although, this method completely eliminates BO vulnerabilities, it imposes substantial costs. The *indirect compiler* approach performs integrity checks on code pointers before dereferencing them. While, indirect compiler approach does not make BO attacks impossible, it stops the most of BO attacks, and the attacks that it does not stop are difficult to create. The newest method of almost completely alleviating the BO attacks is memory randomization [184]. However, successful performing of BO attack is not impossible, but requires a large number of attempts.

4.3 IDS Evaluation Datasets

In this section, we summarize basic properties and characteristics of public IDS evaluation datasets. We propose partition of datasets into two categories. The first one represents datasets containing raw network dumps and the second one represents datasets containing high level features extracted upon underlying network dumps as well as upon any other relevant resources of data. These two categories are closely discussed in the following subsections.

4.3.1 Datasets Consisting of Network Data

Datasets, from the category of datasets, consisting of network data for evaluation of intrusion detection methods has one property in common – they contain network data dumps with optional data serving for labeling purpose. No high level features extracted upon network and host data is available. This fact make the category more challenging than the category of datasets consisting of high level features.

The first 4 representatives of this category are large collections of datasets and are referred to as projects – MWS [68], PREDICT [137], CAIDA [27], NETRESEC [121]. The following 4 examples of the category represent just one specific collection of network data and are referred to as datasets – DARPA [50], CCRC [110], CDX [164], CONTIAGO [39].

Project MWS

The MWS Datasets are a collection of various types of datasets that are designed for use in anti-malware research [68]. Authors assumed three types of researchers, when they collected the first version of the datasets: a) highly-professional on malware analysis; b) packet analysis expert such as intrusion detection; c) entry level researcher who is unfamiliar with malware analysis and / or packet analysis.

The authors have shared a summary of the MWS Datasets in Japanese [4, 69, 70, 71, 87] which covers three phases of attacks:

- 1) probing,
- 2) infection,
- 3) malware activities after infection.

These three phases of attacks with associated MWS collections and analysis methods are shown in Figure 4.4.

Cyber Clean Center (**CCC**) dataset consists of a malware sample, honeypot packet trace, and malware collection log. The dataset was collected from server-side, high-interaction honeypots operated by the CCC in a distributed manner. Over a hundred of honeypots gathered attacks and collected malware through multiple ISPs. These honeypots were based on Windows 2000 and Windows XP SP1 virtual machines.

The **MARS** (Malware / Minimal-attack Analysis Result Set) [114], is a set of dynamic and static analysis data for the malware samples of the CCC dataset 2008, 2009, and 2010. One part of the metadata contains the hash digest, file name, file size, file type, detection method name, and timestamp of the anti-virus software, as well as the version of the anti-virus software with regards to a particular malware sample. The next is a reference list of each analysis result file with its analysis time stamp and tool information.

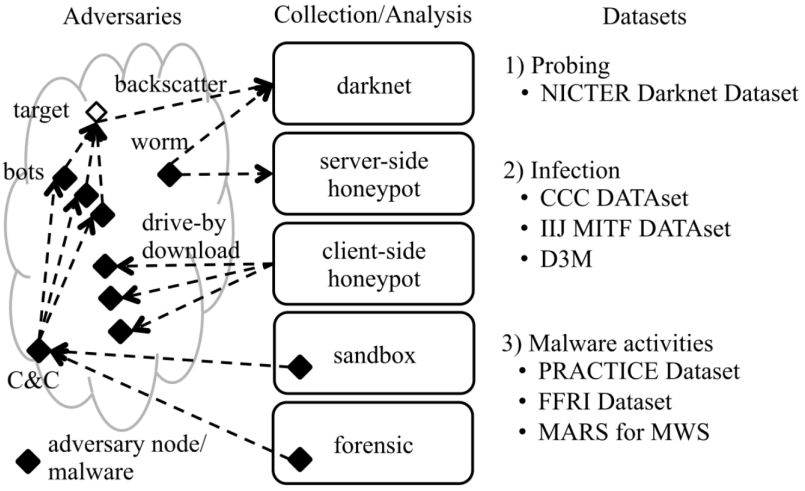

Figure 4.4: Attack phases of malware applicable to the MWS Datasets [68]

Drive-by Download Data by Marionette (**D3M**) dataset is a set of packet traces collected from the web client based high-interaction honeypot system – Marionette [3, 5], which is based on Internet Explorer with several vulnerable plugins, such as Adobe Reader, Flash Player, WinZip, QuickTime and Java. The datasets contain packet traces for the two periods: at infection and after infection.

The **IIJ MITF** dataset is collected by server-side, low interaction honeypots based on the open source honeypot – Dionaea [55]. This dataset contains attack communication and malware collection logs from a hundred honeypots between July 2011 and April 2012 in order to discover the trends of bot and bot nets. Deployments of honeypots were realized at one ISP.

The **PRACTICE** dataset (associated to Proactive Response Against Cyber-attacks Through International Collaborative Exchange) contains the packet traces obtained during long-term dynamic analysis of five malware samples (Zbot , SpyEye , etc.) and their metadata. The associated project's approach focuses on the long-term network activity of malware, using dynamic analysis system [9] connected to the Internet. The analysis period of the dataset is one week in the middle of May 2013.

The **FFRI** dataset focuses on the internal activities that occurred at a host by influence of malware and are generated by dynamic analysis systems – Cuckoo sandbox [63] and FFR yarai Analyzer Professional [59].

The **NICTER** darknet dataset is a set of packet traces collected since April 2011 to 2014 using the darknet monitoring system – NICTER [81]. The packet traces contain scan packets to explore the reachable hosts by worms and researchers, backscatter packets caused by source IP address spoofing, distributed reflection denial of service (DRDoS) attacks using DNS and NTP, etc. A featured difference between this dataset and others is the ability for researchers to access past and real-time data using the NONSTOP [180], Platform as a Service (PaaS) environment, which provides an isolated environment where users can remotely access and analyze various information resources of security related data.

From the perspective of network intrusion detection evaluation and current category of datasets consisting of network data, we consider as related following datasets of MWS: PRACTICE, D3M, CCC. As a source of expert knowledge in network intrusion detection problem could be used following datasets of MWS: FFRI, IIJ MITF, D3M, CCC.

Project PREDICT

Protected Repository for the Defense of Infrastructure Against Cyber Threats (PREDICT) [137] shares 430 datasets in 14 categories contributed by several data providers. Researchers in the USA and other selected countries are approved for creating accounts and accessing the repository. PREDICT provides sharing approval mechanism which distinguishes 5 data classes: a) Unrestricted (click-through agreement); b) Quasi-Restricted (click-through agreement); c) Unrestricted Non-Commercial (click-through agreement); d) Quasi-Restricted Non-Commercial (click-through agreement); and e) Restricted (formal written agreement is necessary).

From the all 14 categories, just three of them are relevant to the network intrusion detection and could be used for evaluation purposes:

- **Blackhole Address Space Data** – is collected by monitoring routed but unused IP address space that does not host any networked devices. Systems that monitor such unoccupied address space have a variety of names, including darkspace, darknets, network telescopes, blackhole monitors, sinkholes, and background radiation monitors.

Packets observed in the darkspace can originate from a wide range of security-related events, such as scanning in search of vulnerable targets, backscatter from spoofed denial-of-service attacks, automated spread of Internet worms or viruses, etc. The related subcategory of this category is UCSD Archived Network Telescope Data. The archived files are in PCAP format. Source IP addresses are not anonymized.

- **IP Packet Headers** – these datasets are comprised of headers of network data, containing information such as anonymized source and destination IP addresses and other IP and transport header fields. No packets' content is included. Depending on the specific dataset, this category of data can be used for characterization of typical internet traffic, or of traffic anomalies such as distributed denial of service attacks, port scans, or worm outbreaks.

- **Synthetically Generated Data** – are generated by capturing information from a synthetic environment, where benign user activity and malicious attacks are emulated by computer programs. In this category, full network packets as well as firewall logs, application logs, and malicious attacks are available, without any risk of compromising the privacy of real people. In this category, one can know and document complete "ground truth," i.e., which traffic is benign and which traffic is malicious. Therefore, this category is well suited for evaluation of NIDS systems.

Notice, that **IDS and Firewall Data** category contains large collection of logs submitted in a standard format but generated from a diverse set of hardware and software systems. It does not contain any PCAP files, therefore it could not be used for IDS evaluation purpose. If we would consider categorization of datasets from 4.3.1, then emphasized datasets of PREDICT would represent probing and infection categories.

Project CAIDA

Center for Applied Internet Data Analysis (CAIDA) [27] collects several different network data types at geographically and topologically diverse locations, and makes this data available to the research community while preserving the privacy of individuals and organizations who donate data or network access.

The CAIDA datasets are dived into three categories which reflect collection status:

- **A) Ongoing** – the data collection for such dataset is still active and has continuing, regularly scheduled collections,

- **B) One-time snapshot** – the dataset comes from a single collection event that only occurred once. Future events will have a different dataset names,

- **C) Complete** – a formerly ongoing data collection that is finished, and will not be resumed.

From the network security and intrusion detection perspective, CAIDA includes datasets containing e.g. DDoS attacks [28, 31], botnet traffic [49], dumps of various well known worms (Conficker [29], Code-Red [30], Witty [32]). These datasets could be utilized for evaluation of intrusion detection approaches after little analysis followed by labeling if it is not available. If we would consider categorization of datasets from 4.3.1, then CAIDA would represents probing and infections categories.

Project NETRESEC

Network Forensics and Network Security Monitoring (NETRESEC) [121] is independent software vendor with focus on the network security field. Netresec specializes in software for network forensics and analysis of network traffic. It additionally maintains a comprehensive list of publicly available PCAP files which can be used for evaluation of intrusion detection approaches as well.

The datasets are divided into six categories:

- **A) Cyber Defence Exercises** – this category includes network traffic from exercises and competitions, such as Cyber Defense Exercises and red-team/blue-team competitions.
- **B) Capture the Flag Competitions** – it contains files from capture-the-flag (CTF) competitions and challenges.
- **C) Malware Traffic** – it contains pcap files of captured malware traffic from honeypots, sandboxes or real world intrusions.
- **D) Network Forensics** – Network forensics training, challenges and contests.
- **E) SCADA/ICS Network Captures** - files with attacks against Industrial Control Systems; files captured at Industrial Control System Village (4SIC, CTF, DEF CON 22).
- **F) Uncategorized PCAP Repositories** – various captures which often represents data for intrusion detection purposes.

If we would consider categorization of datasets from 4.3.1, then NETRESEC datasets would represent probing and infection categories.

DARPA 1998 and 1999 Datasets

The Cyber Systems and Technology Group [50] (formerly the DARPA Intrusion Detection Evaluation Group) of MIT Lincoln Laboratory, under Defense Advanced Research Projects Agency (DARPA ITO) and Air Force Research Laboratory (AFRL/SNHS) sponsorship, has collected and distributed the first standard corpora for evaluation of computer network intrusion detection systems. They have also coordinated, with the Air Force Research Laboratory, the first formal, repeatable, and statistically significant evaluations of intrusion detection systems. These evaluation efforts have been carried out in 1998 and 1999.

There were collected two datasets DARPA 1998 and 1999 which were intended for statistically significant evaluations of intrusion detection systems. Later, there were released three datasets marked as DARPA 2000 which address specific network scenarios.

According to [186] DARPA dataset contains all traffic generated by software that is not publicly available and hence it is not possible to determine how accurate the background traffic inserted into the evaluation is. Also evaluation criteria do not account for system resources used, ease of use, or what type of system it is. If we would consider categorization of datasets from 4.3.1, then DARPA datasets would represent probing and infection categories.

CCRC 2006 Dataset

The authors F. Massicotte et al. developed a framework for automatic evaluation of intrusion detection systems in their paper [110] as well they collected example dataset consisted of many of network attack simulations. Moreover, the authors made the dataset publicly available for IDS research community and can be obtained by sending emails to authors. We denote this dataset as CCRC 2006, because the main author was, at the time of article was written, an employee of Canada Communication Research Center in Ottawa.

The dataset is specific to signature-based, network intrusion detection systems and contains only well known attacks, without background traffic. The purpose of the dataset is testing and evaluation of the detection accuracy of IDS in the case of successful and failed attack attempts. The paper also reports an initial evaluation of the framework on two well-known IDS, namely SORT [160] and Bro [133].

The results are encouraging as the authors are able to automatically generate a large, properly documented dataset which showed itself to be able automatically test and evaluate intrusion detection systems. Notice, that the framework also contains mutation layer which is able to perform various L2 and L3 protocol based obfuscations using tools Fragroute [174] and Whisker [140]. If we would consider categorization of datasets from 4.3.1, then CCRC 2006 dataset would represent infection category.

CDX 2009 Dataset

The CDX 2009 dataset [164] was created during network warfare competition, in which one of the goal was to generate labeled dataset. By labeled dataset authors mean TCP dump traces of all simulated communications and SORT [160] log with information about occurrences of intrusions and other security incidents. It contains data capture by NSA, data capture outside of west point network border (in TCP dump format) and SNORT intrusion prevention log as relevant sources for any experiments. Network infrastructure contained 4 servers with 4 vulnerable services (one per each server). These services with IP addresses of their hosted servers are described in Table 4.1. The expert knowledge of the dataset is provided by SNORT tool in the standard exported log format of the SNORT.

Service	OS	Internal IP	External IP
Postfix Email	FreeBSD	7.204.241.161	10.1.60.25
Apache Web Server	Fedora 10	154.241.88.201	10.1.60.187
OpenFire Chat	FreeBSD	180.242.137.181	10.1.60.73
BIND DNS	FreeBSD	65.190.233.37	10.1.60.5

Table 4.1: List of CDX 2009 vulnerable servers

If we would consider categorization of datasets from 4.3.1, then CDX 2009 dataset would represent probing and infection categories.

Contagio Dataset

Contagio dataset [39] contains collection of PCAP files from malware analysis. The authors collected several malicious and exploit PCAPs (almost 1000) from various public sources and stated the references on original providers or authors. The collection is irregularly updated with new PCAP files. PCAPs in Contagio dataset include implicit expert knowledge about

occurrence of malware/exploit and therefore, is useful for IDS and signature testing and development, general education, and malware identification.

If we would consider categorization of datasets from 4.3.1, then Contagio dataset would represent infection and malware activities categories.

4.3.2 Datasets Consisting of High Level Features

The category of datasets consisting of high level features contains representatives which were built upon network traffic dumps as well as upon any other data originating from host machines. It does not contain any raw network traffic dumps. This category can be interpreted as preprocessed version of the former category and optionally may include other data than ones originating from network traffic. Considering the fact that datasets are preprocessed and might be built upon manifold input data, we can a priori designate it as less challenging in prediction task than former category, and also as enabling testing IDSs to provide more plausible and precise results.

The current category of datasets contains 5 representatives - KDD Cup '99 [187], NSL KDD '99 [181], Moore's 2005 [119], Kyoto 2006+ [173] and OptiFilter 2014 [163].

KDD Cup '99

In 1999, KDD Cup '99 [187] dataset was created and is based on the DARPA 1998 dataset of network dumps. It has been used for evaluating of new intrusion detection methods based on analyzing of features extracted from network traffic and host machines. The training dataset consists of approximately 4,9 mil. single connection vectors from seven weeks of network traffic, each labeled as either normal or attack, containing 41 features per connection record (see Appendix B for full list of features). Similarly, the two weeks of testing data yielded around two million connection records. The datasets contain a total number of 24 training attack types, with an additional 14 types in the testing dataset.

The simulated attacks fall into four main categories [187, 181]:

- **Denial of Service Attack (DOS)**: is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine, e.g. syn flood,
- **Remote to Local Attack (R2L)**: occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine, e.g. guessing password, remote buffer overflow attacks,
- **User to Root Attack (U2R)**: is a class of attacks where the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack or social engineering) and then, is able to exploit some vulnerability to gain superuser access to the system, e.g. various local buffer overflow attacks,
- **Probing**: is an attempt to gather information about a network of computers for the purpose of circumventing its security controls, e.g. port scanning for vulnerable services.

The features of the KDD '99 dataset are, according to [187], divided into three categories:

- **A) basic features** of individual TCP connections. This category encapsulates all the attributes that can be extracted from a TCP/IP connections. Most of the features lead to an implicit delay in detection. For full list of features of this category see Appendix B.1.
- **B) content features** within a connection suggested by domain knowledge. Unlike most of the DoS and Probing attacks, the R2L and U2R attacks cannot be described by any intrusion frequent sequential pattern. This is because the DoS and Probing attacks involve many connections to some hosts in a very short period of time, however, the R2L and U2R attacks are embedded in the data portions of the packets associated with single connection. In order to detect these kinds of attacks, we need some features which would be able to look for suspicious behavior in the data portion, e.g. the number of failed login attempts. These features are called content features because extract payload of the packet regardless of whether it is ciphered or no and therefore, have to be extracted at host machines. For full list of features of this category see Appendix B.2.
- **C) traffic features** computed using a two-second time window interval (see Appendix B.3). This category of traffic features, moreover, contains two subcategories of features [177]:
  - **1) same host** features – examine only the connections in the past two seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc,
  - **2) same service** features – examine only the connections in the past two seconds that have the same service as the current connection.

  The same host and same service features are together called time-based traffic features of the connection records.

The paper [177] criticize time based features of the connection records which are computed using time window of 2 seconds. There exists several slow probing that scan host in using a much larger time interval than 2 seconds. Rather than using a time window of 2 seconds, the authors of [177] used a connection window of 100 connections, and constructed a mirror set of host-based traffic features replacing original time-based traffic features.

The dataset is criticized mainly because it does not seem to be similar to traffic in real networks, and also there are some critiques of attack taxonomies and performance issues. Since then, many researches have proposed new measures to overcome existing deficiencies of KDD Cup '99 dataset.

NSL KDD '99

Deficiencies of the KDD Cup 99 dataset were discused in [181]. The main deficiency of orginal dataset are redundant replicated entries (78% in training set and 75% in testing set). The next deficiency is non-uniform distribution of target classes resulting into very poor results achived by classification models, which use neural networks or SVM[1].

[1]Support Vector Machines. URL: http://www.support-vector.net/.

The original dataset was modified, reduced an repaired into proposed NSL KDD '99 dataset. Training dataset contains about 130 thousands of entries and testing about 23 thousands. In NSL KDD '99 dataset are data sorted into original 24 classes as well as into 2 classes – legitimate communication or attack. Complete NSL KDD '99 dataset is available at [182].

Moore's 2005

The 2005 Moore datasets [119] are intended to aid in the assessment of network traffic classification. A number of datasets are described; each dataset consists of a number of objects and each object is described by a group of features (also referred to as discriminators). Leveraged by a quantity of hand-classified data, each object within each dataset represents a single flow of TCP packets between client and server. Features for each object consist of processed input data by discriminators extraction functions and these features serves as the input to probabilistic classification techniques. Input data is obtained by Architecture of a Network Monitor designed in [116]. There are accessible public perl scripts[2] which performs TCP connection extraction and next they perform computation of discriminators from input data in TCP dump format.

In contrast to previously described KDD datasets, the Moore's dataset is based purely on network traffic dumps and there is not leveraged any information from host machines during extraction of the features.

Kyoto 2006+

The authors J. Song et al. in their paper [173] realized in 2011, that the only widely used dataset intended for evaluation of intrusion detection systems – KDD Cup '99, was more than 10 years deprecated these days and could not reflect current network situations and the latest attack trends. Therefore, they presented new evaluation dataset, called Kyoto 2006+, built on the 3 years of real traffic data (since Nov. 2006 to Aug. 2009) which were obtained from diverse types of honeypots. Furthermore, the authors provided detailed analysis results of honeypot data and shared their experiences in order to make security researchers able to get insights into the trends of latest cyber security attacks and the Internet situations.

The total number of honeypots used during collecting dataset, is 348 including two black hole sensors with 318 unused IP addresses. The most of honeypots were rebooted and restored original HDD image immediately after a malicious packet was observed. For inspection of captured traffic, the authors use three independent security SW: SNS7160 IDS system [179], Clam AntiVirus software [35] and Ashula [10] which is dedicated for shellcode detection purpose. Since April 1st 2010, the authors have deployed another IDS – SNORT [160].

The overall data distribution of the dataset is depicted in Table 4.2. During the observation period, there were over 50 millions of normal sessions and over 43 millions of attack sessions. The authors regarded all traffic data captured from their honeypots as attack data and all traffic data captured at their legitimate mail and DNS server as normal data. Also, among the attack sessions, there were observed over 425 thousands of sessions which were related to unknown attacks, because they did not trigger any IDS alerts, but they contained shellcodes detected by Ashula. There also occurred several situations, when IDS failed to

[2]URL: http://www.cl.cam.ac.uk/research/srg/netos/brasil/downloads/index.html.

	Number of Sessions	Avg. Number of Sessions per Day
Total	93,076,270	93,638
Normal	50,033,015	50,335
Known attack	42,617,536	42,874
Unknown attack	425,719	428

Table 4.2: Data distribution of Kyoto 2006+ dataset [173]

detect known attacks, i.e., false negative, but ratio of the failure was negligibly small. There was leveraged Bro IDS tool [133] for conversion of raw traffic data into session data.

The Kyoto 2006+ dataset consists of 14 statistical features derived from KDD Cup '99 dataset as well as 10 additional features which can be used for further analysis and evaluation of NIDSs. The reason why authors extracted just the 14 statistical features is that among the original 41 features of the KDD Cup 99 dataset [187] there exist substantially redundant and insignificant features, and also many of the KDD Cup '99 features are content based which makes them impossible to extract them by NIDS which do not use without domain knowledge. Therefore, content based features are kind of features which are not suitable for network based intrusion detection systems. The 14 features adopted from KDD Cup '99 are depicted in Appendix C.1.

Additionally, to above mentioned statistical features, the authors extracted additional 10 features which may enable them to investigate more effectively what kinds of attacks happened on their networks. Some of these features (the first four ones) reflects "ground thruth" and therefore can be considered as expert knowledge and should not be included into machine learning process as regular attributes, but special ones which represents labels of records. The list of additional features with their descriptions is depicted in the Appendix C.2. Kyoto 2006+ dataset is available to the public at [94].

OptiFilter 2014 – Persistent Dataset Generation

The authors Salem et al. proposed OptiFilter [163] – a framework which efficiently constructs connection vectors from online data flow. The framework collects network packets and host events continuously in real-time and parses them to a queue of dynamic windows, then it generates connection vectors accordingly. Datasets generated by the framework can be leveraged for evaluation of intrusion detection models.

OptiFilter handles ARP, ICMP, IP/TCP and IP/UDP protocols. Moreover, it utilizes a finite state machine on TCP and UDP connections to constantly monitor their state until a connection is closed or a certain condition is satisfied. As capturing method for network data, TCP dump tool [183] is utilized.

All host based features are collected using SNMP traps, a mechanism which allows systems to send messages to a trap receiver. Regarding Windows systems, the authors utilize the Windows Management Instrumentation (WMI) that is able to filter events and send them as SNMP traps via WMI-SNMP-Provider. In contrast, the Linux systems utilize syslog daemon to generate SNMP traps using the NetSNMP agent.

The extracted features of OptiFilter framework are influenced by KDD Cup 99 [187] and Kyoto 2006 [173] datasets and consit of three categories:

- **Network based features** – timestamp, source and destination IPs with ports, protocol type, service, transferred source Bytes, flags describing connection state (calculation from BRO [133]), the number of fragmentation errors.

- **Traffic features** – the features are statistical and get derived from the basic features. They are divided into two types, time-based traffic features and connection-based traffic features, which are distinguished and treated differently by OptiFilter. The former type is calculated based on a dynamic time window, e.g. the last 5 seconds, while the latter type is calculated on a configurable connection window, e.g. the last 1000 connections. Instances of this group are not self explanatory and not described in the original paper.

- **Content features** – the features are obtained directly from monitored host using SNMP. Instances are: the number of failed login attempts, indication of successful login, indication of obtaining root shell.

For the evaluation purpose, the authors selected 17 common services: ftp, ssh, telnet, smtp, smb, nfs, xmpp, http, ntp, dhcp, syslog, snmp, rdp, IMAP, pop3, rsync. But, it is not clear whether SecMonet – the dataset they generated – contains any malicious traffic. Experiments evaluating performance of supervised classifiers also include KDD Cup 99 dataset. Therefore, the result dataset of this work is KDD Cup 99 enriched by SecMonet probably containing only legitimate traffic.

4.4 Machine Learning in Anomaly Intrusion Detection

This section discusses research performed in the field of anomaly intrusion detection and machine learning. We distinguish two categories of anomaly intrusion detection approaches utilizing machine learning according to a type of input data they work with:

- **A)** the first category represents ***general anomaly intrusion detection techniques*** which intake all possible data sources including host-based features as well as network-based ones. We will discuss several examples of this category – [1, 79, 84, 90, 93, 108, 136] (see Section 4.4.1). This category of anomaly intrusion detection techniques can be considered as adjacent to the category of datasets consisting of high level features (see Section 4.3.2),

- **B)** the second category represents pure ***network anomaly intrusion detection techniques*** which assume only the features extracted from network traffic. We will discuss several examples of this category – [16, 88, 95, 99, 102, 134, 167] (see Section 4.4.2). Similarly, it can be considered as adjacent to the category of datasets consisting of network traffic dumps (see Section 4.3.1).

The former category is more spread as it works with high amount of input information having enough entropy to perform adequate decisions. Therefore, it can provide low false positive rate and high precision. But on the other hand, this category is more complicated to deploy in real world because of various OS platforms and versions. Also, it could be considered as subject to privacy issues, as its instances may analyze e.g. deciphered packet payload, system calls sequences, running processes, established network connections etc.

The second category is more challenging, as its input data is limited to only network traffic features – offering lower information entropy than former category. Our current research

is related to the second category and can be referred to as *machine learning based network anomaly intrusion detection*. The following subsections will discuss several examples of each proposed category.

4.4.1 General Anomaly Intrusion Detection Techniques

The category of general anomaly intrusion detection techniques contains several examples with their brief description and achieved results. The examples include supervised as well as unsupervised machine learning approaches to anomaly intrusion detection. Note that all presented state-of-the-art approaches are ordered by the year they were published, starting from the earliest one.

Intrusion Detection with Unlabeled Data Using Clustering

The authors of the paper [136] present an intrusion detection algorithm, which is based on unsupervised anomaly detection. This algorithm takes as inputs a set of unlabeled data and attempts to find intrusions within the data. After these intrusions are detected, training of misuse detection algorithm or a traditional anomaly detection algorithm is applied on the data. The approach of the paper clusters the data instances together into clusters using a simple distance-based metric, which is referred as single-linkage clustering. This clustering is performed on unlabeled data, requiring only feature vectors without labels to be presented. Once the data is clustered, all of the instances that appear in small clusters are labeled as anomalies. Both the training and testing was done using different subsets of KDD Cup 99 dataset [187]. The authors utilized 10-fold cross validation method for performance evaluation. On average, the detection rate around $40\% - 55\%$ with a $1.3\% - 2.3\%$ false positive rate was achieved.

Bayesian Networks for Intrusion Detection

The authors of the paper [93] present a method for performing Bayesian classification of input events intended for intrusion detection. There was improved the naive threshold-based schemes traditionally used to combine model outputs by employing Bayesian networks. This allows the authors to naturally incorporate model confidence and dependencies between models into the event classification process.

The evaluation of the scheme was performed on MIT Lincoln Labs 1999 dataset which contained network traffic dumps as well as information about system call sequences. The experimental results show that a significant reduction of false alerts was achieved. When all attacks in testing dataset were detected, the Bayesian event classification reports only half as many false alerts as the traditional approach, based on the same model outputs. The resulting ROC graph comparing threshold-based method with Bayesian Networks is depicted in Figure 4.5.

Octopus-IIDS

The authors Mafra et al. presents an anomaly intrusion detection system Octopus-IIDS, which is based on the behavior of network traffic and performs analysis and classification of messages [108]. Two machine learning techniques are applied for the detection of anomalies – Kohonen Neural Network (KNN) and SVM. These techniques are used in sequence in order to improve an accuracy of the system. The system is able to identify known and new

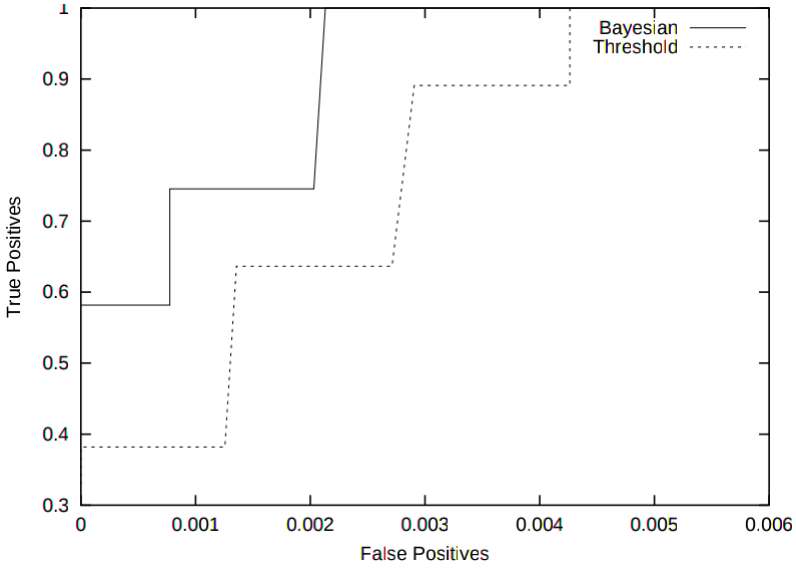

Figure 4.5: ROC curves for Bayesian Networks and threshold method

attacks in real-time. The evaluation of Octopus-IIDS is performed on KDD Cup '99 dataset together with simulated data for normal traffic, which were generated without the presence of fragmented packets, disordered packets, etc. Thus, the authors consider 4 attack classes (DoS, Probe, R2L and U2R) and one class for normal traffic. The comparison of detection rates by Octopus-IIDS and some of state-of-the-art anomaly intrusion detection systems is shown in Table 4.3. According to the comparison, the Octopus-IIDS outperforms other anomaly intrusion detection approaches.

Anomaly Intrusion Detector	Average Detection Rate	Max. Deviation
Anomalous Payload-based IDS [20]	58.80%	41.20%
HPCANN [105]	77.49%	22.53%
MADAM ID [97]	77.97%	17.97%
Multi-level Hybrid Classifier [199]	89.19%	22.52%
Octopus-IIDS [108]	97.40%	8.57%

Table 4.3: Octopus-IIDS and state-of-the-art intrusion detectors [108]

Moreover, authors perform an analysis of the features utilized for classification of input data, and define which of them are important for each class of attack. Note that the results of this paper are limited to drawbacks of KDD Cup '99 dataset.

Self Organizing Map Artificial Neural Network

Ibrahim et al. analyze the performance of unsupervised machine learning-based anomaly intrusion detection system, which is instantiated by Self Organization Map (SOM) Artificial Neural Network (ANN) [79]. The system employs SOM ANN for detection and separation of normal traffic from the malicious one. Proposed IDS is validated on detection of anomalies

in KDD Cup '99 and NSL-KDD datasets and the results show that SOM ANN achieved 92.37% accuracy with KDD Cup '99 dataset, and 75.49% accuracy in the case of NSL KDD Cup '99 dataset.

Holistic Network Defense

Ji Jenny et al. in the paper [84] presents a hybrid network-host monitoring strategy, which fuses data from both the network and the host to recognize malware infections. The thesis focuses on three categories: normal, scanning, and infected. The network-host sensor fusion is accomplished by extracting 248 features from network traffic using the public available discriminators extraction tools of A. Moore [119] (see Appendix E). The next part of information is obtained from host using text mining, looking at the frequency of the 500 most common strings and analyzing them as word vectors. Improvements to detection performance are made by synergistically fusing network features obtained from IP packet flows and host features obtained from text mining. In addition, the thesis compares three different machine learning algorithms and updates the script required to obtain network features. The results of hybrid method outperformed host only classification by 31.7% and network only classification by 25% in classification accuracy.

Host features were obtained using memory dump utility win32dd[3] and Microsoft SysInternals Suite [162] on active target servers of the CDX 2009 [164] competition mentioned above. Three classification techniques were employed in the thesis: LVQ, SOM and SVM. The best performance was achived by SVM (the 2nd by SOM and 3rd by LVQ). Using the host only features there was achieved an accuracy of 76%; network only features achieved 87% and the hybrid approach achieved an accuracy of 98.4%.

Combination of Random Tree and Naive Bayes Decision Tree

The authors Kevric et al. developed a combining classifier model utilizing tree-based algorithms for network intrusion detection [90]. The NSL-KDD dataset [181], a much improved version of the original KDD Cup '99 dataset [187], was used for evaluation of the performance. The task of the detection algorithm was to classify whether the incoming network connection in normal or an attack, based on 41 features describing every pattern of network traffic. The detection accuracy of 89.24% was achieved using the combination of random tree and Naive Bayes Tree algorithms based on the sum rule scheme, outperforming the individual random tree algorithm. According to the authors, this result represents the highest result achieved so far using the complete NSL-KDD dataset. Therefore, combining classifier approach based on the sum rule scheme can yield better results than individual classifiers. The main limitation of this work is utilization of old NSL-KDD dataset.

SSAD – Semi-supervised Statistical Approach for ADS

The paper [1] proposes a two-stage Semi-supervised Statistical approach for Network Anomaly Detection called SSAD. The first stage of SSAD aims to build a probabilistic model of normal instances and measures any deviation that exceeds an established threshold. This threshold is deduced from a regularized discriminant function of maximum likelihood. The purpose of the second stage is to reduce False Alarm Rate (FAR) through an iterative process that reclassifies anomaly cluster from the first stage, using a similarity distance and dispersion

[3] URL: http://www.debuggingexperts.com/win32dd–memory-imaging.

rate of anomaly cluster. The authors of the paper evaluated a performance of proposed approach on the well-known intrusion detection dataset NSL-KDD [181] and Kyoto 2006+ [173]. The experiments compare proposed approach to the Naive Bayes classifier, and the results show that SSAD outperforms the Naive Bayes model in terms of detection rate and false positive rate.

4.4.2 Network Anomaly Intrusion Detection Techniques

The category of network anomaly intrusion detection techniques contains several examples with their brief description and achieved results if they are present. The examples include supervised as well as unsupervised machine learning approaches to network anomaly intrusion detection. Note that all presented state-of-the-art approaches are ordered by the year they were published, starting from the earliest one.

Rule Induction Algorithm and Connection Features

One of the first approaches which utilize machine learning for network anomaly intrusion detection is described in the paper [99]. The authors also apply machine learning for analysis of system calls in the another part of the paper, however, it is not relevant to the current category of intrusion detection techniques. The connection level features are utilized in order to describe properties and characteristics of particular connection and later are utilized as the input for the Rule Induction algorithm designed in the paper [37]. The feature set consists of start time, duration, participating hosts, ports, the statistics of the connection (e.g., bytes sent in each direction, resent rate, etc.), protocol (TCP or UDP), and flag ('normal' or one of the recorded connection/termination errors). However, the performance of the proposed framework was improved by including more features covering:

- examination of all connections in the past n seconds, and counting the number of: connection establishment errors (e.g., 'connection rejected'), all other types of errors (e.g., 'disconnected'), connections to designated system services (e.g., ftp), connections to user applications, and connections to the same service as the current connection
- calculation of the per-connection average duration and data bytes (on both directions) of all connections for the past n seconds, and the same averages of connections to the same service.

Proposed framework consists of classification, association rules which can be used to construct detection models. The experiments on network TCP dump [183] data shown promising results of classification models in detecting of anomalies, which was evaluated by 5-fold cross validation method. The accuracy of the detection models depends on sufficient training data and the correct feature set.

ADAM

Audit Data Analysis and Mining (ADAM) [16] has been designed and implemented as an anomaly detection system, but using a module that classifies the suspicious events into false alarms or real attacks. ADAM is unique in two ways. First, ADAM uses data mining to build a customizable profile of rules of normal behavior, and then a classifier sifts the suspicious activities, classifying them into real attacks and false alarms. Secondly, ADAM

is designed to be used on-line (in real-time), which is achieved by using incremental mining algorithms that use a sliding window of time to find suspicious events.

ADAM uses a combination of association rules, mining and classification to discover attacks in a TCP dump [183] audit trail. First, ADAM builds a repository of normal frequent sets of items that hold during attack-free periods. It does so by mining data that is known to be free of attacks. Secondly, ADAM runs a sliding-window as on-line algorithm that finds frequent itemsets in the last D connections and compares them with those stored in the normal itemset repository, discarding those that are deemed normal. With the rest, ADAM uses a classifier which has been previously trained to classify the suspicious connections as a known type of attack, an unknown type or a false alarm.

The performance of ADAM was evaluated on DARPA '99 dataset [50] and it achieved overall accuracy of circa 45% for DOS attack detection and accuracy of 28% for Probe detection, while in the case of U2R and U2L detection, it achieved 0% accuracy.

Comparison of Unsupervised Anomaly Detection Methods

Several anomaly detection schemes for detecting network intrusions are proposed in the paper [95], and are based on unsupervised machine learning techniques. To support applicability of anomaly detection schemes, a procedure for extracting useful statistical content based and temporal features from network traffic is also implemented, while features are inspired by KDD Cup 99 ones [187].

The list of unsupervised machine learning techniques compared in the paper includes: the Nearest Neighbor approach (NN), the Mahalanobis-based approach, the local outlier factor (LOF) scheme and the unsupervised SVM approach.

Experimental results performed on DARPA 98 dataset [50] indicate that the most successful anomaly detection techniques were able to achieve the detection rate of 74% for attacks involving multiple connections and detection rate of 56% for more complex single connection attacks, while keeping the false alarm rate at 2%. If the false alarm rate is increased to 4%, then detection rate reaches 89% for bursty attacks and 100% for single-connection attacks. Computed ROC curves depicted in Figure 4.6 indicate that the most promising technique for detecting single connection intrusions in DARPA '98 data is the LOF approach. The LOF technique showed also great promise in detecting novel intrusions in real network data and during the few months it had been successful in automatically identifying several novel intrusions at the University of Minnesota.

Genetic Algorithm for Network Intrusion Detection

The authors of the paper [102] discuss a methodology of applying Genetic Algorithm into the area of network intrusion detection. Unlike other implementations of the same problem, this implementation considers both temporal and spatial information of network connections in encoding the network connection information into the rules of IDS. Then, Genetic Algorithm can be used to evolve simple rules for network traffic which will be able to differentiate normal network connections from anomalous connections. This is helpful for identification of complex anomalous behaviors. A rule includes particular features together with their acceptable ranges. Features utilized in this work are derived from the headers of network traffic as well as from simple statistics. They include: IP addresses, port, duration of connection, the number of sent bytes, the number of received bytes and flag indicating correct termination of a connection. Note that the focus of the work is pointed on the

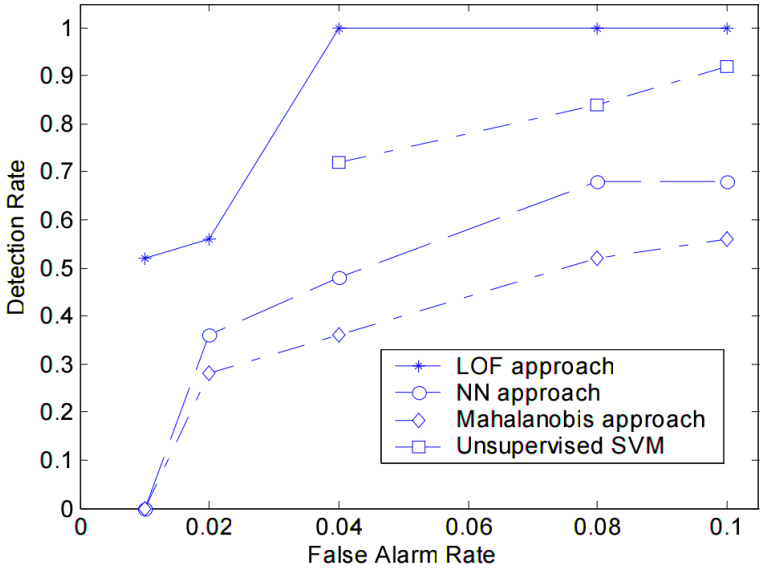

Figure 4.6: ROC curves for single connection attacks

TCP/IP network protocols. The proposed method was evaluated on DARPA '99 dataset [50], however, the authors do not discuss resulting performance.

IDS using Decision Trees and SVM

Peddabachigari et al. in their paper [134] investigate and evaluate the Decision Tree classifier as an intrusion detection mechanism. They compare it with SVM[4]. Intrusion detectors based on Decision Tree and SVM were tested using benchmark 1998 DARPA intrusion detection dataset. The authors achieved better overall performance by Decision Tree than by SVM.

Authors used four attacks classes of DARPA 1998 dataset [50] and one class for legitimate communications. To compare Decision Tree performance with SVM which is a binary classifier, they used binary Decision Tree classifier although it is capable of handling a 5-class classification problem. They constructed five different classifiers for SVM and for Decision Tree methods. The data is partitioned into the two classes of "Normal" and "Attack" patterns where Attack is the collection of four classes of attacks (Probe, DOS, U2R, and R2L). The objective is to separate normal and attack patterns. Input dataset was divided into training data with 5092 records and testing with 6890 records. Obtained results are show in Table 4.4.

Class	Decision Tree	SVM
Normal	99.64%	99.64%
Probe	**99.86%**	98.57%
DOS	96.83%	**99.78%**
U2R	**68.00%**	40.00%
R2L	**84.19%**	34.00%

Table 4.4: Performance Comparison of Decision Tree with SVM [134]

[4]Support Vector Machines.

APAN

Advanced Probabilistic Approach for Network Intrusion Forecasting and Detection (APAN) described in [167] is a novel probabilistic approach which proposes utilization of Markov chain for probabilistic modeling of abnormal events in network systems.

First, to define the network states, K-means clustering is performed and then the authors introduce the concept of an outlier factor. Based on the defined states, the degree of abnormality of the incoming data is stochastically measured in real-time. The performance of the proposed approach is evaluated through experiments using the well-known DARPA 2000 dataset and further analyzed.

The main objective of APAN is to describe the state of the network with its probability and then to decide the abnormality of the network on the basis of this probability. APAN is composed of three main phases: in the first phase, the network states, including the outlying state, are newly defined. Based on these states, the state transition probability matrix and the initial probability distribution of the Markov model is built in the next phase. In the third phase, the chance of abnormal activity for online data is stochastically computed in real-time.

The proposed approach achieved high detection performance while representing the level of attacks in stages. In particular, this approach is shown to be very robust to training datasets and to the number of states in the Markov model.

Adaptive NIDS with Hybrid Approach

Karthick et al. describe an adaptive network intrusion detection system in their paper [88], which utilize a two stage architecture. In the first stage, a probabilistic classifier is employed in order to detect potential anomalies in the network traffic. In the second stage, a Hidden Markov Model (HMM) is applied to narrow down the potential attack IP addresses. The HMM is proposed to profile TCP based communication channel for intrusions. Any normal TCP connection has three phases during its lifetime, i.e., connection establishment, data transmission and connection termination phase. There is an inherent sequential nature in such mode of communication and makes it convenient for authors to model them using HMM, which can leverage this nature of TCP traffic.

The goal of the work is to implement suitable model which can function effectively in real-time. When implementing pure HMM as a real-time system, it turns into two pitfalls. The first drawback comes out from the fact, that HMM needs to keep track of all incoming addresses, and thus spoofed IP addresses can cause exhaustion of resources of a server. The second limitation of real-time implementation is the fact, that pure HMM solution needs complete TCP flows, however in practice, there is no way of telling when connection ends, and thus HMM cannot make real-time detection. The solution for this problem would be storing of all packets, which would also potentially cause resource exhaustion of a server. Note that author evaluate their HMM approach on DARPA dataset and achieved high detection rate of suspicious IP addresses (97.1%) in the case of 9 states of HMM.

Therefore, the authors propose hybrid model combining their HMM with Naive Bayes approach designed in [190]. The hybrid model employs Naive Bayes for online learning and HMM for offline one. The online model monitors incoming traffic and flag traffic blocks that are suspicious. The offline model is fed with the traffic flagged by online model. HMM then performs source separation for the connections present in the flagged traffic and classifies the connections as either attack or normal. The output of the offline model is update of firewall

rule, which prohibits incoming connections from source IP addresses with suspicious traffic characteristics. The proposed hybrid model also performed well in detecting intrusions on mixture of DARPA [50] and CAIDA [27] datasets. 100% detection rate was achieved in hybrid model case which also utilized 9 states of HMM.

4.4.3 Critique of Machine Learning in Network Intrusion Detection

Sommer et al. [172] examine the surprising imbalance between the extensive amount of research on machine learning-based anomaly detection pursued in the academic intrusion detection community, versus the lack of operational deployments of such systems. They claim that the task of finding attacks is fundamentally different from other tasks, making it significantly difficult for the intrusion detection community to employ machine learning effectively. They support this claim by identifying challenges particular to network intrusion detection and then, they provide a set of guidelines meant to strengthen future research on anomaly detection.

However, machine learning methods can be employed in the area of anomaly intrusion detection, which inherently imposes certain level of false positive rate. Therefore, anomaly intrusion detection techniques can be exploited in combination with other intrusion detection approaches, for providing more information about occurred incidents and potentially help in forensic analysis.

4.5 Machine Learning in Network Traffic Classification

The section describes various network traffic classification approaches leveraging machine learning and it states references to input testing datasets and achieved results if these are present at original sources. Note that all presented state-of-the-art approaches are ordered by the year they were published, starting from the earliest one.

Flow Clustering Using Machine Learning Techniques

McGregor et al. in [112] identify traffic with similar observable properties by applying an untrained classifier to the problem. The untrained classifier identifies classes of traffic having similar properties, but does not directly assist in understanding what or why applications have been grouped in this way. Authors are left to guess the precise type of each application without content-derived information. This work demonstrated that the properties of network traffic, using features on a per-flow basis, allow some differentiation to be inferred. Each class includes a relatively large number of network-based applications, but the authors were never able to establish this method as anything more than a property-grouping technique. The authors of this work provide a useful contribution by illustrating that (unsupervised) machine learning is plausible, but achieved results are not comparable with other methods like [11].

Bayesian Classifier with Kernel Distribution

In the report [118] of A. Moore, previously mentioned discriminators (see 4.3.2) were applied onto Moore's 2005 datasets and internet traffic classification was performed. During classification, applications with similar behavioral and statistical traits were classified into the same class. The Naive Bayesian classifier was used in the algorithm, where the Bayes formula was utilized in order to calculate posterior probability of a testing sample and selected the

class with the highest probability as the classification result. A total of about 200 features (discriminators) of a network flow were used to train the model and a kernel-based function was utilized to estimate the distribution function. The total accuracy achieved about 95% in the dimension of flow number being correctly classified and 84% in the dimension of flow size.

Discriminators extraction method in conjunction with classifier model can be used to detect network intrusions in the case of having ground truth for training data of the classifier. The authors leveraged hand made ground truth of traffic classes and in many cases they used port numbers.

Bayesian Neural Networks for Internet Traffic Classification

Authors of paper [11] demonstrate functionality of Bayesian framework using a neural network model that allows identification of traffic without using any port or host (Internet protocol address) information. This represents the same situation as faced when attempting to classify anonymized traffic. There is illustrated classification process operating as an offline tool, as in the auditing of forensic work.

Classification accuracy which authors achieved reach over 99% when training and testing on homogeneous traffic from the same site on the same day.

Experiments which were realized in a more realistic situations achieved classification accuracy of 95% in the training on one day of traffic from one site and testing on traffic from that site for a day, eight months later. Authors proclaims significantly higher results than in the case of naive Bayesian approach adopted in [118]. The next contribution of this work is finding, that small number of features carry high significance regardless of the classification scheme. Authors also show, that there is some overlap in features of high importance to either method.

There are described experiments with preclassified data described originally in [117], which were obtained by high-performance network monitor described in [116]. This data, also used in [118], consists of descriptions of Internet traffic that have been manually classified. Hand-classification of two distinct days of data for an active Internet facility provides the input for sets of the training and testing phases. The data was provided as a set of flows taken from two distinct days, where each day consisted of ten sets of classified transport control protocol (TCP) traffic flows with each object described by its membership class and a set of features.

Network Traffic Classification Based on Improved DAG-SVM

Given error accumulation in traditional Directed Acyclic Graph Support Vector Machine (DAG-SVM) algorithm, the authors Hao et al. propose an improved of DAG-SVM classification method using two different possibility metrics in their paper [67]. Differing from traditional DAG-SVM, the improved DAG-SVM algorithm eliminates one class only under the condition of the classification error probability is less than threshold. The results of the experiment show that in comparison to traditional DAG-SVM, the both methods proposed in this paper have higher classification accuracy with acceptable time cost. Improved DAG-SVM based on distance has a better performance than improved DAG-SVM based on decision function. All the experiments were performed on the Moore dataset [119].

Expectation Maximization & Internet Traffic Classification

The traditional Expectation Maximization (EM) algorithm suffer from the disadvantage of sensitivity to initial value, and thus often converges to local optimum values. Therefore, the authors Liu and Hu et al. of the paper [106] propose a new improved EM algorithm based on the q-DAEM method. The improvement of the algorithm resides in application of the EM algorithm to generate a constrained matrix, then combine the constrained matrix with the q-DAEM algorithm to reduce the search range, and thus a better Gaussian mixture model can be derived from this algorithm. The algorithm is validated on the Moore datasets [119] for evaluation, and the experimental results show that this method improved the EM algorithm and outperforms its performance measures like precision and overall accuracy.

4.6 Obfuscation and Evasion Approaches in ADS and IDS

This section describes various approaches related to obfuscation of network protocols as well as obfuscation of payload and application behavior of network attacks. These obfuscations can be utilized as attempts evading the detection by network anomaly detection systems, by intrusion detection systems and partially by firewalls. In several cases, such attempts may be successful and cause evasion of ADS or IDS. We divide obfuscation (potentially evasion) approaches into four categories:

- **Tunneling obfuscations** – the first category of obfuscations deals with tunneling of one protocol in another one (see Section 4.6.1). From the perspective of TCP/IP stack, it leverages application layer for tunneling.
- **Application layer obfuscations** – the second category covers modifications of packet payload by various mutations and morphism of application level behavior of malicious attacks (see Section 4.6.2). From the perspective of TCP/IP stack, it exploits application layer for payload-based modifications.
- **Obfuscations of network protocols and characteristics** – the third category aims on modification of network protocol packet exchanges as well as on modification of network traffic characteristics (see Section 4.6.3). From the perspective of TCP/IP stack, the category exploits network and transport layer.
- **Combinations of application layer and network protocol obfuscations** – the last category considers combinations of the previous two categories (see Section 4.6.4). From the perspective of TCP/IP stack, the category exploits network, transport and application layer.

Instances belonging into each category will be further described, and moreover, several techniques intended for detection and defence of some obfuscation techniques will also be discussed.

4.6.1 Tunneling Obfuscations

One of the most spread kind of obfuscation is payload tunnel. Payload tunnel is covert channel that tunnel one protocol in the payload of another protocol. One of the purposes of such channel is circumventing firewalls that limit outgoing traffic to few allowed application protocols. A variety of tools exist for tunneling over application protocols that are often

not blocked, such as ICMP [48, 176, 203] or HTTP [57, 96, 131]. Some of the papers related to obfuscation by covert channels were revealed in the survey of covert channels and countermeasures in computer network protocols collected in the year 2007 [202] by authors Zander S. et al.

