Czech Technical University in Prague

Faculty of Electrical Engineering

Doctoral Thesis

August 2016 ***Martin Grill***

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Combining Network Anomaly Detectors

Doctoral Thesis

Martin Grill

Prague, August 2016

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Supervisor: Ing. Tomáš Pevný, Ph.D.
Supervisor-Specialist: Martin Rehák, Ph.D., Ingénieur ECP

Acknowledgments

Foremost, I would like to express my deep and sincere gratitude to my supervisors Tomáš Pevný and Martin Rehák. Tomáš educated me generously in how to be a good researcher with his immense knowledge in the field of machine learning. He guided me in my research and pointed me in the right directions. Martin has introduced me into the field of network anomaly detection and supported me since the days I began working on the CAMNEP intrusion detection system as an undergraduate. He was mentoring and encouraging me in the past ten years.

My sincere thanks also goes to Michal Pěchouček who provided me an opportunity to join the Agent Technology Center group at the Czech Technical University. The group has been a source of friendships as well as good advice and collaboration. Namely I would like to thank to Viliam Lisý and Branislav Bošanský for sharing their thorough knowledge in the field of game theory.

I would like to thank my fellow graduate students and coworkers for providing a stimulating and fun environment in which to learn and grow. I am especially grateful to Karel Bartoš, Jan Jusko and Jan Stiborek with whom I have worked closely over the last seven years.

I am also very grateful to all collaborators, and undergraduates who contributed to this research. Namely I would like to thank Sebastián García with whom I collaborated on the creation of the labeled NetFlow datasets. I am also very thankful to Verónica Valeros for providing huge amount of high quality labels for the HTTP(s) access log data and Ivan Nikolaev for helping me with the implementation and experimental evaluation of the DGA malware detection algorithm.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by the ITC-A of the U.S Army (contracts N62558-07-C-001, W911NF-08-1-0250, W911NF-10-1-0070 and W911NF-12-1-0028); ONR Global under the Department of the Navy (contract N62909-11-1-7036); Air Force Office of Scientific Research, Air Force Material Command, USAF (grant FA8655-10-1-3016); Czech Ministry of Education (grants ME10051 and MEB111008) and the Czech Ministry of Interior (grants VG2VS/189 and VG20122014079).

Last but not the least, I would like to thank my family for all their love and encouragement. For my parents who supported me in all my pursuits and put me where I am today. And most of all for my loving, supportive, encouraging, and patient wife Tereza and daughter Adélka whose faithful support during the final stages of this Ph.D. is so appreciated. Thank you.

Abstract

The anomaly-based network intrusion detection systems (IDS) typically suffer from high false alarm rate rendering them useless in practice as the subsequent analysis done by the network operator is costly and can be done only for a small number of raised alarms. This thesis introduces several novel anomaly detectors and develop techniques for their combination to achieve much smaller false positive rates.

We propose an architecture of an IDS that uses a number of simple network anomaly detectors that are able identify anomalies relevant to malicious network communication using the NetFlow (CAMNEP IDS) or HTTP access log (Cisco Cognitive Threat Analytics — CTA) telemetry data. We introduce several novel network anomaly detection techniques that enrich the ensemble of the state-of-the-art network anomaly detection methods used in both detection systems. The detectors are designed to use different anomaly detection algorithms applied to different subsets of features to introduce diversity and detect wider range of malicious behaviors.

The outputs of the anomaly detectors are combined using two parallel aggregation functions constructed in supervised and unsupervised manner. The unsupervised combination uses a state-of-the-art method that is robust to presence of low accuracy detectors. The supervised combination is created using a novel technique that finds a convex combination of outputs of the anomaly detectors maximizing the accuracy in τ-quantile of the most anomalous samples. An extensive experimental evaluation and comparison to prior art on real network data using anomaly detectors of both CAMNEP and CTA intrusion detection systems shows that the proposed method not only outperforms prior art, but is also more robust to noise in training data labels, which is another important feature for deployment in practice.

Moreover, we propose to smooth the outputs of the ensembles by online Local Adaptive Multivariate Smoothing (LAMS) to further reduce the amount of the false positives. LAMS can reduce the number of false positives introduced by the anomaly detection by replacing the anomaly detector's output on a network event with an aggregate of its output on all similar network events observed in the past. The arguments are supported by extensive experimental evaluation involving ensembles of anomaly detectors of both CTA and CAMNEP intrusion detection systems. We also describe an effective implementation of the proposed solution to process large streams of non-stationary data.

Finally, the extensive experimental evaluation using real network data collected in a number of corporate networks with a large number of labeled samples shows that each of these techniques significantly improves the efficacy of the anomaly-based intrusion detection system.