Tunnel Hunter

Dusi et al. presented a mechanism called Tunnel Hunter, which can successfully identify protocols tunneled inside tunneling protocols such as HTTP, DNS and SSH [56]. Tunnel Hunter performs statistical analysis of simple IP level flow features (i.e., packets sizes, inter-arrival time and packet order). The technique suffers from the problem of sensitivity to packet-size and timing value manipulation.

Tunneling over ICMP

One of the first approaches for tunneling protocols over ICMP was Loki, which tunneled data in the payload of ICMP echo messages [48]. Zelenchuk implemented an indirect IP over ICMP tunnel [203]. The covert sender sends echo request packets to a bounce host with spoofed source address (set to the address of the covert receiver) and the covert data encoded in the payload. The bounce host then sends echo replies to the covert receiver with the same payload as in the requests.

Sohn et al. [170] used the SVM-based approach to evaluate the accuracy of detecting covert channels embedded in ICMP echo packets and achieved classification accuracy of up to 99% when training a classifier on normal and abnormal packets.

Tunneling over HTTP

Another popular method is to tunnel protocols over HTTP. Padgett developed a tool that tunnels SSH through HTTP proxies [131]. Dyatlov and LeBoutillier implemented tools for tunneling UDP and TCP over HTTP [57, 96].

The authors Crotti et al. in their paper [43] proposed application of a statistically-based traffic classification technique to detect tunneled protocols inside of HTTP by the analysis of inter-arrival time, size and order of the packets crossing a gateway. They describe the technique which effectively enhance application level gateways and firewalls, helping to better apply network security policies on such tunneled traffic.

Pack et al. [130] proposed detecting HTTP tunnels by using behavior profiles of traffic flows. Behavior profiles are based on a number of metrics such as the average packet size, ratio of small and large packets, change of packet size patterns, total number of packets sent and received, and connection duration. If the behavior of a flow under observation deviates from the normal HTTP behavior profile it is likely to be an HTTP tunnel.

The Web Tap's [21] focus is aimed at detecting attempts to send significant amounts of information via HTTP tunnels to rogue web servers from within firewalled internal network. A related goal of Web Tap is to help in detection of spyware programs, which often send out personal data to servers using HTTP transactions and may open up security holes in the network. Based on the analysis of HTTP traffic over a training period, the authors use filters to help in detection of anomalies in outbound HTTP traffic using metrics such as request regularity, bandwidth usage, inter-request delay time, and transaction size. The Web Tap can be evaded by the adversary by monitoring and analysis of users outbound traffic and then mimic the access patterns of a legitimate site.

Tunneling over IMAP

Magnus Lundström implemented a tool that can establish a bi-directional tunnel over the exchange of emails [107]. In principle, it is necessary to have one mail box on the blocked (internal) network, and an account on an outside internet-connected system.

4.6.2 Application Layer Obfuscations

The category of application layer obfuscations performs modifications of networks attacks solely at the application layer of the TCP/IP stack, and thus is also referred to as category of payload-based obfuscations. It may include e.g. morphing, encrypting or mimicry of attack's code contained in payload of packets.

Polymorphic Blending Attacks

The authors Fogla et al. realize the obfuscation of network attacks by proposing a new class of polymorphic attacks, called polymorphic blending attacks (PBA) [61], which can effectively evade byte frequency-based network anomaly IDS. The attacks carefully match the statistics of the mutated attack instances to the normal profiles. They demonstrate the efficiency of PBA attacks on PAYL. In the next paper [60] they show that in general, generating a PBA that optimally matches the normal traffic profile is a hard problem (NP-complete), but can be reduced to SAT or ILP problems. They present a formal framework for PBA attacks and also propose a technique to improve the performance of an IDS against PBAs.

Mimicry Attacks

The authors Wagner D. et al. [192] introduced the notion of a mimicry attack which can cloak the attacks behavior to avoid detection of IDS by generating usual system calls into system calls sequence of an attack. Also, the authors develop a theoretical framework for evaluating the security of an IDS against mimicry attacks.

Whiskers

The next work dealing with payload-based evasion is described in [140] and presents a tool called Whisker. The author aims at anti-intrusion detection tactics by performing mutations of the HTTP request in a way that the web server is able to understand the request, but intrusion detection systems can be confused. The principles of Whisker can also be applied to other protocols than HTTP.

Mutant Exploits

Vigna et al. [189] proposed a framework generating exploit mutations to change the appearance of a malicious payload, which bypasses detection of network intrusion detection. The proposed framework was evaluated on two well-known signature based NIDSs – SORT [160] and RealSecure [158].

Malware Morphism

The paper [201] explores the malware obfuscation techniques such as dead-code insertion, register reassignment, subroutine reordering, instruction substitution, code transposition

and code integration, which have been mainly used by polymorphic and metamorphic malwares to evade signature based scanners (antivirus). Among polymorphic and metamorphic malware, the paper also reviews encrypted and oligomorphic malware.

4.6.3 Obfuscations of Network Protocols and Characteristics

The previous two categories of methods can evade payload-based NIDS systems primarily by morphing (or encrypting) the payload, but do not need to be efficient against network ADS. Network ADSs are the most sensitive on the morphing at the network and transport layers of the TCP/IP stack. The following techniques and tools are primary dealing with such modification of network traffic as well as with evasion of intrusion detectors by such modifications. Also, this category can be referred to as non-payload-based obfuscations, as its principles does not modify the payload of the packet and thus application layer content.

Advanced Evasion Techniques

Stonesoft found evasion techniques that extend earlier research to include a new set of techniques and combinations of the prior techniques [19]. Together, these Advanced Evasion Techniques (AETs) prey upon protocol weaknesses and the permissive nature of network-based communication, exponentially increasing the number of evasions that can bypass even the most up-to-date IPS technologies.

The AET evasions mostly build on well-known principles of de-synchronizing detection systems relying on the network view of the traffic, from the perspective of the target host. TCP/IP protocol suite used on the Internet and the vast majority of all computer networks, is based on the requirements from RFC 791 [83] which was written in 1981. Among other things, the RFC says: "In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior" Although, an implementation must be careful to send well-formed datagrams, but must accept any datagram that it can interpret. That means there will be multiple ways to form messages that will be interpreted identically by the receiving host. While this permissive stance was intended to make interoperability between systems as reliable as possible, simultaneously paved the way for a number of attacks and ways to hide those and other attacks from a detection. As different operating systems and applications behave in different ways when receiving packets, the destination host application may see something quite different than what was in the network traffic.

Evasion possibilities have been found on IP and transport layers (TCP, UDP) of TCP/IP stack as well as on application layer protocols, including but not limited to SMB and RPC protocols. The representative examples of methods leveraged in AET are TCP segmentation and IP fragmentation.

Protocol Obfuscations Inhibiting Statistical Analysis

The paper [74] shows how obfuscated application layer protocols, such as BitTorrent's MSE [36] or Skype [115], can be identified by an analysis of statistically measurable properties of TCP and UDP sessions. The authors depict that many of the analyzed protocols have statistically measurable properties in payload data, flow behavior, or both. Based on their insight, they propose a few techniques for improving protocol obfuscation which inhibits traffic identification by statistical analysis. These techniques include better obfuscation of payload data and flow properties as well as hiding inside tunnels of well-known protocols. The purpose of this work is not to provide more effective intrusion classification of NBAD

systems, but rather to provide feedback to protocol creators who want to contribute on the network neutrality of the Internet.

Realtime Classification for Encrypted and Obfuscated Traffic

Bar-Yanai et al. presented a method for real-time classification of encrypted traffic [13]. The proposed statistical classifier is based on a hybrid combination of k-means and k-nearest neighbor geometrical classifiers and is shown to be very robust, even to obfuscated traffic such as Skype and encrypted BitTorrent. The statistical feature set is composed of 17 parameters based on packet and payload byte counts, packet sizes and packet rates for each direction.

Fragroute

Fragroute [174] is a tool which was written to test intrusion detection systems by using simple ruleset language enabling interception and modification of egress traffic with minimal support for randomized or probabilistic behavior. It can intercept, modify, and rewrite egress traffic destined for a specified host and contains a simple ruleset language to delay, duplicate, drop, fragment, overlap, print, reorder, segment, source-route of all outbound packets with minimal support for randomized or probabilistic behavior. Fragroute implements three classes of attacks – insertion, evasion, and denial of service which were described in paper [139].

AGENT

A common way to elude a signature-based NIDS is to transform an attack instance that the NIDS recognizes into another instance that it misses. For example, to avoid matching the attack payload to a NIDS signature, attackers split the payload into several TCP packets or hide it between benign messages. The authors of the paper [161] derive different attack instances from each other using simple transformations as TCP fragmentation, TCP permutation, TCP retransmission, FTP padding etc. Then, these transformations are modeled as inference rules in a natural-deduction system. Starting from an exemplary attack instance, the authors use an inference engine to automatically generate all possible instances derived by a set of rules. The result is a simple tool capable of both generating attack instances for NIDS testing dataset as well as determining whether a given sequence of packets is an attack. In several testing phases using different sets of rules, the proposed tool exposed serious vulnerabilities in SORT [160]. Attackers acquainted with these vulnerabilities would have been able to construct instances that elude SNORT for any TCP-based attack, any Web-CGI attack, and any attack whose signature is a certain type of regular expression.

Protocol Scrubbing

Watson et al. [193] proposed a method called *Protocol Scrubbing* which represents active mechanisms for transparent removing of network attacks from protocol layers in order to allow passive IDS systems to operate correctly against evasion techniques.

Network Normalizers as Defence for Protocol Obfuscations

In order to answer non-payload-based evasions of NIDS, the concept of network traffic normalizer was introduced by Handley et al. [66]. The authors proposed the implementation

of normalizer called *norm*. Norm performs normalizations of ambiguities in the TCP traffic stream which can be seen by NIDS. However, introducing a network normalization brought problems related to platform dependent semantic of network ambiguities interpretation as well as throughput reduction. These problems lead Shankar et al. [166] to introduce the concept and implementation of *Active Mapping*, which eliminates them with minimal runtime cost by building profiles of the network topology including the TCP/IP policies of hosts on the network. A NIDS may then use the host profiles to disambiguate the interpretation of the network traffic on a per-host basis. Because of the shortcomings of network normalizers, their usage in a network can result in side-effects and can even be prone to various attacks, e.g. state holding, and CPU overload [53, 132, 168].

4.6.4 Combinations of Obfuscations

Evasions based on modifications at each of the application, transport and network layers of the TCP/IP stack are described in papers [33, 86]. Cheng et al. [33] described general evasion techniques and examined the detection performance of signature based NIDS when performing mutation of known attacks. Juan et al. described NIDS evaluation framework called *idsprobe* [86] which takes original network traces, creates several altered traces based on predefined transformation profiles, then runs NIDS against generated traces and evaluates feedback of the detection regarding different evasion scenarios.

Chapter 5

Automated Intrusion Prevention System

The Automated Intrusion Prevention System (AIPS) is conceptual and practical model of Network Behavioral Anomaly Detection (NBAD) system intended for intrusion detection.[1] AIPS was developed at the Faculty of Information Technology in Brno within research plan of the Security-Oriented Research in Information (MSM0021630528). The project related with the AIPS system had four main participants: Maroš Barabas, Michal Drozd, Petr Chmelař and Ivan Homoliak (me). I participated the project since 2010 until its successful completion in 2013.

Schematic model of original AIPS concept is illustrated in Figure 5.1. The model consists of five parts: *Honeypots*, *Communication extractor*, *Metrics extractor*, *Dataset* and *Intrusion Detection and Prevention System* (IDPS). My responsibility in the project involved the design and implementation of Connection Extractor and Metrics Extractor components as well as experiments with machine learning based classification models of IDPS component. The full concept of the AIPS system is described in Section 5.1, while my particular contributions to the design and definition of the system's features are described in Section 5.2 and Section 5.3.

5.1 Full Concept

The schematic model of AIPS system, illustrated in Figure 5.1, considers expert knowledge (ground truth), which helps to identify the real attack execution. The expert knowledge is provided by the shadow honeypot system [8] Argos, which is based on the dynamic tainted data analysis technique of the process memory [122]. By the nature of this technique, the honeypot can recognize only buffer overflow attacks. Network data capture is performed by *tcpdump* utility [183] listening on network interfaces of the honeypots. Data captured by tcpdump are passed to Communication extractor component, which groups related packets into connection objects. Next, Advanced Security Network Metrics (ASNM) [76] feature extraction is performed by Metrics extractor component, which receives connection objects from Communication extractor. This process involves statistical and behavioral analysis of packet header fields, which is the most time-consuming part of the AIPS framework. Then, values of ASNM features extracted above communication objects are stored in database as ASNM records. The database is represented by Dataset component in schematic model [75].

[1] The term NBAD can be interchangeably referred to as Anomaly Detection System (ADS) too.

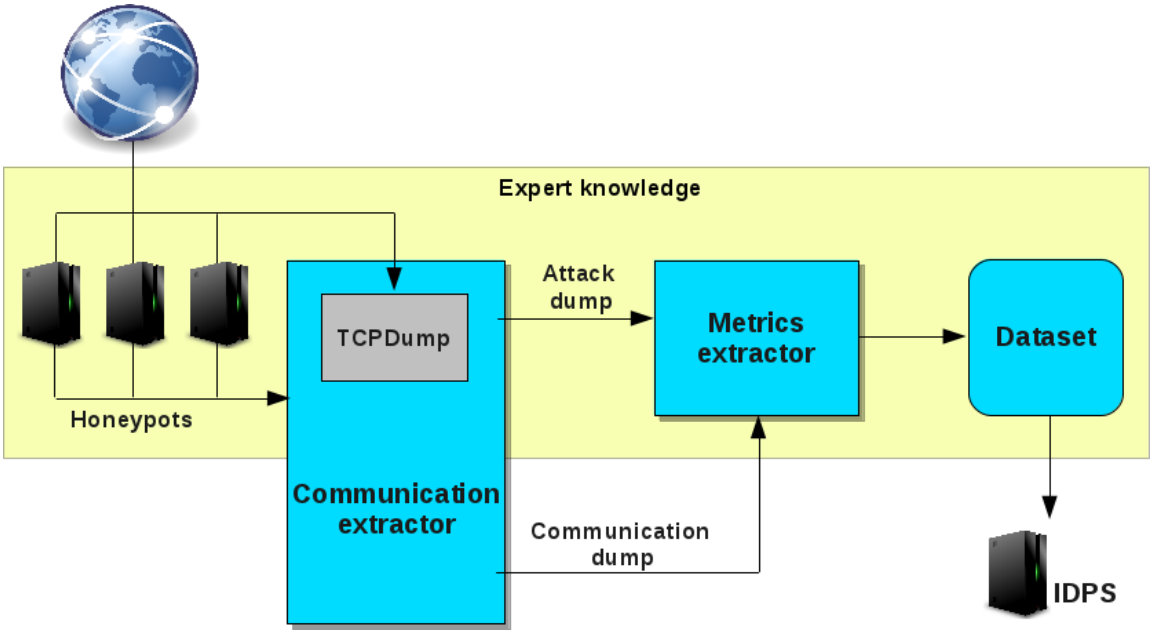

Figure 5.1: Model of the AIPS system [14]

Another part of the AIPS model is the IDPS component, which utilize ASNM records of all analyzed communications to train its classification model and analyze new traffic occurred in a network. Using accurate and actual classification model, it is supposed to detect already known and also new (zero day) network buffer overflow attacks, which in many cases exhibit similar behavioral characteristics as training data. IDPS can detect attacks on all stations in monitored network. AIPS framework should have more honeypots with divergent operating systems and services installed [14]. Note that honeypots are supposed to include vulnerable services as well as ones with not already known vulnerabilities in order to strengthen detection of zero-day buffer overflow attacks.

Successfull experiments were performed with the architecture of the AIPS system and there were defined 112 features divided into five categories according to their nature: *statistic, dynamic, distribution, localization* and *behavioral* [15]. The AIPS framework is restricted to perform only TCP communications analysis.

The next aspect of AIPS system is that it considers the context of analyzed connection, which represents previous and actual time bounded communications having the same source and destination IP addresses. Thus, the context of an analyzed TCP connection object represents communication log between pairs of source and destination machines, and will be formally defined later.

5.1.1 Methods for Gathering Expert Knowledge

Expert knowledge of connection records is designed to be provided by shadow honeypots. But in fact, it does not matter how is expert knowledge provided. Therefore, the concept of the system is extendable by various expert knowledge sources, e.g. SNORT log, audit log, hand-made labeling, custom one.

In the current scheme of AIPS, if an attacker performs an attack on vulnerable service of a target system and he causes a buffer overflow attack, then consequences of his activity are detected and logged thanks to involvement of honeypots. The communication entry (ob-

tained by tcpdump) and attack packet are passed to Communication extractor component where these data are processed. The result of this process is passed to Metrics extractor component, which creates a set of ASNM features and stores it with expert knowledge into database. The set of ASNM features with the expert knowledge is then used to actualize detection model of IDPS component by repetitive learning [75].

Similar process, as described above for malicious communication cases, is necessary to perform for legitimate communication cases in order to endeavour of balanced amount of training data for both classes. Also, the classification model of the AIPS framework needs to have enough amount of training data for both classes. Therefore, the training should be performed on all attack entries and representative set of legitimate communication entries. Note that in practice are legitimate communications entries present in much greater amount than attack communications entries.

5.1.2 Principle of Detection

IDPS component analyzes all network traffic from SPAN port of border network device. Traffic is consequently grouped to form TCP communication objects using functionality of Connection extractor. Every distinguished TCP communication object is analyzed by Metrics extractor component, which extract all defined and allowed features associated with a TCP communication object. Next, extracted features of a TCP communication object are classified by previously trained classification model of IDPS component [75]. According to the settings, the filtering decision is made for analyzed TCP communication or there is generated detection alert message indicating possible occurrence of intrusion.

5.1.3 The Second Generation

In the experiments performed in [15], we found several limitations of the original idea and some parts of the architecture were changed. We extended the ASNM dataset to 167 features[2] containing approximately 4000 parameters. The main goals of this version are

- to design the architecture of detection framework that will enhance the overall network security level with the ability to learn new behaviors of attacks without intervention of human by using the expert knowledge from Honeypot (or similar) systems,
- to find the most suitable set of features that will successfully describe the behavior of attacks in the network traffic and will significantly increase the detection rate and lower the false positive rate.

The conceptual schema of the next AIPS version [15] is illustrated in Figure 5.2. It includes the AIPS Network Detector (AIPS ND) working as a network probe capable of detecting intrusions using a knowledge base from the AIPS Attack Processor (AIPS AP). Further, the schema includes the Intrusion Detection and Prevention system (IDPS) for a real-time detection/prevention of attacks and a database (DB) for storing data and signatures (not included in the scheme).

[2] Later, ASNM set was extended to 195 features in 2014.

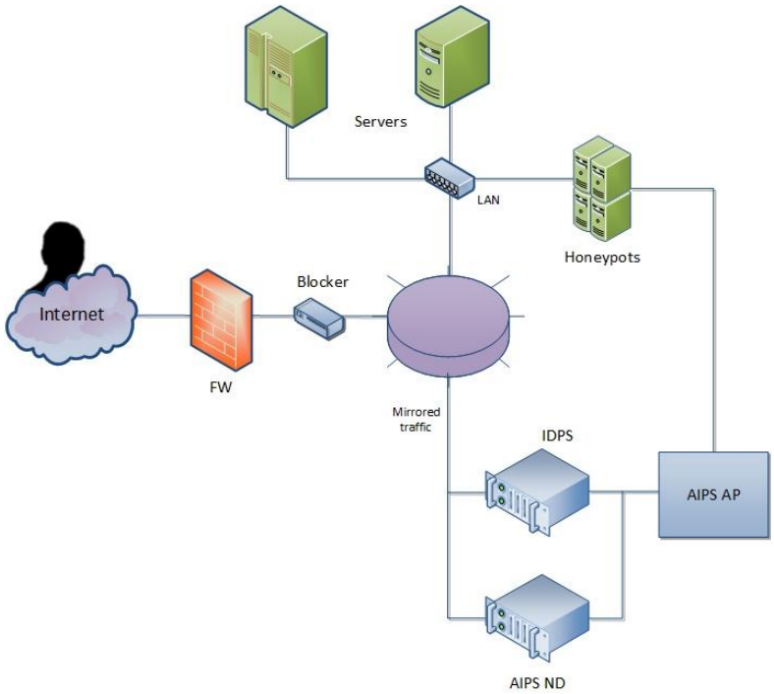

Figure 5.2: AIPS network architecture [15]

The last (optional) part of the AIPS architecture is a group of highly interactive Honeypot systems which are used to create expert knowledge of detected attacks. The knowledge is sent to AIPS AP where the knowledge base is concentrated. The AIPS AP is also responsible for learning the artificial intelligence of the AIPS ND. The AIPS ND works as a network probe capable of detecting intrusions using a knowledge base from the AIPS AP. The mirrored traffic (from a backbone router/firewall) is captured by a tcpdump probe and separated to individual flows. All individual connections are extracted from the pre-processed traffic flow and further processed to create a vector of ASNM features.

The architecture of the framework is designed by modular principles to allow the flexibility of exchanging or enhancing each part of framework model to cover all potential use cases. The part with deployed honeypots is designed to be either a regular instance of the honeypot system or a part of real operating system within a DMZ (demilitarized zone) as a service. There were performed experiments with Windows XP, Windows 2000 and Linux systems, all with successful deployment and successful detection of tested buffer overflow attacks. PostgreSQL database was utilized for interaction between the parts of the framework. The use of database also fastened the processes working with high amount of data [15].

5.2 ASNM Extraction with Context Analysis

Features used in AIPS system are formally specified and extraction of them can be generally performed for each TCP network connection. We can interpret the specification of ASNM feature set as extended protocol NetFlow [34] but describing more than statistical properties of network communications. The ASNM features specification includes statistical, dynamical, localization and especially behavioral properties of network communication [15]. Some of defined features consider context of an analyzed connection [76] which will be defined and described later.

5.2.1 Definitions of Packet and Connection

The method of our approach [76] is based on the extraction of various types of properties from each analyzed TCP connection. We suppose having all packets set P:

$$P = \{p_i\},\ i \in \{1, \ldots, N\}, \tag{5.1}$$

where N represents all packets count. The identification of each packet is represented by its index i. A packet $\boldsymbol{p}_i$ can be expressed as a tuple:

$$\boldsymbol{p}_i = (t, size, eth_{src}, eth_{dst}, ip_{off}, ip_{ttl}, ip_p, ip_{sum}, ip_{src}, ip_{dst}, ip_{dscp}, tcp_{sport}, tcp_{dport}, tcp_{sum},$$

$$tcp_{seq}, tcp_{ack}, tcp_{off}, tcp_{flags}, tcp_{win}, tcp_{urp}, data).$$

Symbols used in the packet tuple as well as their origin from TCP/IP stack are described in Table 5.1. The notation B^* denotes the iteration of the set B containing all possible byte values.

Symbol	Description
$t \in \Re_0^+$	Relative time of the packet capture (L1).
$size \in \aleph$	Size of the whole Ethernet frame including Ethernet header (L1).
$eth_{src} \in \{0, \ldots, 2^{48} - 1\}$	Source MAC address of the frame (L2, Ethernet).
$eth_{dst} \in \{0, \ldots, 2^{48} - 1\}$	Destination MAC address of the frame (L2, Ethernet).
$ip_{off} \in \{0, \ldots, 2^{13} - 1\}$	Offset field (L3, IPv4).
$ip_{ttl} \in \{0, \ldots, 2^{8} - 1\}$	Time to live field (L3, IPv4).
$ip_p \in \{0, \ldots, 2^{8} - 1\}$	Protocol field (L3, IPv4).
$ip_{sum} \in \{0, \ldots, 2^{16} - 1\}$	Checksum of the header (L3, IPv4).
$ip_{src} \in \{0, \ldots, 2^{32} - 1\}$	Source IP address of the packet (L3, IPv4).
$ip_{dst} \in \{0, \ldots, 2^{32} - 1\}$	Destination IP address of the packet (L3, IPv4).
$ip_{dscp} \in \{0, \ldots, 2^{8} - 1\}$	Differentiated services code point field (L3, IPv4).
$tcp_{sport} \in \{0, \ldots, 2^{16} - 1\}$	Source port of the packet (L4, TCP).
$tcp_{dport} \in \{0, \ldots, 2^{16} - 1\}$	Destination port of the packet (L4, TCP).
$tcp_{sum} \in \{0, \ldots, 2^{16} - 1\}$	Checksum of the header (L4, TCP).
$tcp_{seq} \in \{0, \ldots, 2^{32} - 1\}$	Sequence number of the packet (L4, TCP).
$tcp_{ack} \in \{0, \ldots, 2^{32} - 1\}$	Acknowledgment number of the packet (L4, TCP).
$tcp_{off} \in \{0, \ldots, 2^{8} - 1\}$	Offset and reserved fields together (L4, TCP).
$tcp_{flags} \in \{0, \ldots, 2^{8} - 1\}$	Control bits (L4, TCP).
$tcp_{win} \in \{0, \ldots, 2^{16} - 1\}$	Window field (L4, TCP).
$tcp_{urp} \in \{0, \ldots, 2^{16} - 1\}$	Urgent pointer field (L4, TCP).
$data \in B^* = \{0, \ldots, 2^{8} - 1\}^*$	Payload of the packet (L7).

Table 5.1: Symbols of the packet tuple

TCP connection $\boldsymbol{c}$ is represented by tuple:

$$\boldsymbol{c} = (t_s, t_e, p_c, p_s, ip_c, ip_s, P_c, P_s).$$

The interpretation of the symbols used in the tuple is briefly described in Table 5.2.

Symbol	Description
$t_s \in \Re^+$	Timestamp of the connection's start.
$t_e \in \Re^+$	Timestamp of the connection's end.
$p_c \in \{0, \ldots, 2^{16} - 1\}$	Port of the client within the TCP connection.
$p_s \in \{0, \ldots, 2^{16} - 1\}$	Port of the server within the TCP connection.
$ip_c \in \{0, \ldots, 2^{32} - 1\}$	IPv4 address of the client.
$ip_s \in \{0, \ldots, 2^{32} - 1\}$	IPv4 address of the server.
$P_c \subset P$	Set of packets sent by client to server.
$P_s \subset P$	Set of packets sent by server to client.

Table 5.2: Symbols of the TCP connection tuple

The source part of a TCP connection is the one which initiate a connection (usually referred to as a client side) and the destination part is the opposite part of the connection (usually referred to as a server side). The set of all packets can be interpreted also as a set of all TCP connections $C = \{c_1, \ldots, c_M\}$, where M is count of TCP connections, which we can identify in the P, and N is the count of all packets in set P. The minimum number of packets, which is necessary to identify a TCP connection, is three. These three packets serve for establishment of a TCP connection according to TCP specification and they must contain the same IP addresses (ip_s, ip_d), ports (p_s, p_d) and fields tcp_{seq}, tcp_{ack} corresponding to a three way handshake specification stated in RFC 793.[3] Therefore, the number of all TCP connections identified in P is $M \leq N/3$.

5.2.2 Context Definition

Considering analyzed TCP connection c_k, we define a sliding window $\boldsymbol{sw}$ of length τ as a set of TCP connections W_k which are delimited by $\pm \frac{\tau}{2}$:

$$\begin{aligned} \boldsymbol{sw}(\boldsymbol{c}_k, \tau) = W_k \subseteq C, \\ W_k = \{\boldsymbol{c}_j\}, \end{aligned} \tag{5.2}$$

where all $\boldsymbol{c}_j$ must satisfy following statements:

$$\begin{aligned} \boldsymbol{c}_j[t_s] > \boldsymbol{c}_k[t_s] - \frac{\tau}{2}, \\ \boldsymbol{c}_j[t_e] < \boldsymbol{c}_k[t_s] + \frac{\tau}{2}. \end{aligned} \tag{5.3}$$

The next fact about each particular TCP connection $\boldsymbol{c}_k$ is an unambiguous association of it to particular sliding window W_k. We can interpret the start time t_s of the TCP connection $\boldsymbol{c}_k$ as a center of the sliding window W_k. Then, we can denote a shift of the sliding window $\Delta(W_j)$ which is defined by start time differences of two consecutive TCP connections in C:

$$\begin{aligned} \Delta(W_j) = \boldsymbol{c}_{j+1}[t_s] - \boldsymbol{c}_j[t_s], \\ j \in \{1, \ldots, |C| - 1\}. \end{aligned} \tag{5.4}$$

Next, we define the context K_k of the TCP connection c_k, which is a set of all connections in a particular sliding window W_k excluding analyzed TCP connection $\boldsymbol{c}_k$:

[3] URL http://www.ietf.org/rfc/rfc793.txt, page 30.

$$K_k = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n\} = \{W_k \setminus \boldsymbol{c}_k\}. \tag{5.5}$$

Defined terms are shown in Figure 5.3. The x axis displays time and the y axis represents TCP connections which are shown in the order of their occurrences. Packets are represented by small squares and TCP connections are represented by a rectangular boundary of particular packets. A bold line and font is used for depicting an analyzed TCP connection $\boldsymbol{c}_k$, which has an associated sliding window W_k and context K_k. TCP connections, which are part of the sliding window W_k, are drawn by full line boundary and TCP connections, which are not part of this sliding window, are drawn by a dashed line boundary.

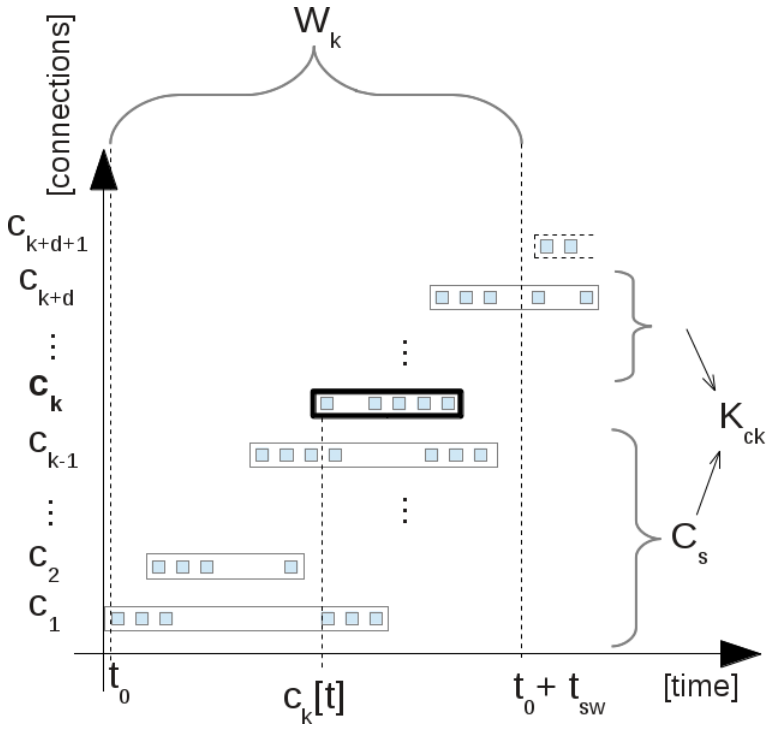

Figure 5.3: Sliding window and context of the first analyzed TCP connection $\boldsymbol{c}_k$ [76]

5.2.3 ASNM Feature Extraction

At this time, we can express the ASNM characteristics of a connection by features. The feature extraction process is defined as a function which maps a connection $\boldsymbol{c}_k$ with its context $K_k = \boldsymbol{sw}(\boldsymbol{c}_k, \tau)$ (and other optional arguments) into feature space F:

$$\begin{aligned} f(\boldsymbol{c}_k, K_k, arg_1, \ldots, arg_n) &\mapsto F, \\ F &= (F_1, F_2, \ldots, F_n), \end{aligned} \tag{5.6}$$

where n represents the number of defined features. Each feature f_i, generating feature space F_i, is defined as a function which maps the connection $\boldsymbol{c}_k$ with its context K_k (and other optional arguments) into feature space F_i:

$$f_i(\boldsymbol{c}_k, K_k, arg_1, \ldots, arg_n) \mapsto F_i, \quad i \in \{1, \ldots, n\}, \tag{5.7}$$

and each element of codomain F_i is defined as

$$\begin{gathered} e = (e_0, \ldots, e_n),\ n \in \aleph_0, \\ e_i \in \aleph \mid e_i \in \Re \mid e_i \in \Gamma^+,\ i \in \{0, \ldots, n\}, \\ \Gamma = \{a - z, A - Z, 0 - 9\}, \end{gathered} \tag{5.8}$$

where Γ^+ denotes positive iteration of the set Γ. It should be noted that stream property of network flow is omitted and replaced by final set notation for the purpose of simplified explanation. Notice that optional arguments of f_i can represent various parameters of functions performing specific feature extraction, e.g. a direction of a TCP connection, order and type of polynomial, thresholds of the data size or packet count, etc.

5.2.4 Description of ASNM Features

All features were defined in order to describe properties, process and behavior of network attacks or legitimate TCP connections. By using these features we are able to identify an attack with a high probability. For the purpose of the best representation of the TCP connections we use 195 features as a vector describing anomaly characteristics of a communication. These 195 features are in many cases, a result of reasonable parametrization of base feature functions. Types of our feature sets are depicted in Table 5.3 together with the number of them in each category. We decided to determine the naming of categories of features according to their principles, not according to static data representation. The list of original proposed features with regard to the categorization, is introduced in my Master's thesis [75].

ASNM Category	Count
Statistical	77
Dynamic	33
Localization	8
Distributed	34
Behavioral	43

Table 5.3: Distribution of ASNM features

Statistical Features

In this category of proposed features, statistical properties of TCP connections are identified. All packets of the TCP connection are considered in order to determine count, mode, median, mean, standard deviation, ratios of some header fields of packets or the packets themselves. This category of features partially uses a time representation of packets occurrences contrary to the dynamic category definition. Therefore, it includes particularly dynamic properties of the analyzed TCP connection, but without any context of it. Most of the features in this category also distinguish inbound and outbound packets of analyzed TCP connection. In total, 77 statistical features were defined. See Appendix D.1 for listing of statistical features with brief description.

Dynamic Features

Dynamic features were defined in order to examine dynamic properties of the analyzed TCP connection and transfer channel such as speed or error rate. These properties can be real or simulated. Eighteen of the features consider the context of an analyzed TCP connection. The difference between some of the statistical and dynamic features from a dynamic view can be demonstrated on two instances of the same TCP connection, which performs the same packet transfers, but in different context conditions and with different packet retrasmitions and attempts to start or finish the TCP connection. There were 33 dynamic features defined in total. Many of them distinguish between inbound and outbound direction of the packets and consider statistical properties of the packets and their sizes as mentioned in statistical features. See Appendix D.4 for listing of dynamic features with brief description.

Localization Features

The principal character of localization features category is that it contains static properties of the TCP connection. These properties represent the localization of participating machines and their ports used for communication. In some features localization is expressed indirectly by a flag, which distinguishes whether participating machines lie in a local network or not. Features included in this category do not consider the context of the analyzed TCP connection, but they distinguish a direction of the analyzed TCP connection. We defined 8 localization features. See Appendix D.2 for listing of localization features with brief description.

Distributed Features

The characteristic property of distributed features category is the fact that they distribute packets or their lengths to a fixed number of intervals per unit time specified by a logarithmic scale (1s, 4s, 8s, 32s, 64s). A logarithmic scale of fixed time intervals was proposed because of a better performance of used classification methods. The next principal property of this category is vector representation. All these features are supposed to work within the context of an analyzed TCP connection. Altogether, we defined 34 features in this category which are a result of parametrization of 2 functions, which accepts parameters as unit time, threshold, direction and the context of an analyzed TCP connection. See Appendix D.3 for listing of distributed features with brief description.

Behavioral Features

Behavioral features are a set of features based on the description of the properties directly associated with TCP connection behavior. Examples include legal or illegal connection closing, polynomial approximation of packet lengths in a time domain or in an index of occurrence domain, count of new TCP flows after starting an analyzed TCP connection, coefficients of Fourier series in a trigonometric representation with distinguished direction of an analyzed TCP connection etc. We defined 43 behavioral features. Most of them use the direction of the analyzed TCP connection and six of them consider the context. See Appendix D.5 for listing of behavioral features with brief description.

5.3 Mathematical Background of ASNM Features

This section will define mathematical and algorithmic background behind the computation of ASNM features and will not consider definition of packet and connection described in the previous section, but rather explain all the definitions in general terms. The only fact related to packet and connection objects is that all the features are computed above single connection object which may contain the variable number of packets depending on connection instance. Therefore, computed features have to produce the output with constant size in order to enable mutual comparison of various connection instances.

For the purpose of all the following definitions, we will suppose having a dataset of size N:

$$\boldsymbol{x} = (x_0, \ldots, x_n), \tag{5.9}$$

where $N = n + 1$, together with corresponding observations of the values of y, denoted as

$$\boldsymbol{y} = (y_0, \ldots, y_n). \tag{5.10}$$

Note that this section considers the indexation of the dataset starting from zero. At first, functions of descriptive statistics will be defined, followed by approximation by polynomials, Fast Fourier Transformation (FFT) method, and finally computation of products with Gaussian curves.

5.3.1 Functions of Descriptive Statistics

Descriptive statistics is a discipline dealing with quantitative description of the main features of a collection of information [109]. It aims to summarize samples, rather than use the data to learn about the population which is represented by data samples.

Commonly used measures for describing a dataset are measures of central tendency and measures of variability (dispersion). Measures of central tendency include the mean, median and mode, while measures of variability include the standard deviation, variance, the minimum and maximum values of the variables [194] among others. In the current work, we utilize following measures of descriptive statistics: mean, median, mode, minimum, maximum and standard deviation. These measures assume only set $\boldsymbol{x}$ for their definition. We consider definitions of the minimum and the maximum values as implicit and will not be explicitly described. The definition of **arithmetic mean** is formulated as follows:

$$\overline{x} = \frac{x_0 + x_1 + \ldots + x_n}{N} = \frac{1}{N} \sum_{i=0}^{n} x_i \ . \tag{5.11}$$

It represents the sum of the dataset values divided by the size of the dataset. Note that features in this work utilize only arithmetic mean.

Median is defined as the value which separates the higher half of a data samples from the lower half ones. Median of dataset of observations $\boldsymbol{x}$ can be obtained by sorting the observations from the lowest value to the highest one, and then picking the middle value. In the case of odd number of observations, the situation is straightforward and median is directly picked. However, in the case of the even number of observations, the median is computed as a mean of two middle values [109]. The principal advantage of the median over the mean in describing dataset in a way which is resilient to extremely large or extremely

Algorithm 3: Computation of median

Input: $\boldsymbol{x} = (x_0, \ldots, x_n)$
Output: median of $\boldsymbol{x}$

1: $\boldsymbol{x^o} = \text{Sort}(\boldsymbol{x})$
2: **if** *(N % 2 == 1)* **then**
3: median = $\boldsymbol{x^o}_{n/2}$
4: **else**
5: median = $\frac{\boldsymbol{x^o}_{\lfloor n/2 \rfloor} + \boldsymbol{x^o}_{n/2+1}}{2}$
6: **end if**

small values. The pseudo code for computation of median value is depicted in Algorithm 3.

The **variance** is the expectation of the squared deviation of a random variable from its mean in probability theory and statistics, however, in the descriptive statistics its meaning can be reformulated and stands for squared deviation of an observation of a dataset from the mean value of a dataset. The variance is defined by the following equation:

$$\begin{aligned} \sigma^2 &= \frac{1}{N}\left[(x_0 - \overline{x})^2 + (x_1 - \overline{x})^2 \ \ldots \ - (x_n - \overline{x})^2\right] \\ &= \frac{1}{N}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2. \end{aligned} \tag{5.12}$$

Standard deviation is a measure that is utilized to quantify the amount of variation or dispersion in a set of data values [18]. And thus, standard deviation is defined as a square root of the variance:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2}. \tag{5.13}$$

A useful property of the standard deviation is that, unlike the variance, it is expressed in the same units as the data.

Mode is defined as the most common value obtained in a set of observations. A dataset with a single mode is said to be unimodal. A dataset with more than one mode is said to be bimodal, trimodal, etc., or in general, multimodal [197]. The pseudo code for computation of mode value primary aimed at unimodal datasets is shown in Algorithm 4. Note that $Map()$ is constructor for dictionary data type mapping integer keys to integer values, and assuming default values of each newly added item as zero. Also, note that function $FindKeyWithMaxVal()$ returns the lowest key, which stores the maximum value in the case of more keys containing maximum as their values. The ASNM features utilizing mode for their computation assume the lowest modus value.

5.3.2 Approximation by Polynomials

Input datasets of variable size can be approximated by polynomials which produce the output with constant size. The following information regarding approximation of polynomials by the least sum-of-square method are primarily taken from [58], revised and filled from [17].

Algorithm 4: Computation of mode

Input: $\boldsymbol{x} = (x_0, \ldots, x_n)$
Output: mode of $\boldsymbol{x}$

```
1: countsMap = new Map(<int>, <int>)
2: for (i = 0; i <= n; i++) do
3:     countsMap[x_i] += 1
4: end for
5: keyMax = FindKeyWithMaxVal(countsMap)
6: mode = countsMap[keyMax]
```

Assume functions $\varphi_0(x), \varphi_1(x), \ldots, \varphi_m(x)$ which are defined as follows

$$\begin{aligned} \varphi_0(x) = y(x) = &\quad 1 \\ \varphi_1(x) = y(x) = &\quad x \\ \varphi_2(x) = y(x) = &\quad x^2 \\ &\vdots \\ \varphi_m(x) = y(x) = &\quad x^m. \end{aligned} \tag{5.14}$$

In particular, we can fit the observations $\boldsymbol{y}$ of input data values $\boldsymbol{x}$ by polynomial function of the form

$$\begin{aligned} P_m(x, \boldsymbol{c}) = y(x, \boldsymbol{c}) &= c_0\varphi_0(x) + c_1\varphi_1(x) + \ldots + c_m\varphi_m(x) \\ &= \sum_{j=0}^{m} c_j\varphi_j(x). \end{aligned} \tag{5.15}$$

Note that although $P_m(x, \boldsymbol{c})$ is nonlinear function of x, it is a linear function of the coefficients $\boldsymbol{c}$. Functions which are linear in unknown parameters are called *linear models* [17].

The values of the coefficients are determined by fitting the polynomial to the input data. This can be done by minimizing an *error function* that measures the misfit between the function $P_m(x, \boldsymbol{c})$, for any given value c, and the input dataset. One simple choice of error function, which is widely used, is given by the sum of the squares of the errors between the predictions $P_m(x_i, \boldsymbol{c})$ for each data point x_i and the corresponding observed values y_i, so that we minimize

$$E(\boldsymbol{c}) = \sum_{i=0}^{n} \Big(y_i - P_m(x, \boldsymbol{c}) \Big)^2. \tag{5.16}$$

$E(\boldsymbol{c})$ is non negative quantity that would be zero if, and only if, the function $P_m(x, \boldsymbol{c})$ passes exactly through each input data point.

We can solve the curve fitting problem by choosing the value of $\boldsymbol{c}$ for which $E(\boldsymbol{c})$ is as small as possible. Because the error function is the quadratic function of the coefficients $\boldsymbol{c}$, its derivatives with respect to the coefficients will be linear in the elements of $\boldsymbol{c}$. Therefore, the minimization of the error function has a unique solution, denoted by $\boldsymbol{c^*}$ [17]. The error function

$$E(\boldsymbol{c}) = \sum_{i=0}^{n} \Big(y_i - c_0\varphi_0(x_i) - c_1\varphi_1(x_i) - \ldots - c_m\varphi_m(x_i) \Big)^2 \tag{5.17}$$

is minimal in the point $\boldsymbol{c}^* = (c_0, \ldots, c_m)$, for which are partial derivatives of $E(\boldsymbol{c})$ with respect to $\boldsymbol{c}$ equal to zero:

$$\frac{\partial(E(\boldsymbol{c}))}{\partial c_j} = \frac{\partial}{\partial c_j}\left[\sum_{i=0}^{n}(y_i - c_0\varphi_0(x_i) - c_1\varphi_1(x_i) - \ldots - c_m\varphi_m(x_i))^2\right] = 0, \tag{5.18}$$

where $j = 0, \ldots, m$. After derivation, we get

$$\sum_{i=0}^{n} 2(y_i - c_0\varphi_0(x_i) - c_1\varphi_1(x_i) - \ldots - c_m\varphi_m(x_i))(-\varphi_j(x_i)) = 0, \tag{5.19}$$

for $j = 0, \ldots, m$. Then, we divide equations by -2 and partition to the particular sums:

$$\sum_{i=0}^{n} y_i\varphi_j(x_i) - \sum_{i=0}^{n} c_0\varphi_0(x_i)\varphi_j(x_i) - \ldots - \sum_{i=0}^{n} c_m\varphi_m(x_i)\varphi_j(x_i) = 0, \tag{5.20}$$

for $j = 0, \ldots, m$. We can detach particular coefficients c_k from each sum and by simple arrangement we get normal equations for unknown variables $c_0, \ldots, c_m$:

$$c_0\sum_{i=0}^{n}\varphi_0(x_i)\varphi_j(x_i) + \ldots + c_m\sum_{i=0}^{n}\varphi_m(x_i)\varphi_j(x_i) = \sum_{i=0}^{n} y_i\varphi_j(x_i), \qquad j = 0, \ldots, m. \tag{5.21}$$

The corresponding system of equations is then specified in the following way:

$$\begin{aligned}
c_0\sum_{i=0}^{n}\varphi_0^2(x_i) + c_1\sum_{i=0}^{n}\varphi_1(x_i)\varphi_0(x_i) + \ldots + c_m\sum_{i=0}^{n}\varphi_m(x_i)\varphi_0(x_i) &= \sum_{i=0}^{n} y_i\varphi_0(x_i)\\
c_0\sum_{i=0}^{n}\varphi_0(x_i)\varphi_1(x_i) + c_1\sum_{i=0}^{n}\varphi_1^2(x_i) + \ldots + c_m\sum_{i=0}^{n}\varphi_m(x_i)\varphi_1(x_i) &= \sum_{i=0}^{n} y_i\varphi_1(x_i)\\
&\ddots \qquad\qquad \vdots\\
c_0\sum_{i=0}^{n}\varphi_0(x_i)\varphi_m(x_i) + c_1\sum_{i=0}^{n}\varphi_1(x_i)\varphi_m(x_i) + \ldots + c_m\sum_{i=0}^{n}\varphi_m^2(x_i) &= \sum_{i=0}^{n} y_i\varphi_m(x_i)
\end{aligned} \tag{5.22}$$

The solution of this system of equations represents coefficients $\boldsymbol{c}^*$ of the polynomial curve $P_m(x, \boldsymbol{c})$ which approximate the input dataset.

5.3.3 Fourier Transformation

At first, we will define continuous Fourier transformation, then generalize it to discrete function and denote it as Discrete Fourier Transformation (DFT), and finally discuss Fast Fourier Transformation (FFT) as optimization step of DFT. The purpose of DFT is to convert a finite sequence of equally spaced samples into the list of coefficients of complex exponential curves, ordered by their frequencies. In other words, DFT converts values of sampled function from its original domain into the frequency domain, and vice versa for the case of inverse DFT.

Continuous and Discrete Fourier Transformation

The continuous Fourier transformation [195] is defined as

$$\begin{aligned} f(\nu) &= \mathcal{F}_t[f(t)](\nu) \\ &= \int_{-\infty}^{\infty} f(t) e^{-2\pi i \nu t} dt. \end{aligned} \tag{5.23}$$

Now consider generalization to the case of a discrete function $f(t) \rightarrow f(t_k)$ by letting $f_k \equiv f(t_k) \equiv y_k$, where $t_k \equiv k_\Delta$, for $k = 0, \ldots, N-1$. This enable us to define discrete Fourier transformation $F_n = \mathcal{F}_k\left[\{f_k\}_{k=0}^{N-1}\right](n)$ as

$$F_n = \sum_{k=0}^{N-1} f_k \; e^{-2\pi i n \frac{k}{N}}. \tag{5.24}$$

Note that that input samples may be complex numbers (in practice, usually real numbers), and the output coefficients are complex numbers as well. The frequencies of the output complex exponential curves are integer multiples of a fundamental frequency, whose corresponding period is given by the length of the input dataset. The inverse Fast Fourier transformation $f_k = \mathcal{F}_n^{-1}\left[\{F_n\}_{n=0}^{N-1}\right](k)$ is defined as

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n \; e^{2\pi i k \frac{n}{N}}. \tag{5.25}$$

The DFT is an invertible, linear transformation

$$\mathcal{F} : \mathbb{C}^N \rightarrow \mathbb{C}^N, \tag{5.26}$$

where $\mathbb{C}$ denotes the vectors of complex numbers. Therefore, N-dimensional complex vector has a DFT and IDFT which are as well N-dimensional complex vectors. The time complexity of computing the DFT or IDFT is $O(n^2)$, which is undesirable especially in the cases of huge input dataset.

Fast Fourier Transformation

As a response to the high time complexity of the DFT algortihm, performance improvement was designed by Cooley and Tukey [40] in 1965. However, the principles of the critical factorization step of FFT [196] can be traced to Gauss' unpublished work in 1805 when he needed it to interpolate the orbit of asteroids Pallas and Juno from sample observations [72]. The FFT algorithm reduces time complexity of the DFT from $O(N^2)$ to $O(N \; log_2(N))$.

A DFT can be computed using an FFT by means of the Danielson-Lanczos lemma if the number of points N is a power of two. If the number of points N is not a power of two, a transformation can be performed on sets of points corresponding to the prime factors of N which is slightly degraded in speed. Fast Fourier transformation algorithms generally fall into two classes: decimation in time, and decimation in frequency. The Cooley-Tukey FFT algorithm [40] first rearranges the input elements in bit-reversed order, then builds the output transformation (decimation in time). The basic idea is to break up a transformation

of length N into two transformations of length $N/2$ using the identity

$$\begin{aligned}\sum_{n=0}^{N-1} y_n \; e^{-2\pi i n \frac{k}{N}} &= \sum_{n=0}^{N/2-1} y_{2n} \; e^{-2\pi i (2n) \frac{k}{N}} + \sum_{n=0}^{N/2-1} y_{2n+1} \; e^{-2\pi i (2n+1) \frac{k}{N}} \\ &= \sum_{n=0}^{N/2-1} y_n^{even} \; e^{-2\pi i n k / \frac{N}{2}} + e^{-2\pi i \frac{k}{N}} \sum_{n=0}^{N/2-1} y_n^{odd} \; e^{-2\pi i n k / \frac{N}{2}},\end{aligned} \tag{5.27}$$

which is sometimes called the Danielson-Lanczos lemma [196].

5.3.4 Products with Gaussian Curves

First, we define algorithm of Gaussian function, which is parametrized by mean μ, variance σ^2 and input value x, for which it returns functional value. The pseudo code of the Gaussian function's value computation is shown in Algorithm 5.

Algorithm 5: Gaussian function's value computation

```
1 Function GaussFn(μ, σ², x) begin
2     return 1/√(2πσ²) e^(−(x−μ)²/(2σ²))
3 end
```

Now, we can compute the normalized sum of products of input data specified by $\boldsymbol{y}$ and one Gaussian function. The corresponding pseudo code for computation of this sum of products is shown in Algorithm 6 as dedicated function which takes $\boldsymbol{y}$ as input. Note

Algorithm 6: The sum of normalized products of $\boldsymbol{y}$ and one Gaussian function

```
1  Function NormalizedProductFn(y) begin
2      if (N % 2 == 1) then
3          μ = N/2
4      else
5          μ = (N + 1)/2 − 1
6      end if
7      σ² = ((N − 1)/6)²
8      product = 0
9      for (i = 0; i <= n; i++) do
10         product += y_i × GaussFn(μ, σ², y_i)
11     end for
12     return product / N
13 end
```

that this function takes as the input $\boldsymbol{y}$ values, while it assumes that $\boldsymbol{x}$ values are implicitly specified by the index of items in $\boldsymbol{y}$. Thus, $\boldsymbol{x}$ is omitted in the computation. Firstly, mean and variance are computed using index domain of $\boldsymbol{y}$. The computation of variance is not precise due to computational efficiency of the function as well as for the purpose of our

Algorithm 7: Normalized products of $\boldsymbol{y}$ and k Gaussian functions

Input: k, $\boldsymbol{y}$
Output: k products of sliced $\boldsymbol{y}$ and k Gaussian curves

```
1  products = new List(<int>)
2  for (i = 0; i < k; i++) do
3      products.append(0.0)
4  end for
   /*                Allow at least 3 packets per product                */
5  if (3 × k > N − 1) then
6      return products
7  end if
8  for (i = 0; i < k; i++) do
9      startIndex = ⌊i × (N−1)/(k+1)⌋
10     endIndex = ⌊(i + 2) × (N−1)/(k+1)⌋
       /*                  Handle end index alignment                    */
11     if (endIndex >= N || endIndex == N - 2) then
12         endIndex = N - 1
13     end if
       /*            Compute single normalized sum of products           */
14     products_i = NormalizedProductFn(y_startIndex, ..., y_endIndex)
15 end for
```

algorithm, it is not necessary to precisely compute it. μ and σ^2 are then used as parameters of Gaussian function, which is called in the cycle as part of a product computation. When all products are summed, final sum of products is normalized by size of the input data.

After the computation of the normalized sum of products of input dataset with one Gaussian curve, we can generalize this computation for more Gaussian curves and output more products. The pseudo code which splits the input dataset into k folds and then computes the sum of products for each fold separately, is depicted in Algorithm 7. First, list of k products is initialized and each of them is set to zero. Then, constraint for enough input packet is executed. If it is not passed, then the algorithm return zero products. When constraint is passed, we can compute the $i-th$ sum of products in each iteration of the cycle by calling the $NormalizedProductsFn()$ function with required (ordered) subset of input dataset y. Note that the computation of products with Gaussian functions may overlap each other by few packets. Also note that this algorithm as well as function $NormalizedProductFn$ does not perform sorting of input dataset, instead assumes that observations of values $\boldsymbol{y}$ are sorted according to required key implicitly, and thus consider ordered data type of $\boldsymbol{y}$.

Chapter 6

Evaluation of ASNM Features

This chapter describes various data mining experiments with ASNM features of AIPS and with several machine learning based classifiers. A number of features with interesting value density distribution are highlighted. The first experiments utilize custom dataset which was captured in laboratory conditions by manual simulations of network buffer overflow attacks. Latter experiments utilize publicly available dataset CDX 2009 [164]. The most of the input data for the experiments described and performed in this chapter were obtained during my participation in the AIPS project, however presented outcomes are the results of my particular contribution.

6.1 Master's Project on Detection of Zero-day Attacks

The first experiments with AIPS and ASNM features are presented in my Master's project *Metrics for intrusion detection in network traffic* [75], which proposes network security features as well as their categorization. The project implements the feature extraction process and presents OOP model of implemented tools. It describes functionality of implemented tools, input processing, configuration settings, features generation and several data mining experiments.

The input data collection considered in the project was obtained by performing of network buffer overflow attacks on vulnerable network services deployed on high interactive honeypot Argos [8], which was the contribution of my colleague, Maroš Barabas. We leveraged exploits and functionality of Metasploit framework [113] serving for penetration purposes. Exploitation of network services was performed in laboratory conditions, and therefore this work also discusses advantages and disadvantages of laboratory conditions. The collected dataset contains 12 buffer overflow attacks and 173 legitimate communication records.

Collected network traffic was passed as input to Connection Extractor and outgoing connection records were then analyzed by Metrics Extractor which produced ASNM features' records. Later, the work presents analysis of experiments and results obtained by utilization of RapidMiner [157] tool on collected feature records. Gaussian kernel density estimation method provided by Naive Bayes classifier component of RapidMiner was utilized for the purpose of value density estimation of particular features. This method represents non-parametric estimation of feature distribution.

We identified 45 features, which achieved interesting value density distribution of input records considering class membership distinguishing between malicious and legitimate communications. The identifiers of these features correspond to the list of ASNM features in

Apendix D and are enumerated in the following listing:

- `CorClosed` – correctly closed connection.
- Distributed features – `InPkt32s10i[2], InPkt32s10i[5], InPkt64s10i[1], InPkt64s20iTr1KB[0], InPkt64s20iTr1KB[1], OutPkt1s10i[1], OutPkt1s20iTr4KB[4], OutPkt32s10i[2], OutPkt64s20iTr1KB[1], OutPkt64s20iTr1KB[2], OutPktLen4s10i[0], OutPkt64s20iTr2KB[1], OutPktLen4s10i[2] and OutPktLen8s10i[3].`
- Statistical features – `MeanPktLenSrc, SigPktLenSrc, SumPktLenSrc, RatInOutPkt, CntNondPktIn` – representing average length of outbound packets, standard deviation of outbound packet lengths, the sum of outbound packet lengths, ratio of outbound to inbound packets and count of non-data inbound packets, respectively.
- Dynamic features – `CntResendPktsIn, SigTdiff2PktsIn` – representing the number of resend inbound packets and standard deviation of time intervals between pairs of consecutive inbound packets, respectively.
- Features approximating discrete slope of communications by polynomials – `PolyInd3ordOut[0], PolyInd3ordOut[2], PolyInd3ordOut[3] and PolyInd8ordIn[7].`
- Features approximating communications by Fourier series – `FourGonAngleOut[14], FourGonAngleOut[1], FourGonAngleOut[3], FourGonAngleOut[8], FourGonModulOut[0], FourGonModulOut[1], FourGonModulOut[2], FourGonModulOut[4], FourGonAngleIn[7], FourGonmodulIn[3] and FourGonModulIn[5].`
- Normalized products of packet lengths with Gaussian curves – `GaussProds2Out[1], GaussProds4In[0], GaussProds4Out[0], GaussProds4Out[2], GaussProds8All[7], gaussProds8Out[2], GaussProds8Out[3] and GaussProds8Out[7].`

Three representative examples of interesting values density distributions were selected from above mentioned list:

- `SigTdiff2PktsIn` – standard deviation of time differences of packets' occurrences in input direction, which is illustrated in Figure 6.1. The red color represents values of the feature in malicious communication cases and blue color in legitimate ones.

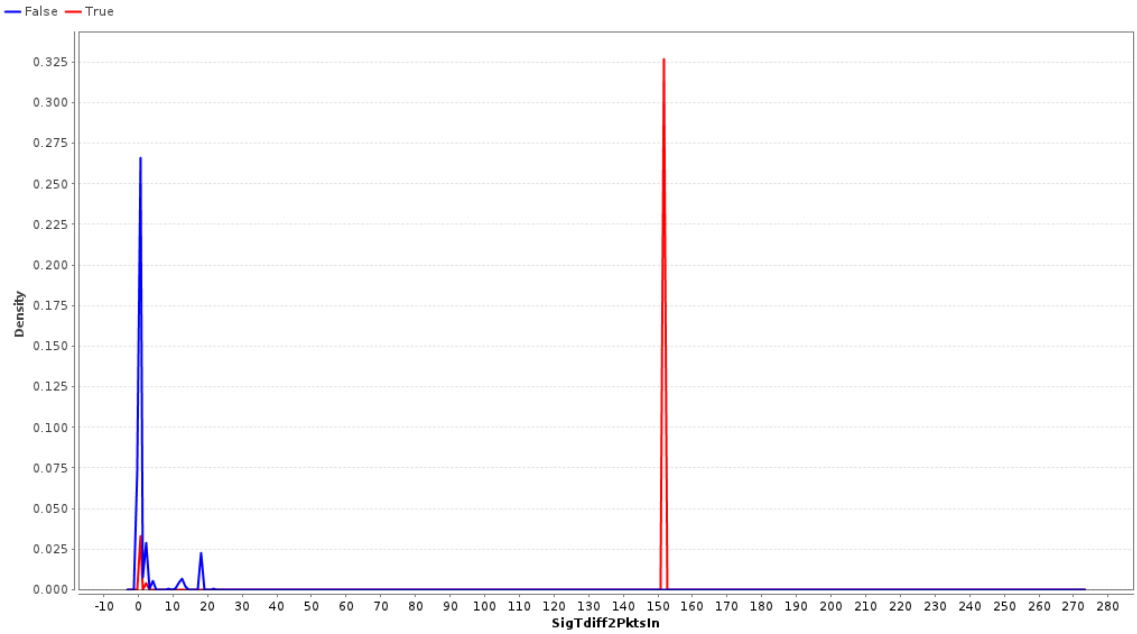

Figure 6.1: Standard deviation of packet IAT in inbound traffic [75] ($SigTdiff2PktsIn$)

- `PolyInd3ordOut[3]` – progress of output communication approximated by polynomial of the 3rd order, which is illustrated in Figure 6.2. The interpretation of colors is the same as in the previous case. Note that the feature represents the 4th coefficient of the approximation.

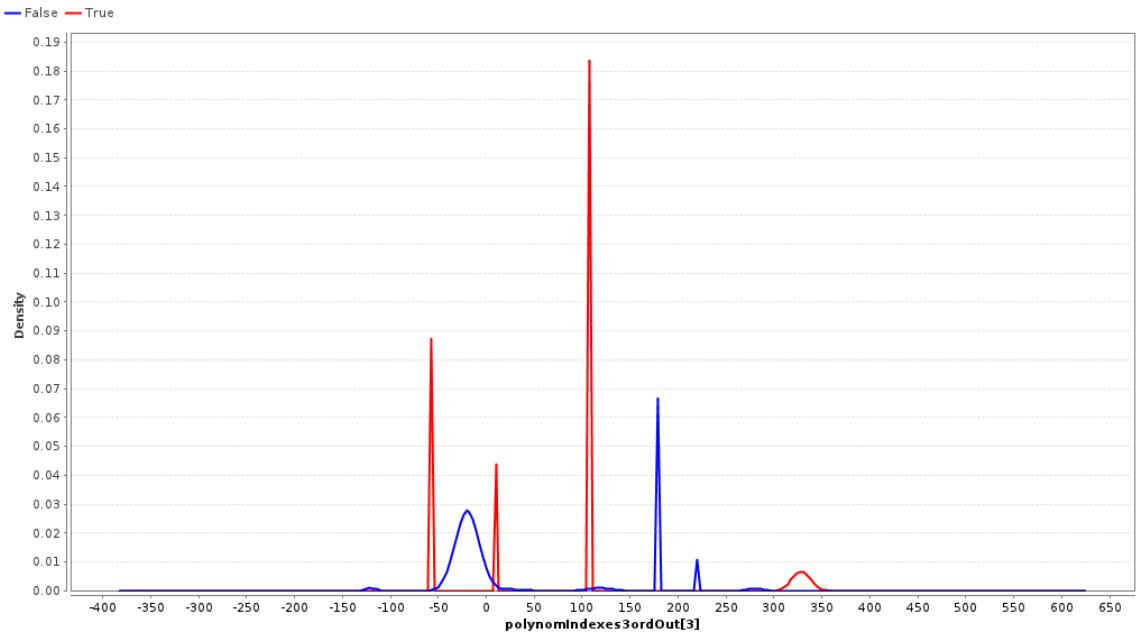

Figure 6.2: Approximation of inbound communication by polynomial of 3 order [75] ($PolyInd3ordOut[3]$)

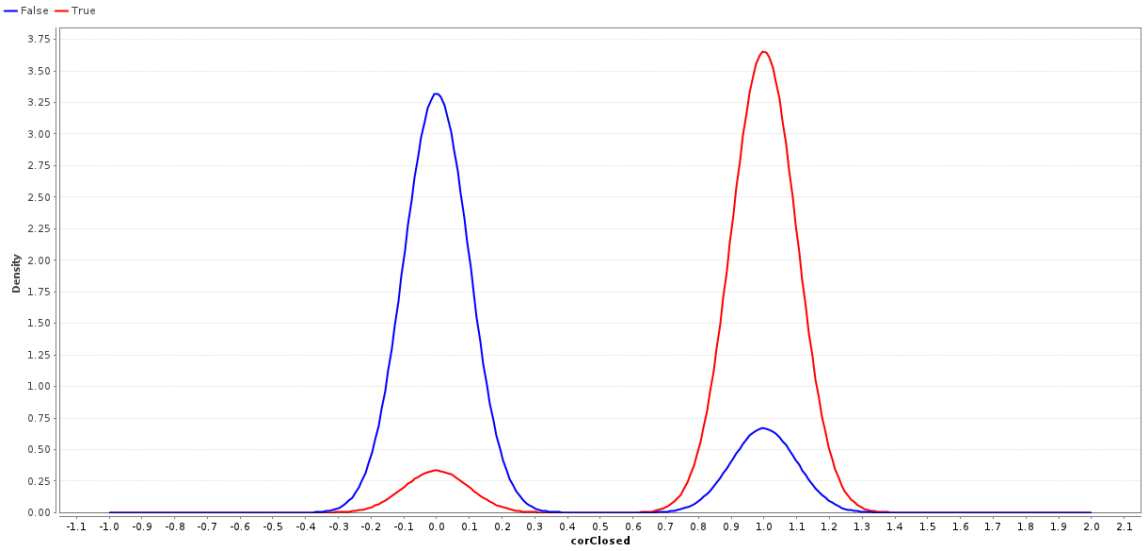

Figure 6.3: Indication of correctly closed connection [75] (*CorClosed*)

- `CorClosed` – indication of correctly closed TCP connection, which is illustrated in Figure 6.3. The interpretation of colors has the same meaning as in the previous cases.

6.1.1 Classification Models

The next experiments of the project are aimed at classification of TCP communications with use of the subset of the original feature set. These features were manually selected according to several criterias e.g. value density analysis and Forward Features Selection (FFS) experiments; and they correspond to the listing of 45 features enumerated in aforementioned rows. The experiments were carried out using three classification models: Support Vector Machines, Naive Bayes Classifier and Decision Tree. Data preprocessing techniques including discretization of attributes into specified number of bins and principal component analysis method were utilized. Also, experiment enabling stratified sampling method included into the training process was performed and showed to be successful. Therefore, we leveraged stratified sampling method in other experiments as well. All the experiments were performed using 5-fold cross validation method. Summary of the results achieved by various classifiers with dedicated adjustments is shown in Table 6.1, which horizontally orders approaches according to classification accuracy.

Prediction Method		Decision tree with gini index	Naive Bayes classifier with discretization of ordinal attributes	Naive Bayes classifier and PCA with automatic count of components	Naive Bayes classifier and PCA with fixed count of components	Naive Bayes classifier	SVM with neural kernel
Legit.	**Recall**	97.69%	94.22%	97.11%	88.44%	87.28%	80.92%
	Precision	97.13%	99.39%	94.38%	98.71%	99.34%	100.00%
	F_1-measure	97.41%	96.73%	95.72%	93.29%	92.92%	89.45%
Attacks	**Recall**	58.33%	91.67%	16.16%	83.33%	91.67%	100.00%
	Precision	63.64%	52.38%	28.57%	33.33%	33.33%	26.67%
	F_1-measure	60.87%	66.66%	20.64%	47.61%	48.88%	42.10%
Average Recall		78.01%	92.95%	56.63%	85.88%	89.47%	90.46%
Overall Accuracy		95.14%	94.06%	91.92%	88.10%	87.57%	82.17%

Table 6.1: Summary of classification results [75]

The only method which was able to correctly predict all the attack instances was SVM. But on the other hand, it suffered from low recall of legitimate traffic class. However, it was not even possible to alleviate this phenomenon by adjusting penalty parameter of the SVM classifier. The best recall of legitimate class was achieved by the Decision Tree model, and thus this fact mostly influenced the overall classification accuracy. Although, in this case we were able to correctly predict only circa half of the attacks. The trade-off between these two models represents Naive Bayes Classifier with attribute discretization, which was able to correctly predict more than 90% of legitimate communications as well as attack ones, and therefore achieved the best average recall of the classes. Confusion matrices related to this section are placed in Appendix F.1.

6.2 Advanced Experiments with BO Attacks

Master's thesis, described in the previous section, was used as the base for latter experiments which were performed on the same input dataset as in Section 6.1, but with improved approaches to settings of classification methods and optimizations. We realized that distribution of records in our dataset is not optimal, but we had not available another dataset for repeating the experiments. Therefore, we slightly alleviate impact of low number of attack instances in our dataset by adjusting cross validation method to utilize only 3-fold cross validations (instead of 5 from the previous section). Also, statistically significant amount of validations performed with each model was performed employing various selections of instances into particular folds of cross validation method. Therefore, the performance results were expected to be more representative than the ones from Section 6.1.

In order to find optimal parameters of each classification method, we used grid combination components of mining tool. We found rough values at first, and then, we tried to optimize them by lower scales. We have also experimented with data preprocessing phase:

Prediction Method		SVM with radial kernel	Decision tree with gini index	Naive Bayes classifier and PCA with automatic count of components	Naive Bayes classifier with discretization of ordinal attributes	Naive Bayes classifier	Naive Bayes classifier and PCA with fixed count of components
Legit.	**Recall**	99.42%	98.27%	98.84%	98.27%	86.13%	85.55%
	Precision	96.09%	94.97%	94.48%	94.97%	93.13%	93.08%
	F_1-measure	97.73%	96.59%	96.61%	96.59%	89.49%	89.16%
Attacks	**Recall**	41.67%	25.00%	16.67%	25.00%	8.33%	8.33%
	Precision	83.33%	50.00%	50.00%	50.00%	4.00%	3.85%
	F_1-measure	55.56%	39.58%	25.00%	39.58%	5.41%	5.26%
Average Recall		70.54%	61.63%	57.76%	61.63%	47.23%	46.94%
Overall Accuracy		95.68%	93.51%	93.51%	93.51%	81.08%	80.54%

Table 6.2: Summary of the results per classification method [15]

we used discretization of ordinal attributes and principal component analysis method for finding principal attributes. Results of these experiments are shown in Table 6.2, where the methods are horizontally ordered by classification accuracy [15].

From the perspective of average recall and classification accuracy, we achieved the best results by the SVM method. Note that SVM utilized radial kernel in comparison to neural kernel of SVM in experiments from Section 6.1. Also, Decision Tree and Naive Bayes achieved interesting results likewise the previous section.

We also compared classification models by ROC method. Considering neutral bias, the best configuration of TPR and FPR were achieved by SVM method, however Naive Bayes also contained one interesting configuration for TPR equal to 0.65. ROC diagram with neutral bias is depicted in Figure 6.4. Note that comparing of ROC ran above cross validation method, and thus generated certain variability, which was amplified by non-optimal dataset distribution. The variability of the performance results are shown in the figure by line-adjacent transparent areas.

All experiments were performed in laboratory conditions, therefore some differences from real environment might influence the results. Laboratory conditions of experiments may differ mainly in context-dependent features, where the context was generated only by two laboratory hosts (the attack machine and the vulnerable one). Also, several features depend on transmission time of packets in the analyzed traffic. In a real traffic, more nodes are present along the route between the attacker and the detector, and also this path can be dynamically selected according to actual network conditions, but in laboratory conditions these parameters are constant. Other influence may relate to TCP retransmission of packets. Also note that distribution and size of our testing dataset was not optimal, and therefore the results presented in the current section and the previous one are considered as preliminary.

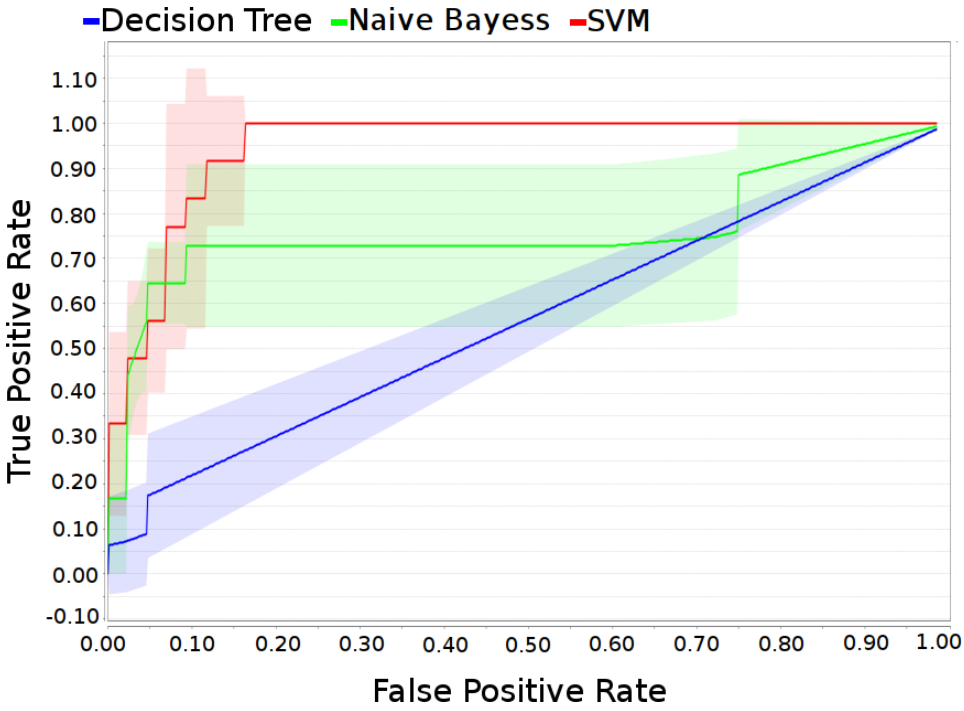

Figure 6.4: ROC diagram comparing classifiers [15]

6.3 Experiments with CDX 2009 Dataset

As our previous results were dependent on our own dataset with the low number of attack instances and also legitimate ones, we were encouraged to test the detection properties of ASNM features on another dataset as well as we would like to compare performance of ASNM with another similar approach applicable in NBAD. Therefore, we searched for similar set of features like ASNM, and found features which are called discriminators and are proposed by Andrew Moore et al. in their paper [119].

The performance of our behavioral ASNM features was evaluated in our paper [76] which compares them with discriminators of A. Moore. The authors of the paper consider only TCP connections to perform extraction of discriminators in the same way as we do. So, there were equivalent conditions for performance comparison between our suggested features and discriminators suggested in the above mentioned work. 248 discriminators are defined in [119], including all particular items of vector types. Unlike their research we considered the whole particular vector metric as one in the ASNM definition. In the work of A. Moore, each TCP flow is described by three modes according to packet transmissions: idle, interactive and bulk. Many discriminators use these three modes as their input. The authors do not mention any explicit categorization of defined discriminators, however, the only possible categorization can implicitly follow from a direction of the TCP flow. We also performed a similarity analysis of discriminators and ASNM features definition, and discovered that there are approximately 20% of discriminators principally similar or the same as ASNM features. Unique properties of discriminators' definitions include, for example, the using of quartiles for a statistical analysis, analysis of selective acknowledgment of TCP, a number of window probe indication, pushed or duplicate packets etc.

Later, we performed comparative data mining experiments of ASNM and discriminators on CDX 2009 dataset [164], which contains SNORT log as source of ground truth. It was

discovered that the SNORT log can be associated only with data capture outside of the West Point network border and only with significant differences of timestamps – approximately 930 days. We did not find any association between the SNORT log and data capture performed by the National Security Agency. We focused only on buffer overflow attacks found in a log from SNORT IDS and a match with the packets contained in the West Point network border capture was performed. It should be noted that buffer overflow attacks were performed only on two services – Postfix Email and Apache Web Server. An example of the buffer overflow SNORT log entry:

```
[**] [124:4:1] (smtp) Attempted specific command
      buffer overflow: HELO, 2320 chars [**]
      [Priority: 3]
      11/09-14:22:25.794792
      10.2.195.251:2792 -> 7.204.241.161:25
      TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:2360
      ***AP*** Seq: 0x68750738  Ack: 0x24941B59
      Win: 0xFDC0  TcpLen: 20.
```

We used IP addresses (5th row), ports (5th row), time of occurrence (4th row), TCP sequence and acknowledgment numbers (7th row) as information to match the SNORT log entries with particular TCP connections identified in TCP dump traces.

Despite all efforts, there were exactly 44 buffer overflow attacks matched out of all 65, and these identified attacks were used as ground truth for the data mining process. In order to correctly match SNORT entries, it was necessary to remap IP addresses of the internal to external network because a SNORT detection was realized in the internal network and TCP dump data capture contains entries from outside the IP address space. Buffer overflow attacks, which were matched with data capture, have their content only in two TCP dump files: *2009-04-21-07-47-35.dmp*, *2009-04-21-07- 47-35.dmp2*. Due to the enormous count of all packets (approximately 4 millions) in all dump files, only two files were considered which contained 1 538 182 packets. We also noticed that network data density was increased in the time when attacks were performed. Consequently, we made another reduction of packets, which filters enough temporal neighborhood of attacks occurrences. In the result, 204 953 packets for the next phases of our experiments were used.

The whole process of ASNM features and discriminators extraction with data mining comparison is illustrated in Figure 6.5. There are four segments and data flow direction from top to bottom depicted in the figure. Empty boxes represent data as input or output of some processes and filled ovals represent working components which perform some action. A working component takes input data and outputs output data. The upper segment represents the input of the whole experiment process and includes input data files: CDX 2009 TCP dump files and CDX 2009 SNORT log file. The CDX 2009 TCP dump files are the mutual input of both extraction processes. The input of ground truth (CDX 2009 SNORT log file) is directly provided to the feature extraction process and is indirectly bounded to extracted discriminators after the end of ASNM extraction process. The left segment contains phases of discriminators extraction and the right segment contains the ASNM feature extraction process with expert knowledge processing.

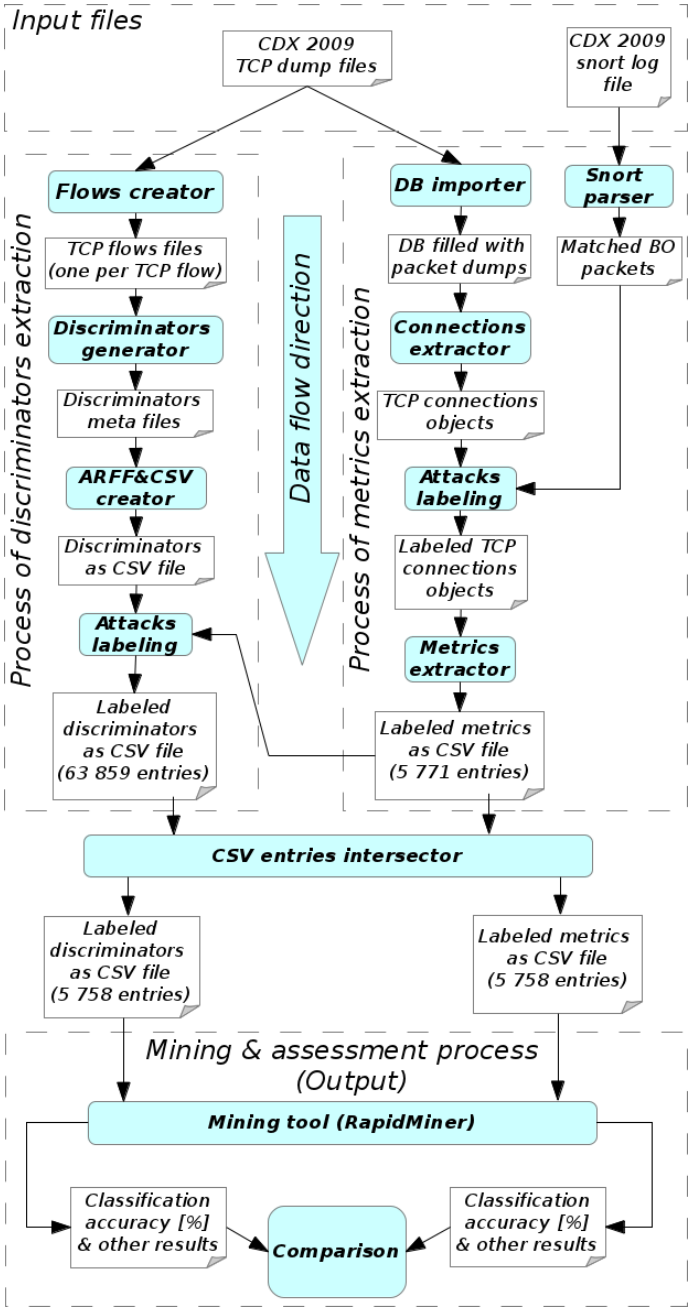

Figure 6.5: The Process of feature extraction and assessment [76]

ASNM Feature Extraction Process

The feature extraction process of the right segment includes a process described in Section 5.2. An all packets set P is represented by the input of CDX 2009 TCP dump files, which are imported into the database by a DB importer component. Next, an active component Connection extractor performs the identification of all TCP connections set C in all packet set P. The extraction of TCP connections was followed by expert knowledge information processing, which means matching of extracted TCP connections with parsed SNORT log information. If a match occurred, the TCP connection is labeled as an attack by the

Attacks labeling component. Then, ASNM features extraction is performed for each TCP connection in C by the Metrics extractor component and the results of this step are ASNM features values for each TCP connection object in CSV file. It should be noted that the ASNM feature extraction process is independent of expert knowledge information.

Discriminators Extraction Process

The input of this process is the same as in the ASNM feature extraction process. The component Flows creator performs the identification of TCP connections by netdude tool[1] and it creates a TCP dump file for each identified TCP connection. These TCP dump files are used as an input for Discriminator generator component, which performs extraction of discriminators for each identified TCP connection. This component performs equivalent operation as Metrics extractor component in the process of ASNM feature extraction. It generates discriminators meta files which contain intermediate results of discriminators values. These meta files are processed and joined by the ARFF&CSV creator component into a CSV file. After this step, the attack TCP connections are labeled, which is performed by the the Attack labeling component.

Mining & Assessment Process

This process is depicted in the lower part of figure 6.5. Before this process takes place, it is necessary to make an intersection between output CSV files of ASNM features and discriminators extraction processes, which is performed by the CSV entries intersector component. At the output of this step there are ASNM features and discriminators of the same TCP connections objects, so there are equivalent conditions for the data mining process. Two intersected CSV files with an equal number of entries are used as the input of the Mining tool component and output consists of classification accuracy and other results suitable for comparison. It should be noted that we found 5 771 TCP connections by our TCP connections extractor and 63 859 TCP connections by the TCP demultiplexer from netdude framework which is used by discriminators extraction. The main reason of it is the fact that in ASNM, we consider only established TCP connections because only an established TCP connection can perform a buffer overflow attack, in contrast to discriminators. The intersection of ASNM features and discriminators outputs contains 5 758 objects and 44 of them represent attacks. The intersected CSV files were used in the data mining process, and thus, there were adjusted the same conditions for both ASNM features and discriminators outputs containing the same TCP connections records.

Thirteen established TCP connections were not found by the TCP demultiplexer which is utilized by Flows creator of discriminators extraction process. The discrimination extraction was performed using a source code available from the author's web[2]. The whole process of discriminators extraction itself was not described in [119], so we deduced it from a source code and README instructions. It was also necessary to debug some functionality of provided tools. During the preparation for discriminators extraction, there occurred some compatibility issues caused by old versions of dependencies. We finally used Linux Fedora 4 as the most suitable operating system for the necessary operation.

[1] URL: http://netdude.sourceforge.net/.

[2] URL: http://www.cl.cam.ac.uk/research/srg/netos/nprobe/data/papers /sigmetrics/index.html

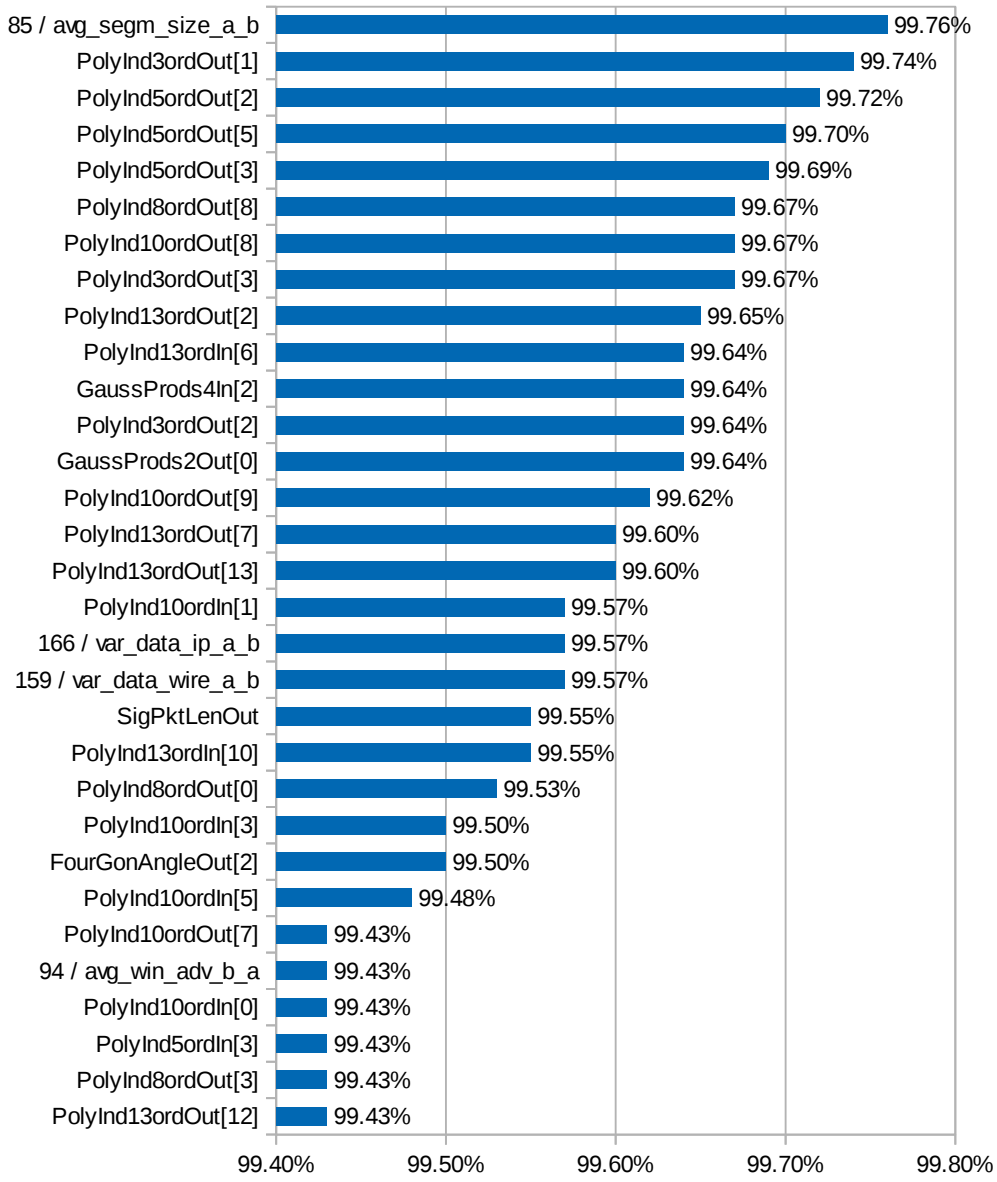

Figure 6.6: List of features sorted by overall accuracy (over 99.43%)

6.3.1 Results of the Experiments

We analyzed joined outputs of ASNM features and discriminators extraction processes by the RapidMiner[3] tool. Our training model used the Naive Bayes classifier with kernel functions for estimation of density distribution, which is considered as non-parametric estimation method. A stratified sampling with 5-cross fold validation for every experiment was performed. At first, we focused only on the accuracy evaluation of particular ASNM features and discriminators. Therefore, this experiment was adjusted for maximal classification accuracy of input data. In Figure 6.6 the best ASNM features and discriminators (over 99.43% overall accuracy) are shown, sorted by the overall accuracy. The names of discriminators consist of a number and label defined in [119], which are also enumerated

[3]URL: http://rapid-i.com/content/view/181/190/lang,en/.

in Appendix E. The names of ASNM features are defined in [75] and also their updated version is available in Appendix D. The names of polynomial approximation features consist of 5 parts: polynomial metric label, method of approximation (indexes or time), order of polynomial, direction and coefficient index. Fourier coefficient features' names consist of the Fourier coefficient metric label, the goniometric representation, the angle or the module, the direction and the coefficient index. Gaussian products features' names are a compound of the Gaussian metric label, the number of Gaussian curves, the direction and the product index (e.g. *PolyInd5OrdOut[1]*).

We can see that the best classification accuracy for the ASNM features set was achieved by several polynomial approximation features. In most of these cases we achieved better results by the output direction (client to server), but we were also able to achieve interesting results with the input direction (server to client). A good performance was also achieved by normalized products with Gaussian curves as well as by Fast Fourier Transformation (FFT). The relevance in the case of standard deviation of packets length in the output direction (*SigPktLenOut*) is also presented.

In the set of discriminators, the best results were achieved by an average segment size discriminator in the direction from client to server (*85/avg_segm_size_a_b*). It could be caused by the fact that the exploit's payload contains a huge amount of data necessary to perform buffer overflow in application and these data is segmented. Another interesting designated discriminator is the variance of bytes count in Ethernet as well as IP datagram in the destination direction (*159/var_data_wire_ab* and *166/var_data_ip_a_b*). This discriminator is equivalent to average standard deviation metric of packet length in the output direction and brings nearly equivalent results. Also, the average window advertisement in the input direction (*94/avg_win_adv_b_a*) holds relevant information potentially useful in the process of classification.

Accuracy is highly dependent on training samples parsed from captured network traffic. The training and testing samples may be biased towards a certain class of traffic. For example, valid communication (according to the separation to valid and attack connections) represents a large majority of the samples in the testing dataset (approximately 99.24%). The reason to the high classification capability of fewer features is that classification of buffer overflow attacks was highly predictable due to the size of data in segmented packets, which caused the overflow and the nature of a valid communication with a small number of segmented packets.

Comparison of ASNM and Discriminators

Next, the Forward Feature Selection (FFS) method was utilized in order to select the most relevant features per each input CSV file (ASNM features and discriminators). FFS started to run with an empty set of features and in each iteration added a new feature contributing by the best improvement of average recall of all classes. The average recall of all classes was computed using the underlying 5-fold cross validation method employing the Naive Bayes classifier. Also, FFS accepted one iteration without improvement as we wanted to avoid the selection process becoming stuck in local extremes. The maximal number of selected features was limited to 20, however, it was not necessary. In the case of ASNM, average recall of all classes equal to 92.04% was achieved. The associated confusion matrix is depicted in Table 6.3. Relevant ASNM features selected by FFS method are enumerated and briefly described in the following listing, where the order of features corresponds to the order of their selection by FFS:

Classification Accuracy: 99.86% ±0.07		True Class		Precision
		Legit. Flows	Attacks	
Predicted Class	Legit. Flows	5726	7	99.88%
	Attacks	1	37	97.37%
	Recall	99.98%	84.09%	$F_1 = 90.24\%$

Table 6.3: FFS on ASNM features and Naive Bayes classifier

- `PolyInd3ordOut[0]` – approximation of outbound communication (from client to server) by polynomial of 3rd order in the index domain of packet occurrences. The feature represents the 1st coefficient of the approximation,
- `PolyInd3ordOut[3]` – the same as the previous one, but the feature represents the 4th coefficient of the approximation,
- `PolyInd8ordOut[6]` – the same as the previous ones, but the feature represents the 7th coefficient of the approximation,
- `InPkt1s10i[7]` – lengths of inbound packets occurred in the first second of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 8th interval,
- `InPkt1s10i[0]` – the same as the previous one, but it represents the 1st interval,
- `InPkt1s10i[1]` – the same as the previous one, but it represents the 2nd interval,
- `GaussProds8All[7]` – normalized products of all packet sizes with 8 Gaussian curves. The feature represents a product of the 8th slice of packets with a Gaussian function which fits to the interval of the packets' slice.

Regarding disciminators, average recall of classes equal to 99.94% was achieved, and associated final confusion matrix, obtained by FFS, is depicted in Table 6.4. The following

Classification Accuracy: 99.94% ±0.04		True Class		Precision
		Legit. Flows	Attacks	
Predicted Class	Legit. Flows	5707	0	100.00%
	Attacks	7	44	86.27%
	Recall	99.88%	100.00%	$F_1 = 92.62\%$

Table 6.4: FFS on discriminators of A. Moore and Naive Bayes classifier

listings enumerates and describes discriminator features selected by FFS method:

- `99/ttl_stream_length_a_b` – the theoretical stream length, which is calculated as the difference between the sequence numbers of the SYN and FIN packets, giving the length of the data stream seen,
- `95/initial_window-bytes_a_b` – the total number of bytes sent in the initial window i.e., the number of bytes seen in the initial flight of data before receiving the first ACK packet from the other endpoint,

- `209/time_since_last_connection` – time elapsed since the last connection between client and server hosts,
- `87/max_win_adv_a_b` – if the connection is using window scaling, this is the maximum window-scaled advertisement seen in the connection,
- `159/var_data_wire_a_b` – variance of bytes in the Ethernet packet,
- `64/FIN_pkts_sent_b_a` – the number of all the packets seen with the FIN bit set in the direction from server to client,
- `238/FFT_a_b_#10` – FFT of packet IAT in outbound direction,
- `100/ttl_stream_length_b_a` – the same as the `99/ttl_stream_length_a_b`, but considering inbound direction,
- `8/max_IAT` – the maximum inter-arrival time of packets considering both directions of a communication,
- `3/min_IAT` – the minimum inter-arrival time of packets considering both directions of a communication,
- `4/q1_IAT` – the first quartile of inter-arrival time of packets considering both directions of a communication,
- `248/FFT_b_a_#10` – FFT of packet IAT in inbound direction.

The order of the features in the listing correspondents to the order of the selection by FFS. Later, we merged ASNM and discriminator features together, and executed FFS on merged feature set. Average recall of classes equal to 99.99% was achieved, and associated confusion matrix is depicted in Table 6.5. The features selected by FFS method are enumerated in

Classification Accuracy: 99.98% ±0.03		**True Class**		**Precision**
		Legit. Flows	**Attacks**	
Predicted Class	**Legit. Flows**	5713	0	100.00%
	Attacks	1	44	97.78%
	Recall	99.98%	100.00%	$F_1 = 98.87\%$

Table 6.5: FFS on merged ASNM and discriminator features

the following listing, which orders them to the order of selection by FFS:

- `99/ttl_stream_length_a_b` – described in the previous listing,
- `95/initial_window-bytes_a_b` – described in the previous listing,
- `FourGonModulIn[5]` – FFT of all packet sizes. The feature represents the angle of the 6th coefficient of the FFT in goniometric representation,
- `PolyInd13ordIn[9]` – approximation of inbound communication by polynomial of 13th order in the index domain of packet occurrences. The feature represents the 10th coefficient of the approximation,

- `InPkt4s10i[8]` – lengths of inbound packets occurred in the first 4 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 9th interval,
- `InPkt1s10i[1]` – the same as the previous one, but computed above the first second of a connection. The feature represents totaled inbound packet lengths of the 2nd interval,
- `InPkt1s10i[0]` – the same as the previous one, but it represents totaled inbound packet lengths of the 1st interval,
- `248/FFT_b_a_#10` – described in the previous listing.

The performance results of the ASNM and discriminator features validated on CDX 2009 dataset were very similar, however, discriminators achieved better average recall as well as F_1-measure, while classification accuracy was almost the same in both cases. Note that in the case of discriminators, 12 features were selected, while in the case of ASNM, FFS picked only 7 features. On the other hand, when we merged both feature sets, the results outperforming performances of individual feature sets were achieved, and selected features contained representatives from both feature sets.

Comparison of Classifiers

Finally, we measured a performance of various binominal classification models with the input compound of values of just ASNM features obtained in the first FFS experiment of the previous section. All the current experiments utilized 5-fold cross validation method, and thus they had equivalent conditions for performance comparison. At first, we evaluated performance of the Decision Tree classifier, which utilized gini index as selection criterion for splitting of attributes. The adjacent confusion matrix is depicted in Table 6.6. Next,

Classification Accuracy: 99.71% ±0.07		**True Class**		**Precision**
		Legit. Flows	**Attacks**	
Predicted Class	**Legit. Flows**	5721	11	99.81%
	Attacks	6	33	84.62%
	Recall	99.90%	75.00%	$F_1 = 79.52\%$

Table 6.6: Performance of the Decision Tree classifier

we evaluated performance of the SVM classifier which utilized radial basis function as non-linear kernel function. The best results were achieved with cost parameter equal to zero. Adjacent confusion matrix is depicted in Table 6.7.

Additionally, we ran FFS with 5-fold cross validation on Decision Tree classifier and achieved average recall of 95.42% and accuracy equal to 99.86%. The associated confusion matrix and the model of the classifier is shown in Appendix F.2.1. The reason why we do not run FFS on SVM classifier model is high time complexity of the FFS combined with the SVM classifier.

At the summary, we emphasize that the best performance results were achieved by the Naive Bayes classifier (see Table 6.3). Moreover, the Naive Bayes classifier is the fastest method among utilized ones. These facts led us to primarily utilize Naive Bayes classifier in our further experiments.

Classification Accuracy: 99.81% ±0.06		True Class		Precision
		Legit. Flows	Attacks	
Predicted Class	Legit. Flows	5726	10	99.83%
	Attacks	1	34	97.4%
	Recall	99.98%	77.27%	$F_1 = 86.07\%$

Table 6.7: Performance of the SVM classifier

Finally, all the classification models were compared by ROC method. Considering neutral bias, the most interesting configuration of classifier was achieved by Naive Bayes classifier. ROC diagram with neutral bias is depicted in Figure 6.7. Note that comparing of ROC ran above cross validation method, and thus generated certain variability, which is shown by line-adjacent transparent areas. In this case, the variability of ROC results was more moderate than in the case of ROC experiments in Section 6.2.

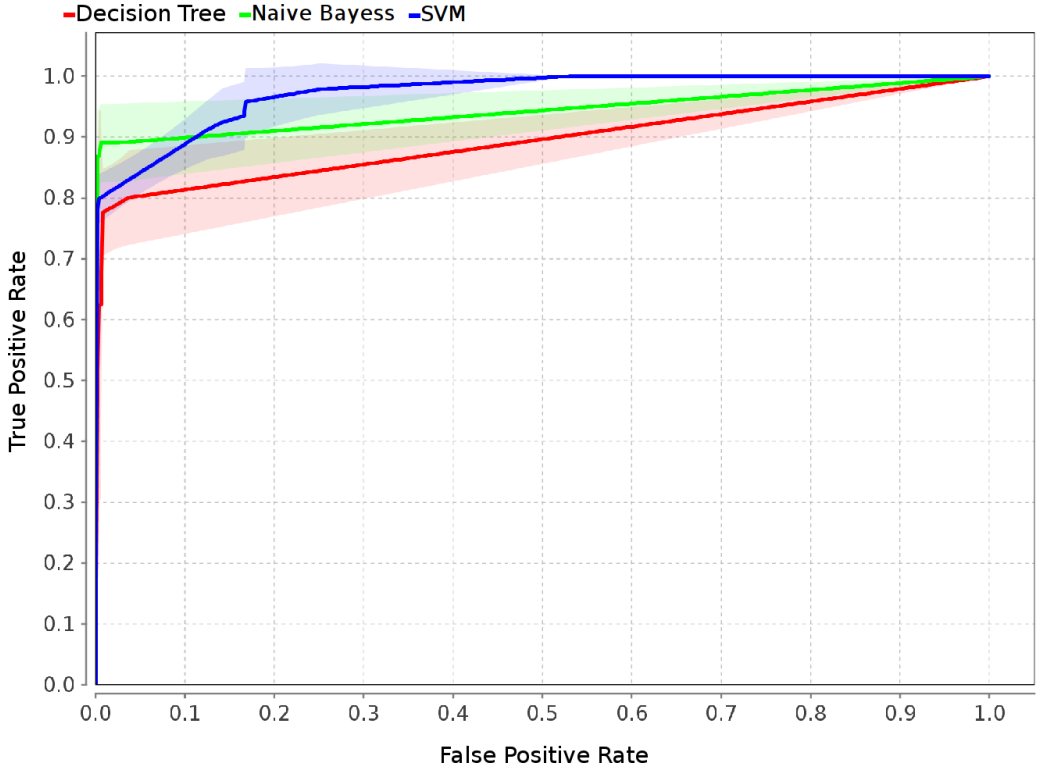

Figure 6.7: ROC diagram comparing classifiers

6.4 Performance Improvement of AIPS

The datasets utilized in our previous experiments were not generated by using any obfuscation techniques, therefore, new challenge for evaluation and comparative study of AIPS ADS arise. Respecting the main goal of this thesis – evaluation and performance improvement of IDS utilizing NBAD principles for detection of network attacks – we aim at weaknesses of the ASNM features in the following two chapters.

The principal technique which will be used to improve performance of AIPS is the idea of evading detection capability of the system's ASNM features. Because ASNM features are primarily based on behavioral and statistical analysis, which often use time and index slope of a connection and analysis of payload size distribution, there can arise question of breaching detection capability of AIPS. The most of the ASNM detection features use information gathered from L3 and L4 layers of packet headers. If such information would be intentionally modified, then it might influence the detection capability of ASNM in negative way. On the other hand, such modifications could be included into the training process of the classifier and might help in further detection of similar modified intrusions evading ADS detection.

6.4.1 Assumption I – Undetected Evasion

Consider a supervised classifier trained by ASNM features extracted from captured TCP dumps of legitimate and malicious traffic and with use of the ground truth. Now, such classifier should be able to detect malicious intrusive traffic which has similar statistical and behavioral properties like training data. But, if we craft the properties of previously known intrusion, then the classifier will be less likely to detect it and undetected evasion may occur.

6.4.2 Assumption II – Learning from Feedback

Consider a supervised classifier trained on network intrusions and legitimate traffic utilizing a number of obfuscation techniques which modify statistical and behavioral properties of a communication. If we perform network intrusion with the crafted properties, then such classifier will be more likely to detect the attack in contrast to situation when it is not trained with inclusion of obfuscated malicious network traffic.

6.4.3 Proposed Obfuscation Techniques

We propose several obfuscation techniques in order to verify designated assumptions. Proposed obfuscation techniques are divided into two categories:

- **A) tunneling obfuscation**

  – the first one represents tunneling obfuscation technique which is based on tunneling a network communication through HTTP and HTTPS protocols, and therefore, enables us to modify the statistical and behavioral properties of a communication representing a network attack,

- **B) non-payload-based network obfuscations**

  – the second one represents various non-packet-payload-based network obfuscation techniques. At the abstract level, non-payload-based obfuscation techniques represent the modifications of various statistical and behavioral properties of network traffic. At the lower level, this category of obfuscation techniques is based on modification of several properties at L3 and L4 layers of TCP/IP reference network model. Proposed examples include the following ideas:

  - spreading out packets in time,
  - segmentation & fragmentation,
  - changing of packet order,

- simulation of unreliable network channel,
- packet loss,
- duplication of packets,
- combinations of these techniques are suggested for use as well.