Contents

Chapter 1
Introduction

Computer network threats against critical enterprise computing infrastructures have become increasingly sophisticated, targeted attacks are on the rise, and cybercriminals are launching their campaigns through a variety of legitimate vectors such as web or email, which are harder to identify. The increase in sophistication drives the need to deploy increasingly more sophisticated defense solutions. An essential component of the defense is Intrusion Detection System (IDS) [150] analyzing network traffic that crosses the defense perimeter looking for evidence of ongoing malicious activities (network attacks). When such an activity is detected, an alarm is raised and then analyzed by network administrator who determines the existence and scope of the damage, repairs it and improves the defense infrastructure against future attacks. The IDS systems are therefore typically used as a last line of defense, after policy enforcement systems (firewalls), proxies, intrusion prevention systems and other perimeter devices [117], and are designed to detect the attacks that successfully breached the perimeter.

IDS can be categorized according to different criteria: by their location (on a host, a wired network, or a wireless network), detection methodology (signature matching or anomaly detection), or capability (simple detection or active attack prevention) [150]. In the context of our work, the most important criterion is the categorization by the detection methodology, which is described in more detail below.

Signature matching techniques identify attacks by matching network packet (i.e. a unit of data carried by the network) contents against specific attack signatures. These signatures are created using already identified and well-described attack samples, which is time consuming and can take from couple of hours up to several days, which gives attackers plenty of time for their criminal activities. The biggest weakness of this solution is that it is detecting only known attacks, which can be due to smart evasion techniques used by malware limiting. With the growing proportion of encrypted traffic, use of self-modifying malware, and other evasion techniques, the use of a detection technique tailored to catch predefined known set of attacks is becoming increasingly irrelevant.

Anomaly-based detection tries to decrease the human work (e.g. manual creation of signatures) by building a statistical model of a normal behavior and detect all deviations from it. This enables to detect new, previously unknown attacks provided that their statistical behavior is different from that of the normal traffic. While anomaly-based methods are attractive conceptually, they have not been widely adopted. This is because they typically suffer from comparatively higher false alarm rate (not every anomaly is related to the attack) rendering them useless in practice, since network operator can analyze only few incidents per day [88, 87]. That is why the pattern based IDS are still widely used even when they are not able to detect new types of attack nor to find anomalous behavior of the network users.

This thesis focuses on the anomaly-based network intrusion detection systems analyzing the network telemetry collected at the perimeter. The main goal is to introduce additional techniques that would reduce the number of false alarms (false positives) to a level that makes the anomaly-based intrusion detection systems applicable in practice. In the next sections, we first discuss the network anomaly detection in more detail and then motivate the use of combination of a number of heterogeneous simple network anomaly detectors.

Network Anomaly Detection

The main assumption of the anomaly-based intrusion detection systems is that intruder's behavior is noticeably different from that of legitimate user and that many unauthorized actions are detectable as statistical anomalies [116]. The anomaly is defined [76] as an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.

The anomaly detector can be seen as one class classifier (this thesis uses the terms detector and classifier interchangeably) and categorized into two groups: *adaptive* and *static*, based on the fact that they use or do not use some kind of normal user model continuously learned from the data. Both groups are acting differently under different conditions. The static detectors are typically exploiting some domain knowledge and use a small subset of features and thresholding to search for the anomalies. The adaptive detectors, on the other hand, are maintaining models of normal user behavior or normal network state and labeling the deviations from this model as anomalous. The models of normal network behavior are generally unknown, differ widely among the networks, and it is very difficult to develop an anomaly detector in the strictest sense. Therefore, they typically utilize the assumption that most of the traffic on the network is normal, and anomalies within are related to malicious activities [130][1].

In general, the anomaly detector can have two[2] types of outputs represented by:

- **Label** (Hard decision) — each observed instance is given a normal or anomalous label, so there is no information about the certainty of the guessed label. Since most of the anomaly detection algorithms produce continuous value outputs, the common practice is to use thresholding to transform the continuous values to normal/anomalous label.
- **Score** (Soft decision) — a continuous *anomaly score* is assigned to each observed instance. The anomaly score (or degree of support) can be interpreted in various ways, the two most common being confidence and estimate of the posterior probability of the instance being an anomaly. The requirement of additional threshold parameter still holds, it is just moved to another step of the detection process.

This thesis focuses solely on the score producing anomaly detectors. They provide more information in form of their certainty of the current observation being anomalous. The certainty can be utilized in following combination process to gain higher accuracy of the whole system as motivated in the next section. Chapters 3 and 4 describe several novel network anomaly detectors in more detail to illustrate their simplicity and various approaches taken in the process of designing them.

Potential of Anomaly Detector Combination

It has been observed [75] that different classifier designs offer complementary information about the attacks, which could be combined to improve the performance of detecting wider range of intrusions. This thesis deals with the combination of multiple classifiers to gain higher accuracy and reduce the number of false positives.