These two categories of obfuscation techniques will be closely analyzed in the two following chapters. The experiments investigating properties of tunneling obfuscations will be performed in Chapter 7, while the experiments investigating potential of non-payload-based obfuscation techniques will be presented in Chapter 8.

Chapter 7

Tunneling Obfuscation Technique

This chapter aims to examine the detection properties of ADS validated on obfuscated network buffer overflow attacks. In this chapter, ADS is represented by ASNM features and a supervised classifier. The current obfuscation is performed by tunneling malicious network intrusions in HTTP and HTTPS protocols with the intention of simulating the usual legitimate characteristics of the HTTP traffic's flow. These protocols wrap a malicious communication between an attacker situated outside of an intranet and a callback located inside of an intranet. Exploitation of services is performed in a virtual network environment by using network traffic modifications simulating real network conditions. Captured data is examined by NIDSs SNORT and SURICATA as well as by the ASNM network features and a supervised classifier. The next purpose of this chapter is to describe characteristics of tunneled network buffer overflow attacks in contrast with characteristics of directly simulated attacks. Note that tunneling obfuscation modules presented in this chapter were originally designed by my colleague Daniel Ovšonka in his Master's thesis [129], and later we employed it for experiments with ASNM which are published in our papers [77] and [78].

7.1 Method Description

This section formally describes data exchanges over network channel followed by explanation of tunneling obfuscation which aim to modify the behavioral characteristics of intrusive communication in a way of having different characteristics from original malicious communication.

Consider a session of a protocol at the application layer of the TCP/IP stack which serves for data transfer between the parts of client/server based application. The data is transferred in both directions one by one in succession:

$$\begin{aligned} &C.delivers(d_C[1], S),\\ &S.delivers(d_S[1], C),\\ &\vdots\\ &C.delivers(d_C[q], S),\\ &S.delivers(d_S[q], C), \end{aligned} \tag{7.1}$$

where C, S, q, $delivers$, $d_C[i]$ and $d_S[i]$ denote the client, the server, the number of data exchanges of the session, the method of delivering data from client to server (and vice versa),

$i-th$ sentence of application protocol's data passed from client to server where $i \in \{1, \ldots, q\}$ and vice versa, respectively.

The interpretation of the previous application data exchanges between client and server can be formulated, considering the TCP/IP stack up to the transport layer, by connection k which is constrained to connection oriented protocol TCP at L4, internet protocol IPv4[1] at L3 and Ethernet protocol at L2 layers of the TCP/IP stack. The TCP connection k is represented by the tuple $k = (t_s, t_e, p_c, p_s, ip_c, ip_s, P_c, P_s)$. Consider interpretation of tuple symbols from Table 5.2. Sets P_c and P_s contain a number of packets, where each of them can be interpreted by the packet tuple $p = (t, size, eth_{src}, eth_{dst}, ip_{off}, ip_{ttl}, ip_p, ip_{sum}, ip_{src}, ip_{dst}, ip_{dscp}\ tcp_{sport}, tcp_{dport}, tcp_{sum}, tcp_{seq}, tcp_{ack}, tcp_{off}, tcp_{flags}, tcp_{win}, tcp_{urp}, data)$. Symbols stated in the packet tuple were described in Table 5.1.

7.1.1 Definition of Tunneling Obfuscation

Consider feature extraction process of ASNM described in Section 5.2, and for simplicity, do not consider context of an analyzed connection nor optional arguments of feature extraction functions. Assume connection k_a representing an intrusive communication executed without any obfuscation. Then, k_a can be represented by ADS features

$$f(k_a) \mapsto F^a = (F_1^a, \quad F_2^a, \ldots, F_n^a) \tag{7.2}$$

which are delivered to the previously trained classifier C. Assume that C can correctly predict the target label as an intrusive one, because its knowledge base is derived from training dataset D_{tr} containing intrusive connections having similar (or the same) behavioral characteristics.

Now, consider connection k'_a which represents intrusive communication k_a executed by employment of tunneling obfuscation. The tunneling obfuscation modifies P_c and P_s packet sets of the original connection k_a by wrapping each original packet into new one. The wrapping may cause fragmentation of IP packets, and thus it can also modify the number of packets in both packet sets P_c and P_s. Also, the obfuscation modifies IP addresses (ip_c, ip_s) and ports (p_c, p_s) of the original connection. Symbols of the packet tuple whose values are sensitive to the obfuscation include all defined fields, as tunneling obfuscation creates new TCP/IP stack with unique values of L2, L3, L4 headers as well as new content of application layer data. All these modifications, especially modifications of P_c and P_s of the connection k_a, cause alteration of the original ADS features' values F^a to new ones. Thus, ADS features extracted over k'_a are represented by

$$f(k'_a) \mapsto F^{a'} = (F_1^{a'}, \quad F_2^{a'}, \ldots, F_n^{a'}) \tag{7.3}$$

and have different values than features F^a of the connection k_a. Therefore, we assume that the probability of a correct prediction of k'_a-connection's features $F^{a'}$ by the previously assumed classifier C is lower than in the case of connection k_a. Also, we assume that classifier C' trained by learning algorithm A on training dataset D'_{tr}, containing obfuscated intrusion instances, will be able to correctly predict higher number of unknown obfuscated intrusions than classifier C. These assumptions will be evaluated and analyzed later.

[1] Our description and experiments are related to only IPv4 traffic, as we assume that substantial behavioral characteristics of IPv4 and IPv6 traffic are very similar.

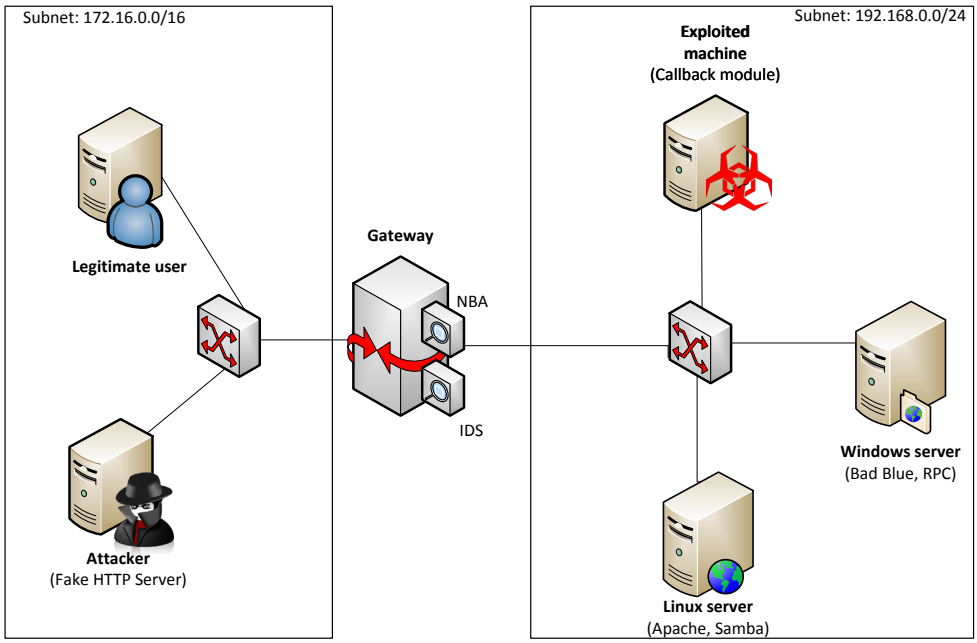

Figure 7.1: Scheme of testing virtual network architecture

7.2 Tunneling Obfuscation System

The obfuscation of malicious communications was created with the aim of similarity maximization of obfuscated network intrusions and real network traffic. The major requirement on the obfuscated traffic is obfuscation's transparency for upper network layers. Based on these requirements, the Hyper Text Transfer Protocol (HTTP) and HTTP Secure (HTTPS) protocols were selected as carrying protocols for our tunneling obfuscation infrastructure. Thus, our tunneling obfuscation is based on the encapsulation of network traffic into standard HTTP or HTTPS packets. This makes the obfuscated network data hard to detect by classic signature based approaches, and as we will show, even by anomaly detection methods. Another reason why these protocols are selected is because they are heavily spread in nearly all computer networks, and thus usually not blocked by firewalls, which gives us a high probability of obfuscated malicious communications not to be dropped or disclosed.

In our approach, we assume a private network connected to an outer network through the gateway which performs Network Address Translation (NAT) and is monitored by IDS as well as by ADS systems. Our testing virtual network architecture is shown in Figure 7.1. The left side of the picture represents the outer network with the legitimate user and the attacker. The private network with vulnerable machines and previously exploited machine are illustrated on the right side of the figure. In the middle of the figure is situated the gateway which interconnects inner and outer networks.The gateway is running on Debian Linux with a 2.6.32-5-686 kernel. The attacker's station and the exploited machine are running on Ubuntu Linux with a 3.11.0-12 kernel. The Windows server is running on the Windows 2000 with SP4 and the Linux server is running on the Red Hat Linux with a 2.4.7-10 kernel.

The obfuscation system is divided into two separate modules. The first one – called `Callback`, is deployed in the internal network, while the second one – called `Fake HTTP Server`, acts as the remote HTTP server and is deployed anywhere in a public network.

7.2.1 Callback Module

The Callback module acts as a proxy because the main function of this module is to translate incoming encapsulated communications, restore their original content and then distribute the communication into the internal network. The Callback module also forges IP addresses inside of processed packets, thus the responses of internal network can be caught by exploited machine. The caught packets are then encapsulated and sent as a callback message to the external Fake HTTP Server machine.

The installation of the Callback module into the internal network has to be done by the exploitation of a target machine laying at the internal network without detection by IDS or ADS system. The successful deployment of the callback module mostly depends on privilege escalation of a host machine. It can be performed by various approaches e.g. an attacker can use an infected website which exploits hosts browser and consequently installs a backdoor to a victim. Another way of the backdoor deployment can be executed by a user who installs it as a part of useful application from an untrusted source (trojan horse). A host machine can be also exploited directly in the case it has exposed open port to public network which is bounded to a vulnerable application. When the attacker gains access to a host machine, the Callback module can be deployed on it. For the simplicity, we did not simulate any of these options, and instead, we assumed the Callback module was previously installed in an internal network.

7.2.2 Fake HTTP Server Module

The Fake HTTP Server module of the obfuscation system waits for a connection from the Callback module. When the connection is established, all traffic affected by obfuscation is tunneled through the carrying protocol. Tunneled traffic is processed on low system level making it transparent to higher network layers. Therefore, communicating entities cannot notice the data is tunneled and they do not need to process obfuscated data. This module represents the main logic of the system because it has to apply advanced filters to all traffic and select only relevant packets for further processing by tunneling obfuscation methods.

7.2.3 Communication between Callback and Attacker

A connection is initiated from the Callback module when a request is generated. Request also contains encapsulated meta-data to identify the Callback module at Fake HTTP Server and it can contain data gathered from internal network, e.g. response from another host in private network. The attacker encapsulates data ready for obfuscation into carrying protocol and sends them to the Callback module as a standard web-server response. The Callback module restores encapsulated data and distributes them into private network. Also, the Callback module catches the responses from internal network, encapsulates them and sends them back to the Fake HTTP Server.

7.2.4 Implementation Notes

All obfuscation routines are implemented at a lower network level. Transmitted encapsulated packets are bypassed using standard `iptables` with custom dynamic rules. Dynamic rules depend on a character of obfuscated communication, therefore it is important not to process all packets due to performance impact on host system. Next, packets are processed through the `libnetfilter_queue` library which allows us to move packet processing from kernel

space to user space. The packets affected by obfuscation are encapsulated here or restored from carrying packets before they reach the kernel space for regular processing. This solution is transparent for upper network layers and the host kernel cannot detect whether packets are modified or not. The solution is also protocol independent and no other configuration of host system is necessary.

7.3 Collection of Malicious and Legitimate Traffic

We have created a set of scenarios simulating malicious and legitimate network traffic in our virtual network environment. Scenarios representing legitimate traffic were performed manually because of adding some human factor and diversity to data, and furthermore, we wanted to reach maximal authenticity of user's behavior. Other relevant legitimate traffic was collected from campus network and further annonymized.

For the purpose of malicious network traffic simulation, we utilized Metaspolit framework [113] as well as some widespread public exploits. The selection of network attacks was aimed at high impact of their possible consequences on attacked machine, and thus could be considered as intrusions. Another selection criterion was the fact, that such intrusions are easily detectable by classic NIDSs, and therefore, we could easily demonstrate functionality of our obfuscation system on signature-based approaches too.

All generated network traffic was dumped and saved on the gateway node because it processed all transmitted packets of our scenarios. Packets were dumped using the Wireshark tool [38] directly from a software network bridge interface which interconnected internal and external networks. Data obtained here served as the input for further pre-processing and data mining experiments.

7.3.1 Vulnerabilities

Four vulnerabilities were exploited – two against a Linux based machine and the next two against a Windows based one. These machines correspond to the PC stations in the right segment of Figure 7.1. Each attack exploited well-known vulnerability whose exploitation led to full privilege escalation, thus the attacker could get the root's permissions. The details about each vulnerability and its exploitation are briefly described in the following listing, where Common Vulnerabilities and Exposures (CVE) IDs [123] with Common Vulnerability Scoring System (CVSS) [124] values are shown in square brackets:

- **Apache web server with `mod_ssl` plugin 2.8.6**
  [CVE-2002-0082: 7.5] – this attack exploits buffer overflow vulnerability in `mod_ssl` plugin of the Apache web server. The plugin does not properly initializes memory in the `i2d_SSL_SESSION` function, which allows remote attacker to exploit a buffer overflow vulnerability in order to execute arbitrary code via a large client certificate which is signed by trusted Certificate Authority (CA), which produces a large serialized session [44]. This allows remote code execution and modification of any file on compromised system [152]. The vulnerable versions of plugin are in range 2.7.1-2.8.6.

- **BadBlue web server 2.72b**
  [CVE-2007-6377: 7.5] – the second attack exploits a stack-based buffer overflow vulnerability in `PassThru` functionality of `ext.dll` in BadBlue 2.72b and earlier [144]. In the attack performing phase, the special crafted packet with a long header is sent which leads to an overflow of processing buffer [47]

- **Microsoft DCOM RPC**
  [CVE-2003-0352: 7.5] – the third attack exploits vulnerability in Microsoft Windows DCOM Remote Procedure Call (DCOM RPC) service of Microsoft Windows NT 4.0, 2000 (up to Service Pack 4), Server 2003 and XP [46]. This vulnerability allows a remote attacker to execute an arbitrary code based on buffer overflow in DCOM interface. The vulnerability was originally found by the Last Stage of Delirium research group and has been widely exploited since then [153]. The vulnerability is well documented and it was used, for example, by Blaster worm.

- **Samba service 2.2.7**
  [CVE-2003-0201: 10.0] – the last attack exploits buffer overflow vulnerability in `call_trans2open` function in `trans2.c` of Samba 2.2.x before 2.2.8a, 2.0.10, earlier versions than 2.0.x and Samba-TNG before 0.3.2 [45]. This vulnerability allows a remote attacker to execute an arbitrary code. In our case public exploit was used which sends malformed packets to a remote server in batches [154]. Packets differ in a one shellcode address only because the return address depends on versions of Samba and host operating systems.

7.3.2 Real Network Conditions

For the purpose of simulating real network conditions, we executed each malicious and legitimate network communication four times in four different network traffic modifications. Our network traffic modifications should improve the relevance of further experiments' outputs and make our synthetic data looking more similar to real network traffic. We used a different configuration of virtual network environment for each network traffic modification. Network traffic modification differ in the alteration degree of the traffic on the gateway node and are divided into four categories:

(a) The first category represents reference output without any modification to configuration. All experiments ran on the same host machine to minimize deviations among different tests.

(b) The second category is dedicated to simulate traffic shaping. Therefore, all packets were forwarded with higher time delays. For this purpose, the special gateway machine with limited processor's performance was used. This machine was also fully loaded to emulate slower packets processing than in the first scenario.

(c) The third category is supposed to simulate traffic policing when some of packets were dropped during the processing on the network gateway node. In this case, a custom packet dropper was used on the gateway node and 25% of packets were dropped, resulting in output which contains re-transmitted packets.

(d) The fourth category represents transmission on an unreliable network channel, thus 25% of packets were corrupted during processing on the network gateway node.

7.3.3 Collected Dataset

The class distribution of our collected dataset is depicted in Table 7.1. The table contains connection entries created from network traffic dumps which were collected during our scenarios representing malicious and legitimate network traffic. Notice, the table also includes

Network Service	Count of TCP Connections			
	Legitimate	Direct Attacks	Obfuscated Attacks	Summary
Apache	38	102	61	201
BadBlue	95	4	10	109
DCOM RPC	4	4	8	16
Samba	15	20	8	43
Other Traffic	25	n/a	n/a	25
Summary	177	130	87	394

Table 7.1: Testing dataset distribution

connection entries created from network dumps captured at campus network. We found out that high amount of network traffic collected in campus network was malicious. Therefore, all legitimate traffic representatives captured in campus network were filtered through SNORT and SURICATA NIDSs in order to eliminate occurrence of known network attacks in legitimate traffic class. The dataset contains similar number of legitimate traffic representatives as attack ones, which is not common in real network traffic but was the result of filtering legitimate traffic.

7.4 Detection by Signature Based NIDSs

We performed detection by signature-based NIDSs as we wanted to analyze their response to our proposed obfuscation technique. In 2014, at the time of writing our papers [77, 78], we performed detection by the most current version of SNORT (with standard rule set), and inspected the results. Later, in 2016, we repeated NIDS detection by SNORT and SURICATA (through VirusTotal API) resulting into different detection properties. The following two subsections state the results of both NIDS analysis.

7.4.1 Detection by SNORT in 2014

For the purposes of signature-based network intrusion detection, we employed SNORT 2.9.4 utilizing rules with version of snapshot 2940. There were replayed few instances of direct attacks (non-obfuscated) on previously described vulnerable services and SNORT inspected them. In the result, SNORT detected all attacks instances except the attack on the Apache web service. In this case, the communication was encrypted, and thus SNORT's signatures could not be matched in the payload of packets. SNORT alert messages captured during replay of direct attacks are listed below:

- **BadBlue**:

```
ET SHELLCODE Rothenburg Shellcode
INDICATOR-SHELLCODE x86 OS agnostic fnstenv
    geteip dword xor decoder
http_inspect: LONG HEADER
```

- **DCOM RPC**:

```
ET SHELLCODE Rothenburg Shellcode
INDICATOR-SHELLCODE x86 OS agnostic fnstenv
    geteip dword xor decoder
OS-WINDOWS DCERPC NCACN-IP-TCP
IActivation remote activation
    overflow attempt
```

- **Samba**:

```
GPL NETBIOS SMB trans2open buffer
    overflow attempt
GPL NETBIOS SMB IPC$ share access
```

Next, we performed replay of our obfuscated attacks on each vulnerable service and there were not generated any alerts by SNORT NIDS. Therefore, the tunneling obfuscation of network intrusions was considered completely successful in evading detection by one representative of the signature based NIDSs.

7.4.2 Detection by SNORT and SURICATA in 2016

In this stage of our research, we performed NIDS detection by SNORT and SURICATA through VirusTotal API [191]. SNORT was equipped with Sourcefire VRT ruleset and SURICATA utilized Emerging Threats ETPro ruleset. Both of them were actualized to 15th of May 2016. The results of direct attacks' detection by both SNORT and SURICATA are shown in Table 7.2. Note that 93 direct attacks on Apache service were detected by high priority rules in both NIDSs but there were also 4 undetected direct attacks which occurred almost at the same time as some of detected attack instances, and thus we considered them as part of other detected direct attacks. Also, we can see that 5 instances of direct attacks were not detected by SNORT nor SURICATA. These 5 instances utilized (c) and (d) network traffic modifications (see Section 7.3.2) which likely influenced the detection rate of both NIDSs.

The resulting detection rates of direct attacks look the same in both NIDSs but there were differences in fired alerts during exploitation of Apache service. Unlike SNORT, SURICATA did not detect any occurrence of buffer overflow nor shellcode nor remote commad execution, but instead fired high prioroty alerts related to potencial corporate privacy violation:

```
ET POLICY
Possible SSLv2 Negotiation in Progress
Client Master Key SSL2_RC4_128_WITH_MD5,
```

which we decided to consider as correct detection. If we did not consider them as correctly detected, then SURICATA would not detect any attack on Apache service.

Next, we analyzed the detection properties of obfuscated attacks by both NIDSs and the results are depicted in Table 7.3, which distinguishes between tunneling obfuscation performed through HTTP and HTTPS protocols. We can see that average detection rate per service is much more lower for obfuscated attacks than in the case of direct attacks detection, and thus tunneling obfuscation was partially capable of evading attacks' detection by utilized NIDSs. Regarding tunneling through the HTTP protocol, both SNORT and SURICATA achieved the same low detection rate for all classes of attacks.

(a) SNORT

	Direct Attacks		
	Detected	**Total**	**%**
Apache	93 +4	102	95.10%
BadBlue	4	4	100.00%
DCOM RPC	4	4	100.00%
Samba	20	20	100.00%
Overall Detection	125	130	96.15%
ADR* per Service			98.77%

*Average detection rate.

(b) SURICATA

	Direct Attacks		
	Detected	**Total**	**%**
Apache	93 +4	102	95.10%
BadBlue	4	4	100.00%
DCOM RPC	4	4	100.00%
Samba	20	20	100.00%
Overall Detection	125	130	96.15%
ADR* per Service			98.77%

*Average detection rate.

Table 7.2: Detection of direct attacks by SNORT and SURICATA

The situation is little different for the case of tunneling through the HTTPS protocol. The SNORT achieved average detection rate per class equal to 68.75% and SURICATA only 23.25%. We found out the same fact about high priority rules fired by SURICATA on exploitation of Apache service as in the case of direct attacks detection – no buffer overflow nor shellcode nor remote command execution rules were matched, and thus we accepted previously mentioned potential corporate privacy violation alert as correct detection again. If we did not accept it, then SURICATA would not detect any tunneled attack on Apache service. Also notice that SURICATA fired one non-high-priority alert classified as potentially bad traffic in several instances of attacks tunneled through HTTPS, which exploited BadBlue, DCOM and Samba services:

```
ET POLICY
FREAK Weak Export Suite From Client (CVE-2015-0204).
```

But we did not consider it as correct detection due to the priority of the alert as well as the scope of CVE-2015-0204 is only the client code of OpenSSL. The plus notation in Table 7.3, alike Table 7.2, denotes undetected attacks that occurred almost at the same time as some other correctly detected attacks, and thus are considered as their parts.

The first difference between our NIDS analysis experiments in 2014 and 2016 is that in latter case SNORT has detected the most of direct attacks on Apache service despite it was encrypted. This indicates that VirusTotal may utilize very paranoic rule set, which causes false positives. Thus, the results of the analysis through VirusTotal API are arguable.

(a) SNORT

Service	Obfuscated Attacks								
	HTTP			HTTPS			All		
	Detected	Total	%	Detected	Total	%	Detected	Total	%
Apache	0	4	0.00	51 +6	57	100.00	57	61	93.40
BadBlue	3	6	50.00	2	4	50.00	5	10	50.00
DCOM	0	4	0.00	3	4	75.00	3	8	37.50
Samba	0	4	0.00	2	4	50.00	2	8	25.00
Summary	3	18	16.67	64	69	92.75	67	87	77.01
ADR*			12.50			68.75			51.49

*Average detection rate per class.

(b) SURICATA

Service	Obfuscated Attacks								
	HTTP			HTTPS			All		
	Detected	Total	%	Detected	Total	%	Detected	Total	%
Apache	0	4	0.00	50 +3	57	92.98	53	61	86.89
BadBlue	3	6	50.00	0	4	0.00	3	10	30.00
DCOM	0	4	0.00	0	4	0.00	0	8	0.00
Samba	0	4	0.00	0	4	0.00	0	8	0.00
Summary	3	18	16.67	53	69	76.81	56	87	64.37
ADR*			12.50			23.25			29.22

*Average detection rate per class.

Table 7.3: Detection of obfuscated attacks by SNORT and SURICATA

However, concluding the results of this section, we can state that detection capabilities of utilized NIDSs are not resistant to our proposed tunneling obfuscation technique, as high number of obfuscated attacks were not detected in comparison to the case where obfuscations were not employed.

7.5 Data Processing and Analysis in ADS

All collected data were passed into the data processing and analysis stage of our experiments. The whole process of data processing and analysis is illustrated in Figure 7.2. There are 3 segments with data flow direction shown from the top to the bottom of the scheme. Empty boxes represent data as input or output of some processes and filled ovals represent working components which perform some action. A working component takes input data and outputs output data. The upper segment represents the input of the whole experiment process and includes:

- **TCP dump files** – collected during the simulation of attacks and legitimate communications.
- **Directory structure** – it contains information about a kind of simulation (attack or legitimate), identification of a network service, identification of vulnerability by CVE.

- **Mapping of services to hosts** – it contains the IP addresses of hosts relevant to our simulations with mapping of the analyzed services to hosts. It included information about obfuscation tunnel's endpoints too.

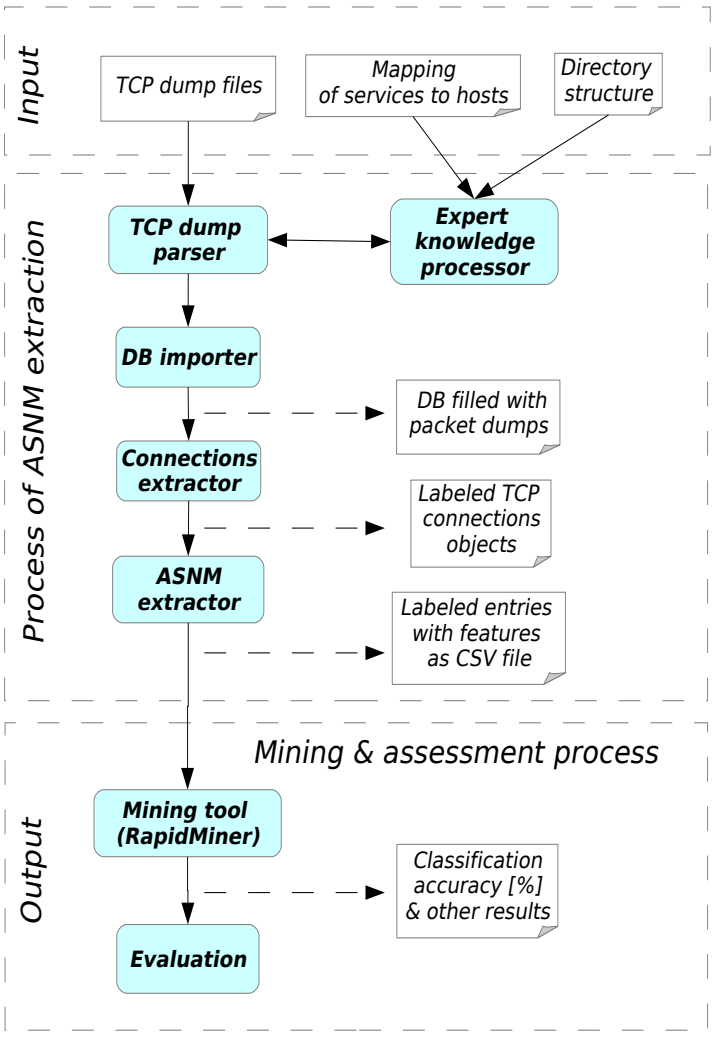

Figure 7.2: Scheme of data processing and analysis

The middle segment of the scheme represents a process of ASNM network feature extraction. Two upper components of this segment serves for parsing TCP dump files with parallel expert knowledge processing. These two components take all inputs and export processed data ready for storing into persistent database storage. The next component, DB importer, enables persisting of processed data. Next, an active component, Connection extractor, performs the identification of all TCP connections in database. It produces a list of TCP connection objects with embedded ground truth information. Then, ASNM feature extraction is performed for each TCP connection by the Metrics extractor component and the results of this step are values of ASNM features for each TCP connection object in CSV representation. The last segment of the scheme illustrates the mining and assessment process, which produces the output results of the analysis. The RapidMiner [157] tool was employed for our data mining experiments.

7.6 Data Mining Experiments

In this section, we describe data mining experiments from three classification perspectives – binominal, trinominal and multi-class. These perspectives reflect the interpretation of processed data and their ground truth. A communication record in collected and processed data can either be interpreted as malicious/legitimate communication – in binary representation – or it can be interpreted as malicious/legitimate communication which distinguishes between direct and obfuscated attacks – in trinominal representation – or it can be interpreted as malicious/legitimate communication at specific network service – in the multi-class representation. Our multi-class classification includes three classes for each service (resulting into 12 classes) plus one class for other captured traffic. The names of classes in the binominal and trinominal classification experiments are intuitive and indicate legitimate or malicious behavior of communication records. The identification of classes in a multi-class case consists of two parts. The first part represents the subclass of the connection specifying its maliciousness and may contain three labels with an intuitive representation: legitimate traffic, direct attacks and obfuscated attacks. The second part of the identification indicates the specific acronym of a service. Every presented experiment employed the Naive Bayes classifier. Almost all results of experiments were based on the k-fold cross validations of the classifier. The only exceptions were the first binominal and the first multi-class classification experiment, which predicted target labels of previously unknown obfuscated attacks. There was utilized Forward Feature Selection method (FFS) to find the best features with emphasis on selection of context-free features, as context-based features were designed to be applicable in real network traffic conditions, not in synthetic traffic from laboratory experiments.

7.6.1 Forward Feature Selection

For the purpose of finding the best subset of ASNM features, we performed the FFS method. FFS started to run with an empty set of features and in each iteration added a new feature contributing by the best improvement of average recall of all classes. The average recall of all classes was computed using the underlying 5-fold cross validation method employing the Naive Bayes classifier. The experiment considered two class prediction – the first for legitimate traffic and the second for intrusive traffic. Therefore, the classifier did not distinguish between obfuscated and direct attacks, and they were represented by the same class.

Some features existed which were inconvenient for comparison of synthetic attacks with legitimate traffic captured in real network, therefore, such features were removed from the dataset in the preprocessing phase of our experiments. The examples include: TTL based features, IP addresses, ports, MAC addresses, occurrence of source/destination host in monitored local network, context-based features etc.

The utilized FFS method allowed the acceptance of one iteration without improvement, as we wanted to alleviate the possibility of the selection process becoming stuck in local extremes. The experiment consisted of two executions of the FFS. The first took as input just legitimate traffic and direct attack entries, and represented the case where ADS was trained without knowledge about obfuscated attacks. The second execution took as input the whole dataset of network traffic – consisting of legitimate traffic, direct attacks as well as obfuscated ones, and therefore, represented the case where ADS was aware of obfuscated attacks. The selected features of both executions are depicted in Table 7.4. The penultimate column of the table (i.e., FFS DOL) denotes the selected features where the whole dataset

Feature ID	Description	FFS DOL	FFS DL
SigPktLenIn	• Std. deviation of inbound (server to client) packet sizes.	X	
ConTcpFinCntIn	• The number of TCP FIN flags occurred in inbound traffic.	X	X
ConTcpSynCntIn	• The number of TCP SYN flags occurred in inbound traffic.	X	X
InPktLen32s10i[0]	• Lengths of inbound packets occurred in the first 32 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 1st interval.	X	
InPktLen1s10i[2]	• The same as the previous one, but computed above the first second of a connection. The feature represents totaled inbound packet lengths of the 3rd interval.	X	
InPktLen8s10i[7]	• The same as the previous one, but computed above the first 8 seconds of a connection. The feature represents totaled inbound packet lengths of the 8th interval.	X	
OutPktLen1s10i[0]	• Lengths of outbound (client to server) packets occurred in the first second of a connection which are distributed into 10 intervals. The feature represents totaled outbound packet lengths of the 1st interval.	X	
FourGonAngleN[9]*	• Fast Fourier Transformation (FFT) of all packet sizes. The feature represents the angle of the 10th coefficient of the FFT in goniometric representation.	X	X
InPktLen8s10i[1]	• Lengths of inbound packets occurred in the first 8 seconds of a connection which are distributed into 10 intervals. The feature represents totaled inbound packet lengths of the 2nd interval.		X
PolyInd8ordOut[5]	• Approximation of outbound packet lengths in index domain by polynomial of 8th order. The feature represents 6th coefficient of the polynomial.		X
PolyInd8ordIn[5]	• Approximation of inbound packet lengths in index domain by polynomial of 8th order. The feature represents 6th coefficient of the polynomial.		X

*Sizes of inbound and outbound packets are represented by negative and positive values, respectively.

Table 7.4: ASNM features selected by FFS method

was utilized for the FFS, and the last one (i.e., FFS DL) denotes the case where only direct attacks and legitimate traffic were taken into account. We have noticed that the final model of the FFS DL case has achieved average recall of classes equal to $99.71\% \pm 0.57\%$ as well as $99.67\% \pm 0.66\%$ accuracy in comparison to the FFS DOL case in which average recall equal to $99.49\% \pm 0.64\%$ and accuracy of $99.49\% \pm 0.62\%$ has been achieved, indicating that it has been a little difficult to train the classifier with inclusion of obfuscated attacks unlike the case not considering them.

Several mutual features were selected in both cases which means they provided a value regardless of whether obfuscation was performed or not. Almost all of the following experiments will utilize the feature set gained from the second execution (i.e., FFS DOL), as we consider them as more appropriate for general behavior representation of both kinds of attacks. The only exceptions are the first binominal, two of the trinominal and the first of the multi-class experiments, which will be described and marked in the following subsections.

7.6.2 Binominal Classification

First, we executed the experiment which performed detection of malicious obfuscated traffic by the classifier trained on direct attacks and legitimate traffic only. It represented the situation when ADS had no previous knowledge about obfuscated attacks, and therefore we utilized FFS DL feature set. As the result, only 35.63% (31 of 87) of obfuscated attacks were correctly detected by the current model of the classifier, and thus average recall and classification accuracy of the classifier were equal to 67.53% and 78.41%, respectively. An associated confusion matrix is depicted in Table 7.5. We realized that 64.36% of obfuscated

Classification Accuracy: 78.41%		**True Class**		**Precision**
		Legit. Flows	**Obfus. Attacks**	
Predicted Class	**Legit. Flows**	176	56	75.86%
	Obfus. Attacks	1	31	96.88%
	Recall	99.44%	35.63%	$F_1 = 52.10\%$

Table 7.5: Detection of unknown obfuscated attacks by binominal classifier

attacks were incorrectly predicted as legitimate traffic, and thus caused evasion of the ADS classifier.

Our second binominal classification experiment considered explicit information about obfuscated attacks in training phase of the classifier. Therefore, we validated the classifier by direct and obfuscated attacks labeled as one class. FFS DOL feature set was utilized for the purpose of this experiment. The resulting confusion matrix with performance measures is shown in Table 7.6. The outcome of this experiment indicates a high class recall, precision, *F_1-measure* and classification accuracy of the model trained with knowledge about obfuscated attacks.

Classification Accuracy: 99.49% ± 0.62%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	176	1	99.44%
	All Attacks	1	216	99.54%
	Recall	99.44%	99.54%	$F_1 = 99.54\%$

Table 7.6: Reference confusion matrix

The third binominal classification experiment differs from the previous one in the assumption: obfuscated attacks or direct attacks can be undetected by ADS, and thus can be

(a) Obfuscated attacks labeled as legitimate traffic

Classification Accuracy: 96.45% ± 2.82%		**True Class**		**Precision**
		Legit. Flows	**Direct Attacks**	
Predicted Class	**Legit. Flows**	256	6	97.71%
	Direct Attacks	8	124	93.94%
	Recall	96.97%	95.38%	$F_1 = 94.65\%$

(b) Direct attacks labeled as legitimate traffic

Classification Accuracy: 93.65% ± 1.40%		**True Class**		**Precision**
		Legit. Flows	**Obfus. Attacks**	
Predicted Class	**Legit. Flows**	298	16	94.90%
	Obfus. Attacks	9	71	88.75%
	Recall	97.07%	81.61%	$F_1 = 85.03\%$

Table 7.7: Neutralization of obfuscated/direct attacks

experimentally considered as legitimate traffic. We performed this assumption by neutralization of the target malicious class – labeling it as legitimate. Accordingly, we performed two validations: the first contained legitimate labels for obfuscated attack instances and the second one contained legitimate labels for direct attack instances. The achieved results are depicted in Table 7.7.

We can observe, that in the case of obfuscated attack class neutralization (Table 7.7a), it was less difficult to train the classifier with relabeled classes than in the case of direct attack class (Table 7.7b), which reveals that obfuscated attacks class may have more similar characteristics with legitimate traffic than direct attacks may have. The performance results of the third experiment are lower in comparison with reference confusion matrix (Table 7.6), which indirectly indicates that obfuscated and direct attack classes have different statistical and behavioral characteristics than legitimate traffic has. This result represents simple form of randomization testing of the null hypothesis saying: binominal classifier is able to distinguish between malicious and legitimate traffic. The outcome of the experiment accepts the null hypothesis, as classifier trained on rearranged labels achieved worse performance than original one.

7.6.3 Trinominal Classification

In trinominal classification experiments, connection records were divided into 3 classes: direct attacks, obfuscated attacks and legitimate traffic. Firstly, we tested the classifier, previously trained on direct attacks and legitimate traffic utilizing FFS DL feature set, on the whole dataset including obfuscated attacks. The resulting confusion matrix representing the result of prediction is depicted in Table 7.8. We expected the results to be very similar as in the first binominal experiment (Table 7.5). We can see that only 33.33% of obfuscated attacks (29 of 87 instances) were predicted as direct attacks, and thus our expectation was fulfilled. Little difference, in comparison with binominal experiment, might be caused by nuance in randomness of validation process. All obfuscated attacks predicted as legitimate traffic (58 instances) caused evasion of the ADS classifier. Thus, this experiment shows

Classification Accuracy: 77.66%		True Class			$PPV(C_i)$	$F_1(C_i)$
		Legit. Flows	**Direct Attacks**	**Obfus. Attacks**		
Predicted Class	**Legit. Flows**	**176**	0	58	75.21%	85.64%
	Direct Attacks	1	**130**	29	81.25%	89.65%
	Obfus. Attacks	0	0	**0**	0.00%	0.00%
	Recall	99.44%	100.00%	0.00%		

Table 7.8: Prediction of unknown obfuscated attacks by trinominal classifier

once again that ADS classifier is not able to correctly distinguish/predict obfuscated attacks without previous knowledge about them.

Later, we utilized FFS DOL feature set obtained by binominal FFS as well as we executed FFS with 3 class labels. The confusion matrices with achieved performance measures are shown in Table 7.9. As we expected, better performance results were achieved by new

(a) FFS DOL feature set

Classification Accuracy: 95.18% ± 1.85%		True Class			$PPV(C_i)$	$F_1(C_i)$
		Legit. Flows	**Direct Attacks**	**Obfus. Attacks**		
Predicted Class	**Legit. Flows**	**174**	3	2	97.21%	97.75%
	Direct Attacks	1	**122**	6	94.57%	94.20%
	Obfus. Attacks	2	5	**79**	91.86%	91.32%
	Recall	98.31%	93.85%	90.80%		

(b) Feature set obtained from FFS considering 3 labels

Classification Accuracy: 97.97% ± 0.63%		True Class			$PPV(C_i)$	$F_1(C_i)$
		Legit. Flows	**Direct Attacks**	**Obfus. Attacks**		
Predicted Class	**Legit. Flows**	**173**	0	1	99.43%	98.57%
	Direct Attacks	2	**127**	0	98.45%	98.06%
	Obfus. Attacks	2	3	**86**	94.51%	96.63%
	Recall	97.74%	97.69%	98.85%		

Table 7.9: Trinominal classification

run of FFS method which considered 3 class labels, and thus it selected different features more appropriate for trinominal classification. But on the other hand, very plausible results were reached by FFS DOL feature set as well, which even approved them as appropriate for distinction between attack classes.

7.6.4 Multi-class Classification

Our multi-class experiment was performed with explicit information about obfuscated attacks and utilized the whole dataset. We performed validation of classifier utilizing FFS DOL feature set as well as we executed FFS method considering multi-class labels. Results

			True Class Distribution														
			Direct Attacks				Obfus. Attacks				Legit. Traffic						
Classification Accuracy: 88.33% +/ 0.91%			Apache	BadBlue	DCOM RPC	Samba	Apache	BadBlue	DCOM RPC	Samba	Apache	BadBlue	DCOM RPC	Samba	Other Traffic	Precision	$F\ measure_i$
Predicted Class Distribution	Direct Attacks	Apache	101	0	0	0	5	0	0	0	0	0	0	1	1	93.52%	96.19%
		BadBlue	0	4	0	0	0	2	0	0	0	0	0	0	0	66.67%	80.00%
		DCOM RPC	0	0	3	0	0	0	0	0	0	0	0	0	0	100.00%	85.71%
		Samba	0	0	0	19	0	0	0	0	0	0	0	0	0	100.00%	97.44%
	Obfus. Attacks	Apache	1	0	1	0	49	2	1	0	2	2	0	1	1	81.67%	80.99%
		BadBlue	0	0	0	0	0	4	0	0	0	0	0	0	0	100.00%	57.14%
		DCOM RPC	0	0	0	0	2	1	5	5	0	0	0	0	0	38.46%	47.62%
		Samba	0	0	0	0	1	0	2	3	0	0	0	0	0	50.00%	42.86%
	Legit. Traffic	Apache	0	0	0	1	1	0	0	0	34	3	0	3	0	80.95%	85.00%
		BadBlue	0	0	0	0	2	1	0	0	2	89	0	0	0	94.68%	94.18%
		DCOM RPC	0	0	0	0	0	0	0	0	0	0	4	0	0	100.00%	100.00%
		Samba	0	0	0	0	0	0	0	0	0	0	0	10	0	100.00%	80.00%
		Other Traffic	0	0	0	0	1	0	0	0	0	1	0	0	23	92.00%	92.00%
Recall			99.02%	100.00%	75.00%	95.00%	80.33%	40.00%	62.50%	37.50%	89.47%	93.68%	100.00%	66.67%	92.00%		

Table 7.10: Multi-class classification – FFS DOL features

of the 3-fold cross validation of the classifier on FFS DOL feature set are depicted in Table 7.10 and the results obtained from new run of FFS method are shown in Table 7.11.

			True Class Distribution														
			Direct Attacks				Obfus. Attacks				Legit. Traffic						
Classification Accuracy: 95.68% +/ 0.73%			Apache	BadBlue	DCOM RPC	Samba	Apache	BadBlue	DCOM RPC	Samba	Apache	BadBlue	DCOM RPC	Samba	Other Traffic	Precision	$F\ measure_i$
Predicted Class Distribution	Direct Attacks	Apache	102	0	0	0	1	0	0	0	0	0	0	0	1	98.08%	99.03%
		BadBlue	0	4	0	0	0	0	0	0	0	0	0	0	0	100.00%	100.00%
		DCOM RPC	0	0	4	0	0	0	0	0	0	0	0	0	0	100.00%	100.00%
		Samba	0	0	0	20	0	0	0	0	0	0	0	0	0	100.00%	100.00%
	Obfus. Attacks	Apache	0	0	0	0	55	1	0	0	2	0	0	0	1	93.22%	91.66%
		BadBlue	0	0	0	0	0	7	0	0	0	0	0	0	0	100.00%	82.35%
		DCOM RPC	0	0	0	0	0	1	7	0	0	0	0	0	0	87.50%	87.50%
		Samba	0	0	0	0	3	0	1	8	0	0	0	0	0	66.67%	80.00%
	Legit. Traffic	Apache	0	0	0	0	0	0	0	0	35	2	0	0	0	94.59%	93.33%
		BadBlue	0	0	0	0	1	1	0	0	1	93	0	0	0	96.88%	97.38%
		DCOM RPC	0	0	0	0	0	0	0	0	0	0	4	0	0	100.00%	100.00%
		Samba	0	0	0	0	0	0	0	0	0	0	0	15	0	100.00%	100.00%
		Other Traffic	0	0	0	0	1	0	0	0	0	0	0	0	23	95.83%	93.88%
Recall			100.00%	100.00%	100.00%	100.00%	90.16%	70.00%	87.50%	100.00%	92.11%	97.89%	100.00%	100.00%	92.00%		

Table 7.11: Multi-class classification – FFS considering 13 labels

Mean recall of classes obtained from cross validation run was equal to 80.46% ±3.26% in the

case of FFS DOL features and 94.87% ±2.06% in the case of new FFS run. Alike trinominal experiment, we obtained better performance in the case of new FFS run. However, FFS DOL features achieved very plausible results again.

7.6.5 Comparison of Various Classifiers

For the purpose of performance comparison of various classifiers, we executed 5-fold cross validation on the next two classification models – Decision Tree and SVM. FFS DOL feature set was utilized in this experiment as the input for the classifiers working with two class prediction. At first, we evaluated performance of the SVM classifier which utilized radial basis function as non-linear kernel function. The adjacent confusion matrix is depicted in Table 7.12.

Classification Accuracy: 80.96% ± 3.51%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	176	74	70.40%
	All Attacks	1	143	99.31%
	Recall	99.44%	65.90%	$F_1 = 79.22\%$

Table 7.12: Performance of the SVM classifier

The next experiment was performed with Decision Tree classifier, which utilized gini index as selection criterion for splitting of attributes. The adjacent result is represented by confusion matrix in Table 7.13. The results of both performance evaluation experiments

Classification Accuracy: 95.93% ± 2.47%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	169	8	95.48%
	All Attacks	8	209	96.31%
	Recall	95.48%	96.31%	$F_1 = 96.31\%$

Table 7.13: Performance of the Decision Tree classifier

can be compared to the result of Naive Bayes classifier represented by reference confusion matrix in Table 7.6. Considering average recall of all class and F_1-measure as the most credible performance measures, we can say that the Naive Bayes classifier achieved the best results, following by Decision Tree, and finally by SVM.

All the classification models were compared by ROC method. Considering neutral bias, the most interesting configuration of classifier was achieved by Naive Bayes classifier. ROC diagram with neutral bias is depicted in Figure 7.3. Note that comparing of ROC ran above cross validation method, and thus generated certain variability, which is shown by line-adjacent transparent areas.

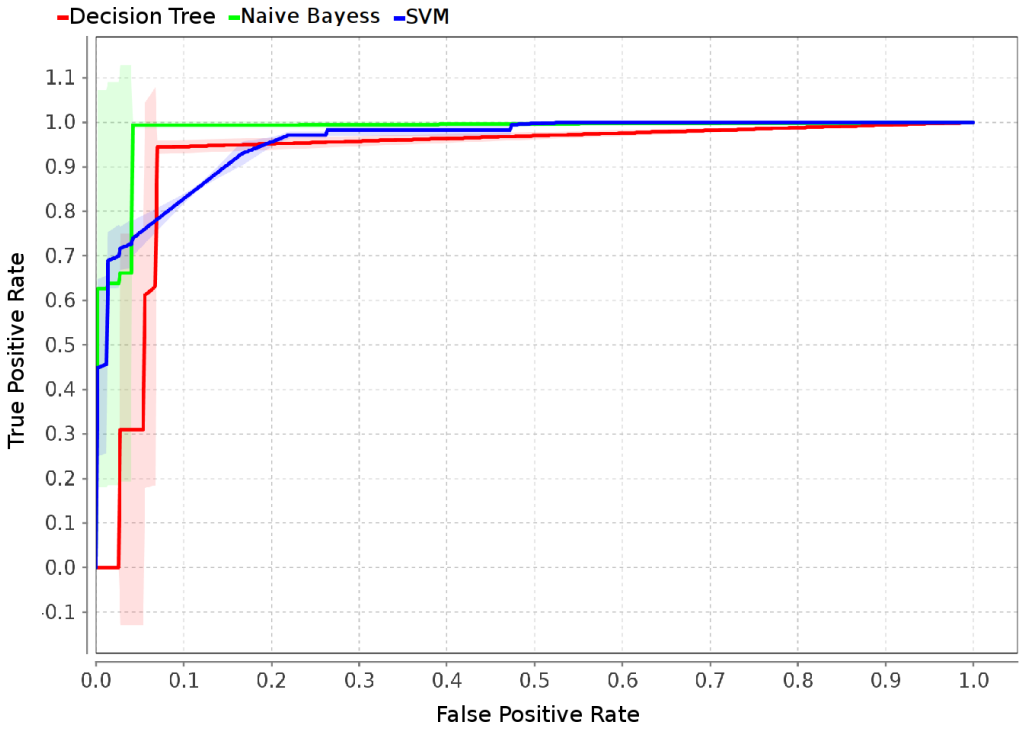

Figure 7.3: ROC diagram comparing classifiers

7.7 Analysis of Network Traffic Characteristics

ASNM [76] features of AIPS and discriminators features of A. Moore [119] are utilized for the analysis of network traffic characteristics. Representative example depicting the traffic flow differences between tunneled and direct malicious traffic is Samba service whose network traffic flow is illustrated in Figure 7.4. The positive sign on the y axis denotes direct malicious traffic and negative sign represents obfuscated (tunneled) oneIt is obvious that tunneling obfuscation may change many network traffic characteristics of direct attack, e.g. the number of incoming and outgoing packets, the size of them, various packet header fields.

We examined the value density distribution of each feature with emphasis on distinctiveness of obfuscated and direct malicious traffic. For the purpose of value density examination among TCP connections of our dataset, we utilized kernel density estimation with Gaussian kernels. However, we performed value density examination by frequency analysis in some features because of their clearer interpretation. We selected some features presenting the interesting characteristics of each network traffic class. At first, we present **discriminating features** enabling good malicious and legitimate traffic separation. Further, we present **obfuscated features** representing specific characteristics of obfuscated attacks which are similar to legitimate traffic, making the detection of such obfuscated attacks more complicated from the view of ADS which does not have any previous knowledge about obfuscated attacks.

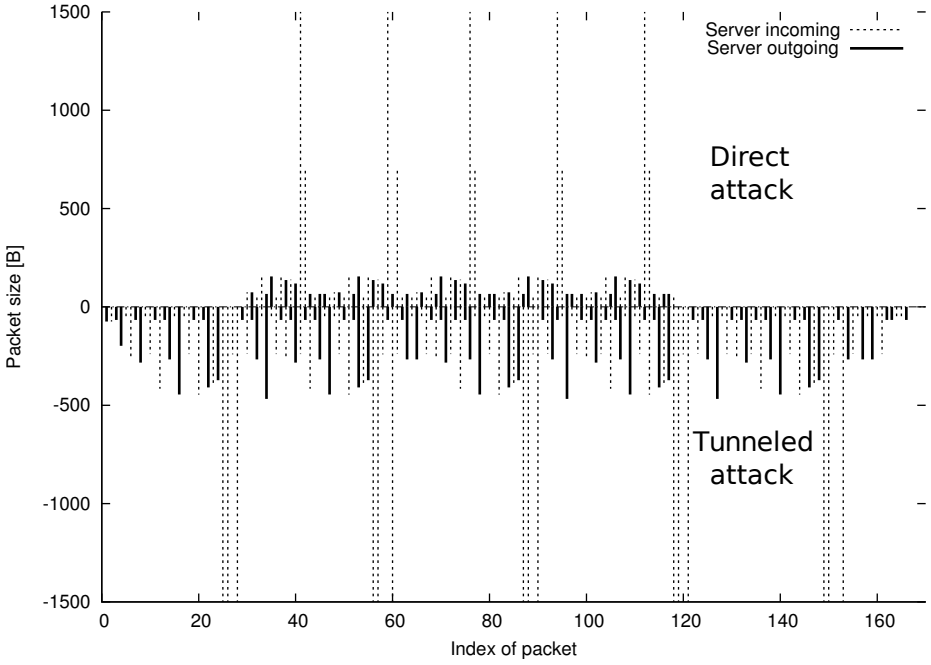

Figure 7.4: Traffic flows of direct and tunneled attack on Samba service

7.7.1 Discriminating Features

The first discriminating feature, which we present, is standard deviation of packet lengths from server to client in the TCP communication ($SigPktLenIn$). The Gaussian kernel density estimation of this feature is shown in Figure 7.5. The obvious observation of the figure is that both of the attack classes have different values than the most of legitimate traffic representatives. There can be seen two value ranges of the legitimate traffic: $\langle 50, 150\rangle$ and $\langle 600, 750\rangle$ – in Figure, while in the case of attacks there is one significant value range $\langle 350, 450\rangle$ and two less significant: $\langle 0, 80\rangle$ and $\langle 200, 300\rangle$. Notice, that feature does not have clear discriminating property in the range $\langle 0, 80\rangle$, however, more important are previously mentioned ranges, as they can be used for distinction between both network traffic classes in the majority of their instances. Also, regarding FFS experiment, we can see that this feature was selected into FFS DOL set (see Table 7.4).

The maximum segment size observed during a lifetime of a connection in outbound direction (`81/max_segm_size_a_b`) is the next example of the current category of features. The histogram of the feature is shown in Figure 7.6. Values of the feature are very similar for obfuscated and direct attack classes in the most of instances, while values of legitimate traffic can be recognized in two intervals: $\langle 150, 180\rangle$ and $\langle 350, 460\rangle$. Thus, there are value density differences between attack classes and legitimate traffic class.

The next interesting feature, representing the current group, is the average segment size observed during the lifetime of the connection in inbound direction (`86/avg_segm_size_b_a`). The Gaussian kernel density estimate of the feature is depicted in Figure 7.7. The majority of malicious and legitimate traffic can be distinguished by the feature.

Another interesting feature from actual group represents the Fast Fourier Transformation (FFT) of lengths of packets transmitted from server to client (`FourGonModulIn[1]`). The Gaussian kernel density estimation of this feature is illustrated in Figure 7.8. Notice that this feature represents the modul of the second frequency of FFT and has goniometric

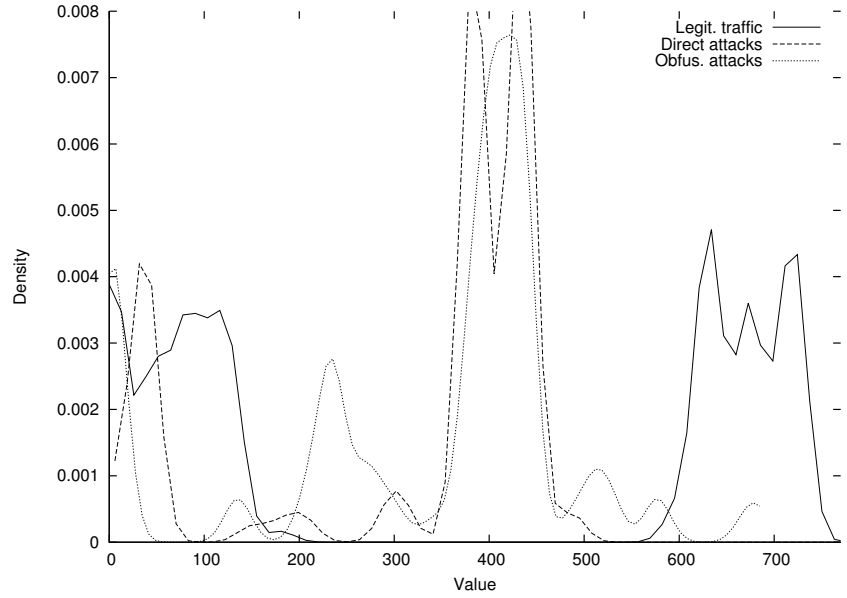

Figure 7.5: Standard deviation of inbound packet lengths (`SigPktLenIn`)

representation. It shows the highest discrimination rate for malicious traffic in the range of values $\langle 900, 1500 \rangle$. The most difficult is to distinguish between legitimate and malicious traffic in the interval of values $\langle 0, 600 \rangle$, but for this purpose, the feature can be combined with another one which may help for recognition.

One of the last examples belonging into the current category of discriminating features is distribution of lengths of inbound packets occurred in the first 32 seconds of a connection (`InPktLen32s10i[0]`). Gaussian kernel density estimation of the feature can be found in

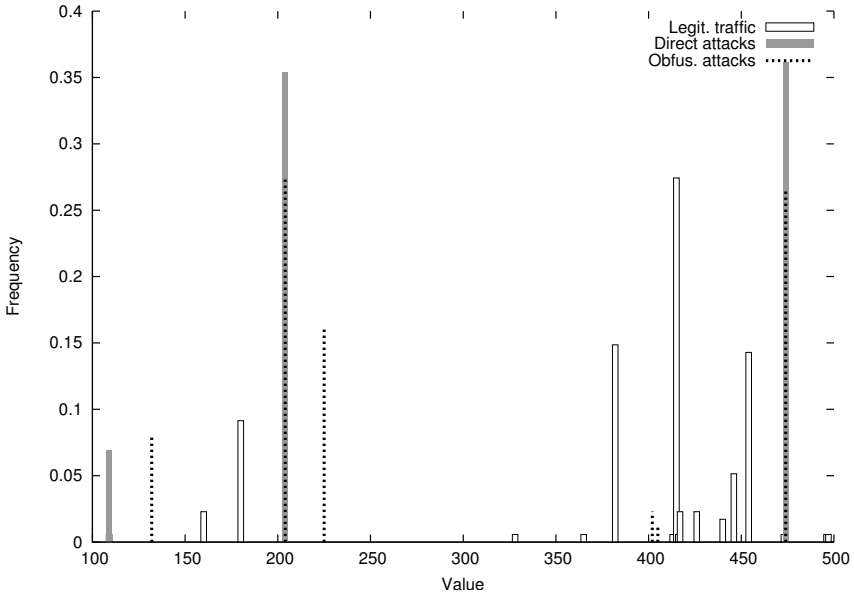

Figure 7.6: Maximum TCP segment size from client to server (`81/max_segm_size_a_b`)

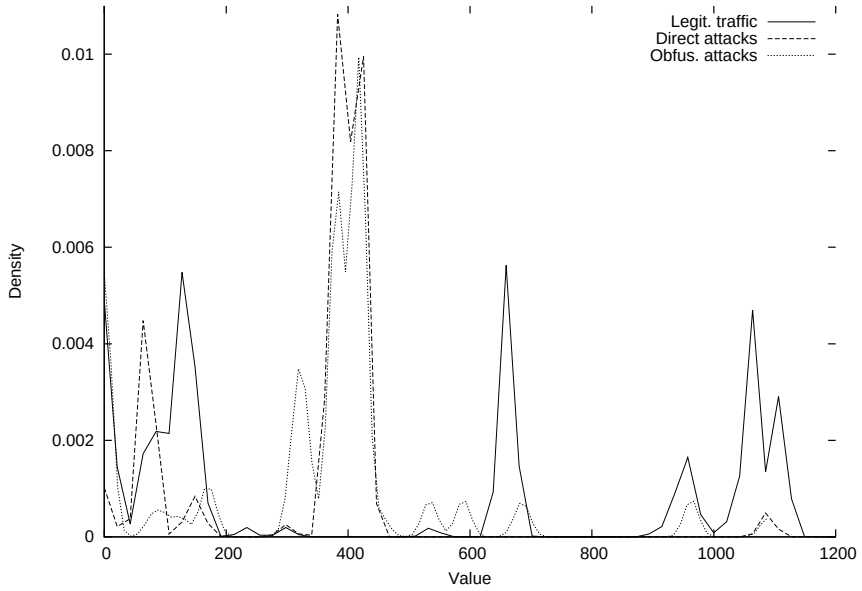

Figure 7.7: The average segment size observed during the lifetime of the connection in direction from server to client (`86/avg_segm_size_b_a`)

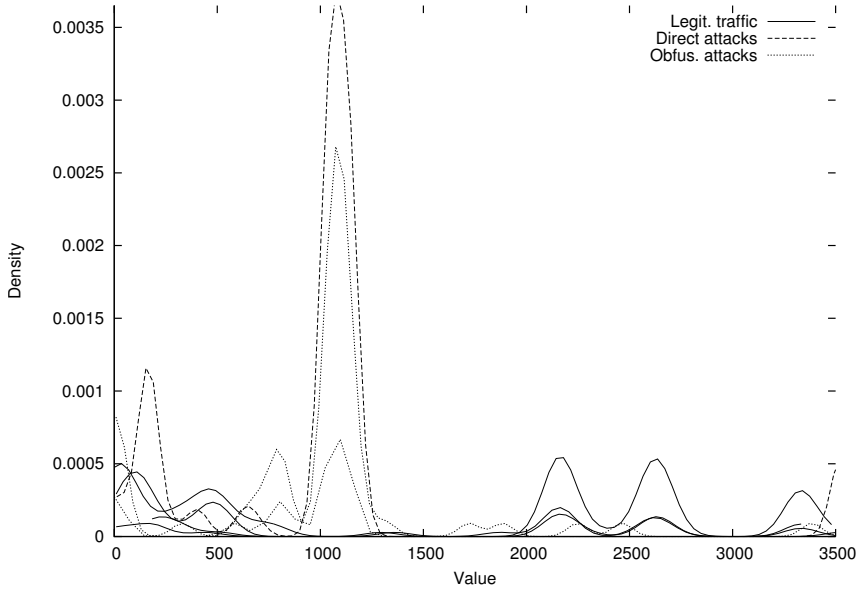

Figure 7.8: FFT of packet lengths from server to client (`FourGonModulIn[1]`)

Appendix G.1. Finally, the last two identified representatives of this category are FFT of packet lengths in both directions of a connection (`FourGonAngleNeg[9]`) – the feature represents the 10th frequency of FFT; and normalized sum of products of all packet lengths of a connection with 8 Gaussian curves (`GaussProds8AllNeg[7]`) – the feature represents the 8th normalized sum of products with Gaussian curves. Both features consider packet lengths from client to server as positive values and opposite lengths as negative ones. The Gaussian kernel density estimations of these two features can be found in Appendix G.1.

7.7.2 Obfuscated Features

The second group of features aims on distinctiveness of network attacks performed by tunneling obfuscation technique, which in the optimal case, leads to similar characteristics as legitimate traffic has. The first example of this group is the FFT of Inter Arrival Time (IAT) of all traffic in a connection (`222/FFT_all_#4`). The histogram of this feature is shown in Figure 7.9. This feature represents just the magnitude of the fourth frequency of approximation by FFT. The most important result shows similar values of tunneled attack instances and legitimate traffic instances, which can be considered as successful obfuscation of malicious traffic imitating the behavior of legitimate traffic in several instances. Another observation shows differences between the majority of tunneled attacks instances and direct attack instances, resulting into fair discrimination between these two attack classes.

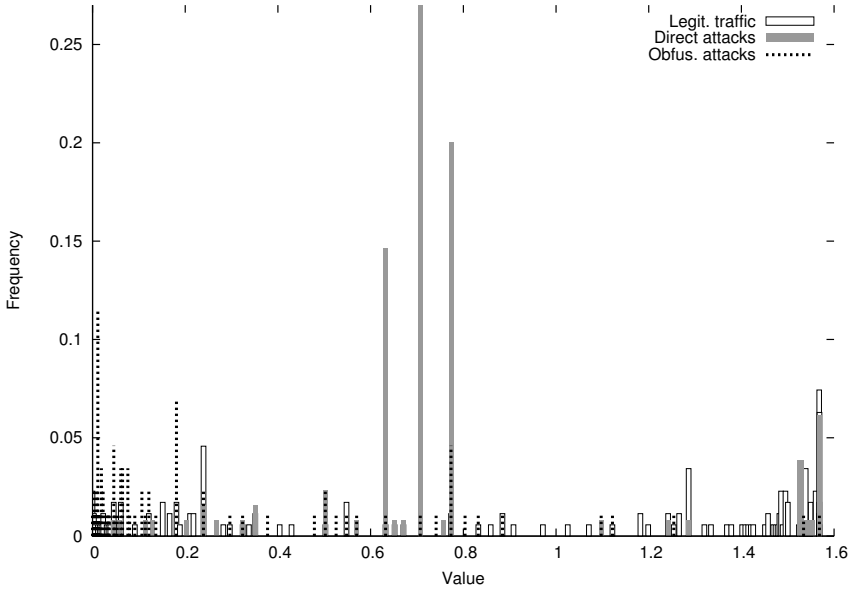

Figure 7.9: FFT of IAT for all traffic in a connection (`222/FFT_all_#4`)

The Gaussian kernel density estimation of the feature representing FFT of packet lengths is illustrated in Figure 7.10. The feature denotes the angle of goniometric representation of the 8th frequency of FFT (`FourGonAngleOut[7]`). The important observation of this figure is specific value distribution of direct attacks instances as well as value distribution of tunneled attacks similar to legitimate traffic's distribution. This disables to detect obfuscated attacks among legitimate traffic and direct attacks by this feature without having any additional information about them. The same statements can be claimed about feature representing the number of transferred packets from client to server during a connection (`PktPerSesOut`), which is illustrated in Figure 7.11. The next example belonging into the current category of obfuscated features is approximation of communication from client to server by polynomial of 8 order in the index domain of packets (`PolyInd8ordOut[3]`). The feature represents the 4th coefficient of the approximation. Gaussian kernel density estimation of this feature can be found in Appendix G.2. The last two identified representatives of this category are lengths of outbound packets occurred in the first 4 and 3 seconds of a connection, respectively. Both of them can be found in Appendix G.2.

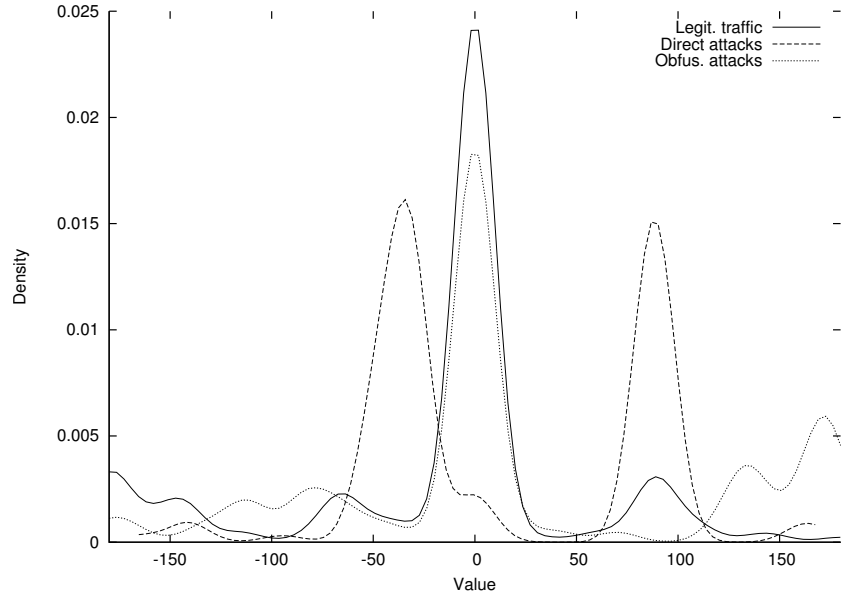

Figure 7.10: FFT of packet lengths from client to server (`FourGonAngleOut[7]`)

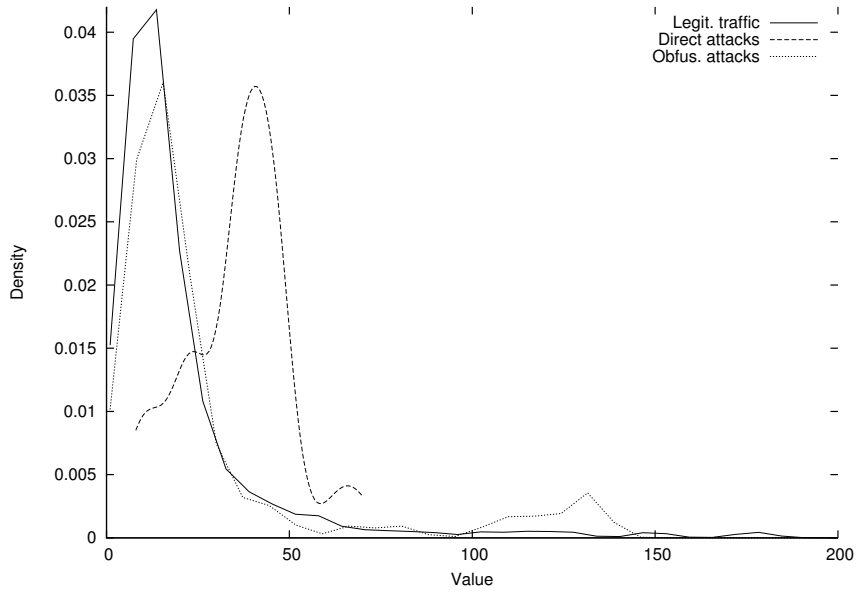

Figure 7.11: The number of transferred packets from client to server (`PktPerSesOut`)

7.8 Concluding Remarks

In this chapter, we presented obfuscation of malicious network traffic by tunneling in HTTP and HTTPS protocols. The main reason of the proposed malicious traffic obfuscation was to imitate the characteristics of the legitimate traffic by the malicious traffic and to show low detection capabilities of NIDSs and ADSs. There were utilized advanced network features in contrast with tunneling obfuscation stated in state-of-the-art paper [56]. We analyzed only network services vulnerable to buffer overflow attacks, which have many characteristics

common. Therefore, the generalization of our results is very likely for this category of malicious network traffic.

We performed exploitation of chosen services with a direct approach and by employing our proposed obfuscation. Also, we performed legitimate traffic simulation as well as a capture of real network traffic in order to balance malicious and legitimate traffic class representatives. All simulations of malicious and legitimate traffic were performed in four network modifications, which simulated traffic shaping, traffic policing and transmission on an unreliable network channel, and thus helped to emulate real network traffic conditions. The achieved results showed low detection rate of obfuscated attacks by our ADS trained on direct attacks and legitimate traffic only; in contrast with high classification performance in the case it was trained with knowledge about obfuscated attacks. Therefore, the assumptions from Subsection 7.1.1 were confirmed.

Further, we examined interesting characteristics of tunneled and direct network attacks. Observed characteristics were divided into two categories. The first one, called discriminating features, has properties which are useful in detection phase of malicious network traffic. These features can help us in distinguishing between legitimate traffic class and malicious traffic class. The later category called obfuscated features, represents features which were significantly influenced by our tunneling obfuscation technique, and therefore, cannot be directly used to distinguish between malicious and legitimate traffic. Examples of this feature group serve as prove of obfuscations success.

Moreover, several experiments with signature-based NIDSs – SNORT and SURICATA – were performed and achieved results show high detection rate of direct attacks by both NIDS, while detection rate of obfuscated attack was poor (considering standard rule set of SNORT).

Chapter 8

Non-payload-based Network Obfuscation Techniques

Although non-payload-based evasions of network attacks in the area of intrusion detection were considered as an actual research subject more than one and half decade ago [66, 139, 140], it turned out to be important to study a few years ago as well [19]. There are several related works considering non-payload-based evasions of network attacks for payload-based intrusion detection, however, there is a lack of works performing investigations into non-payload-based network behavior anomaly detection and this kind of evasion.

Exploitation of several vulnerable network services is performed in a virtual network environment by employing of non-payload-based obfuscations of malicious network traffic. Captured data is examined by NIDSs SNORT and SURICATA as well as by the ASNM network features and a supervised classifier. The goal of this chapter is to train a classifier to be aware of the behavior of obfuscated attacks, and thus correctly predict other similar obfuscated attacks. In this chapter, ASNM features and a supervised classifier represent ADS detection engine.

The obfuscation techniques leveraged in the chapter are based on non-payload-based modifications of connection-oriented communications accomplished by NetEm [73] utility and ifconfig [80] Linux command. The next purpose of this chapter is to analyze discriminating characteristics of network attacks also containing obfuscated representatives in contrast with legitimate traffic ones. The obfuscation tool and techniques which are presented in this chapter were designed in Master's thesis [185] of Martin Teknös which was created under my supervision.

8.1 Network Traffic Normalizers

Generally, to face the non-payload-based obfuscations, and thus possible evasions, IDSs utilize network normalizers which aim to eliminate impact of such evasions by transformation of network traffic by several rules unifying the network flow. However, issues with performance and platform dependency of the normalizers, as well as differences between various implementations, were described. Because of the shortcomings of network normalizers, their usage in a network can result in side-effects and can even be prone to various attacks, e.g. state holding and CPU overload [53, 132, 168]. Thus, we specify the next goal of this chapter which is based on the previous facts about network normalizers: investigation of network normalizer's exigency in our ADS. Experimentally, this chapter does not consider

a normalizer of network traffic, instead considers traffic without application of proposed obfuscations as normalized one. Similarly, network traffic containing some non-payload-based obfuscations is considered as un-normalized one.

8.2 Method Description

This section formally describes data exchanges over network channel followed by explanation of non-payload-based obfuscations, which aim to modify the behavioral characteristics of intrusive communication in a way of having different characteristics from original malicious network traffic.

Consider a session of a protocol at the application layer of the TCP/IP stack which serves for data transfer between the parts of client/server based application. The data is transferred in both directions one by one in succession:

$$\begin{aligned} &C.delivers(d_C[1], S),\\ &S.delivers(d_S[1], C),\\ &\vdots\\ &C.delivers(d_C[q], S),\\ &S.delivers(d_S[q], C), \end{aligned} \tag{8.1}$$

where C, S, q, $delivers$, $d_C[i]$ and $d_S[i]$ denote the client, the server, the number of data exchanges of the session, the method of delivering data from client to server (and vice versa), $i-th$ sentence of application protocol's data passed from client to server where $i \in \{1, \ldots, q\}$ and vice versa, respectively.

The interpretation of the previous application data exchanges between client and server can be formulated, considering the TCP/IP stack up to the transport layer, by connection k which is constrained to connection oriented protocol TCP at L4, internet protocol IPv4[1] at L3 and Ethernet protocol at L2 layers of the TCP/IP stack. The TCP connection k is represented by the tuple $k = (t_s, t_e, p_c, p_s, ip_c, ip_s, P_c, P_s)$. Consider interpretation of tuple symbols from Table 5.2. Sets P_c and P_s contain a number of packets, where each of them can be interpreted by the packet tuple $p = (t, size, eth_{src}, eth_{dst}, ip_{off}, ip_{ttl}, ip_p, ip_{sum}, ip_{src}, ip_{dst}, ip_{dscp}\ tcp_{sport}, tcp_{dport}, tcp_{sum}, tcp_{seq}, tcp_{ack}, tcp_{off}, tcp_{flags}, tcp_{win}, tcp_{urp}, data)$. Symbols stated in the packet tuple were described in Table 5.1.

8.2.1 Definition of Non-payload-based Obfuscations

Consider feature extraction process of ASNM described in Section 5.2, and for simplicity, do not consider context of an analyzed connection as well as optional arguments of feature extraction functions. Assume connection k_a representing an intrusive communication executed without any obfuscation. Then, k_a can be represented by ADS features

$$f(k_a) \mapsto F^a = (F_1^a, \quad F_2^a, \ldots, F_n^a) \tag{8.2}$$

which are delivered to the previously trained classifier C. Assume that C can correctly predict the target label as an intrusive one, because its knowledge base is derived from

[1]Our description and experiments are related to only IPv4 traffic, as we assume that substantial behavioral characteristics of IPv4 and IPv6 traffic are very similar.

training dataset D_{tr} containing intrusive connections having similar (or the same) behavioral characteristics.

Now, consider connection k'_a which represents intrusive communication k_a executed by employment of non-payload-based obfuscations aimed at modification of its network behavioral properties. The obfuscations can modify P_c and P_s packet sets of the original connection k_a by insertion, removal and transformation of the packets. Symbols of a packet tuple whose values are sensitive to the obfuscations include: $t, size$, ip_{off}, ip_{sum}, tcp_{sum}, tcp_{seq}, tcp_{ack}, tcp_{off}, tcp_{flags}, tcp_{win}, tcp_{urp} and $data$. [2] The modifications of P_c and P_s of the connection k_a can cause alteration of the original ADS features' values F^a to new ones. Thus, ADS features extracted over k'_a are represented by

$$f(k'_a) \mapsto F^{a'} = (F_1^{a'}, \quad F_2^{a'}, \ldots, F_n^{a'}) \tag{8.3}$$

and have different values than features F^a of the connection k_a. Therefore, we assume that the probability of a correct prediction of k'_a-connection's features $F^{a'}$ by the previously assumed classifier C is lower than in the case of connection k_a. Also, we assume that classifier C' trained by learning algorithm A on training dataset D'_{tr}, containing obfuscated intrusion instances, will be able to correctly predict higher number of unknown obfuscated intrusions than classifier C. These assumptions will be evaluated and analyzed later.

8.3 Obfuscation Tool

We designed and implemented a tool for automatic exploitation of network services which is able to perform various obfuscation techniques based on NetEm utility and *ifconfig* Linux command. Execution of direct attacks (non-obfuscated ones) is also supported by the tool as well as capturing network traffic related to the attacks. The tool can restore all modified system settings and consequences of attacks on a compromised machine. The behavioral state diagram of the obfuscation tool is depicted in Figure 8.1. After successful execution of each specified obfuscation of selected attack, the output directory contains several network traffic dumps associated with the obfuscations.

8.3.1 Supported Obfuscation Techniques

Several obfuscation techniques are proposed which are able to influence network flow characteristics at network and transport layers of the TCP/IP stack. Table 8.1 presents instances of these techniques and contains appropriate empirically recognized parameters. It had been experimented with various parameter values until attacks could be successfully performed and appropriate divergent behavior had been achieved.

8.3.2 Implementation Notes

The obfuscation tool is based on third party open source tools and is written in python programming language. For the purpose of an automatic attack execution, a utility from Metasploit framework [113] called *msfconsole*, is used. Tcpdump [183] tool is chosen to perform network traffic capture between the attacker's machine and the legitimate one.

[2] Notice, the *data* field is sensitive to the obfuscations only in the manner of damaging or splitting the original packet's data.

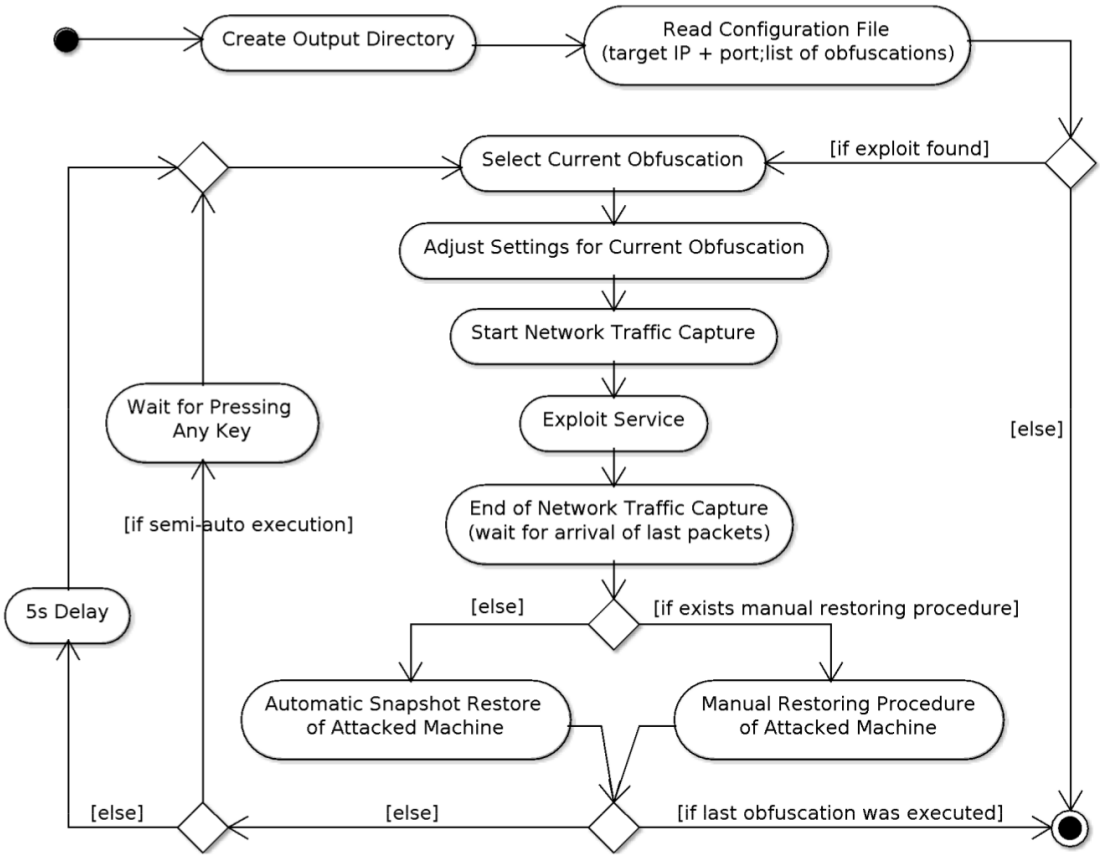

Figure 8.1: Behavioral state diagram of the obfuscation tool

The most suggested obfuscations are performed by *tc* [89] utility and its extension NetEm [73], respectively. NetEm enables us to add latency of packets, loss of packets, duplication of packets, reordering of packets and other outgoing traffic characteristics of the selected network interface. The modification of MTU is performed by the linux utility *ifconfig* [80]. The tool supports an automatic snapshot restoring of a compromised machine by use of the *vboxmanage* [127] utility. Also, the tool supports adjustment of special restoring procedure for each attack.

Technique	Instance	ID
Spread out packets in time	• constant delay: 1s	(a)
	• constant delay: 8s	(b)
	• normal distribution of delay with 5s mean 2.5s standard deviation (25% correlation)	(c)
Packets' loss	• 25% of packets	(d)
Unreliable network channel simulation	• 25% of packets damaged	(e)
	• 35% of packets damaged	(f)
	• 35% of packets damaged with 25% correlation	(g)
Packets' duplication	• 5% of packets	(h)
Packets' order modification	• reordering of 25% packets; reordered packets are sent with 10ms delay and 50% correlation	(i)
	• reordering of 50% packets; reordered packets are sent with 10ms delay and 50% correlation	(j)
Fragmentation	• MTU 1000	(k)
	• MTU 750	(l)
	• MTU 500	(m)
	• MTU 250	(n)
Combinations	• normal distribution delay ($\mu = 10ms$, $\sigma = 20ms$) and 25% correlation; loss: 23% of packets; corrupt: 23% of packets; reorder: 23% of packets	(o)
	• normal distribution delay ($\mu = 7750ms$, $\sigma = 150ms$) and 25% correlation; loss: 0.1% of packets; corrupt: 0.1% of packets; duplication: 0.1% of packets; reorder: 0.1% of packets	(p)
	• normal distribution delay ($\mu = 6800ms$, $\sigma = 150ms$) and 25% correlation; loss: 1% of packets; corrupt: 1% of packets; duplication: 1% of packets; reorder 1% of packets	(q)

Table 8.1: Experimental obfuscation techniques with parameters

8.4 Virtual Network Infrastructure

We utilized a virtual private network environment for network vulnerability exploitation because of legal aspects related to this research. VirtualBox [128] virtualization tool was utilized for this purpose. Virtual network infrastructure used in our research is shown in

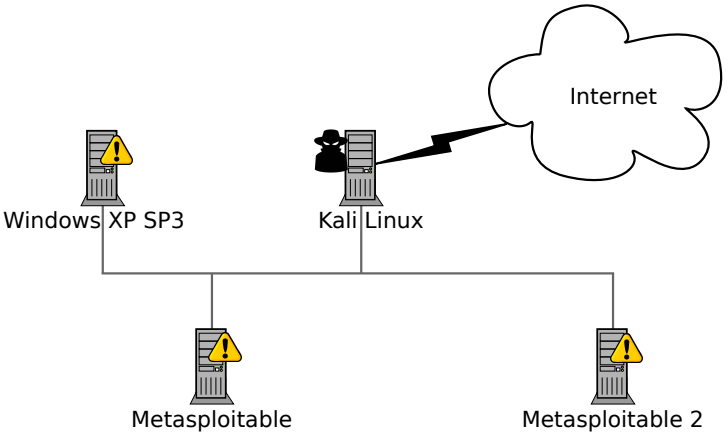

Figure 8.2: Testing network infrastructure

Figure 8.2. All virtual machines (VM) were configured with private static IP addresses in order to ensure easy automatization of the whole exploitation process. Snapshots of all VMs were taken in order to ensure consistent post attack recovery. Our testing network infrastructure consisted of the attacker's machine equipped with Kali Linux 1.1.0 and vulnerable machines which were running Metasploitable 1, 2 and Windows XP with Service Pack 3 (WinXPSP3). WinXPSP3 had Microsoft SQL Server 2005 installed and the firewall disabled. Metasploitable [146] machines represented publicly available vulnerable Linux machines serving for penetration testing purposes and validation of security tools.

8.4.1 Chosen Vulnerabilities

Network intrusion attacks were executed on various available vulnerable network services. The selection of the services was aimed at the high severity of their successful exploitation leading to remote shell code execution through established backdoor communication. The following listing contains a brief description of vulnerable services considered in our experiments. CVE IDs [123] with CVSS [124] values are shown in square brackets:[3]

- **Apache Tomcat 5.5**
  [CVE-1999-0502: 7.5; CVE-2009-3843: 10.0] – firstly, a dictionary attack was executed in order to obtain access credentials into the application manager instance [156]. Further, the server's application manager was exploited for transmission and execution of malicious code [143].

- **Microsoft SQL Server 2005**
  [CVE-1999-0506: 7.2; CVE-2000-1209: 10.0] – a dictionary attack was employed to obtain access credentials of MSSQL user [149] and then the procedure `xp_cmdshell` enabling the execution of an arbitrary code was exploited [148].

[3] In those cases when there is more CVE and CVSS scores present, the order of their exploitation matches with the order stated in the description.

- **Samba 3.0.20-Debian**
  [CVE-2007-2447: 6.0] – vulnerability in Samba service enabled the attacker of arbitrary command execution which exploited MS-RPC functionality when configuration `username map script` [155] was allowed. There was no need of authentication in this attack.

- **Server service of Windows XP**
  [CVE-2008-4250: 10.0] – the service enabled the attacker of arbitrary code execution through crafted RPC request resulting into stack overflow during path canonicalization [147].

- **PostgreSQL 8.3.8**
  [CVE-1999-0502: 7.5; CVE-2007-3280: 9.0] – a dictionary attack was executed in order to obtain access credentials into the PostgreSQL instance [151]. Standard PostgreSQL linux installation had write access to `/tmp` directory and it could call user defined functions (UDF) which utilized shared libraries located on an arbitrary path (e.g. `/tmp`). An attacker exploited this fact and copied its own UDF code to `/tmp` directory and then executed it [150].

- **DistCC 2.18.3**
  [CVE-2004-2687: 9.3] – vulnerability enabled the attacker remote execution of an arbitrary command through compilation jobs which were executed on the server without any permission check [145].

8.4.2 Collected Network Traffic Dataset

Our obfuscation tool was employed for automatic exploitation of the described vulnerable services and the capturing of related intrusive network traffic. Legitimate network traffic was collected from two sources. The first source represented legitimate traffic simulation in our virtual network architecture which also employed non-payload-based obfuscations for the purpose of real network simulation. As the second source, common usage of all previously mentioned services was captured in real network, and all traffic was anonymized and filtered by signature based NIDS Suricata [178] and SORT [160] through Virus Total API [191]. The final dataset is summarized in Table 8.2. Notice, that dataset contains higher number of legitimate traffic representatives than malicious one, which is also common in real network conditions.

8.5 Detection by Signature Based NIDS

In this section, the detection by signature-based NIDSs was performed because we wanted to analyze their detection capabilities to our proposed obfuscation techniques in similar manner like we did in Section 7.4. There were investigated detection results of SNORT and SURICATA NIDSs (through VirusTotal API [191]) equipped by rules of the same version as in the tunneling obfuscation experiments.

First, we let NIDSs to inspect direct attacks exploiting our current network vulnerabilities. The results of the inspection summarize the detection properties of SNORT and SURICATA, and are depicted in Table 8.3. Regarding to mutual comparison of SNORT's and SURICATA's detection capabilities, we can see that SNORT overcame SURICATA and correctly detected 100.00% of direct attacks. Notice that only 33 direct attack instances on

Network Service	Count of TCP Connections			
	Legitimate	Direct Attacks	Obfuscated Attacks	Summary
Apache Tomcat	34	57	164	255
DistCC	100	10	23	133
MSSQL	75	31	103	209
PostgreSQL	42	13	45	100
Samba	4635	18	43	4696
Server	3339	26	102	3467
Other Traffic	144	n/a	n/a	144
Summary	8369	155	480	9004

Table 8.2: Testing dataset distribution

(a) SNORT

	Direct Attacks		
	Detected	Total	%
Apache Tomcat	33 +24	57	100.00
DistCC	10	10	100.00
MSSQL	31	31	100.00
PostgreSQL	13	13	100.00
Samba	18	18	100.00
Server	26	26	100.00
Overall Detection	155	155	100.00
ADR* per Service			100.00

*Average detection rate.

(b) SURICATA

	Direct Attacks		
	Detected	Total	%
Apache Tomcat	56 +1	57	100.00
DistCC	0	10	0.00
MSSQL	31	31	100.00
PostgreSQL	0	13	0.00
Samba	0	18	0.00
Server	26	26	100.00
Overall Detection	114	155	73.55
ADR* per Service			50.00

*Average detection rate.

Table 8.3: Detection of direct attacks by SNORT and SURICATA

Apache service were detected by high priority rules of SNORT and 24 attacks were undetected. Despite of it, we considered these attacks as correctly detected, as they occurred almost at the same time like other correctly predicted direct attacks instances, and thus were part of their execution. In the case of SURICATA, only one such undetected direct attack occurred. However, unlike SNORT, SURICATA did not fire any alert representing buffer overflow nor shellcode nor remote commad execution, but instead fired combination of high priority alerts related to potential corporate privacy violation:

- `ET POLICY`
  `Incoming Basic Auth Base64 HTTP`
  `Password detected unencrypted`

- `ET POLICY`
  `Outgoing Basic Auth Base64 HTTP`
  `Password detected unencrypted`

- `ET POLICY`
  `HTTP Request on Unusual Port Possibly Hostile`

- `ET POLICY`
  `Internet Explorer 6 in use`
  `Significant Security Risk,`

which we decided to consider as correctly detected. If we did not consider them as correctly detected, then SURICATA would not detect any attack on Apache service.

Next, we analyzed detection capabilities of both NIDSs on obfuscated attacks and the results are depicted in Table 8.4. Comparing the detection rate of SNORT and SURICATA on obfuscated attacks, we can conclude that SNORT overcame SURICATA again and the ratio of their correct detection was almost the same as in the case of direct attacks (Table 8.3). The only difference happened during exploitation of vulnerability in Server service, where 2 instances of obfuscated attacks were not detected by any NIDS. These 2 instances utilized obfuscations with IDs (f) and (g), both from category of unreliable network traffic channel simulation techniques (see Table 8.1). There were also several undetected obfuscated attacks on Apache service in both NIDSs, but we were able to track their occurrences and associate them as part of other detected attacks, thus the detection rate for Apache service achieved 100.00% for both NIDSs. Regarding to Apache service, SURICATA once again did not fired any alert detecting malicious content, but instead fired previously mentioned combination of high priority alerts stating corporate privacy violation, which were, once again, considered as correct detection. Also, notice that SURICATA fired one non-high-priority alert classified as potentially bad traffic in all instances of direct and obfuscated attacks exploiting PostgreSQL service:

`ET POLICY`
`Suspicious inbound to PostgreSQL port 5432.`

But we did not considered it as correct detection due to the priority and the classification of the alert. As discussed in Section 7.4.2, VirusTotal likely use paranoic rule set, and thus fired alerts may contain false positives. However comparing fired alerts before and after obfuscation, we can see that detection properties of utilized NIDSs are resistant to the non-payload-based obfuscations in the majority of the cases.

(a) SNORT

	Obfuscated Attacks		
	Detected	**Total**	**%**
Apache Tomcat	128 +36	164	100.00
DistCC	23	23	100.00
MSSQL	103	103	100.00
PostgreSQL	45	45	100.00
Samba	43	43	100.00
Server	100	102	98.04
Overall Detection	478	480	99.58
ADR* per Service			99.67

*Average detection rate.

(b) SURICATA

	Obfuscated Attacks		
	Detected	**Total**	**%**
Apache Tomcat	162 +2	164	100.00
DistCC	0	23	0.00
MSSQL	103	103	100.00
PostgreSQL	0	45	0.00
Samba	0	43	0.00
Server	100	102	98.04
Overall Detection	367	480	76.46
ADR* per Service			49.67

*Average detection rate.

Table 8.4: Detection of obfuscated attacks by SNORT and SURICATA

8.6 Data Mining Experiments

The purpose of this section is to perform supervised classification in order to evaluate the effectiveness of the proposed obfuscation techniques as well as feedback of the classifier having obfuscated data included in its training process.

All experiments were performed in Rapid Miner Studio [157] using a k-fold cross validation and conditional probability based Naive Bayes classifier which is also employed in other works [11, 77, 118]. The optimization of the Naive Bayes classifier was not the subject of our experiments, and therefore we utilized fixed settings of the classifier.

Considering our current dataset's class distribution, we utilized 5-fold cross validation which creates big enough D_t sets for binominal classification. In the case of multi-class classification, we utilized 3-fold cross validation. All cross validation experiments of our work have been adjusted to employ stratified sampling during assembling of data folds. The stratified sampling ensures equally balanced class distribution of each fold.

8.6.1 Forward Feature Selection Experiment

This principle of the experiment is the same as in one described in 7.6.1. The experiment consisted of two executions of the FFS. The first took as input just legitimate traffic and direct attack entries, and represented the case where ADS was trained without knowledge about obfuscated attacks. The second execution took as input the whole dataset of network traffic – consisting of legitimate traffic, direct attacks as well as obfuscated ones, and therefore, represented the case where ADS was aware of obfuscated attacks. The selected features of both executions are depicted in Table 8.5.The penultimate column of Table (i.e., FFS DOL) denotes the selected features where the whole dataset was utilized for the FFS, and the last one (i.e., FFS DL) denotes the case where only direct attacks and legitimate traffic were considered. We have noticed that the final model of the FFS DL case has achieved average recall of classes equal to $99.68\% \pm 0.65\%$ as well as $99.99\% \pm 0.02\%$ accuracy in comparison to the FFS DOL case in which average recall equal to $99.62\% \pm 0.40\%$ and accuracy of $99.40\% \pm 0.16\%$ has been achieved, indicating that it has been a little difficult to train the classifier with inclusion of obfuscated attacks unlike the case not considering them.

Several mutual features were selected in both cases which means they provided a value regardless of whether obfuscation was performed or not. From another point of view, we noticed that in the FFS DL case some features more susceptible to suggested obfuscations were selected in comparison with the FFS DOL case, e.g. MeanPktLenOut, OutPktLen{4s10i[2], 4s10i[3], 1s10i[1]}, as modification of packets' lengths is the part of fragmentation-like obfuscations as well as packets can be re-transmitted and spread in time which can markedly influence mentioned features. Therefore, almost all of the following experiments will utilize the feature set gained from the second execution (i.e., FFS DOL), as we consider them as more appropriate for general behavior representation of both kinds of attacks. The only exception is the second binominal classification experiment which will be described in Subsection 8.6.3.

8.6.2 Binominal Classification Experiment I

Similarly to the previous experiment, the current one also worked with two classes prediction. A 5-fold cross validation was performed using direct attacks and legitimate traffic and further prediction of the whole dataset. The results of the cross validation experiment which use data consisting of legitimate traffic and direct attacks are shown in Table 8.6a. The classifier trained on all direct attacks and legitimate traffic instances was applied in the prediction of the whole dataset (including obfuscated attacks) and it correctly predicted 71.25% of obfuscated attacks and 78.26% of all attacks respectively. An associated confusion matrix is depicted in Table 8.6b. The achieved result proclaimed the existence of some successful obfuscations of attacks which were predicted as legitimate traffic (138 instances[4]).

In the next part of the current binominal classification experiment, we performed 5-fold cross validation of the whole dataset including obfuscated attacks. The results of this experiment are shown in Table 8.7. The high precision and recall of both classes represents the fulfilled assumption that a classifier trained with knowledge about some obfuscated attacks is able to detect the same or similar obfuscated attacks later.

[4]We notice that Table 8.6a contains 2 false negative instances of direct attacks, but they are not present in Table 8.6b. It happened because Table 8.6b evaluates the model built on all instances of direct and legitimate traffic in comparison with k-cross validation which evaluates k models trained on $k-1$ folds.

Feature ID	Description	FFS DOL	FFS DL
SigPktLenOut	• Std. deviation of outbound (client to server) packet sizes.	X	
SigPktLenIn	• Std. deviation of inbound (server to client) packet sizes.	X	
HasFragIP	• The feature indicates occurrence of fragmented packets.	X	
MeanPktLenOut	• Mean of packets' sizes in outbound traffic of a connection.		X
SumTCPHdrLen	• The sum of TCP header lengths of all traffic.	X	
BytesTCPOverhead	• The number of bytes transferred during establishment and finishing of a connection.	X	X
ConTcpFinCntIn	• The number of TCP FIN flags occurred in inbound traffic.	X	X
ConTcpSynCntIn	• The number of TCP SYN flags occurred in inbound traffic.	X	X
GaussProds8AllN[2]*	• Normalized products of all packet sizes with 8 Gaussian curves. The feature represents a product of the 3rd slice of packets with a Gaussian function which fits to the interval of the packets' slice.	X	
GaussProds8AllN[7]*	• The same as the previous one, but it represents a product of the 8th slice of packets with a Gaussian function which fits to the interval of the packets' slice.	X	
GaussProds8All[6]	• The same as the previous one, but it represents a product of the 7th slice of packets with a Gaussian function which fits to the interval of the packets' slice.		X
GaussProds8In[3]	• The same as the previous one, but computed above inbound packets and represents a product of the 4th slice of packets with a Gaussian function which fits to the interval of the packets' slice.	X	
FourGonAngleN[9]*	• Fast Fourier Transformation (FFT) of all packet sizes. The feature represents the angle of the 10th coefficient of the FFT in goniometric representation.	X	X
FourGonModulN[0]*	• The same as the previous one, but it represents the module of the 1st coefficient of the FFT in goniometric representation.		X
FourGonModulIn[1]	• FFT of inbound packet sizes. The feature represents the module of the 2nd coefficient of the FFT in goniometric representation.		X
OutPktLen4s10i[2]	• Lengths of outbound packets occurred in the first 4 seconds of a connection which are distributed into 10 intervals. The feature represents totaled outbound packet lengths of the 3rd interval.		X
OutPktLen4s10i[3]	• The same as the previous one, but it represents totaled outbound packet lengths of the 4th interval.		X
OutPktLen1s10i[1]	• The same as the previous one, but computed above the first second of a connection. The feature represents totaled outbound packet lengths of the 2nd interval.		X

*Sizes of inbound and outbound packets are represented by negative and positive values, respectively.

Table 8.5: ASNM features selected by FFS method

(a) Direct attacks & legitimate traffic cross validation

Classification Accuracy: 99.95% ± 0.04%		**True Class**		**Precision**
		Legit. Flows	**Direct Attacks**	
Predicted Class	**Legit. Flows**	8367	2	99.98%
	Direct Attacks	2	153	98.71%
	Recall	99.98%	98.71%	$F_1 = 98.71\%$

(b) Whole dataset prediction

Classification Accuracy: 98.46%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	8368	138	98.38%
	All Attacks	1	497	99.80%
	Recall	99.99%	78.27%	$F_1 = 87.73\%$

Table 8.6: Confusion matrices of binominal classification (FFS DOL features)

Classification Accuracy: 99.84% ± 0.08%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	8359	4	99.95%
	All Attacks	10	631	98.44%
	Recall	99.88%	99.37%	$F_1 = 98.90\%$

Table 8.7: Legitimate traffic & all attacks cross validation (FFS DOL features)

8.6.3 Binominal Classification Experiment II

The second binominal classification experiment basically did the same as the first one but the only difference resided in the subset of features utilized in it. The experiment considered features collected by FFS method which ran on the dataset consisted of direct attacks and legitimate traffic only (previously referred to as FFS DL features).

The results of the cross validation experiment are shown in Table 8.8a. The classifier trained on the data collection containing direct attacks and legitimate traffic was applied in the prediction of the whole dataset (including obfuscated attacks) and it correctly predicted 67.50% of obfuscated attacks (75.43% of all attacks, respectively). An associated confusion matrix is depicted in Table 8.8b.

Finally, the cross validation of the whole dataset (including obfuscated attacks) was performed and the obtained results are depicted in Table 8.9. To summarize differences in the results of the both binominal classification experiments, we conclude that the second model (using FFS DL features) has achieved slightly better results in learning direct attacks and legitimate traffic characteristics than the first model (using DOL features), but on the other hand, later, it has resulted into more misclassified cases of obfuscated attacks than the first model (i.e., 155:138 instances), as well as it has achieved worse results in cross validation of the whole dataset than the first one. Therefore, our consideration about more

(a) Direct attacks & legitimate traffic cross validation

Classification accuracy: 99.99% ± 0.02%		**True class**		**Precision**
		Legit. flows	**Direct attacks**	
Predicted class	**Legit. flows**	8368	0	100.00%
	Direct attacks	1	155	99.36%
	Recall	99.99%	100.00%	$F_1 = 99.67\%$

(b) Whole dataset prediction

Classification accuracy: 98.27%		**True class**		**Precision**
		Legit. flows	**All attacks**	
Predicted class	**Legit. flows**	8369	156	98.17%
	All attacks	0	479	100.00%
	Recall	100.00%	75.43%	$F_1 = 85.99\%$

Table 8.8: Confusion matrices of binominal classification (FFS DL features)

Classification accuracy: 99.40% ± 0.16%		**True class**		**Precision**
		Legit. flows	**All attacks**	
Predicted class	**Legit. flows**	8330	15	99.82%
	All attacks	39	620	94.08%
	Recall	99.53%	97.64%	$F_1 = 95.82\%$

Table 8.9: Legitimate traffic & all attacks cross validation (FFS DL features)

appropriate feature set has been confirmed and there will be no longer considered FFS DL features.

8.6.4 Trinominal Classification Experiment

The current experiment divides the attack class into two subclasses (obfuscated and direct attacks) resulting into trinominal classification. The assumption of having different statistical and behavioral properties in direct and obfuscated attacks cases is considered. Then, the classifier should be able to distinguish between these two attacks' subclasses.

Confusion matrix related to the experiment is depicted in Table 8.10. 5-fold cross validation of the classifier confirms the previous assumption for over 80% of all attack cases. There are almost 22% of direct attacks (34 instances) predicted as obfuscated attacks and almost 14% of obfuscated attacks (67 instances) predicted as direct ones. The gray color of the cells' background denotes these two cases.

Notice, that the overall prediction preserves attack's superclass – which refers to the comparable results to those achieved in the first binominal classification experiment (Table 8.7). No direct attacks are predicted as legitimate traffic and vice versa, however, in the case of obfuscated attacks 8 obfuscated attacks are predicted as legitimate traffic and 10 instances of legitimate traffic are predicted as obfuscated attacks, which may indirectly indicate more similar behavioral characteristics of these two classes than in the case of direct attacks and

Classification Accuracy: 98.68% ± 0.20%		True Class			$PPV(C_i)$	$F_1(C_i)$
		Legit. Flows	**Direct Attacks**	**Obfus. Attacks**		
Predicted Class	**Legit. Flows**	**8359**	0	8	99.90%	99.89%
	Direct Attacks	0	**121**	67	64.36%	70.55%
	Obfus. Attacks	10	34	**405**	90.20%	87.19%
	Recall	99.88%	78.06%	84.38%		

Table 8.10: Confusion matrix of trinominal classification

legitimate traffic pair. Aside from the previous statements, there is achieved average recall of all classes equal to $87.44\% \pm 3.43\%$ which can be still considered as interesting one from the classification perspective, but it is not the purpose of the experiment.

8.6.5 Multi-class Classification Experiment

Different view on the results of the obfuscations is provided by multi-class classification experiment. The experiment distributes the input dataset into 19 classes according to the class of a communication on a particular network service. The communication classes are represented by direct and obfuscated attacks versus legitimate traffic. The Cartesian product of 3 communication classes with all considered services yields 18 classes plus 1 class representing other legitimate traffic.

The confusion matrix of 3-fold cross validation is shown in Table 8.11. Mean recall of the classes is equal to $69.94\% \pm 2.61\%$ which does not indicate excellent performance of the multi-class classifier, however, it is not the purpose of the experiment. Cells with a gray background depict incorrectly predicted attacks – obfuscated attacks predicted as direct ones and vice versa. These cells shape a parallel line to a great diagonal of the matrix. In these cases, ASNM are not able to distinguish between obfuscated and direct attacks. Nevertheless, more important obfuscated cases, having discriminating properties and distinguished by FFS DOL features, lay on the great diagonal. The next important fact is that attacks' superclass is preserved in the majority of the cases.

			True Class Distribution																			
			Direct Attacks						Obfus. Attacks						Legit. Traffic							
		Classification Accuracy: 95.88% +/ 0.23%	**Apache Tomcat**	**DistCC**	**MSSQL**	**PostgreSQL**	**Samba**	**Server**	**Apache Tomcat**	**DistCC**	**MSSQL**	**PostgreSQL**	**Samba**	**Server**	**Apache Tomcat**	**DistCC**	**MSSQL**	**Other flows**	**PostgreSQL**	**Samba**	**Server**	**Precision**	**F measure$_1$**
Predicted Class Distribution	**Direct Attacks**	**Apache Tomcat**	**52**	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0	**70.27%**	**79.39%**
		DistCC	0	**10**	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	**58.82%**	**74.07%**
		MSSQL	0	0	**24**	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	**85.71%**	**81.35%**
		PostgreSQL	0	0	0	**11**	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	**78.57%**	**81.48%**
		Samba	0	0	0	0	**18**	0	0	0	0	0	12	0	0	0	0	0	0	0	0	**60.00%**	**75.00%**
		Server	0	0	0	0	0	**8**	0	0	0	0	0	14	0	0	0	0	0	0	0	**36.36%**	**33.33%**
	Obfus. Attacks	**Apache Tomcat**	5	0	0	0	0	0	**137**	1	0	0	0	0	0	0	0	0	0	0	2	**94.48%**	**88.67%**
		DistCC	0	0	0	0	0	0	0	**11**	0	0	2	0	0	0	0	0	0	0	0	**84.62%**	**61.12%**
		MSSQL	0	0	6	0	0	0	0	0	**99**	0	0	0	0	0	0	0	0	0	0	**94.29%**	**95.20%**
		PostgreSQL	0	0	0	2	0	0	0	0	0	**41**	0	0	0	0	0	0	0	0	0	**95.35%**	**93.18%**
		Samba	0	0	0	0	0	0	0	3	0	0	**26**	0	0	0	0	0	0	0	0	**89.66%**	**72.23%**
		Server	0	0	1	0	0	18	1	0	0	1	0	**86**	0	0	0	4	0	0	0	**77.48%**	**80.75%**
	Legit. Traffic	**Apache Tomcat**	0	0	0	0	0	0	0	0	0	0	0	0	**3**	4	4	2	6	5	0	**12.50%**	**10.34%**
		DistCC	0	0	0	0	0	0	0	1	0	0	2	0	11	**55**	28	2	17	6	7	**42.64%**	**48.04%**
		MSSQL	0	0	0	0	0	0	2	0	0	0	0	0	8	21	**26**	2	9	6	9	**31.33%**	**32.92%**
		Other flows	0	0	0	0	0	0	2	0	0	0	0	2	1	2	2	**99**	0	3	5	**85.34%**	**76.15%**
		PostgreSQL	0	0	0	0	0	0	0	0	0	0	1	0	9	16	12	0	**7**	3	4	**13.46%**	**14.89%**
		Samba	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	25	0	**4608**	0	**99.42%**	**99.42%**
		Server	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	10	3	4	**3312**	**99.34%**	**99.26%**
	Recall		**91.23%**	**100.00%**	**77.42%**	**84.62%**	**100.00%**	**30.77%**	**83.54%**	**47.83%**	**96.12%**	**91.11%**	**60.47%**	**84.31%**	**8.82%**	**55.00%**	**34.67%**	**68.75%**	**16.67%**	**99.42%**	**99.19%**		

Table 8.11: Confusion matrix of multi-class classification

8.6.6 Comparison of Various Classifiers

For the purpose of performance comparison of various classifiers, we executed 5-fold cross validation on already discussed and employed classification models – Decision Tree and SVM. FFS DOL feature set was utilized in this experiment as the input for the classifiers working with two class prediction. At first, we evaluated performance of the SVM classifier which utilized radial basis function as non-linear kernel function. The results of the evaluation are represented by confusion matrix which is depicted in Table 8.12.

Classification Accuracy: 99.51% ± 0.12%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	8329	4	99.95%
	All Attacks	40	631	94.04%
	Recall	99.52%	99.37%	$F_1 = 96.63\%$

Table 8.12: Performance of the SVM classifier

The following experiment was performed with Decision Tree classifier utilizing gini index as selection criterion for splitting of attributes. The associated confusion matrix which represents the result is shown in Table 8.13. Comparing the results of the current two experiments with performance achieved by the Naive Bayes classifier, we can declare that Naive Bayes achieved the best performance, while the second best was Decision Tree, followed by SVM. This statement is the same as in the tunneling obfuscation case (Section 7.6.5).

Classification Accuracy: 99.74% ± 0.13%		**True Class**		**Precision**
		Legit. Flows	**All Attacks**	
Predicted Class	**Legit. Flows**	8358	12	99.86%
	All Attacks	11	623	98.26%
	Recall	99.87%	98.11%	$F_1 = 98.18\%$

Table 8.13: Performance of the Decision Tree classifier

Also, all the classification models were compared by ROC method, which considered neutral bias. The most interesting configurations of classifier adjustment were achieved in the case of the Naive Bayes classifier and the SVM one. ROC diagram with neutral bias is depicted in Figure 8.3. Note that we executed ROC method in one run with stratified distribution of samples, which utilized 90% of the input dataset for training and 10% for validation purpose. Therefore, the figure does not contain any variability of the ROC curves.

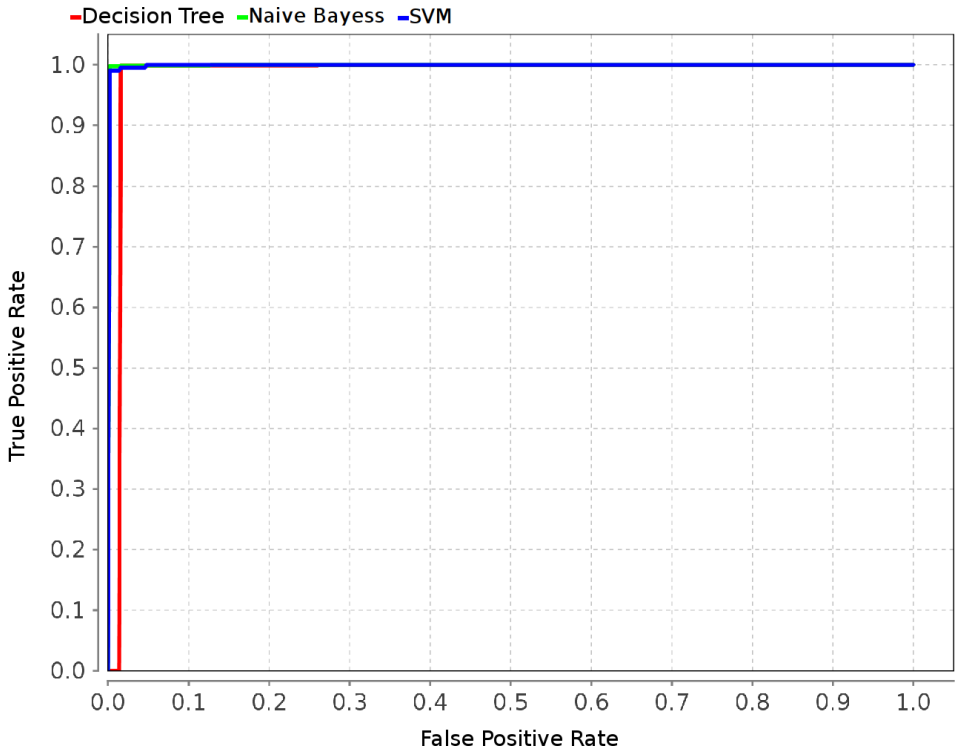

Figure 8.3: ROC diagram comparing classifiers

8.7 Summary of the Obfuscation Techniques

The section compares and analyzes utilized obfuscation techniques identified by IDs listed in Table 8.1. The results presented in the first part of the section originate from a binominal classification experiment in which the classifier is trained without obfuscated attack knowledge and validated on the whole dataset. The obfuscations are considered successful if they are predicted as legitimate traffic, and therefore the situation represents the ADS evasion case. Table 8.14 presents ordered ratios of successfully obfuscated attacks per technique. The most successful obfuscations use combinations of more techniques (i.e., o, q, p), damaging of packets (i.e., f, e) and spreading out packets in time with delays specified by normal distribution (i.e., c). From the MTU modification techniques, (n) appear to be the most successful. We can observe inverse relationship – the lower the MTU value is, the higher number of successful instances obfuscation achieves. It is caused by a direct relationship of some features to modification of the data packet's lengths. On the other hand, the least successful obfuscations utilize spreading out packets in time with a short delay (a), modification of MTU by higher values than 500 (i.e., k, l) and reordering of packets (i.e., i, j).

Table 8.15 presents ordered ratios of successfully obfuscated attacks per service. Comparing both features' sets, it can be noted that the average ratio of successful obfuscations is lower in the FFS DOL case (i.e., 30.24%) than in the case of FFS DL (i.e., 37.10%). Therefore, it endorses better resistance to obfuscated attacks of the FFS DOL feature set than the FFS DL one. Regarding the FFS DOL feature set (Table 8.15a), the most successful ob-

Obfuscation Technique	All Instances	Successful Instances	Ratios of Successful Instances
(o)	27	14	51.85%
(f)	29	14	48.28%
(c)	30	14	46.67%
(q)	35	16	45.71%
(p)	33	15	45.45%
(e)	27	10	37.04%
(n)	27	10	37.04%
(d)	30	10	33.33%
(m)	27	9	33.33%
(g)	27	8	29.63%
(b)	22	5	22.73%
(h)	30	6	20.00%
(i)	27	2	7.41%
(j)	27	2	7.41%
(l)	27	2	7.41%
(a)	28	1	3.57%
(k)	27	0	0.00%

Table 8.14: Ratios of successfully obfuscated attacks per technique

fuscated attacks are those exploiting DistCC and Samba services. From the analysis of the distributions of features' values, it was found out that instances of direct attacks executed on these services had very flat distribution of values of many features in comparison with other direct attacks. Example features are standard deviation of inbound and outbound packet sizes of the connection, followed by other features dependent on the packets' length variability of the flow. Therefore, obfuscated attacks on these services cause more variability of the features which is in many cases similar to legitimate traffic.

On the other hand, in the cases of MSSQL and PostgreSQL many features' values of the direct attacks are more divergent through their instances, and thus obfuscations contribute to the divergence only in low scale. Therefore, the most of the obfuscated attacks have similar characteristics like direct ones, which enables their correct detection.

8.7.1 Discriminative Analysis of Prediction

The results of the current subsection evaluate discriminative characteristics of the classifier with emphasis on distinction among obfuscated attacks, direct attacks and legitimate traffic. We know that there exist obfuscated attack instances which are easier to correctly predict than other ones. We denote characteristics of such attacks as discriminative. The previous fact also occurs in direct attacks and legitimate traffic classes. Therefore, obfuscated attacks having discriminative characteristics should be correctly predicted as intrusions. Following results utilize gathered data of trinominal and multi-class classification experiments, and thus, assume that obfuscated attacks' ground truth is available in the training process of the ADS classifier. Table 8.16 presents ordered ratios of correctly predicted obfuscated attacks per obfuscations technique, i.e., ones having discriminative characteristics.Spreading

(a) FFS DOL features

Service	Ratios of Successful Instances
DistCC	43.48%
Samba	41.86%
Apache Tomcat	35.37%
Server	26.47%
PostgreSQL	17.78%
MSSQL	16.50%

(b) FFS DL features

Service	Ratios of Successful Instances
DistCC	52.17%
Samba	51.16%
Server	46.08%
PostgreSQL	28.89%
Apache Tomcat	26.83%
MSSQL	17.48%

Table 8.15: Ratios of successfully obfuscated attacks per service

out packets in time (i.e., b, c) together with the combinations of various techniques (i.e., p, q) achieve the highest discrimination rate in the both classification cases. Moreover, they can be considered as resilient to a granularity of the classes. As obfuscation techniques not resilient to granularity of the classes can be considered one which performs spreading out packets in time using short constant delay (i.e., a) and one which performs damage of packets (i.e., g). The least discriminative characteristics are obvious at MTU modification techniques with the highest values (i.e., k, l).

(a) Trinominal classification

Obfuscation ID	True Prediction
(b)	100.00%
(p)	100.00%
(q)	100.00%
(c)	96.67%
(f)	96.55%
(e)	85.19%
(g)	85.19%
(o)	85.19%
(d)	83.33%
(a)	82.14%
(j)	81.48%
(n)	81.48%
(h)	76.67%
(i)	74.07%
(m)	70.37%
(k)	66.67%
(l)	62.96%

(b) Multi-class classification

Obfuscation ID	True Prediction
(b)	100.00%
(p)	100.00%
(q)	97.14%
(c)	93.33%
(n)	88.89%
(o)	88.89%
(e)	85.19%
(d)	83.33%
(h)	83.33%
(f)	79.31%
(i)	77.78%
(j)	77.78%
(m)	77.78%
(g)	74.07%
(a)	71.43%
(k)	66.67%
(l)	66.67%

Table 8.16: Ratios of obfuscated attacks having discriminative characteristics

Table 8.17 depicts ordered ratios of attacks' obfuscations per service having discriminating characteristics. As can be seen in Table 8.17, obfuscated attacks exploiting MSSQL and PostgreSQL services are the most discriminative. In the case of MSSQL, it is caused by the fact, that direct attacks contain around 20-50 times and 5-10 times more data packets transferred from client to server and vice versa, respectively, in comparison with other direct attack types and legitimate traffic classes. The second, but not less important observation of direct attacks at MSSQL service, is the fact that average size of data packets uploaded to the server is approx. 15 times greater than downloaded one. Therefore, lot of obfuscations modify characteristics of the attacks in correlated scale which reflects the distinctness of them.

The situation regarding the second observation of MSSQL is very similar in the case of PostgreSQL as well. The substantial difference resides in the fact that average size of uploaded packets is approx. 2 times greater than in the case of average downloaded size. This ratio is much more lower in other direct attack cases.

Contrary, there has been achieved the lowest discrimination rate of the obfuscated attacks exploiting DistCC service. It can be explained by the two facts: a) first, direct attacks on the service contain the lowest number of data packets among all other direct attacks classes, and therefore, some obfuscation techniques using only low percentage of attack's flow modification are applied on them in that scale; b) second, the most of the data packets of direct attacks on the service have lower size than 250 Bytes, and therefore MTU modification obfuscations do not expressively influence the TCP flow of the attack.Comparing the

(a) Trinominal classification

Service	True Prediction
MSSQL	96.12%
PostgreSQL	95.56%
Apache Tomcat	85.29%
Server	79.27%
Samba	72.09%
DistCC	65.22%

(b) Multi-class classification

Service	True Prediction
MSSQL	96.12%
PostgreSQL	91.11%
Apache Tomcat	84.31%
Server	83.54%
Samba	60.47%
DistCC	47.83%

Table 8.17: Ratios of discriminative obfuscated attacks per service

results which contains Table 8.17 to those present at Table 8.15a, we realize that they almost have inverse relationship. The interpretation of this observation can be formulated by the statement: The more discriminative characteristics obfuscated attacks have, the higher is the probability of their detection.

8.8 Analysis of Network Traffic Characteristics

Alike Section 7.7 of Chapter 7, we examined the value density distribution of each feature. This time, the emphasis was put on the distinctiveness of malicious and legitimate network traffic, as the current obfuscation techniques were not as efficient as tunneling one, and thus it was difficult to find significant differences between obfuscated and direct malicious traffic by analyzing features independently. For the purpose of value density examination among TCP connections of our dataset, we utilized frequency analysis of the features' values. Also, kernel density estimation with Gaussian kernels was utilized in few cases because of their clearer interpretation. Several features presenting the interesting characteristics of each network traffic class will be discussed and described.

Considering categorization of the features from Section 7.7, we primarily aim at **discriminating features** enabling good malicious and legitimate traffic separation. However, we present one obvious example of **obfuscated features'** group which represents specific characteristics of obfuscated attacks which are similar to legitimate traffic.

8.8.1 Discriminating Features

We selected the mean of packet lengths from client to server (`MeanPktLenOut`) as the first example of the discriminating features' category. The corresponding histogram of the feature is depicted in Figure 8.4. The obvious observation of the figure is that both of the attack

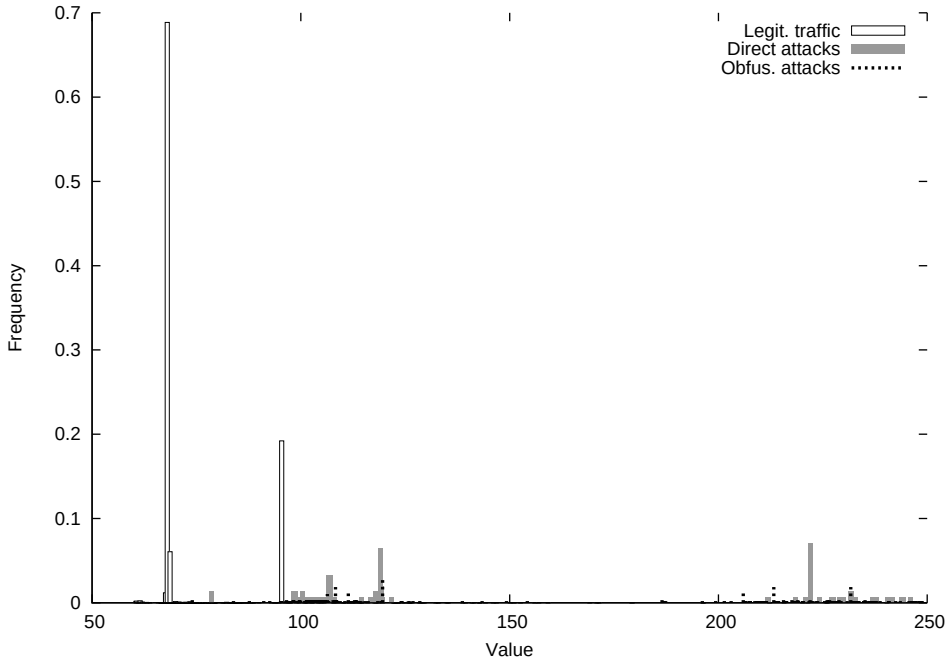

Figure 8.4: Mean of packet lengths from client to server (`MeanPktLenOut`)

classes have different values than the most of legitimate traffic representatives. There can be seen two values which revealed to be characteristic for legitimate traffic – 70 and 95 as well as their near surroundings; while in the case of malicious traffic, primarily two value ranges were observed: $\langle 100, 125\rangle$ and $\langle 200, 250\rangle$. Also, regarding to FFS experiment of Subsection 8.6.1, we can see that this feature was selected into FFS DL set (see Table 8.5).

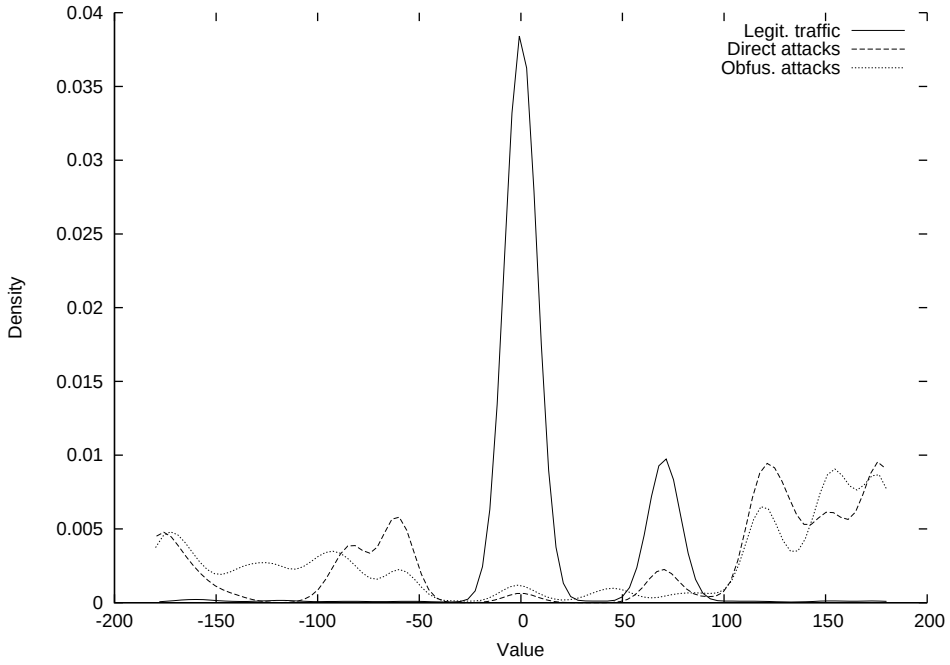

Figure 8.5: FFT of packet lengths from client to server (`FourGonAngleOut[2]`)

The next discriminating feature is FFT of lengths of packets which were transferred from client to server (`FourGonAngleOut[2]`). The feature denotes the angle of goniometric representation of the 3rd frequency of FFT. The Gaussian kernel density estimation of the feature is depicted in Figure 8.5. The important observation of the figure is that values of legitimate traffic instances are spread in two intervals: $\langle$-*25*, *25*$\rangle$ and $\langle 50, 100\rangle$; while intervals $\langle$-*180*, -*50*$\rangle$ and $\langle 100, 180\rangle$ are characteristic for malicious traffic.

The histogram of the feature representing standard deviation of lengths of packets transferred from server to client (`SigPktLenIn`) is depicted in Figure 8.6. The legitimate traffic representatives are characteristic by two value ranges: $\langle$-*10*, *10*$\rangle$ and $\langle 150, 220\rangle$. The interval $\langle$*150*, *160*$\rangle$ does not have discriminating properties but representatives falling into this interval can be distinguished by another features.

Another example belonging into the current category of discriminating features is distribution of lengths of outbound packets occurred in the first 4 seconds of a connection (`OutPktLen4s10i[9]`). The histogram of the feature can be found in Appendix H.1. The next identified examples of discriminating features are standard deviation of lengths of packets transferred from client to server (`SigPktLenOut`) and the sum of TCP header lengths of all packets transferred in TCP connection (`SumTCPHdrLen`). The corresponding histograms of the features are depicted in Appendix H.2 and H.3, respectively. Finally, the last representative of this category is FFT of packet lengths which are considered as negative values for incoming direction and positive values for outgoing one (`FourGonAngleNeg[5]`). The Gaussian kernel density estimation of the feature is depicted in Appendix H.4.

8.8.2 Obfuscated Features

Since the current obfuscation techniques were not as efficient as tunneling one, it was more difficult to find obfuscated features. The only representative feature which we present is normalized sum of products of a connection with 8 Gaussian curves (`GaussProds8All[6]`),

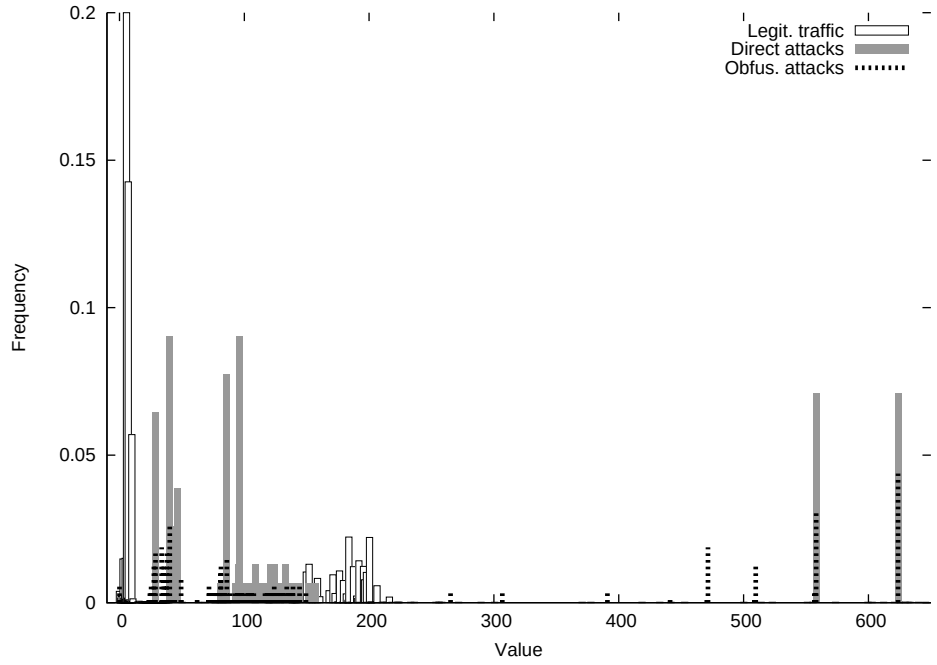

Figure 8.6: Standard deviation of lengths of packets transferred from server to client (`SigPktLenIn`)

which is depicted in Figure 8.7. The feature represents the 7th sum of products and the figure shows just the most interesting sub-interval of the feature. The most important observation of the figure occurs around the value of 16, where exists significant amount of obfuscated

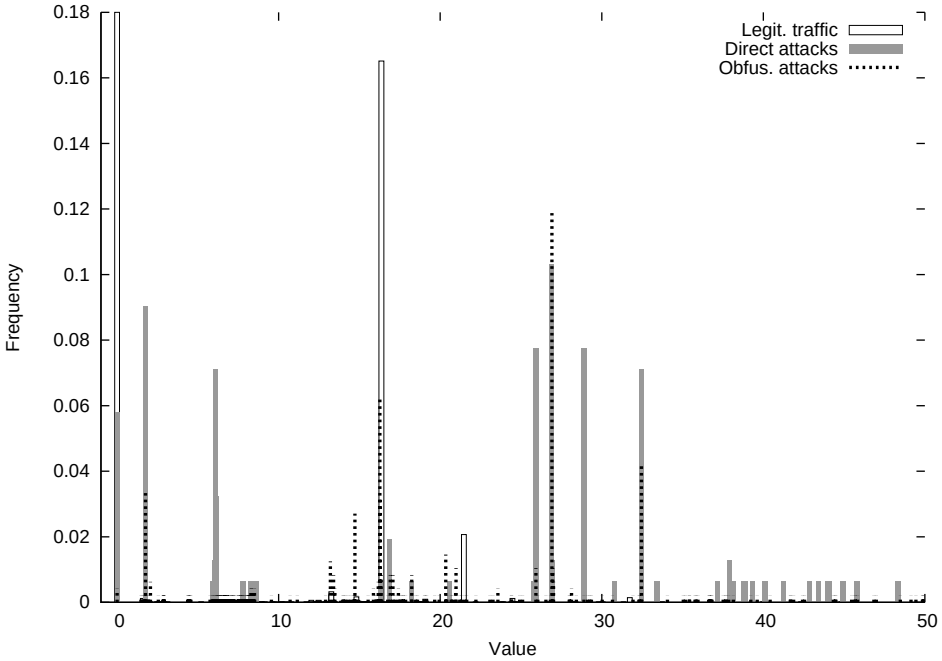

Figure 8.7: Normalized sum of products of a connection with 8 Gaussian curves (`GaussProds8All[6]`)

attacks which have very similar values of the feature as legitimate traffic has. Therefore, it can be difficult to distinguish between malicious and legitimate traffic classes at this value, and thus obfuscation was partially successful from the perspective of this feature.

8.9 Concluding Remarks

In this chapter, we performed remote intrusive attacks on selected vulnerable network services employing various non-payload-based obfuscation techniques based on NetEm and MTU modifications with intention to make behavioral characteristics of the attacks, represented by ASNM features, being similar to legitimate traffic ones, and thus cause false negative predictions of machine learning based ADS (AIPS).

The summary of the presented results revealed non-payload-based obfuscation techniques as partially successful in evading the detection by Naive Bayes classifier trained without knowledge about them. On the other hand, if some of the obfuscated attacks were included into the training process of the classifier, then it was able to detect other (similar) obfuscated attacks with high recall and precision. Therefore, the assumptions from Subsection 8.2.1 were confirmed. Other results were achieved by considering the ground truth of obfuscated attacks, and demonstrated that ASNM features are not able to distinguish between obfuscated and direct attacks in some cases, but they are able to keep attacks' superclass. Further, we examined interesting characteristics of all network attacks and legitimate traffic. Lot of discriminating features were identified, while obfuscated features were rare. The obfuscation-like properties of simulated attacks were not as evident as in the case of tunneling obfuscation.

Moreover, several experiments with signature-based NIDSs – SNORT and SURICATA – were performed and the results revealed that detection capabilities of utilized NIDSs are resistant to our proposed obfuscation techniques in almost all instances. In the case of SNORT, there were only 2 instances of undetected obfuscated attacks, while all direct attacks were detected. However, the situation was little different in the case of SURICATA, which did not contain rules for detection of attacks on DistCC, PostgreSQL and Samba service, and thus did not detect any of direct and obfuscated attacks on those services. Despite of it, SURICATA detected all instances of direct and obfuscated attacks executed on remaining services. Therefore, we can say that if signature based NIDSs had rules detecting direct variants of attacks, then these rules also match in almost all instances of obfuscated attacks.

8.9.1 (Non) Exigency of Network Normalization

If we would assume existence of an optimal network normalizer for ADS which would be able to completely eliminate the impact of proposed non-payload-based obfuscation techniques, then these obfuscation techniques would be useless. If such optimal network normalizer would exists, then it would still be prone to state holding and CPU overload attacks. Contrary, if we would not assume network normalizer as part of ADS system, then:

"non-payload-based obfuscation techniques may be employed as training data driven approximation of network normalizer,"

which would not be prone to previously mentioned issues and attacks. The situation can be demonstrated by our binomal classification experiments described in 8.6.2 and 8.6.3. Consider ADS model validated on direct attacks and legitimate traffic whose performance achieved average recall of 99.99% and F_1-*measure* equal to 99.67% (see Table 8.8a). Training and testing data of ADS were built upon normalized malicious network traffic represented by direct attacks. Then, the model trained on the direct attacks and legitimate traffic was applied onto prediction of the whole dataset which included obfuscated attacks as well. In this case obfuscated attacks represented un-normalized malicious network traffic, and thus classifier of ADS achieved worse performance than in the previous case – average recall of 87.72% and F_1-*measure* equal to 85.99% were achieved (see Table 8.8b).

In order to alleviate negative performance impact of un-normalized malicious network traffic (represented by obfuscated attacks) on our ADS system not performing normalization, we can include obfuscated attacks into the training process of the classifier. This case is interpreted by confusion matrix depicted in Table 8.7. The average recall of 99.63% and F_1-*measure* equal to 98.90% was achieved which were improved by **11.91%** and **12.91%**, respectively. Thus network normalizer element may be omitted from ADS infrastructure and can be approximated by appropriate training data.

Chapter 9

Conclusion

In this thesis, I firstly compiled taxonomy of network intrusion detection approaches. Considering the taxonomy by Lim [103], I decided to concentrate my attention and research to the category of model-based behavioral network intrusion detection which utilize data mining techniques. From the perspective of the Axelsson's taxonomy [12], my research belongs to group of programmable systems which require ground truth provided by the user or by another resource of expert knowledge. Although, in general terms of data mining, this approach is falling into supervised machine learning.

Afterwards, I presented network anomaly intrusion detection system called Automated Intrusion Detection System (AIPS), which is aimed at the detection of intrusions caused by successful execution of network buffer overflow attacks and was designed at FIT BUT. The detection of AIPS is based on extraction of Advanced Security Network Metrics (ASNM features) which describe statistical and behavioral characteristics of a network connection, and then are employed as the input for the supervised machine learning classifiers.

The detection capability of ASNM features was firstly evaluated on the dataset collected in laboratory conditions, which contained only small number of samples. Later, I evaluated the performance of ASNM on CDX 2009 dataset [164], and also compared the detection properties of ASNM with the performance of the state-of-the-art feature set designed by Andrew Moore [119] (also referred to as discriminators). The results of the experiments from both feature sets achieved promising results, offering high detection capability of network buffer overflow attacks, while keeping low false positive rate.

9.1 Evading the Detection by ASNM

Because ASNM is primarily based on behavioral and statistical analysis which often use time and index slope of a connection and analysis of payload size distribution, there can arise the question of breaching detection capability of ASNM. The most of the ASNM detection features use information gathered from L3 and L4 layers of packet headers. If such information would be intentionally modified, then it might influence the detection capability of ASNM in negative way. On the other hand, if such modifications would be included into the training process of a classifier, then they could help in further detection of similar modified intrusions evading ADS detection utilizing ASNM features.

9.2 Performance Improvement of ASNM Intrusion Detection

The principal technique which was proposed to be used as the performance improvement of ASNM is the previous idea of evading the detection capability of a classifier. I proposed to utilize two categories of obfuscation techniques aimed at the evasion of the detection by the ASNM features:

- The first category is referred to as *tunneling obfuscation* and resides in tunneling TCP communications in HTTP and HTTPS protocols. Achieved results showed that the tunneling obfuscation can evade the detection by ASNM with Naive Bayes classifier in 64.4% cases of all tunneled attack instances, however if the classifier had previous information about tunneled attacks, then it could successfully detect 98.9% of obfuscated attacks, while keeping low false positive rate (0.56%).

- The second category is referred to as *non-payload-based obfuscations* and aims on modification of various statistical and behavioral properties of network traffic. At the lower level, this category of obfuscation techniques is based on altering of several properties at L3 and L4 layers of TCP/IP reference network model. Proposed examples include the following ideas: spreading out packets in time, fragmentation of IP packets, changing of packet order, simulation of unreliable network channel, packet loss, duplication of packets as well as their combinations. The results of the experiments showed that non-payload-based obfuscations can evade the detection by ASNM and Naive Bayes classifier in 28.8% cases of all obfuscated attack instances. If the classifier was trained with knowledge about some of obfuscated attacks, then it could successfully detect the majority of obfuscated attacks (99.4%), while keeping low false positive rate (0.12%).

9.3 Summary of Research Contributions

Although, I can mention many small contributions of the current work, primarily I want to emphasize 5 main research contributions and draw attention to them:

- I formally defined the network feature set (ASNM) capable of remote anomaly intrusion detection, whose extracted values for particular TCP communications were utilized as the input of supervised machine learning classifiers. Also, the extraction process of the ASNM features was formally defined together with the context analysis of a TCP communication (see Section 5.2).

- I evaluated the performance of ASNM on three supervised classifiers (Naive Bayes, Support Vector Machines and Decision Tree), and mutually compared their detection capabilities at each stage of my research (see Section 6.3.1, 7.6.5 and 8.6.6).

- The performance of the ASNM features was compared to the performance of the state-of-the-art discriminator features using CDX 2009 dataset [164] (see Section 6.3).

- I formally defined two categories of obfuscation techniques primarily aimed at evading the detection by the ASNM features. These obfuscation techniques were employed as the training data driven performance improvement of the ASNM's detection capability (see Chapter 7 and 8).

- I revealed an alternative view on the non-payload-based obfuscation techniques, and demonstrated how they can be employed as a training data driven approximation of network traffic normalizer (see Section 8.9.1).

List of Publications

Journals Ranked by SCImago[1]

- HOMOLIAK Ivan, BREITENBACHER Dominik and HANÁČEK Petr.: Convergence Optimization of Backpropagation Artificial Neural Network Used for Dichotomous Classification of Intrusion Detection Dataset. *Journal of Computers*. Chengdu, Sichuan: 2017, 12(2), pp. 143–155. ISSN 1796-203X, SJR = 0.172.

  `Contribution`: Main idea, state-of-the-art description, experiments and evaluation, presentation of the paper. Overall engagement 90%.

Other Journals

- BARABAS Maroš, HOMOLIAK Ivan, DROZD Michal and HANÁČEK Petr.: Automated Malware Detection Based on Novel Network Behavioral Signatures. *International Journal of Engineering and Technology*. Singapore: International Association of Computer Science and Information Technology, 2013, 5(2), pp. 249–253. ISSN 1793-8236.

  `Contribution`: Data mining experiments and evaluation of the AIPS framework, concluding remarks. Overall engagement 35%.

Book Chapters

- HOMOLIAK Ivan, BARABAS Maroš, CHMELAŘ Petr, DROZD Michal and HANÁČEK Petr.: Advanced Security Network Metrics. Emerging Trends in ICT Security. Waltham: Elsevier Science, 2013, pp. 187–201. ISBN 978-0-12-411474-6.

  `Contribution`: Formal description of ASNM features and context analysis, experiments and evaluation. Overall engagement 65%.

Conference Proceedings

- HOMOLIAK Ivan, OVŠONKA Daniel, GRÉGR Matěj and HANÁČEK Petr.: NBA of Obfuscated Network Vulnerabilities' Exploitation Hidden into HTTPS Traffic. In: *Proceedings of International Conference for Internet Technology and Secured Transactions*(ICITST-2014). London: IEEE Computer Society, 2014, pp. 311–318. ISBN 978-1-908320-40-7.

  `Contribution`: Main idea of using tunneling obfuscation for NBAD, state-of-the-art, idea of using testing scenarios, data processing and analysis followed by data mining experiments, presentation at conference. Overall engagement 45%.

[1] SJR – SCImago Journal & Country Rank. URL: http://www.scimagojr.com

- HOMOLIAK Ivan, OVŠONKA Daniel, KORANDA Karel and HANÁČEK Petr.: Characteristics of Buffer Overflow Attacks Tunneled in HTTP Traffic. In: *International Carnahan Conference on Security Technology*. Rome: IEEE Computer Society, 2014, pp. 188–193. ISBN 978-1-4799-3531-4.

  `Contribution`: Main idea that compares characteristics of tunneled malicious traffic with directly simulated one as well as comparing them with legitimate traffic, state-of-the-art, data processing and analysis, plotting and description of characteristics, presentation at conference. Overall engagement 45%.

- HOMOLIAK Ivan, BARABAS Maroš, CHMELAŘ Petr, DROZD Michal and HANÁČEK Petr.: ASNM: Advanced Security Network Metrics for Attack Vector Description. In: *Proceedings of the 2013 International Conference on Security & Management*. Las Vegas: Computer Science Research, Education, and Applications Press, 2013, pp. 350–358. ISBN 1-60132-259-3.

  `Contribution`: Formal description of ASNM features and context analysis, experiments and evaluation, presentation at conference. Overall engagement 65%.

- BARABAS Maroš, HANÁČEK Petr, HOMOLIAK Ivan and KAČIC Matej.: Detection of Network Buffer Overflow Attacks: A Case Study. In: *The 47th Annual International Carnahan Conference on Security Technology*. Mendellín: Institute of Electrical and Electronics Engineers, 2013, pp. 128–131. ISBN 978-958-8790-65-7.

  `Contribution`: Description of ASNM metrics, some data mining experiments, presentation at conference. Overall engagement 10%.

Other Publications

- HOMOLIAK Ivan. Increasing Classification Accuracy in LibSVM Using String Kernel Functions. In: *Proceedings of the 18th Conference STUDENT EEICT*. Brno: Faculty of Electrical Engineering and Communication BUT, 2012, pp. 281–283. ISBN 978-80-214-4461-4.

  `Contribution`: Main idea of using binary operation for kernel functions, experiments and evaluation, presentation at conference. Overall engagement 100%.

- BREITENBACHER Dominik, HOMOLIAK Ivan and HANÁČEK Petr.: Parallelized Self-Initializing Quadratic Sieve using OpenMP. In: *Santa's Crypto Get-Together 2015*. Praha: Trusted Network Solutions, a.s., 2015, pp. 39–40. ISBN 978-80-904257-7-4.

  `Contribution`: Supervising the adjacent Master's thesis with feedback and some ideas. Overall engagement 20%.

- SMETKA Tomáš, HOMOLIAK Ivan and HANÁČEK Petr.: Cryptanalysis of Symmetric Encryption Algorithm Using Symbolic Regression and Genetic Programming. In: *Santa's Crypto Get-Together 2015*. Praha: Trusted Network Solutions, a.s., 2015, pp. 43–44. ISBN 978-80-904257-7-4.

  `Contribution`: Supervising the adjacent Master's thesis with feedback and some ideas. Overall engagement 10%.

- KAČIC Matej, HANÁČEK Petr, HENZL Martin and HOMOLIAK Ivan.: A Concept of Behavioral Reputation System in Wireless Networks. In: *The 47th Annual International Carnahan Conference on Security Technology*. Medellín: Institute of Electrical and Electronics Engineers, 2013, pp. 86–90. ISBN 978-958-8790-65-7.

  `Contribution`: Presentation at conference. Overall engagement 1%.

- BREITENBACHER Dominik, HOMOLIAK Ivan, JAROŠ Jiří and HANÁČEK Petr: Impact of Optimization and Parallelism on Factorization Speed of SIQS. In *Proceedings of The 20th World Multi-Conference on Systemics, Cybernetics and Informatics, WMSCI'16*, Orlando, United States, 2016, pp. 55–62. ISBN 978-1-941763-41-4.

  `Contribution`: Supervising the adjacent Master's thesis with feedback and some ideas, preparation of the paper, partial presentation at conference. Overall engagement 25%.

Accepted Publications

- HOMOLIAK Ivan, ŠULÁK Ladislav and HANÁČEK Petr: Features for Behavioral Anomaly Detection of Connectionless Network Buffer Overflow Attacks. In *Proceedings of The 17th Workshop on Information Security Applications, WISA'16*, Jeju Island, South Korea, 2016.

  `Contribution`: Main idea of designing connection-less traffic features for detection of buffer overflow attacks, data mining experiments, presentation at conference. Overall engagement 65%.

Bibliography

[1] Aissa, N. B.; Guerroumi, M.: Semi-supervised Statistical Approach for Network Anomaly Detection. *Procedia Computer Science*, vol. 83, 2016, pp. 1090–1095.

[2] Aizerman, M.; Braverman, E.; Rozonoer, L.: The Probability Problem of Pattern Recognition Learning and the Method of Potential Functions. *Automation and Remote Control*, vol. 25, 1965, pp. 1175–1190.

[3] Akiyama, M.; Iwamura, M.; Kawakoya, Y.: Design and Implementation of High Interaction Client Honeypot for Drive-by-download Attacks. *IEICE Transactions on Communications*, vol. 93, no. 5, 2010, pp. 1131–1139.

[4] Akiyama, M.; Kamizono, M.; Matsuki, T.; et al.: Datasets for Anti-Malware Research – MWS Datasets 2014. Technical report, IPSJ SIG, 2014.

[5] Akiyama, M.; Takeshi, Y.; Kadobayashi, Y.; et al.: Client Honeypot Multiplication with High Performance and Precise Detection. *IEICE Transactions on Information and Systems*, vol. 98, no. 4, 2015, pp. 775–787.

[6] Allwein, E. L.; Schapire, R. E.; Singer, Y.: Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, vol. 1, no. Dec, 2000, pp. 113–141.

[7] Alpaydin, E.: *Introduction to Machine Learning*. MIT press, second edn., 2014, ISBN 978-0-262-01243-0.

[8] Anagnostakis, K. G.; Sidiroglou, S.; Akritidis, P.; et al.: Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA*, 2005.

[9] Aoki, K.; Yagi, T.; Iwamura, M.; et al.: Controlling Malware HTTP Communications in Dynamic Analysis System Using Search Engine. In *Proceedings of the 3rd International Workshop on Cyberspace Safety and Security*, IEEE, 2011, pp. 1–6.

[10] Ashula. [Online], Cited 2016-03-08. Available at `<https://sites.google.com/a/secure-ware.com/securewareproducts/ashula>`

[11] Auld, T.; Moore, A. W.; Gull, S. F.: Bayesian Neural Networks for Internet Traffic Classification. *IEEE Transactions on Neural Networks*, vol. 18, no. 1, 2007, pp. 223–239.

[12] Axelsson, S.: Intrusion Detection Systems: A Survey and Taxonomy. Technical report, Chalmers University of Technology, Department of Computer Engineering, 2000.

[13] Bar-Yanai, R.; Langberg, M.; Peleg, D.; et al.: Realtime Classification for Encrypted Traffic. In *Experimental Algorithms*, Springer, 2010, pp. 373–385.

[14] Barabas, M.; Drozd, M.; Hanáček, P.: Behavioral Signature Generation Using Shadow Honeypot. *World Academy of Science, Engineering and Technology*, vol. 6, no. 5, 2012, pp. 829–833, ISSN 2010-376X.

[15] Barabas, M.; Homoliak, I.; Drozd, M.; et al.: Automated Malware Detection Based on Novel Network Behavioral Signatures. *International Journal of Engineering and Technology*, vol. 5, no. 2, 2013, pp. 249–253.

[16] Barbara, D.; Couto, J.; Jajodia, S.; et al.: ADAM: Detecting Intrusions by Data Mining. In *Proceedings of the IEEE Workshop on Information Assurance and Security*, Citeseer, 2001, pp. 11–16.

[17] Bishop, C.: *Pattern Recognition and Machine Learning*. Information Science and Statistics, Springer, 2006, ISBN 9780387310732.

[18] Bland, J. M.; Altman, D. G.: Statistics notes: measurement error. *British Medical Journal*, vol. 313, no. 7059, 1996, p. 744.

[19] Boltz, M.; Jalava, M.; Walsh, J.: New Methods and Combinatorics for Bypassing Intrusion Prevention Technologies. Technical report, Stonesoft, 2010.

[20] Bolzoni, D.; Zambon, E.; Etalle, S.; et al.: Poseidon: A 2-tier Anomaly-based Intrusion Detection System. In *Proceedings of the 4th IEEE International Workshop on Information Assurance, IWIA'06, Egham, Surrey, UK*, 2006, pp. 144–156.

[21] Borders, K.; Prakash, A.: Web Tap: Detecting Covert Web Traffic. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS'04*, ACM, 2004, pp. 110–120.

[22] Boser, B. E.; Guyon, I. M.; Vapnik, V. N.: A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, ACM, 1992, pp. 144–152.

[23] Breiman, L.; Friedman, J.; Stone, C. J.; et al.: *Classification and Regression Trees*. CRC press, 1984.

[24] Breiman, L.; Ihaka, R.: *Nonlinear Discriminant Analysis via Scaling and ACE*. Department of Statistics, University of California, 1984.

[25] Writing buffer overflow exploits – A tutorial for beginners. [Online], Cited 2014-12-01. Available at <http://www.eecis.udel.edu/bmiller/cis459/2007s/readings/buff-overflow.html>

[26] Büschkes, R.; Borning, M.; Kesdogan, D.: Transaction-based Anomaly Detection. In *Workshop on Intrusion Detection and Network Monitoring*, 1999, pp. 129–140.

[27] CAIDA: the Cooperative Association for Internet Data Analysis. [Online], Cited 2016-03-05. Available at <http://www.caida.org/>

[28] The UCSD CAIDA Backscatter dataset. [Online], Cited 2016-03-05. Available at <http://www.caida.org/data/passive/backscatter_dataset.xml>

[29] The CAIDA UCSD network Telescope "Three Days Of Conficker". [Online], Cited 2016-03-05. Available at <http://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml>

[30] The UCSD CAIDA Dataset on the Code-Red Worms. [Online], Cited 2016-03-05. Available at <http://www.caida.org/data/passive/codered_worms_dataset.xml>

[31] The CAIDA UCSD "DDoS Attack 2007" dataset. [Online], Cited 2016-03-05. Available at <http://www.caida.org/data/passive/ddos-20070804_dataset.xml>

[32] The CAIDA UCSD dataset on the Witty worm. [Online], Cited 2016-03-05. Available at <http://www.caida.org/data/passive/witty_worm_dataset.xml>

[33] Cheng, T.; Lin, Y.; Lai, Y.; et al.: Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems. *IEEE Communications Surveys and Tutorials*, vol. 14, no. 4, 2012, pp. 1011–1020.

[34] Claise, B.: Cisco systems NetFlow services export version 9. [Online], Cited 2016-04-04. Available at <http://tools.ietf.org/html/rfc3954.html>

[35] ClamAV: Open source antivirus engine for detecting trojans, viruses, malware & other malicious threats. [Online], Cited 2016-03-08. Available at <http://www.clamav.net>

[36] Cohen, B.: The BitTorrent protocol specification. 2008.

[37] Cohen, W. W.: Fast Effective Rule Induction. In *Proceedings of the 12th International Conference on Machine Learning*, 1995, pp. 115–123.

[38] Combs, G.; et al.: Wireshark – network protocol analyzer. [Online], Cited 2014-12-01. Available at <www.wireshark.org>

[39] Contagio malware dump: Collection of PCAP files from malware analysis. [Online], Cited 2016-03-07. Available at <http://contagiodump.blogspot.cz/2013/04/collection-of-pcap-files-from-malware.html>

[40] Cooley, J. W.; Tukey, J. W.: An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, vol. 19, no. 90, 1965, pp. 297–301.

[41] Cowan, C.; Pu, C.; Maier, D.; et al.: StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *Proceedings of USENIX Security Symposium*, vol. 98, 1998, pp. 63–78.

[42] Cowan, C.; Wagle, P.; Pu, C.; et al.: Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *Proceedings of DARPA Information Survivability Conference and Exposition, DISCEX'00*, vol. 2, IEEE, 2000, pp. 119–129.

[43] Crotti, M.; Dusi, M.; Gringoli, F.; et al.: Detecting HTTP Tunnels with Statistical Mechanisms. In *Proceedings of IEEE International Conference on Communications, ICC'07*, IEEE, 2007, pp. 6162–6168.

[44] CVE-2002-0082: Buffer overflow vulnerability of mod_ssl and Apache-SSL. [Online], Cited 2014-12-01. Available at <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2002-0082>

[45] CVE-2003-0201: Buffer overflow in the Samba service. [Online], Cited 2014-12-01. Available at <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-0201>

[46] CVE-2003-0352: Buffer overflow in DCOM interface for RPC in MS Windows NT 4.0, 2000, XP and Server 2003. [Online], Cited 2014-12-01. Available at <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-0352>

[47] CVE-2007-6377: Stack-based buffer overflow vulnerability in BadBlue. [Online], Cited 2014-12-01. Available at <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-6377>

[48] Daemon9; Alhambra: Project Loki: ICMP Tunnelling. *Magazine, Phrack*, vol. 7, no. 49, 1997, pp. 1–9.

[49] Dainotti, A.; King, A.; Papale, F.; et al.: Analysis of a/0 Stealth Scan from a Botnet. In *Proceedings of the ACM Internet Measurement Conference, IMC'12*, ACM, 2012, pp. 1–14.

[50] DARPA Intrusion Detection Evaluation. [Online], Cited 2014-01-13. Available at <http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/>

[51] Denning, D. E.; Neumann, P. G.: Requirements and Model for IDES – a Real-time Intrusion Detection Expert System. Technical report, SRI International, Computer Science Laboratory, 1985.

[52] Devroye, L.; Györfi, L.; Lugosi, G.: *A Probabilistic Theory of Pattern Recognition*, vol. 31. Springer New York, 1996.

[53] Dharmapurikar, S.; Paxson, V.: Robust TCP Stream Reassembly in the Presence of Adversaries. In *Proceedings of the 14th USENIX Security Symposium*, 2005, pp. 65–80.

[54] Dietterich, T. G.; Bakiri, G.: Solving Multiclass Learning Problems via Error-correcting Output Codes. *Journal of Artificial Intelligence Research*, vol. 2, no. 263, 1995, pp. 263–286.

[55] Dionaea – A malware capturing honeypot. [Online], Cited 2016-03-07. Available at <http://www.edgis-security.org/honeypot/dionaea/>

[56] Dusi, M.; Crotti, M.; Gringoli, F.; et al.: Tunnel Hunter: Detecting Application-layer Tunnels with Statistical Fingerprinting. *Computer Networks*, vol. 53, no. 1, 2009, pp. 81–97.

[57] Dyatlov, A.: Firepass – A tunneling tool. [Online], Cited 2015-03-15. Available at <http://gray-world.net/pr_firepass.shtml>

[58] Fajmon, B.; Růžičková, I.: Matematika 3. [Online], Cited 2012-01-12, in Czech language. Available at <http://www.umat.feec.vutbr.cz/~novakm/matematika3.pdf>

[59] FFR Yarai analyzer professional. [Online], Cited 2016-03-07, in Japanese language. Available at <www.ffri.jp/products/yarai_analyzer_pro/>

[60] Fogla, P.; Lee, W.: Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ACM, 2006, pp. 59–68.

[61] Fogla, P.; Sharif, M. I.; Perdisci, R.; et al.: Polymorphic Blending Attacks. In *Proceedings of the 15th USENIX Security Symposium*, 2006, pp. 241–256.

[62] Grossman, R.; Kasif, S.; Moore, R.; et al.: Data Mining Research: Opportunities and Challenges. In *Report of Three NSF Workshops on Mining Large, Massive, and Distributed Data*, 1999.

[63] Guarnieri, C.; Tanasi, A.; Bremer, J.; et al.: The Cuckoo sandbox. [Online], Cited 2016-03-07. Available at `<http://www.cuckoosandbox.org>`

[64] Habra, N.; Charlier, B. L.; Mounji, A.; et al.: ASAX: Software Architecture and Rule-based Language for Universal Audit Trail Analysis. In *Proceedings of European Symposium on Research in Computer Security*, *Lecture Notes in Computer Science*, vol. 648, edited by Y. Deswarte; G. Eizenberg; J.-J. Quisquater, Springer, 1992, ISBN 3-540-56246-X, pp. 435–450.

[65] Hand, D. J.; Yu, K.: Idiot's Bayes – Not So Stupid after All? *International Statistical Review*, vol. 69, no. 3, 2001, pp. 385–398.

[66] Handley, M.; Paxson, V.; Kreibich, C.: Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proceedings of the 10th USENIX Security Symposium*, 2001, pp. 115–131.

[67] Hao, S.; Hu, J.; Liu, S.; et al.: Network Traffic Classification Based on Improved DAG-SVM. In *Proceedings of International Conference on Communications, Management and Telecommunications, ComManTel'15*, IEEE, 2015, pp. 256–261.

[68] Hatada, M.; Akiyama, M.; Matsuki, T.; et al.: Empowering Anti-malware Research in Japan by Sharing the MWS Datasets. *Journal of Information Processing*, vol. 23, no. 5, 2015, pp. 579–588.

[69] Hatada, M.; Nakatsuru, I.; Akiyama, M.: Datasets for Anti-malware Research – MWS 2011 Datases. *IPSJ Malware Workshop, MWS'11*, 2011, in Japanese language.

[70] Hatada, M.; Nakatsuru, Y.; Akiyama, M.; et al.: Datasets for Anti-malware Research – MWS 2010 Datasets. In *IPSJ Malware Workshop, MWS'10*, 2010, pp. 1–5.

[71] Hatada, M.; Nakatsuru, Y.; Terada, M.; et al.: Dataset for Anti-malware Research and Research Achievements Shared at the Workshop. In *Proceedings of the Computer Security Symposium*, 2009, pp. 1–8.

[72] Heideman, M.; Johnson, D.; Burrus, C.: Gauss and the History of the Fast Fourier Transform. *IEEE ASSP Magazine*, vol. 1, no. 4, 1984, pp. 14–21.

[73] Hemminger, S.; et al.: Network Emulation with NetEm. In *Proceedings of Australia's 6th National Linux Conference*, Citeseer, 2005, pp. 18–23.

[74] Hjelmvik, E.; John, W.: Breaking and Improving Protocol Obfuscation. Technical Report 05, Department of Computer Science and Engineering, Chalmers University of Technology, 2010.

[75] Homoliak, I.: *Metrics for Intrusion Detection in Network Traffic*. Master's thesis, University of Technology Brno, Faculty of Information Technology, 2012, in Slovak language. Available at <http://www.fit.vutbr.cz/study/DP/DP.php.en?id=13755&y=2013>

[76] Homoliak, I.; Barabas, M.; Chmelar, P.; et al.: ASNM: Advanced Security Network Metrics for Attack Vector Description. In *Proceedings of the 2013 International Conference on Security & Management, SAM'13*, Computer Science Research, Education, and Applications Press, 2013, ISBN 1-60132-259-3, pp. 350–358.

[77] Homoliak, I.; Ovšonka, D.; Grégr, M.; et al.: NBA of Obfuscated Network Vulnerabilities' Exploitation Hidden into HTTPS Traffic. In *9th International Conference for Internet Technology and Secured Transactions, ICITST'14*, IEEE, 2014, ISBN 978-1-908320-40-7, pp. 311–318.

[78] Homoliak, I.; Ovšonka, D.; Koranda, K.; et al.: Characteristics of Buffer Overflow Attacks Tunneled in HTTP Traffic. In *Proceedings of the 47th Annual International Carnahan Conference on Security Technology, ICCST'14*, IEEE, 2014, pp. 188–193.

[79] Ibrahim, L. M.; Basheer, D. T.; Mahmod, M. S.: A Comparison Study for Intrusion Database (KDD'99, NSL-KDD) Based on Self Organization Map (SOM) Artificial Neural Network. *Journal of Engineering Science and Technology*, vol. 8, no. 1, 2013, pp. 107–119.

[80] Ifconfig(8) – Linux manual page. [Online], Cited 2015-07-26. Available at <http://man7.org/linux/man-pages/man8/ifconfig.8.html>

[81] Inoue, D.; Eto, M.; Yoshioka, K.; et al.: Nicter: An Incident Analysis System Toward Binding Network Monitoring with Malware Analysis. In *Workshop on Information Security Threats Data Collection and Sharing, WISTDCS'08*, IEEE, 2008, pp. 58–66.

[82] Insecure.Org: How to write buffer overflows. [Online], Cited 2014-12-01. Available at <http://insecure.org/stf/mudge_buffer_overflow_tutorial.html>

[83] RFC 791 – Internet Protocol. [Online], Cited 2016-06-20. Available at <https://www.ietf.org/rfc/rfc791.txt>

[84] Ji, J.; Peterson, G.; Grimaila, M.; et al.: Holistic Network Defense: Fusing Host and Network Features for Attack Classification. Technical report, Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2013.

[85] John, G. H.; Langley, P.: Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[86] Juan, L.; Kreibich, C.; Lin, C.-H.; et al.: A Tool for Offline and Live Testing of Evasion Resilience in Network Intrusion Detection Systems. In *5th International Conference Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA'08*, Springer, 2008, pp. 267–278.

[87] Kamizono, M.; Hatada, M.; Terada, M.; et al.: Datasets for Anti-Malware Research – MWS Datasets 2013. In *Proceedings of IPSJ Computer Security Symposium*, 2013, pp. 1–8.

[88] Karthick, R. R.; Hattiwale, V. P.; Ravindran, B.: Adaptive Network Intrusion Detection System Using a Hybrid Approach. In *Proceedings of the 4th International Conference on Communication Systems and Networks, COMSNETS'12*, IEEE, 2012, pp. 1–7.

[89] Kerrisk, M.: Linux manual page: tc. [Online], Cited 2015-03-15. Available at <http://man7.org/linux/man-pages/man8/tc.8.html>

[90] Kevric, J.; Jukic, S.; Subasi, A.: An Effective Combining of Classifier Approach Using Tree Algorithms for Network Intrusion Detection. *Neural Computing and Applications*, 2016, pp. 1–8.

[91] Ko, C.; Ruschitzka, M.; Levitt, K.: Execution Monitoring of Security-critical Programs in Distributed Systems: A Specification-based Approach. In *Proceedings of IEEE Symposium on Security and Privacy, S&P'97*, IEEE, 1997, pp. 175–187.

[92] Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.

[93] Kruegel, C.; Mutz, D.; Robertson, W.; et al.: Bayesian Event Classification for Intrusion Detection. In *Proceedings of 19th Annual Computer Security Applications Conference, ACSAC'03*, IEEE, 2003, pp. 14–23.

[94] Kyoto 2006+ Dataset. [Online], Cited 2016-03-08. Available at <http://www.takakura.com/Kyoto_data/>

[95] Lazarevic, A.; Ertöz, L.; Kumar, V.; et al.: A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In *Proceedings of the 3rd SIAM International Conference on Data Mining*, SIAM, 2003, pp. 25–36.

[96] LeBoutillier, P.: HTTunnel. [Online], Cited 2015-03-15. Available at <http://sourceforge.net/projects/httunnel/>

[97] Lee, W.; Stolfo, S. J.: A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security, TiSSEC*, vol. 3, no. 4, 2000, pp. 227–261.

[98] Lee, W.; Stolfo, S. J.; Mok, K. W.: A Data Mining Framework for Building Intrusion Detection Models. In *Proceedings of the IEEE Symposium on Security and Privacy, S&P'99*, IEEE, 1999, pp. 120–132.

[99] Lee, W.; Stolfo, S. J.; et al.: Data Mining Approaches for Intrusion Detection. In *Proceedings of USENIX Security Symposium*, 1998.

[100] Lee, Y.; Lin, Y.; Wahba, G.: Multicategory Support Vector Machines: Theory and Application to the Classification of Microarray Data and Satellite Radiance Data. *Journal of the American Statistical Association*, vol. 99, no. 465, 2004, pp. 67–81.

[101] Lemonnier, E.: Protocol Anomaly Detection in Network-based IDSs. Technical report, Defcom, 2001.

[102] Li, W.: Using Genetic Algorithm for Network Intrusion Detection. *Proceedings of the United States Department of Energy Cyber Security Group*, vol. 1, 2004, pp. 1–8.

[103] Lim, S. Y.; Jones, A.: Network Anomaly Detection System: The State of Art of Network Behaviour Analysis. In *Proceedings of International Conference on Convergence and Hybrid Information Technology, ICHIT'08*, IEEE, 2008, pp. 459–465.

[104] Liskov, B.; Snyder, A.: Exception Handling in CLU. *IEEE Transactions on Software Engineering*, vol. 5, no. 6, 1979, pp. 546–558.

[105] Liu, G.; Yi, Z.; Yang, S.: A Hierarchical Intrusion Detection Model Based on the PCA Neural Networks. *Neurocomputing*, vol. 70, no. 7, 2007, pp. 1561–1568.

[106] Liu, S.; Hu, J.; Hao, S.; et al.: Improved EM method for Internet Traffic Classification. In *Proceedings of the 8th International Conference on Knowledge and Smart Technology, KST'16*, IEEE, 2016, pp. 13–17.

[107] Lundström, M.: MailTunnel. [Online], Cited 2015-03-15. Available at `<http://gray-world.net/tools/mailtunnel-0.2.tar.gz>`

[108] Mafra, P. M.; Moll, V.; da Silva Fraga, J.; et al.: Octopus-IIDS: An Anomaly Based Intelligent Intrusion Detection System. In *Proceedings of Symposium on Computers and Communications, ISCC'10*, IEEE, 2010, pp. 405–410.

[109] Mann, P. S.: *Introductory Statistics*. John Wiley & Sons, 7 edn., 2010, ISBN 978-0-470-44466-5.

[110] Massicotte, F.; Gagnon, F.; Labiche, Y.; et al.: Automatic Evaluation of Intrusion Detection Systems. In *Proceedings of the 22nd Annual Computer Security Applications Conference, ACSAC'06*, IEEE, 2006, pp. 361–370.

[111] Mathworld: Markov process. [Online], Cited 2013-12-24. Available at `<http://mathworld.wolfram.com/MarkovProcess.html>`

[112] McGregor, A.; Hall, M.; Lorier, P.; et al.: Flow Clustering Using Machine Learning Techniques. In *International Workshop on Passive and Active Network Measurement*, edited by C. Barakat; I. Pratt, Springer, 2004, ISBN 3-540-21492-5, pp. 205–214.

[113] Metasploit project. [Online], Cited 2014-12-01. Available at `<http://www.metasploit.org>`

[114] Miwa, S.; Miyachi, T.; Eto, M.; et al.: Design and Implementation of an Isolated Sandbox with Mimetic Internet Used to Analyze Malwares. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test*, USENIX Association, 2007, pp. 6–6.

[115] Mohajeri Moghaddam, H.; Li, B.; Derakhshani, M.; et al.: SkypeMorph: Protocol Obfuscation for Tor Bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ACM, 2012, pp. 97–108.

[116] Moore, A.; Hall, J.; Kreibich, C.; et al.: Architecture of a Network Monitor. In *Proceedings of the Passive & Active Measurement Workshop, PAM'03*, 2003.

[117] Moore, A. W.; Papagiannaki, K.: Toward the Accurate Identification of Network Applications. In *Proceedings of International Workshop on Passive and Active Network Measurement*, *Lecture Notes in Computer Science*, vol. 3431, edited by C. Dovrolis, Springer, 2005, ISBN 3-540-25520-6, pp. 41–54.

[118] Moore, A. W.; Zuev, D.: Internet Traffic Classification Using Bayesian Analysis Techniques. In *Proceedings of ACM SIGMETRICS*, vol. 33, ACM, 2005, pp. 50–60.

[119] Moore, A. W.; Zuev, D.; Crogan, M.: Discriminators for Use in Flow-based Classification. Technical report, Intel Research, Cambridge, 2005.

[120] Murata, T.: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, vol. 77, no. 4, 1989, pp. 541–580.

[121] Netresec – publicly available PCAP files. [Online], Cited 2016-03-06. Available at <http://www.netresec.com/?page=PcapFiles>

[122] Newsome, J.; Song, D.: Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the Network and Distributed System Security Symposium, NDSS'05, San Diego, California, USA*, Internet Society, 2005, ISBN 1-891562-20-7.

[123] National Institutes of Standards and Technology: CVE identifiers. [Online], Cited 2015-03-15. Available at <http://cve.mitre.org/cve/identifiers/index.html>

[124] National Institutes of Standards and Technology: Common Vulnerability Scoring System. [Online], Cited 2015-03-17. Available at <https://nvd.nist.gov/cvss.cfm>

[125] National Institutes of Standards and Technology: National Vulnerability Database. [Online], Cited 2015-03-15. Available at <https://web.nvd.nist.gov/>

[126] One, A.: Smashing the Stack for Fun and Profit. *Phrack Magazine*, vol. 7, no. 49, 1996, pp. 14–16.

[127] Oracle: VBoxManage. [Online], Cited 2015-03-15. Available at <http://www.virtualbox.org/manual/ch08.html>

[128] Oracle: VirtualBox. [Online], Cited 2015-03-15. Available at <https://www.virtualbox.org/>

[129] Ovšonka, D.: *Network Traffic Obfuscation for IDS Detection Avoidance*. Master's thesis, University of Technology Brno, Faculty of Information Technology, 2013, in Slovak language.

[130] Pack, D. J.; Streilein, W.; Webster, S.; et al.: Detecting HTTP tunneling activities. In *Proceedings Of the IEEE Workshop On Information Assurance, IWIA'02*, Citeseer, 2002.

[131] Padgett, P.: Corkscrew. [Online], Cited 2015-03-15. Available at <http://www.agroman.net/corkscrew/>

[132] Papadogiannakis, A.; Polychronakis, M.; Markatos, E. P.: Tolerating Overload Attacks Against Packet Capturing Systems. In *Proceedings of USENIX Annual Technical Conference*, 2012, pp. 197–202.

[133] Paxson, V.: BRO: A System for Detecting Network Intruders in Real-time. *Computer Networks*, vol. 31, no. 23, 1999, pp. 2435–2463.

[134] Peddabachigari, S.; Abraham, A.; Thomas, J.: Intrusion Detection Systems Using Decision Trees and Support Vector Machines. *International Journal of Applied Science and Computations*, vol. 11, no. 3, 2004, pp. 118–134.

[135] Platt, J.; Cristianini, N.; Shawe, J.: Large Margin Dags for Multiclass Classification. *Advances in Neural Information Processing Systems, Cambridge*, vol. 12, 2000, pp. 547–553.

[136] Portnoy, L.; Eskin, E.; Stolfo, S.: Intrusion Detection with Unlabeled Data Using Clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security, DMSA'01*, Citeseer, 2001, pp. 5–8.

[137] PREDICT dataset: Protected Repository for the Defense of Infrastructure against Cyber Threats. [Online], Cited 2016-03-07. Available at <https://www.predict.org>

[138] Provos, N.; Holz, T.: *Virtual Honeypots – From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 2008, ISBN 978-0-321-33632-3.

[139] Ptacek, T. H.; Newsham, T. N.: Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[140] Puppy, R. F.: A look at Whisker's Anti-IDS Tactics. [Online], Cited 2015-03-17. Available at <http://www.ussrback.com/docs/papers/IDS/whiskerids.html>

[141] Quinlan, J. R.: Induction of Decision Trees. *Machine Learning*, vol. 1, no. 1, 1986, pp. 81–106.

[142] Quinlan, J. R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993, ISBN 1-55860-238-0.

[143] Rapid7: Apache Tomcat manager application deployer authenticated code execution. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/exploit/multi/http/tomcat_mgr_deploy>

[144] Rapid7: Badblue 2.72b passthru buffer overflow. [Online], Cited 2015-03-17. Available at <https://www.rapid7.com/db/modules/exploit/windows/http/badblue_passthru>

[145] Rapid7: DistCC daemon command execution. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/exploit/unix/misc/distcc_exec>

[146] Rapid7: Metasploitable – virtual machine to test metasploit. [Online], Cited 2015-03-15. Available at <https://information.rapid7.com/metasploitable-download.html>

[147] Rapid7: Microsoft Server service relative path stack corruption. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/exploit/windows/smb/ms08_067_netapi>

[148] Rapid7: Microsoft SQL server payload execution. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/exploit/windows/mssql/mssql_payload>

[149] Rapid7: MSSQL login utility. [Online], Cited 2015-03-17. Available at <https://www.rapid7.com/db/modules/auxiliary/scanner/mssql/mssql_login>

[150] Rapid7: PostgreSQL for Linux payload execution. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/exploit/linux/postgres/postgres_payload>

[151] Rapid7: PostgreSQL login utility. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/auxiliary/scanner/postgres/postgres_login>

[152] Rapid7: Remotely exploitable buffer overflow in mod_ssl. [Online], Cited 2015-03-17. Available at <https://www.rapid7.com/db/vulnerabilities/HTTP-MODS-0003>

[153] Rapid7: MS03-026 Microsoft RPC DCOM interface overflow. [Online], Cited 2015-03-17. Available at <https://www.rapid7.com/db/modules/exploit/windows/dcerpc/ms03_026_dcom>

[154] Rapid7: Samba trans2open overflow (Linux x86). [Online], Cited 2015-03-17. Available at <https://www.rapid7.com/db/modules/exploit/linux/samba/trans2open>

[155] Rapid7: Samba username map script command execution. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/exploit/multi/samba/usermap_script>

[156] Rapid7: Tomcat application manager login utility. [Online], Cited 2015-03-17. Available at <http://www.rapid7.com/db/modules/auxiliary/scanner/http/tomcat_mgr_login>

[157] RapidMiner: RapidMiner Studio. [Online], Cited 2015-05-20. Available at <https://rapidminer.com/products/studio/>

[158] IBM: RealSecure. [Online], Cited 2016-06-07. Available at <http://www-935.ibm.com/services/th/en/it-services/realsecure-server-sensor.html>

[159] RFC 4193. [Online], Cited 2016-03-10. Available at <http://www.ietf.org/rfc/rfc4193.txt>

[160] Roesch, M.: Snort – Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, USENIX Association, 1999, pp. 229–238.

[161] Rubin, S.; Jha, S.; Miller, B. P.: Automatic Generation and Analysis of NIDS Attacks. In *Proceedings of the 20th Annual Computer Security Applications Conference, ACSAC'04*, IEEE, 2004, pp. 28–38.

[162] Russinovich, M.: Microsoft Sysinternals Suite. [Online], Cited 2015-06-07. Available at <http://technet.microsoft.com/en-us/sysinternals/bb842062.aspx>

[163] Salem, M.; Reissmann, S.; Buehler, U.: Persistent Dataset Generation Using Real-time Operative Framework. In *Proceedings of International Conference on Computing, Networking and Communications, ICNC'14*, IEEE, 2014, pp. 1023–1027.

[164] Sangster, B.; O'Connor, T.; Cook, T.; et al.: Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets. In *Proceedings of the 2nd Workshop on Cyber Security Experimentation and Test, CSET'09*, 2009.

[165] Sekar, R.; Gupta, A.; Frullo, J.; et al.: Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM conference on Computer and Communications Security, CCS'02*, ACM, 2002, pp. 265–274.

[166] Shankar, U.; Paxson, V.: Active Mapping: Resisting NIDS Evasion without Altering Traffic. In *Proceedings of Symposium on Security and Privacy, S&P'03*, IEEE, 2003, pp. 44–61.

[167] Shin, S.; Lee, S.; Kim, H.; et al.: Advanced Probabilistic Approach for Network Intrusion Forecasting and Detection. *Expert Systems with Applications*, vol. 40, no. 1, 2013, pp. 315–322.

[168] Singh, A.; Mora dos Santos, A.; Nordstrom, O.; et al.: Stateless Model for the Prevention of Malicious Communication Channels. *International Journal of Computers and Applications*, vol. 28, no. 3, 2006, pp. 285–297.

[169] Smith, N. P.: Stack Smashing Vulnerabilities in the UNIX Operating System. Technical report, Southern Connecticut State University, 1997.

[170] Sohn, T.; Moon, J.; Lee, S.; et al.: Covert Channel Detection in the ICMP Payload Using Support Vector Machines. In *International Symposium on Computer and Information Sciences, ISCIS'03*, Springer, 2003, pp. 828–835.

[171] Sokolova, M.; Japkowicz, N.; Szpakowicz, S.: Beyond Accuracy, F-score and ROC: A Family of Discriminant Measures for Performance Evaluation. In *AI 2006: Advances in Artificial Intelligence*, Springer, 2006, pp. 1015–1021.

[172] Sommer, R.; Paxson, V.: Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Symposium on Security and Privacy, S&P'10*, IEEE, 2010, pp. 305–316.

[173] Song, J.; Takakura, H.; Okabe, Y.; et al.: Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ACM, 2011, pp. 29–36.

[174] Song, D.: Fragroute. [Online], Cited 2016-03-05. Available at `<http://www.monkey.org/~dugsong/fragroute/>`

[175] Stergiou, C.; Siganos, D.: Neural networks. [Online], Cited 2012-01-01. Available at `<http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html>`

[176] Stødle, D.: Ptunnel – Ping Tunnel. [Online], Cited 2014-02-13. Available at `<http://www.cs.uit.no/daniels/PingTunnel>`

[177] Stolfo, S. J.; Fan, W.; Lee, W.; et al.: Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. In *Proceedings of DARPA Information Survivability Conference and Exposition, DISCEX'00*, vol. 2, IEEE, 2000, pp. 130–144.

[178] Suricata. [Online], Cited 2015-07-17. Available at <http://suricata-ids.org/>

[179] Symantec network security 7100 series. [Online], Cited 2016-03-08. Available at <https://support.symantec.com/en_US/article.TECH112388.html>

[180] Takehisa, T.; Inoue, D.; Eto, M.; et al.: NONSTOP: Secure Remote Analysis Platform for Cybersecurity Information. Technical report, Information and Communication System Security, 2013.

[181] Tavallaee, M.; Bagheri, E.; Lu, W.; et al.: A Detailed Analysis of the KDD Cup 99 Data Set. In *Proceedings of the 2nd IEEE International Conference on Computational Intelligence for Security and Defense Applications*, IEEE Press, 2009, pp. 53–58.

[182] Tavallaee, M.; Bagheri, E.; Lu, W.; et al.: NSL-KDD Dataset. [Online], Cited 2013-07-26. Available at <https://web.archive.org/web/20150205070216/http://nsl.cs.unb.ca/NSL-KDD/>

[183] Tcpdump.org: Tcpdump. [Online], Cited 2015-03-15. Available at <http://www.tcpdump.org/>

[184] Team, T. P.: ASLR: Address Space Layout Randomization. [Online], Cited 2014-06-07. Available at <https://pax.grsecurity.net/docs/aslr.txt>

[185] Teknos, M.: *Extension of Behavioral Analysis of Network Traffic Focusing on Attack Detection*. Master's thesis, University of Technology Brno, Faculty of Information Technology, 2015, in Slovak language.

[186] Thomas, C.; Sharma, V.; Balakrishnan, N.: Usefulness of DARPA Dataset for Intrusion Detection System Evaluation. In *Proceedings of the SPIE Defense and Security Symposium*, vol. 6973, International Society for Optics and Photonics, 2008, pp. 69730G–1.

[187] University of California, I. U.: KDD Cup 99. [Online], Cited 2012-03-01. Available at <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

[188] Vapnik, V. N.: *Statistical Learning Theory*, vol. 1. Wiley New York, 1998.

[189] Vigna, G.; Robertson, W.; Balzarotti, D.: Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS'04*, ACM, 2004, pp. 21–30.

[190] Vijayasarathy, R.; Raghavan, S. V.; Ravindran, B.: A System Approach to Network Modeling for DDoS Detection Using a Naive Bayesian Classifier. In *Proceedings of 3rd International Conference on Communication Systems and Networks, COMSNETS'11*, IEEE, 2011, pp. 1–10.

[191] VirusTotal – virus, malware and URL scanner. [Online], Cited 2016-03-23. Available at <https://www.virustotal.com/>

[192] Wagner, D.; Soto, P.: Mimicry Attacks on Host-based Intrusion Detection Systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS'02*, ACM, 2002, pp. 255–264.

[193] Watson, D.; Smart, M.; Malan, G. R.; et al.: Protocol Scrubbing: Network Security through Transparent Flow Modification. *IEEE/ACM Transactions on Networking, TON'04*, vol. 12, no. 2, 2004, pp. 261–273.

[194] Weisberg, H.: *Central Tendency and Variability*. 83, SAGE Publications, 1992, ISBN 9780803940079.

[195] Weisstein, E. W.: Discrete Fourier transform. [Online], Cited 2016-06-07. Available at <http://mathworld.wolfram.com/DiscreteFourierTransform.html>

[196] Weisstein, E. W.: Fast Fourier transform. [Online], Cited 2016-06-07. Available at <http://mathworld.wolfram.com/FastFourierTransform.html>

[197] Weisstein, E. W.: Mode. [Online], Cited 2016-06-07. Available at <http://mathworld.wolfram.com/Mode.html>

[198] Wu, T. M.: Intrusion Detection Systems. [Online], Cited 2014-01-17. Available at <http://iac.dtic.mil/csiac/download/intrusion_detection.pdf>

[199] Xiang, C.; Lim, S.: Design of Multiple-level Hybrid Classifier for Intrusion Detection System. In *IEEE Workshop on Machine Learning for Signal Processing*, IEEE, 2005, pp. 117–122.

[200] Ye, N.; Chen, Q.; Emran, S. M.; et al.: Chi-square Statistical Profiling for Anomaly Detection. In *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, New York*, 2000, pp. 187–193.

[201] You, I.; Yim, K.: Malware Obfuscation Techniques: A Brief Survey. In *Proceedings of the 5th International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA'10*, IEEE, 2010, ISBN 978-1-4244-8448-5, pp. 297–300.

[202] Zander, S.; Armitage, G. J.; Branch, P.: A Survey of Covert Channels and Countermeasures in Computer Network Protocols. *IEEE Communications Surveys and Tutorials*, vol. 9, no. 1–4, 2007, pp. 44–57.

[203] Zelenchuk, I.: Skeeve – ICMP Bounce Tunnel. [Online], Cited 2016-03-05. Available at <http://www.gray-world.net/poc_skeeve.shtml>

[204] Zhang, X.-S.; Zhi, L.; Chen, D.-P.: A Practical Taint-based Malware Detection. In *International Conference on Apperceiving Computing and Intelligence Analysis, ICACIA'08*, IEEE, 2008, pp. 73–77.

[205] Zheng, S.; Peng, C.; Ying, X.; et al.: A Network State Based Intrusion Detection Model. In *Proceedings of International Conference on Computer Networks and Mobile Computing*, IEEE, 2001, pp. 481–486.

List of Tables

List of Figures

Appendices

List of Appendices

Appendix A

Performance Measures

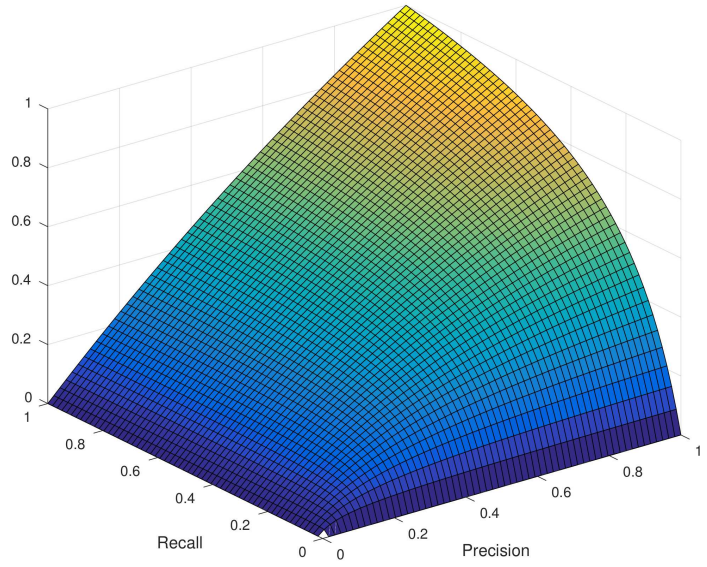

Figure A.1: F-measure for $\beta = 0.5$

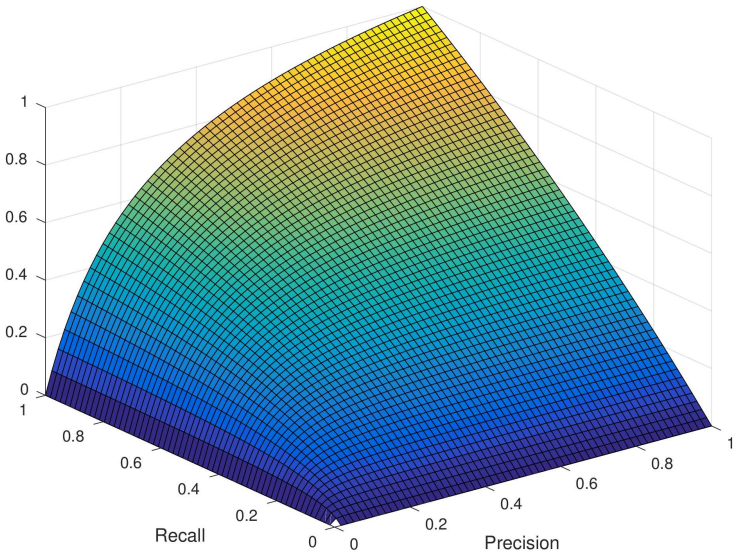

Figure A.2: F-measure for $\beta = 2$

Appendix B

Full List of KDD Cup '99 Features

B.1 Basic Features

Feature Name	Description	type
duration	• length (number of seconds) of the connection	continuous
protocol_type	• type of the protocol, e.g. tcp, udp, etc.	discrete
service	• network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	• number of data bytes from source to destination	continuous
dst_bytes	• number of data bytes from destination to source	continuous
flag	• normal or error status of the connection	discrete
land	• 1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	• number of "wrong" fragments	continuous
urgent	• number of urgent packets	continuous

Table B.1: Basic features of individual TCP connections [187]

B.2 Content Features

Feature Name	Description	Type
hot	• number of "hot" indicators	continuous
num_failed_logins	• number of failed login attempts	continuous
logged_in	• 1 if successfully logged in; 0 otherwise	discrete
num_compromised	• number of "compromised" conditions	continuous
root_shell	• 1 if root shell is obtained; 0 otherwise	discrete
su_attempted	• 1 if "su root" command attempted; 0 otherwise	discrete
num_root	• number of "root" accesses	continuous
num_file_creations	• number of file creation operations	continuous
num_shells	• number of shell prompts	continuous
num_access_files	• number of operations on access control files	continuous
num_outbound_cmds	• number of outbound commands in an ftp session	continuous
is_hot_login	• 1 if the login belongs to the "hot" list; 0 otherwise	discrete
is_guest_login	• 1 if the login is a "guest"login; 0 otherwise	discrete

Table B.2: Content features within a connection suggested by domain knowledge [187]

B.3 Traffic Features

Feature Name	Description	Type
count	• the number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	• % of connections that have "SYN" errors	continuous
rerror_rate	• % of connections that have "REJ" errors	continuous
same_srv_rate	• % of connections to the same service	continuous
diff_srv_rate	• % of connections to different services	continuous
srv_count	• number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	• % of connections that have "SYN" errors	continuous
srv_rerror_rate	• % of connections that have "REJ" errors	continuous
srv_diff_host_rate	• % of connections to different hosts	continuous

Table B.3: Traffic features computed using a two-second time window [187]

Appendix C

Full List of Kyoto 2006+ Features

C.1 Statistical Features

Feature Name	Description
Duration	• the length (in seconds) of the connection
Service	• the connection's service type, e.g. http, telnet
Source_bytes	• the number of data bytes sent by the source IP address
Destination_bytes	• the number of data bytes sent by the destination IP address
Count	• the number of connections whose source IP ad-dress and destination IP address are the same to those of the current connection in the past two seconds.
Same_srv_rate	• % of connections to the same service in Count feature
Serror_rate	• % of connections that have "SYN" errors in Count feature
Srv_serror_rate	• % count of connections that have "SYN" errors in Srv_count (the number of connections whose service type is the same to that of the current connection in the past two seconds) feature
Dst_host_count	• among the past 100 connections whose destination IP address is the same as in the current connection, the number of connections whose source IP address is also the same as in the current connection
Dst_host_srv_count	• among the past 100 connections whose destination IP address is the same as in the current connection, the number of connections whose service type is also the same as in the current connection
Dst_host_same-_src_port_rate	• % of connections whose source port is the same to that of the current connectionin Dst_host_count feature
Dst_host_serror_rate	% of connections that have "SYN"errors in Dst_host_count feature
Dst_host_srv-_serror_rate	• % of connections that "SYN"errors in Dst_host_srv_count feature
Flag	• the state of the connection at the time the connection was written

Table C.1: Statistical features adopted from KDD Cup '99 [173]

C.2 Additional Features

Feature Name	Description
IDS_detection	• reflects whether IDS(Intrusion Detection System) triggered an alert for the connection; '0' means any alerts were not triggered, and an arabic numeral(except '0') means the different kinds of the alerts. Parenthesis indicates the number of the same alert observed during the connection. The authors used Symantec IDS [179] to extract this feature.
Malware_detection	• indicates whether malware, also known as malicious software,was observed in the connection; '0' means no malware was observed, and a string indicates the corresponding malware observed at the connection. The authors used ClamAV [35] software to detect malwares. Parenthesis indicates the number of the same malware observed during the connection.
Ashula_detection	• means whether shellcodes and exploit codes were used in the connection by using the dedicated software [10]; '0' means no shellcodes and exploit codes were observed, and an arabic numeral(except '0') means the different kinds of the shellcodes or exploit codes. Parenthesis indicates the number of the same shellcode or exploit code observed during the connection.
Label	• indicates whether the session was attack or not; '1' means the session was normal, '-1' means known attack was observed in the session, and '-2' means unknown attack was observed in the session.
Source_IP_Address	• means the source IP address used in the session. The original IP address on IPv4 was sanitized to one of the Unique Local IPv6 Unicast Addresses [159]. Also, the same private IP addresses are only valid in the same month: if two private IP addresses are the same within the same month. It means their IP addresses on IPv4 were also the same, otherwise they are different.
Source_Port_Number	• indicates the source port number used in the session.
Destination_IP_Address	• it was also sanitized.
Destination_Port_Number	• indicates the destination port number used in the session.
Start_Time	• indicates when the session was started.
Duration	• indicates how long the session was being established

Table C.2: Additional features of the Kyoto 2006+ dataset [173]

Appendix D

Full List of ASNM Features

D.1 Statistical Features

Feature ID	Description	Context
sumTTLIn	• sum of TTL value for inbound traffic.	
medTTLOut	• median of TTL value in outbound traffic.	
modTTLOut	• mode of TTL value in outbound traffic.	
sigTTLOut	• standard deviation of TTL value in outbound traffic.	
meanTTLOut	• mean of TTL value in outbound traffic.	
sumTTLOut	• sum of TTL value for outbound traffic.	
cntDataPktIn	• the number of inbound data packets. A data packets contain non-zero length payload of application layer of TCP/IP model.	
cntNondPktIn	• the number of inbound non-data packets. A non-data packets contain zero length payload of application layer of TCP/IP model.	
cntDataPktOut	• the number of outbound data packets.	
cntNondPktOut	• the number of outbound non-data packets.	
modTosIp	• mode of TOS (type of service) field in IP headers of all traffic.	
medTosIp	• median of TOS (type of service) field in IP headers of all traffic.	
sigTCPHdrLen	• standard deviation of TCP header lengths in all traffic.	
meanTCPHdrLen	• mean of TCP header lengths in all traffic.	
sumTCPHdrLen	• totaled TCP header lengths in all traffic.	
modTCPHdrLen	• mode of TCP header lengths in all traffic.	
medTCPHdrLen	• median of TCP header lengths in all traffic.	
hasFragIp	• indication of occurrence of fragmented packets in a connection.	
sumFragPkt	• the number of fragmented packets in all traffic.	
sumNfragPkt	• the number of non-fragmented packets in all traffic.	
ratFragNfrag	• ratio of the number of fragmented packets to the number of non-fragmented ones.	
BytesPerSecOut	• the number of bytes per seconds in outbound direction (from client to server).	
BytesPerSecIn	• the number of bytes per seconds in outbound direction (from server to client).	
BytesPerSessOut	• the number of transferred bytes during TCP session in outbound direction.	

Table D.1: Statistical features (part 1/3)

Feature ID	Description	Context
BytesPerSessIn	• the number of transferred bytes during TCP session in inbound direction.	
Bytes3WH2FIN	• the number of all transferred bytes from the start to the end of a communication including session initiation and destruction packets.	
BytesTCPSess	• the number of all transferred bytes from the start to the end of a communication excluding session initiation and destruction packets.	
BytesTCPOverhead	• the number of all transferred bytes in TCP session initiation and destruction phases.	
MedPktLenOut	• median of packet sizes in outbound traffic of a connection.	
ModPktLenOut	• mode of packet sizes in outbound traffic of a connection.	
MeanPktLenOut	• mean of packet sizes in outbound traffic of a connection.	
SigPktLenOut	• standard deviation of outbound packet lengths.	
SumPktLenOut	• the totaled outbound packet lengths of a connection.	
MedPktLenIn	• median of packet sizes in inbound traffic of a connection.	
ModPktLenIn	• mode of packet sizes in inbound traffic of a connection.	
MeanPktLenIn	• mean of packet sizes in inbound traffic of a connection.	
SigPktLenIn	• standard deviation of packet sizes in inbound traffic of a connection.	
SumPktLenIn	• the totaled inbound packet lengths of a connection.	
CntPktsOut	• the number of outbound packets per second.	
CntPktsIn	• the number of inbound packets per second.	
SumPktOut	• the overall number of outbound packets.	
SumPktIn	• the overall number of inbound packets.	
SumSYNPerSess	• the overall number of packets with SYN flag set.	
SumSACKPerSess	• the overall number of packets with SYN + ACK flags set.	
RatInOutB	• the ratio of inbound to outbound bytes.	
RatInOutPkt	• the ratio of inbound to outbound packet counts.	
MedTTLIn	• median of TTL (time to live) value in inbound traffic.	
ModTTLIn	• mode of TTL value in inbound traffic.	
SigTTLIn	• standard deviation of TTL value in inbound traffic.	
MeanTTLIn	• mean of TTL value in inbound traffic.	
ConTcpFinCntIn	• the number of inbound packets of a connection with FIN flag set.	
ConTcpSynCntIn	• the number of inbound packets of a connection with SYN flag set.	
ConTcpRstCntIn	• the number of inbound packets of a connection with RST flag set.	
ConTcpPshCntIn	• the number of inbound packets of a connection with PSH flag set.	
ConTcpAckCntIn	• the number of inbound packets of a connection with ACK flag set.	
ConTcpUrgCntIn	• the number of inbound packets of a connection with URG flag set.	
ConTcpEceCntIn	• the number of inbound packets of a connection with ECE flag set.	
ConTcpCwrCntIn	• the number of inbound packets of a connection with CWR flag set.	
ConTcpRstAckIn	• the number of inbound packets of a connection with RST+ACK set.	
ConTcpFinCntOut	• the number of outbound packets of a connection with FIN flag set.	
ConTcpSynCntOut	• the number of outbound packets of a connection with SYN flag set.	
ConTcpRstCntOut	• the number of outbound packets of a connection with RST flag set.	
ConTcpPshCntOut	• the number of outbound packets of a connection with PSH flag set.	
ConTcpAckCntOut	• the number of outbound packets of a connection with ACK flag set.	
ConTcpUrgCntOut	• the number of outbound packets of a connection with URG flag set.	
ConTcpEceCntOut	• the number of outbound packets of a connection with ECE flag set.	
ConTcpCwrCntOut	• the number of outbound packets of a connection with CWR flag set.	
ConTcpRstAckOut	• the number of outbound packets of a connection with RST+ACK flags set.	

Table D.2: Statistical features (part 2/3)

Feature ID	Description	Context
ConTcpFinCntAll	• the number of all packets of a connection with FIN flag set.	
ConTcpSynCntAll	• the number of all packets of a connection with SYN flag set.	
ConTcpRstCntAll	• the number of all packets of a connection with RST flag set.	
ConTcpPshCntAll	• the number of all packets of a connection with PSH flag set.	
ConTcpAckCntAll	• the number of all packets of a connection with ACK flag set.	
ConTcpUrgCntAll	• the number of all packets of a connection with URG flag set.	
ConTcpEceCntAll	• the number of all packets of a connection with ECE flag set.	
ConTcpCwrCntAll	• the number of all packets of a connection with CWR flag set.	
ConTcpRstAckAll	• the number of all packets of a connection with RST+ACK flags set.	

Table D.3: Statistical features (part 3/3)

D.2 Localization Features

Feature ID	Description	Context
SrcIP	• the IP address of the source machine (client).	
SrcIPInVlan	• indication of occurrence of source IP in local network.	
DstIP	• the IP address of the destination machine (server).	
DstIPInVlan	• indication of occurrence of destination IP in local network.	
SrcPort	• a port of the client machine.	
DstPort	• a port of the server machine.	
SrcMAC	• MAC address of the source machines network adapter.	
DstMAC	• MAC address of the destination machines network adapter.	

Table D.4: Localization features

D.3 Distributed Features

Feature ID	Description	Context
InPkt1s10i	• the number of inbound packets occurred in the first second of a connection which are distributed into 10 intervals.	
InPkt4s10i	• the same as the previous one, but computed above the first 4s.	
InPkt8s10i	• the same as the previous one, but computed above the first 8s.	
InPkt32s10i	• the same as the previous one, but computed above the first 32s.	
InPkt64s10i	• the same as the previous one, but computed above the first 64s.	
OutPkt1s10i	• the number of outbound packets occurred in the first second of a connection which are distributed into 10 intervals.	
OutPkt4s10i	• the same as the previous one, but computed above the first 4s.	
OutPkt8s10i	• the same as the previous one, but computed above the first 8s.	
OutPkt32s10i	• the same as the previous one, but computed above the first 32s.	
OutPkt64s10i	• the same as the previous one, but computed above the first 64s.	
InPkt1s20iTr4KB	• the accumulated number of inbound packets occurred in the 1st second of a connection distributed into 20 intervals, each after accumulation of 4KB of data.	
InPkt4s20iTr4KB	• the same as the previous one, but computed over the first 4s.	
InPkt8s20iTr4KB	• the same as the previous one, but computed over the first 8s.	
InPkt32s20iTr4KB	• the same as the previous one, but computed over the first 32s.	
InPkt64s20iTr4KB	• the same as the previous one, but computed over the first 64s.	
InPkt64s20iTr1KB	• the same as the previous one, but assumes threshold of 1KB.	
InPkt64s20iTr2KB	• the same as the previous one, but assumes threshold of 2KB.	
OutPkt1s20iTr4KB	• the accumulated number of outbound packets occurred in the 1st second of a connection distributed into 20 intervals, each after accumulation of 4KB of data.	
OutPkt4s20iTr4KB	• the same as the previous one, but computed over the first 4s.	
OutPkt8s20iTr4KB	• the same as the previous one, but computed over the first 8s.	
OutPkt32s20iTr4KB	• the same as the previous one, but computed over the first 32s.	
OutPkt64s20iTr4KB	• the same as the previous one, but computed over the first 64s.	
OutPkt64s20iTr1KB	• the same as the previous one, but assumes threshold of 1KB.	
OutPkt64s20iTr2KB	• the same as the previous one, but assumes threshold of 2KB.	
InPktLen1s10i	• lengths of inbound packets occurred in the first second of a connection which are distributed into 10 intervals.	
InPktLen4s10i	• the same as the previous one, but computed above the first 4s.	
InPktLen8s10i	• the same as the previous one, but computed above the first 8s.	
InPktLen32s10i	• the same as the previous one, but computed above the first 32s.	
InPktLen64s10i	• the same as the previous one, but computed above the first 64s.	
OutPktLen1s10i	• lengths of outbound packets occurred in the first second of a connection which are distributed into 10 intervals.	
OuPktLen4s10i	• the same as the previous one, but computed above the first 4s.	
OuPktLen8s10i	• the same as the previous one, but computed above the first 8s.	
OuPktLen32s10i	• the same as the previous one, but computed above the first 32s.	
OuPktLen64s10i	• the same as the previous one, but computed above the first 64s.	

Table D.5: Distributed features

D.4 Dynamic Features

Feature ID	Description	Context
BPerSecIn	• the number of bytes per second transferred in inbound traffic.	X
BPerSecOut	• the number of bytes per second transferred in outbound traffic.	X
BPerSesIn	• the number of transferred bytes between client and server hosts in inbound traffic during TCP session.	X
BPerSesOut	• the number of transferred bytes between client and server hosts in outbound traffic during TCP session.	X
PktPerSIn	• the number of packets per second in inbound traffic.	X
PktPerSOut	• the number of packets per second in outbound traffic.	X
PktPerSesIn	• the number of inbound packets transferred between client and server hosts during the duration of a connection.	X
PktPerSesOut	• the number of outbound packets transferred between client and server hosts during time of a connection.	X
TSesStart	• start time of a connection.	
TSesEnd	• end time of a connection.	
SessDuration	• a duration of a connection.	
CntResendPktsIn	• the number resend inbound packets.	
CntResendPktsOut	• the number resend outbound packets.	
CntPktOutOfOrd	• the number of packets occurred out of order.	
MedTdiff2Pkts	• median of packet IAT in all traffic.	
MeanTdiff2Pkts	• mean of packet IAT in all traffic.	
SigTdiff2Pkts	• standard deviation of packet IAT in all traffic.	
MedTdiff2PktsIn	• median of packet IAT in inbound traffic.	
MeanTdiff2PktsIn	• mean of packet IAT in inbound traffic.	
SigTdiff2PktsIn	• standard deviation of packet IAT in inbound traffic	
MedTdiff2PktsOut	• median of packet IAT in outbound traffic.	
MeanTdiff2PktsOut	• mean of packet IAT in outbound traffic.	
SigTdiff2PktsOut	• standard deviation of packet IAT in outbound traffic	
IntervalsPortsSig	• standard deviation of time intervals between consecutive connections of the two hosts running on the same ports as an analyzed connection. The feature assumes only beginnings of connection for computation of intervals.	X
IntervalsPortsSig2	• the same as the previous one, but it assumes the beginnings as well as ends of connections for computation of intervals.	X
IntervalsIPsSig	• standard deviation of time intervals between consecutive connections of the two hosts running on the same IP addresses as an analyzed connection. The feature assumes only beginnings of connection for computation of intervals.	X
IntervalsIPsSig	• the same as the previous one, but it assumes the beginnings as well as ends of connections for computation of intervals.	X
CntSYNIn	• the number of inbound packets with SYN flag set.	X
CntSYNOut	• the number of outbound packets with SYN flag set.	X
CntACKIn	• the number of inbound packets with ACK flag set.	X
CntACKOut	• the number of outbound packets with ACK flag set.	X
CntFINIn	• the number of inbound packets with FIN flag set.	X
CntFINOut	• the number of outbound packets with FIN flag set.	X

Table D.6: Dynamic features

D.5 Behavioral Features

Feature ID	Description	Context
CntOfOldFlows	• the number of mutual flows between client and server hosts of analyzed connection 5 minutes before it started.	X
CntOfNewFlows	• the number of mutual flows between client and server hosts of analyzed connection 5 minutes after it finished.	X
CorClosed	• indication of correctly closed connection. It exploits 3-way end-shake.	
PolyInd3ordIn	• approximation of inbound communication by polynomial of 3rd order in the index domain of packet occurrences.	
PolyInd5ordIn	• the same as the previous one, but utilizes polynomial of 5th order.	
PolyInd8ordIn	• the same as the previous one, but utilizes polynomial of 8th order.	
PolyInd10ordIn	• the same as the previous one, but utilizes polynomial of 10th order.	
PolyInd13ordIn	• the same as the previous one, but utilizes polynomial of 13th order.	
PolyInd3ordOut	• approximation of outbound communication by polynomial of 3rd order in the index domain of packet occurrences.	
PolyInd5ordOut	• the same as the previous one, but utilizes polynomial of 5th order.	
PolyInd8ordOut	• the same as the previous one, but utilizes polynomial of 8th order.	
PolyInd10ordOut	• the same as the previous one, but utilizes polynomial of 10th order.	
PolyInd13ordOut	• the same as the previous one, but utilizes polynomial of 13th order.	
PolyTime3ordIn	• approximation of inbound communication by polynomial of 3rd order in the time domain of packet occurrences.	
PolyTime5ordIn	• the same as the previous one, but utilizes polynomial of 5th order.	
PolyTime8ordIn	• the same as the previous one, but utilizes polynomial of 8th order.	
PolyTime10ordIn	• the same as the previous one, but utilizes polynomial of 10th order.	
PolyTime13ordIn	• the same as the previous one, but utilizes polynomial of 13th order.	
PolyTime3ordOut	• approximation of outbound communication by polynomial of 3rd order in the time domain of packet occurrences.	
PolyTime5ordOut	• the same as the previous one, but utilizes polynomial of 5th order.	
PolyTime8ordOut	• the same as the previous one, but utilizes polynomial of 8th order.	
PolyTime10ordOut	• the same as the previous one, but utilizes polynomial of 10th order.	
PolyTime13ordOut	• the same as the previous one, but utilizes polynomial of 13th order.	
PolyInd3ordAllN	• the same as the previous one, but utilizes polynomial of 3th order for approximation of all packet sizes, while inbound packets are assumed with negative sign and outbound packets by positive sign.	

Table D.7: Behavioral features (part 1/2)

Feature ID	Description	Context
FourGonModulIn	• Fast Fourier Transformation (FFT) of inbound packet sizes. The feature represents angle coefficients of the FFT in goniometric representation.	
FourGonAngleIn	• the same as previous, but the feature represents module coefficients of the FFT.	
FourGonModulOut	• FFT of outbound packet sizes. The feature represents angle coefficients of the FFT in goniometric representation.	
FourGonAngleOut	• the same as previous, but the feature represents module coefficients of the FFT.	
FourGonModulAllN	• FFT of all packet sizes, while inbound packets are represented with negative sign and outbound packets with positive sign. The feature represents module coefficients of the FFT in goniometric representation.	
FourGonAngleAllN	• the same as the previous one, but the feature represent angle coefficients of the FFT.	
GaussProds1In	• normalized products of inbound packet sizes with 1 Gaussian curve.	
GaussProds2In	• normalized sums of products of inbound packet sizes with 2 Gaussian curves. Packets are divided into 2 slices and products are computed per each slice by summing of products of relevant packets with fitted Gaussian function. Each product is normalized by the number of packets in a slice.	
GaussProds4In	• the same as the previous one, but products are computed using 4 Gaussian curves.	
GaussProds8In	• the same as the previous one, but products are computed using 8 Gaussian curves.	
GaussProds1Out	• normalized products of outbound packet sizes with 1 Gaussian curve.	
GaussProds2Out	• normalized products of outbound packet sizes with 2 Gaussian curves. Packets are divided into 2 slices and products are computed per each slice by summing of products of relevant packets with fitted Gaussian function. Each product is normalized by the number of packets in a slice.	
GaussProds4Out	• the same as the previous one, but products are computed using 4 Gaussian curves.	
GaussProds8Out	• the same as the previous one, but products are computed using 8 Gaussian curves.	
GaussProds1All	• normalized products of all packet sizes with 1 Gaussian curve.	
GaussProds2All	• the same as gaussProds2Out, but considers all traffic.	
GaussProds4All	• the same as the previous one, but products are computed using 4 Gaussian curves.	
GaussProds8All	• the same as the previous one, but products are computed using 8 Gaussian curves.	
GaussProds8AllN	• the same as the previous one, but it considers incoming packet sizes with negative sign and outgoing with positive sign.	

Table D.8: Behavioral features (part 2/2)

Appendix E

Discriminators of Andrew Moore

ID	Short Name	Description
1	Server Port	• port number of a server.
2	Client Port	• port number of a client.
3	min_IAT	• minimum packet inter-arrival time (IAT) for all packets of the flow.
4	q1_IAT	• the first quartile of packet IAT.
5	med_IAT	• median of packet IAT.
6	mean_IAT	• mean of packet IAT.
7	q3_IAT	• the 3rd quartile of packet IAT
8	max_IAT	• the maximum packet IAT.
9	var_IAT	• variance in packet IAT.
10	min_data_wire	• the minimum of bytes in Ethernet packet.
11	q1_data_wire	• the first quartile of bytes in Ethernet packet.
12	med_data_wire	• median of bytes in Ethernet packet.
13	mean_data_wire	• mean of bytes in Ethernet packet.
14	q3_data_wire	• the 3rd quartile of bytes in Ethernet packet.
15	max_data_wire	• the maximum of bytes in Ethernet packet.
16	var_data_wire	• variance of bytes in Ethernet packet.
17	min_data_ip	• the minimum of total bytes in IP packet, using the size of payload declared by the IP packet.
18	q1_data_ip	• the first quartile of total bytes in IP packet.
19	med_data_ip	• median of total bytes in IP packet.
20	mean_data_ip	• mean of total bytes in IP packet.
21	q3_data_ip	• the 3rd quartile of total bytes in IP packet.
22	max_data_ip	• the maximum of total bytes in IP packet.
23	var_data_ip	• variance of total bytes in IP packet.
24	min_data_control	• the minimum of control bytes in packet.
25	q1_data_control	• the first quartile of control bytes in packet.
26	med_data_control	• median of control bytes in packet.
27	mean_data_control	• mean of control bytes in packet.
28	q3_data_control	• the 3rd quartile of control bytes in packet.
29	max_data_control	• the maximum of control bytes in packet.
30	var_data_control	• variance of control bytes in packet.
31	total_packets_a_b	• the total number of packets seen from client to server.
32	total_packets_b_a	• the total number of packets seen from server to client.
33	ack_pkts_sent_a_b	• the total number of packets with ACK flag set (client to server).
34	ack_pkts_sent_b_a	• the total number of packets with ACK flag set (server to client).
35	pure_acks_sent_a_b	• the total number of packets with ACK flag that were not piggy-backed with data (client to server).
36	pure_acks_sent_b_a	• the same as the previous one, but considers opposite direction.

Table E.1: Discriminator features [119] (part 1/7)

ID	Short Name	Description
37	sack_pkts_sent_a_b	• the total number packets with SYN + ACK set (client to server).
38	sack_pkts_sent_b_a	• the same as the previous one, but for opposite direction.
39	dsack_pkts_sent_a_b	• the total number of packets with duplicate SYN + ACK flags set (client to server).
40	dsack_pkts_sent_b_a	• the same as the previous one, but for opposite direction
41	max_sack_blks/ack_a_b	• the maximum number of SYN + ACK blocks seen in any SYN + ACK packet (client to server).
42	max_sack_blks/ack_b_a	• the same as the previous one, but for opposite direction.
43	unique_bytes_sent_a_b	• the number of unique bytes sent, i.e., the total bytes of data sent excluding retransmitted bytes and any bytes sent doing window probing (client to server).
44	unique_bytes_sent_b_a	• the same as the previous one, but for opposite direction.
45	actual_data_pkts_a_b	• the count of all the packets with at least a byte of TCP data payload (client to server).
46	actual_data_pkts_b_a	• the same as the previous one, but for opposite direction.
47	actual_data_bytes_a_b	• the total number of data bytes seen, including bytes from retransmissions and window probe packets (client to server).
48	actual_data_bytes_b_a	• the same as the previous one, but for opposite direction.
49	rexmt_data_pkts_a_b	• the count of all retransmitted packets (client to server).
50	rexmt_data_pkts_a_b	• the same as the previous one, but for opposite direction.
51	rexmt_data_bytes_a_b	• the total bytes of data found in retransmitted packets (client to server).
52	rexmt_data_bytes_b_a	• the same as the previous one, but for opposite direction
53	zwnd_probe_pkts_a_b	• the count of all the window probe packets seen (client to server).
54	zwnd_probe_pkts_b_a	• the same as the previous one, but for opposite direction.
55	zwnd_probe_bytes_a_b	• the total bytes of data sent in the window probe packet (client to server).
56	zwnd_probe_bytes_b_a	• the same as the previous one, but for opposite direction.
57	outoforder_pkts_a_b	• the count of all the packets arrived out of order (client to server).
58	outoforder_pkts_b_a	• the same as the previous one, but for opposite direction.
59	pushed_data_pkts_a_b	• the count of all the packets seen with the PUSH flag set in the TCP header (client to server).
60	pushed_data_pkts_b_a	• the same as the previous one, but for opposite direction.
61	SYN_pkts_sent_a_b	• The count of all the packets seen with the SYN flag set in the TCP header (client to server).
62	FIN_pkts_sent_a_b	• the count of all the packets seen with the FIN flag set in the TCP header (client to server).
63	SYN_pkts_sent_b_a	• the same as SYN_pkts_sent_a_b, but for oppposite direction.
64	FIN_pkts_sent_b_a	• the same as FIN_pkts_sent_a_b, but for oppposite direction.
65	req_1323_ws_a_b	• if the endpoint requested Window Scaling option as specified in RFC 1323 a 'Y' is printed on the respective field. If the option was not requested, an 'N' is printed (client to server).
66	req_1323_ts_a_b	• if the endpoint requested Timestamp option as specified in RFC 1323 a 'Y' is printed on the respective field. If the option was not requested, an 'N' is printed (client to server).
67	req_1323_ws_b_a	• the same as req_1323_ws_a_b, but for oppposite direction.
68	req_1323_ts_b_a	• the same as req_1323_ts_a_b, but for oppposite direction.
69	adv_wind_scale_a_b	• the window scaling factor used. This field is valid only if the connection was captured fully to include the SYN packets (client to server).
70	adv_wind_scale_b_a	• the same as the previous one, but for opposite direction.
71	req_sack_a_b	• if the endpoint sent a SYN + ACK permitted option in the SYN packet opening the connection, a 'Y' is printed; otherwise 'N' is printed (client to server).
72	req_sack_b_a	• the same as the previous one, but for opposite direction.

Table E.2: Discriminator features [119] (part 2/7)

ID	Short Name	Description
73	sacks_sent_a_b	• the total number of ACK packets seen carrying SYN + ACK information (client to server).
74	sacks_sent_b_a	• the same as the previous one, but for opposite direction.
75	urgent_data_pkts_a_b	• the total number of packets with the URG flag turned on in the TCP header (client to server).
76	urgent_data_pkts_b_a	• the same as the previous one, but for opposite direction.
77	urgent_data_bytes_a_b	• the total bytes of urgent data sent. This field is calculated by summing the urgent pointer offset values found in packets having the URG flag set in the TCP header (client to server).
78	urgent_data_bytes_b_a	• the same as the previous one, but for opposite direction.
79	mss_requested_a_b	• the Maximum Segment Size (MSS) requested as a TCP option in the SYN packet opening the connection (client to server).
80	mss_requested_b_a	• the same as the previous one, but for opposite direction.
81	max_segm_size_a_b	• the maximum segment size observed during the lifetime of the connection (client to server).
82	max_segm_size_b_a	• the same as the previous one, but for opposite direction.
83	min_segm_size_a_b	• the minimum segment size observed during the lifetime of the connection (client to server).
84	min_segm_size_b_a	• the same as the previous one, but for opposite direction.
85	avg_segm_size_a_b	• the average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data packets found (client to server).
86	avg_segm_size_b_a	• the same as the previous one, but for opposite direction.
87	max_win_adv_a_b	• the maximum window advertisement seen if the connection is using window scaling (client to server).
88	max_win_adv_b_a	• the same as the previous one, but for opposite direction.
89	min_win_adv_a_b	• the minimum window advertisement seen. This is the minimum window-scaled advertisement seen if both sides negotiated window scaling (client to server).
90	min_win_adv_b_a	• the same as the previous one, but for opposite direction.
91	zero_win_adv_a_b	• the number of times a zero receive window was advertised (client to server).
92	zero_win_adv_b_a	• the same as the previous one, but for opposite direction.
93	avg_win_adv_a_b	• the average window advertisement seen, calculated as the sum of all window advertisements divided by the total number of packets seen (client to server).
94	avg_win_adv_b_a	• the same as the previous one, but for opposite direction.
95	initial_window-bytes_a_b	• the total number of bytes sent in the initial window i.e., the number of bytes seen in the initial flight of data before receiving the first ACK packet from the other endpoint (client to server).
96	initial_window-bytes_b_a	• the same as the previous one, but for opposite direction.
97	initial_window-packets_a_b	• the total number of segments (packets) sent in the initial window (client to server).
98	initial_window-packets_b_a	• the same as the previous one, but for opposite direction.
99	ttl_stream_length_a_b	• The theoretical stream length, which is calculated as the difference between the sequence numbers of the SYN and FIN packets, giving the length of the data stream seen (client to server).
100	ttl_stream_length_b_a	• the same as the previous one, but for opposite direction.
101	missed_data_a_b	• the missed data, calculated as the difference between the theoretical stream length and unique bytes sent (client to server).
102	missed_data_b_a	• the same as the previous one, but for opposite direction.
103	truncated_data_a_b	• The truncated data, calculated as the total bytes of data truncated during packet capture (client to server).
104	truncated_data_b_a	• the same as the previous one, but for opposite direction.

Table E.3: Discriminator features [119] (part 3/7)

ID	Short Name	Description
105	truncated_packets_a_b	• the total number of packets truncated as explained above (client to server).
106	truncated_packets_b_a	• the same as the previous one, but for opposite direction.
107	data_xmit_time_a_b	• total data transmission time, calculated as the difference between the times of capture of the first and last packets carrying non-zero TCP data payload (client to server).
108	data_xmit_time_b_a	• the same as the previous one, but for opposite direction.
109	idletime_max_a_b	• the maximum idle time, calculated as the maximum time between consecutive packets seen in the direction (client to server).
110	idletime_max_b_a	• the same as the previous one, but for opposite direction.
111	throughput_a_b	• the average throughput calculated as the unique bytes sent divided by the elapsed time (client to server).
112	throughput_b_a	• the same as the previous one, but for opposite direction.
113	RTT_samples_a_b	• the total number of Round-Trip Time (RTT) samples found (client to server).
114	RTT_samples_b_a	• the same as the previous one, but for opposite direction.
115	RTT_min_a_b	• the minimum RTT sample seen (client to server).
116	RTT_min_b_a	• the same as the previous one, but for opposite direction.
117	RTT_max_a_b	• the maximum RTT sample seen (client to server).
118	RTT_max_b_a	• the same as the previous one, but for opposite direction.
119	RTT_avg_a_b	• the average value of RTT found, calculated in straightforward way as the sum of all the RTT values found divided by the total number of RTT samples (client to server).
120	RTT_avg_b_a	• the same as the previous one, but for opposite direction.
121	RTT_stdv_a_b	• The standard deviation of the RTT samples (client to server).
122	RTT_stdv_b_a	• the same as the previous one, but for opposite direction.
123	RTT_from_3WHS_a_b	• the RTT value calculated from the TCP 3-Way Hand-Shake (client to server).
124	RTT_from_3WHS_b_a	• the same as the previous one, but for opposite direction.
125	RTT_full_sz_smpls_a_b	• The total number of full-size RTT samples, calculated from the RTT samples of full-size segments. Full-size segments are defined to be the segments of the largest size seen in the connection (client server).
126	RTT_full_sz_smpls_b_a	• the same as the previous one, but for opposite direction.
127	RTT_full_sz_min_a_b	• the minimum full-size RTT sample (client to server).
128	RTT_full_sz_min_b_a	• the same as the previous one, but for opposite direction.
129	RTT_full_sz_max_a_b	• the maximum full-size RTT sample (client to server).
130	RTT_full_sz_max_b_a	• the same as the previous one, but for opposite direction.
131	RTT_full_sz_avg_a_b	• the average full-size RTT sample (client to server).
132	RTT_full_sz_avg_b_a	• the same as the previous one, but for opposite direction.
133	RTT_full_sz_stdev_a_b	• the standard deviation of full-size RTT sample (client to server).
134	RTT_full_sz_stdev_b_a	• the same as the previous one, but for opposite direction.
135	post-loss_acks_a_b	• the total number of ACK packets received after losses, which were detected and a retransmission occurred (client to server).
136	post-loss_acks_b_a	• the same as the previous one, but for opposite direction.
137	segs_cum_acked_a_b	• the count of the number of segments that were cumulatively acknowledged and not directly acknowledged (client to server).
138	segs_cum_acked_b_a	• the same as the previous one, but for opposite direction.
139	duplicate_acks_a_b	• the total number of duplicate acknowledgments received (client to server).
140	duplicate_acks_b_a	• the same as the previous one, but for opposite direction.
141	triple_dupacks_a_b	• the total number of triple duplicate acknowledgments received (client to server).
142	triple_dupacks_a_b	• the same as the previous one, but for opposite direction.

Table E.4: Discriminator features [119] (part 4/7)

ID	Short Name	Description
143	max_#_retrans_a_b	• the maximum number of retransmissions seen for any segment during the lifetime of the connection (client to server).
144	max_#_retrans_b_a	• the same as the previous one, but for opposite direction.
145	min_retr_time_a_b	• the minimum time seen between any two (re)transmissions of a segment among all the retransmissions seen (client to server).
146	min_retr_time_b_a	• the same as the previous one, but for opposite direction.
147	max_retr_time_a_b	• the maximum time seen between any two (re)transmissions of a segment (client to server).
148	max_retr_time_b_a	• the same as the previous one, but for opposite direction.
149	avg_retr_time_a_b	• the average time seen between any two (re)transmissions of a segment (client to server).
150	avg_retr_time_b_a	• the same as the previous one, but for opposite direction.
151	std_retr_time_a_b	• the standard deviation of the retransmission-time samples obtained from all the retransmissions (client to server).
152	std_retr_time_b_a	• the same as the previous one, but for opposite direction.
153	min_data_wire_a_b	• the minimum number of bytes in Ethernet packet (client to server).
154	q1_data_wire_a_b	• the first quartile of bytes in Ethernet packet (client to server).
155	med_data_wire_a_b	• median of bytes in Ethernet packet (client to server).
156	mean_data_wire_a_b	• mean of bytes in Ethernet packet (client to server).
157	q3_data_wire_a_b	• the third quartile of bytes in Ethernet packet (client to server).
158	max_data_wire_a_b	• the maximum of bytes in Ethernet packet (client to server).
159	var_data_wire_a_b	• variance of bytes in Ethernet packet (client to server).
160	min_data_ip_a_b	• the minimum number of total bytes in IP packet (client to server).
161	q1_data_ip_a_b	• the first quartile of total bytes in IP packet (client to server).
162	med_data_ip_a_b	• median of total bytes in IP packet (client to server).
163	mean_data_ip_a_b	• mean of total bytes in IP packet (client to server).
164	q3_data_ip_a_b	• the third quartile of total bytes in IP packet (client to server).
165	max_data_ip_a_b	• the maximum of total bytes in IP packet (client to server).
166	var_data_ip_a_b	• variance of total bytes in IP packet (client to server).
167	min_data_control_a_b	• the minimum number of control bytes in packet (client to server).
168	q1_data_control_a_b	• the first quartile of control bytes in packet (client to server).
169	med_data_control_a_b	• median of control bytes in packet (client to server).
170	mean_data_control_a_b	• mean of control bytes in packet (client to server).
171	q3_data_control_a_b	• the third quartile of control bytes in packet (client to server).
172	max_data_control_a_b	• the maximum of control bytes in packet (client to server).
173	var_data_control_a_b	• variance of control bytes in packet (client to server).
174	min_data_wire_b_a	• the minimum number of bytes in Ethernet packet (server to client).
175	q1_data_wire_b_a	• the first quartile of bytes in Ethernet packet (server to client).
176	med_data_wire_b_a	• median of bytes in Ethernet packet (server to client).
177	mean_data_wire_b_a	• mean of bytes in Ethernet packet (server to client).
178	q3_data_wire_b_a	• the third quartile of bytes in Ethernet packet (server to client).
179	max_data_wire_b_a	• the maximum of bytes in Ethernet packet (server to client).
180	var_data_wire_b_a	• variance of bytes in Ethernet packet (server to client).

Table E.5: Discriminator features [119] (part 5/7)

ID	Short Name	Description
181	min_data_ip_b_a	• the minimum number of total bytes in IP packet (server to client).
182	q1_data_ip_b_a	• the first quartile of total bytes in IP packet (server to client).
183	med_data_ip_b_a	• median of total bytes in IP packet (server to client).
184	mean_data_ip_b_a	• mean of total bytes in IP packet (server to client).
185	q3_data_ip_b_a	• the third quartile of total bytes in IP packet (server to client).
186	max_data_ip_b_a	• the maximum of total bytes in IP packet (server to client).
187	var_data_ip_b_a	• variance of total bytes in IP packet (server to client).
188	min_data_control_b_a	• the minimum number of control bytes in packet (server to client).
189	q1_data_control_b_a	• the first quartile of control bytes in packet (server to client).
190	med_data_control_b_a	• median of control bytes in packet (server to client).
191	mean_data_control_b_a	• mean of control bytes in packet (server to client).
192	q3_data_control_b_a	• the third quartile of control bytes in packet (server to client).
193	max_data_control_b_a	• the maximum of control bytes in packet (server to client).
194	var_data_control_b_a	• variance of control bytes in packet (server to client).
195	min_IAT_a_b	• the minimum of packet inter-arrival time (client to server).
196	q1_IAT_a_b	• the first quartile of packet inter-arrival time (client to server).
197	med_IAT_a_b	• median of packet inter-arrival time (client to server).
198	mean_IAT_a_b	• mean of packet inter-arrival time (client to server).
199	q3_IAT_a_b	• the third quartile of packet inter-arrival time (client to server).
200	max_IAT_a_b	• the maximum of packet inter-arrival time (client to server).
201	var_IAT_a_b	• variance of packet inter-arrival time (client to server).
202	min_IAT_b_a	• the minimum of packet inter-arrival time (server to client).
203	q1_IAT_b_a	• the first quartile of packet inter-arrival time (server to client).
204	med_IAT_b_a	• median of packet inter-arrival time (server to client).
205	mean_IAT_b_a	• mean of packet inter-arrival time (server to client).
206	q3_IAT_b_a	• the third quartile of packet inter-arrival time (server to client).
207	max_IAT_b_a	• the maximum of packet inter-arrival time (server to client).
208	var_IAT_b_a	• variance of packet inter-arrival time (server to client).
209	Time_since _last_connection	• time elapsed since the last connection between involved client and server.
210	No._transitions _bulk/trans	• the number of transitions between transaction mode and bulk transfer mode, where bulk transfer mode is defined as the time when there are more than three successive packets in the same direction without any packets carrying data in the other direction.
211	Time_spent_in_bulk	• amount of time spent in bulk transfer mode.
212	Duration	• connection duration.

Table E.6: Discriminator features [119] (part 6/7)

ID	Short Name	Description
213	%_bulk	• ratio of time spent in bulk transfer mode.
214	Time_spent_idle	• the time spent idle, where idle time is the accumulation of all periods of 2 seconds or greater when no packet was seen in either direction.
215	%_idle	• ratio of time spent in idle mode.
216	Effective_Bandwidth	• effective bandwidth based upon entropy (both directions).
217	Effective_Bandwidth_a_b	• effective bandwidth based upon entropy (client to server).
218	Effective_Bandwidth_b_a	• effective bandwidth based upon entropy (server to client).
219	FFT_all_#1	• FFT of packet IAT – Frequency #1 (both directions).
220	FFT_all_#2	• FFT of packet IAT – Frequency #2 (both directions).
221	FFT_all_#3	• FFT of packet IAT – Frequency #3 (both directions).
222	FFT_all_#4	• FFT of packet IAT – Frequency #4 (both directions).
223	FFT_all_#5	• FFT of packet IAT – Frequency #5 (both directions).
224	FFT_all_#6	• FFT of packet IAT – Frequency #6 (both directions).
225	FFT_all_#7	• FFT of packet IAT – Frequency #7 (both directions).
226	FFT_all_#8	• FFT of packet IAT – Frequency #8 (both directions).
227	FFT_all_#9	• FFT of packet IAT – Frequency #9 (both directions).
228	FFT_all_#10	• FFT of packet IAT – Frequency #10 (both directions).
229	FFT_a_b_#1	• FFT of packet IAT – Frequency #1 (client to server).
230	FFT_a_b_#2	• FFT of packet IAT – Frequency #2 (client to server).
231	FFT_a_b_#3	• FFT of packet IAT – Frequency #3 (client to server).
232	FFT_a_b_#4	• FFT of packet IAT – Frequency #4 (client to server).
233	FFT_a_b_#5	• FFT of packet IAT – Frequency #5 (client to server).
234	FFT_a_b_#6	• FFT of packet IAT – Frequency #6 (client to server).
235	FFT_a_b_#7	• FFT of packet IAT – Frequency #7 (client to server).
236	FFT_a_b_#8	• FFT of packet IAT – Frequency #8 (client to server).
237	FFT_a_b_#9	• FFT of packet IAT – Frequency #9 (client to server).
238	FFT_a_b_#10	• FFT of packet IAT – Frequency #10 (client to server).
239	FFT_b_a_#1	• FFT of packet IAT – Frequency #1 (server to client).
240	FFT_b_a_#2	• FFT of packet IAT – Frequency #2 (server to client).
241	FFT_b_a_#3	• FFT of packet IAT – Frequency #3 (server to client).
242	FFT_b_a_#4	• FFT of packet IAT – Frequency #4 (server to client).
243	FFT_b_a_#5	• FFT of packet IAT – Frequency #5 (server to client).
244	FFT_b_a_#6	• FFT of packet IAT – Frequency #6 (server to client).
245	FFT_b_a_#7	• FFT of packet IAT – Frequency #7 (server to client).
246	FFT_b_a_#8	• FFT of packet IAT – Frequency #8 (server to client).
247	FFT_b_a_#9	• FFT of packet IAT – Frequency #9 (server to client).
248	FFT_b_a_#10	• FFT of packet IAT – Frequency #10 (server to client).
249	Classes	• application class.

Table E.7: Discriminator features [119] (part 7/7)

Appendix F

Performance Experiments with ASNM

F.1 Confusion Matrices from Master's Thesis

Classification Accuracy: 87.57%		**True Class**		**Precision**
		Legit. Flows	**Attacks**	
Predicted Class	**Legit. Flows**	151	1	99.34%
	Attacks	22	11	33.33%
	Recall	87.28%	91.67%	$F_1 = 48.88\%$

Table F.1: Naive Bayes Classifier

Classification Accuracy: 91.92%		**True Class**		**Precision**
		Legit. Flows	**Attacks**	
Predicted Class	**Legit. Flows**	168	10	94.38%
	Attacks	5	2	28.57%
	Recall	97.11%	16.16%	$F_1 = 20.64\%$

Table F.2: Naive Bayes classifier and PCA with automatic count of components

Classification Accuracy: 88.10%		**True Class**		**Precision**
		Legit. Flows	**Attacks**	
Predicted Class	**Legit. Flows**	153	2	98.71%
	Attacks	20	10	33.33%
	Recall	88.44%	83.33%	$F_1 = 47.61\%$

Table F.3: Naive Bayes classifier and PCA with fixed count of components

Classification Accuracy: 94.06%		True Class		Precision
		Legit. Flows	Attacks	
Predicted Class	**Legit. Flows**	163	1	99.39%
	Attacks	10	11	52.38%
	Recall	94.22%	91.67%	$F_1 = 66.66\%$

Table F.4: Naive Bayes classifier with discretization of ordinal attributes

Classification Accuracy: 82.17%		True Class		Precision
		Legit. Flows	Attacks	
Predicted Class	**Legit. Flows**	140	0	100.00%
	Attacks	33	12	26.67%
	Recall	80.92%	100.00%	$F_1 = 42.10\%$

Table F.5: SVM classifier with neural kernel

Classification Accuracy: 95.14%		True Class		Precision
		Legit. Flows	Attacks	
Predicted Class	**Legit. Flows**	169	5	97.13%
	Attacks	4	7	63.64%
	Recall	97.69%	58.33%	$F_1 = 60.87\%$

Table F.6: Decision Tree classifier with gini index split criterion

F.2 CDX 2009 and ASNM

We placed the results achieved by Forward Feature Selection (FFS) method which ran on the Decision Tree classifier into the current appendix, as our attention was primarily focused on the Naive Bayes classifier. For detailed information see Section 6.3.1.

F.2.1 FFS with Decision Tree

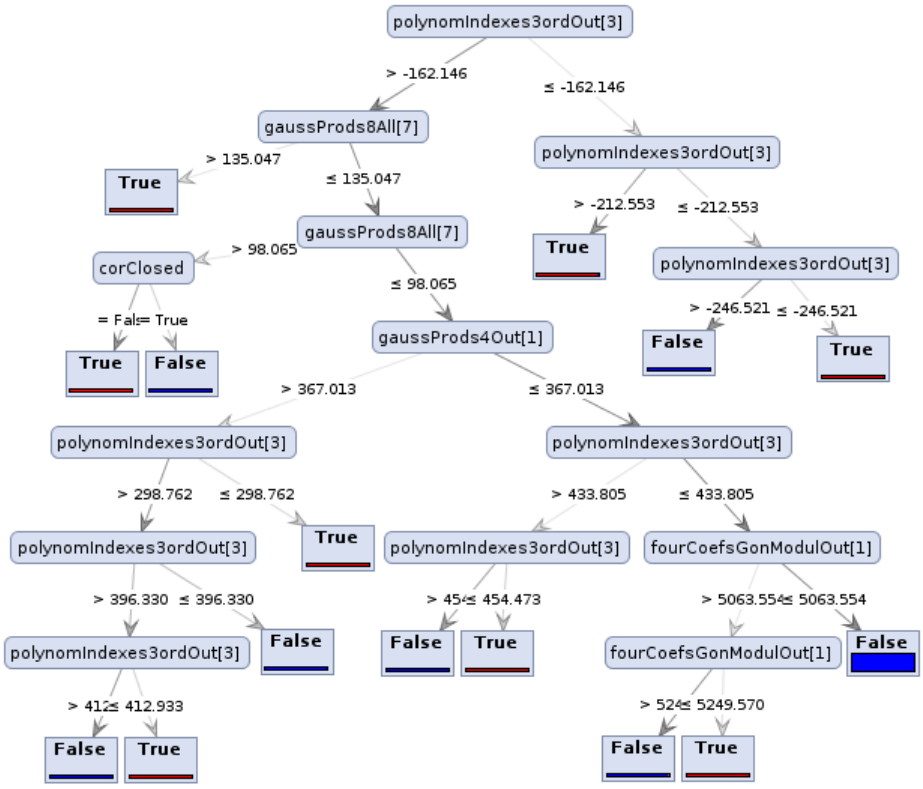

Figure F.1: Model of the Decision Tree classifier (ASNM features)

Classification Accuracy: 99.86% ±0.10		**True Class**		**Precision**
		Legit. Flows	**Attacks**	
Predicted Class	**Legit. Flows**	5723	4	99.93%
	Attacks	4	40	90.91%
	Recall	99.93%	90.91%	$F_1 = 90.91\%$

Table F.7: FFS on ASNM features and Decision Tree classifier

Appendix G

Tunneling Obfuscation

Several examples of discriminating and obfuscated features are placed into the current appendix, as they are not representatives of particular category, and thus their outcome may be unclear or inexpressive in some ranges of the feature distribution. For more information about associated experiments see Section 7.7.

G.1 Discriminating Features

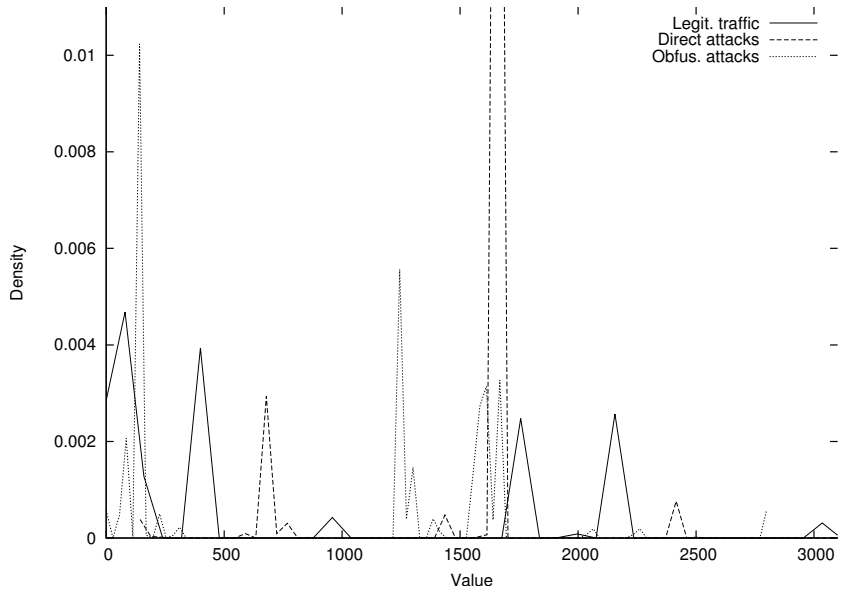

Figure G.1: Lengths of inbound packets occurred in the first 32 seconds of a connection (`InPktLen32s10i[0]`)

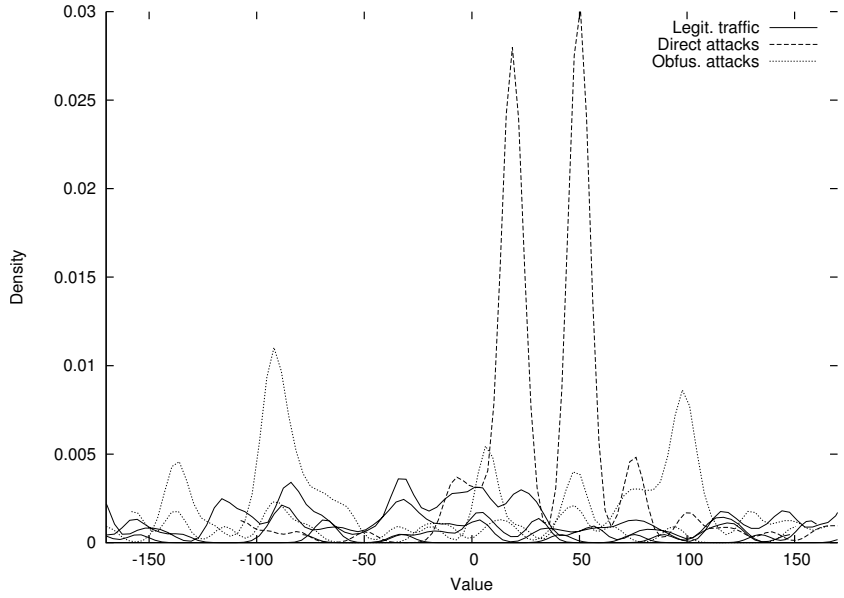

Figure G.2: FFT of packet lengths (both directions)
(`FourGonAngleNeg[9]`)

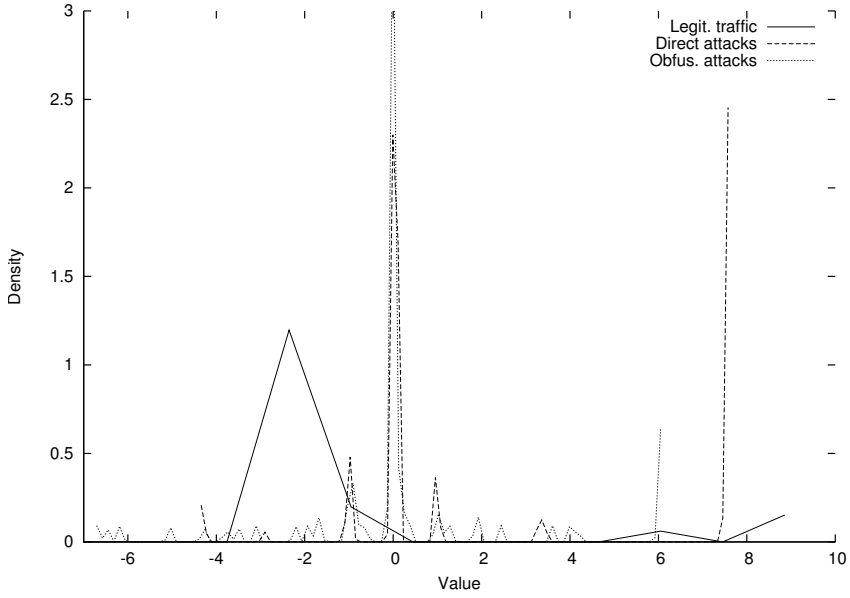

Figure G.3: Normalized sum of products of a connection with 8 Gaussian curves
(`GaussProds8AllNeg[7]`)

G.2 Obfuscated Features

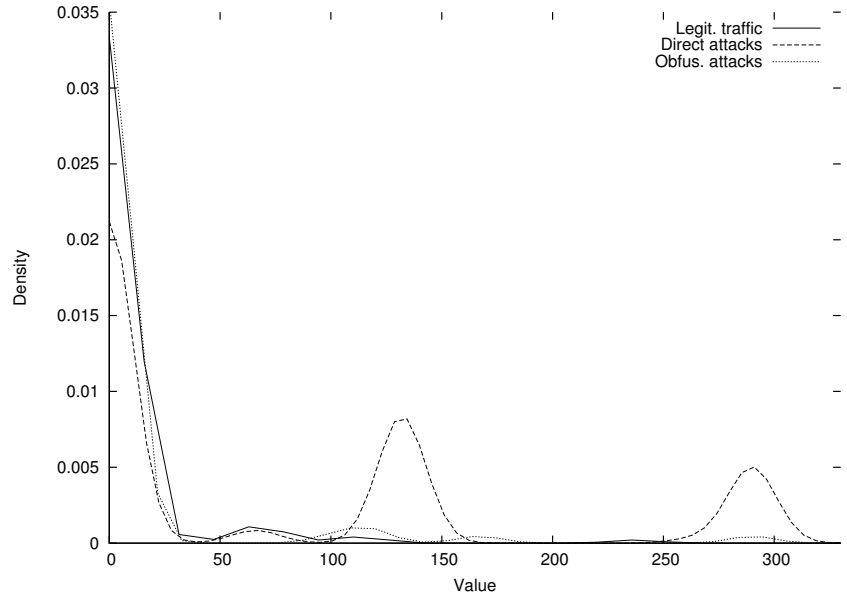

Figure G.4: Lengths of outbound packets occurred in the first 4 seconds of a connection (`OutPktLen4s10i[3]`)

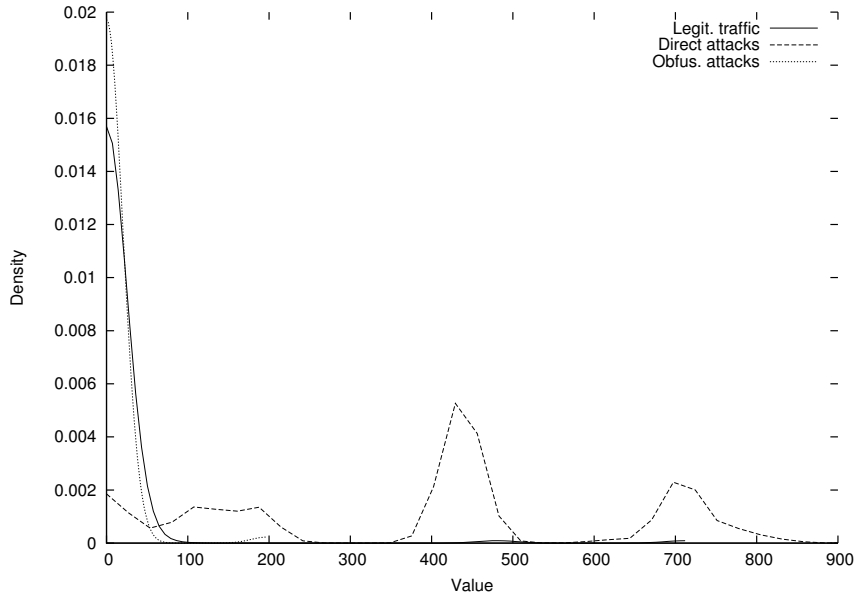

Figure G.5: Lengths of outbound packets occurred in the first second of a connection (`OutPktLen1s10i[1]`)

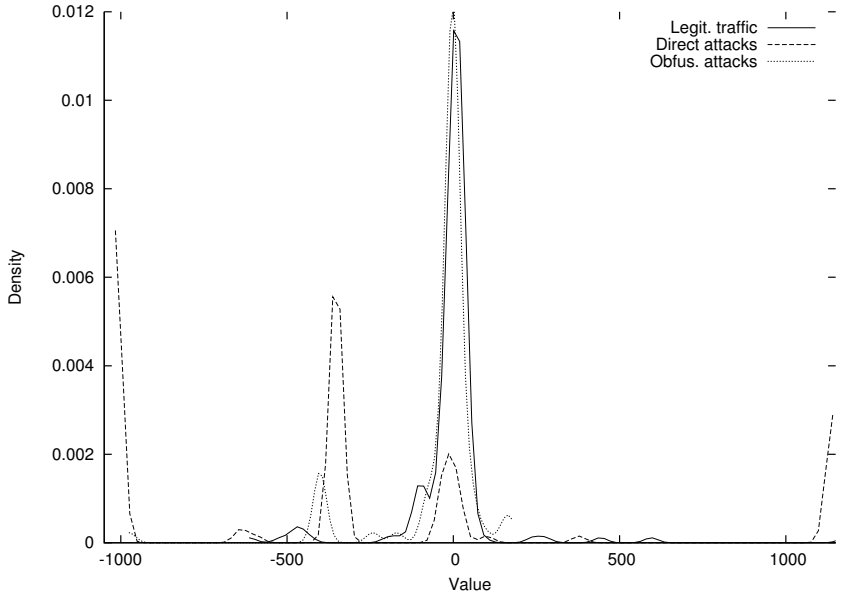

Figure G.6: Approximation of inbound communication by polynomial of 8 order (`PolyInd8ordOut[3]`)

Appendix H

Non-payload Based Obfuscations

Several examples of discriminating and obfuscated features are placed into the current appendix, as they are not representatives of particular category, and thus their outcome may be unclear or inexpressive in some ranges of the feature distribution. For more information about associated experiments see Section 8.8.

H.1 Discriminating Features

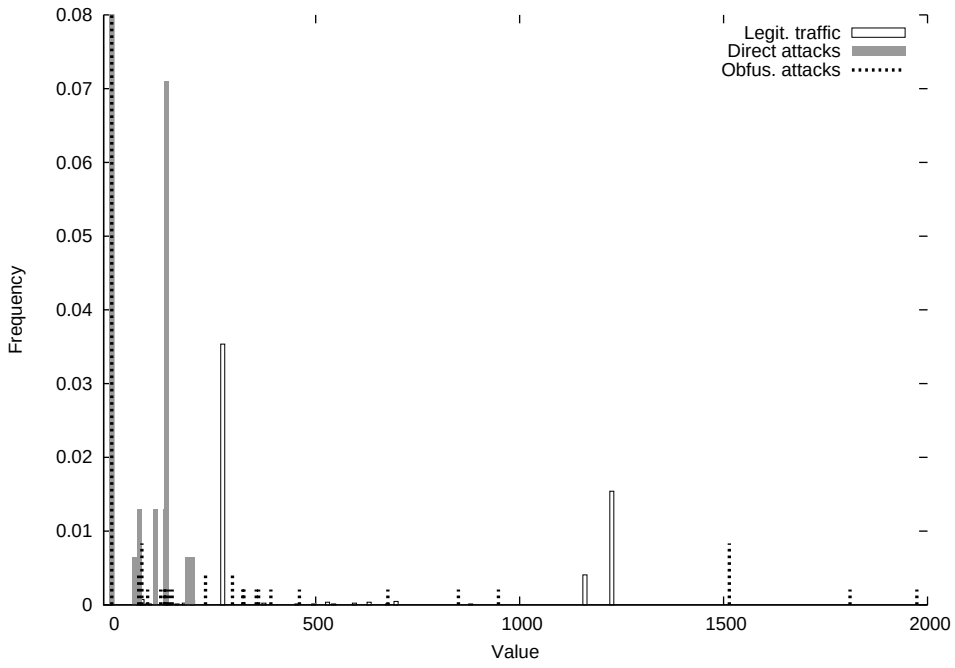

Figure H.1: Lengths of outbound packets occurred in the first 4 seconds of a connection (`OutPktLen4s10i[9]`)

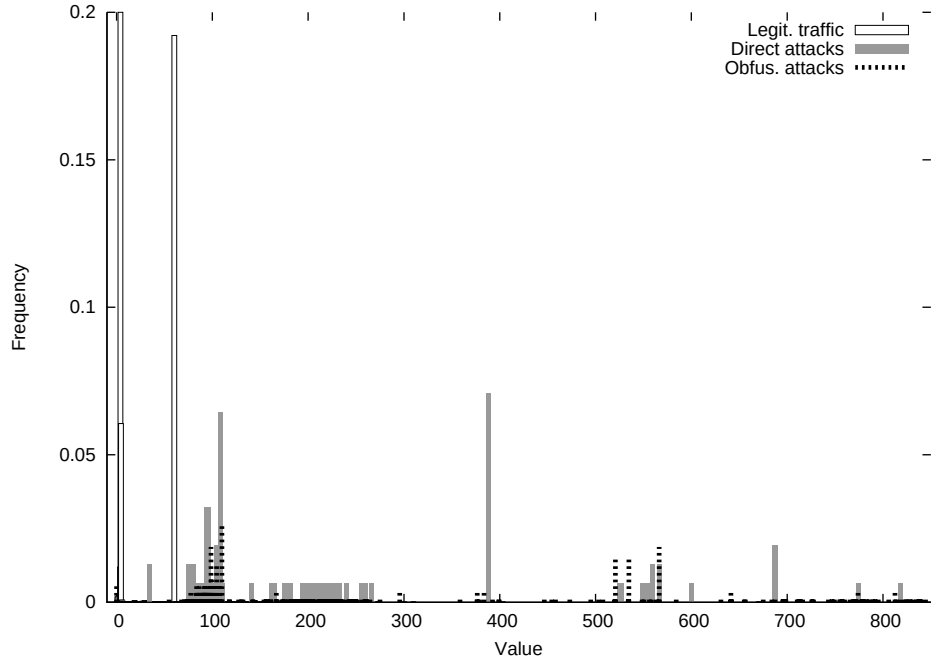

Figure H.2: Standard deviation of lengths of packets transferred from client to server (`SigPktLenOut`)

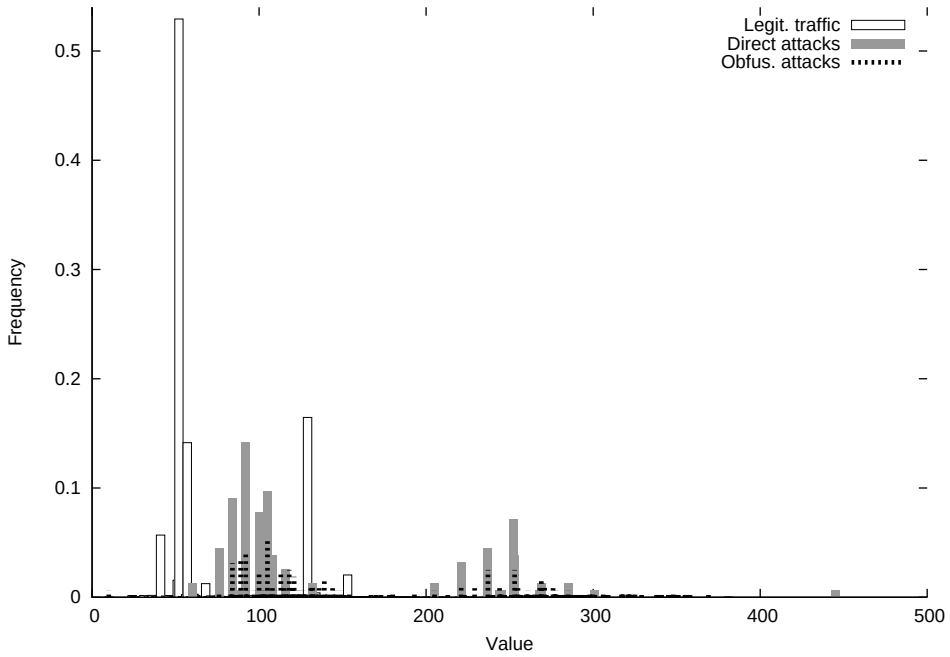

Figure H.3: Sub-interval of sum of TCP header lengths (`SumTCPHdrLen`)

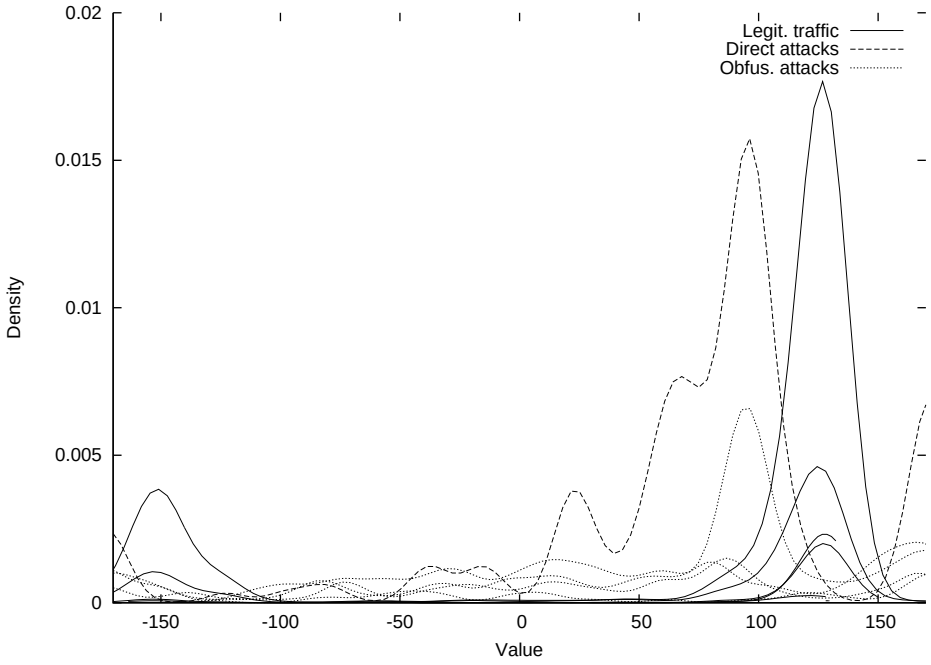

Figure H.4: FFT of packet lengths in both directions (`FourGonAngleNeg[5]`)