[1] This assumption might not always hold, e.g. during botnet co-ordinated or DDoS attacks in which the number of attackers is really high, the majority of the traffic can be malicious, but this scheme is typically able to detect the beginning of such attacks which should be sufficient in the practice to identify the origins of the attack.

[2] It should be noted that the general classifiers can also output *Ranks*. In that case, classifier ranks all the observations in a queue with the observation at the top being the first choice. But this approach is not applicable for the one-class classifiers, so we will not consider them.

The multiple classifier systems are often practical and effective solutions for difficult pattern recognition tasks and have been successfully applied in remote sensing domain, person recognition, and medicine. In literature they can be found under various names, like combining of multiple classifiers [90], decision combination [78], mixture of experts [85], classifier ensembles [73], classifier fusion [94], consensus aggregation [57], dynamic classifier selection [61], hybrid methods [37] and others. We will refer to this approach as *detector combination* or *ensemble system.*

The strength of the ensemble systems comes from the concept called "wisdom of crowd" characterized by statement of Surowiecki [166]:

,,When our imperfect judgments are aggregated in the right way, our collective intelligence is often excellent".

This can be illustrated by one of the oldest and best-known combination strategies, the majority vote. It is a basic, but still powerful combiner that constructs the final decision via simple majority of the detectors outputs. The probability of making the correct decision with the majority vote P_{maj} depends on the probability of each detector providing the correct output that is:

$$P_{maj}(L) = \sum_{m=L/2+1}^{L} \binom{L}{m} p^m (1-p)^{L-m}, \tag{1.1}$$

where L is the number of detectors and p is their accuracy. The result, also known as the Condorcet Jury Theorem, is as follows; if $p > 0.5$, then $P_{maj}(L)$ is monotonically increasing in L and $P_{maj}(L) \to 1$ as $L \to \infty$. If $p < 0.5$, then $P_{maj}(L)$ is monotonically decreasing in L and $P_{maj}(L) \to 0$ as $L \to \infty$. If $p = 0.5$, then $P_{maj}(L) = 0.5$ for any L. This theorem highlights the benefit of combining reasonable detectors (i.e., with an accuracy $p > 0.5$) over the use of a single detector.

The usage of an ensemble of anomaly detectors in network intrusion detection systems is motivated by the fact that there are many features that have to be considered by the detection mechanism to detect the intrusions efficiently. It is very hard to take into account the domain knowledge as a whole in the design of the detector. It is also very difficult for individual classifiers to effectively process features that have very different semantic meanings [62]. More practical approach is to use multiple detectors, where each of them is specialized in detecting several types of anomalies using a small subset of features. In this way, the detection problem is divided into several simple sub-problems, each of them can be decomposed and analyzed in more detail. The specialized detectors are able to create more precise models of normal/anomalous behavior in comparison to the one detector that is analyzing all the features at once. Additionally, the individual anomaly detectors can be specialized in detection of some parts of the attack utilizing the structural knowledge to achieve higher precision [142]. Thus the intrusion detection domain can be seen as a union of different sub-domains, each with different properties of normal behavior and different types of anomalies.

All the benefits of using combination of multiple detectors for the intrusion detection can be summarized in following bullets:

- ensemble method takes advantage of the strong points of each individual detector to induce a better final outcome [63, 179, 90],
- reduction of false positives,
- even when using simple and not effective detectors, the combination can make the result strong [94],
- reduction of the complexity of the detectors – instead of having one really complex detector based on high-dimensional feature vector we have a number of simple low-dimensional detectors designed specially for some sub-domain that can process only the features that have common semantic meaning, this allows us to incorporate more precise sub-domain knowledge and model the normal/anomaly behaviors more precisely,

- ensemble detector systems are more robust to fluctuations introduced by dynamic environments,
- it is harder for the attacker to manipulate ensemble of detectors that are running in parallel, when compared to a single anomaly detector system [124, 147, 19].

Many researchers have demonstrated that the ensembles often outperform their base models (the component models of the ensemble) if the base models perform well on positive examples and tend to make errors on different examples [78, 168, 63, 154]. Therefore, the base detectors must exhibit some level of diversity among themselves [94] to make the ensemble system effective. The diversity can be achieved using different detection algorithms, training parameters or feature subsets. But we have to emphasize that uncorrelated features do not guarantee uncorrelated detectors outputs [62]. Typically, it is almost impossible to have an ensemble of statistically independent detectors but as shown in many works [122, 120] the ensemble of detectors violating this assumption outperformed the base detectors in various research fields.

This thesis proposes to use an ensemble of network anomaly detectors that are using various anomaly detection approaches, applied to different subsets of network features to introduce diversity in the ensemble.

1.1 Challenges in Network Anomaly Detection

Anomaly-based network intrusion detection systems face number of various problems that need to be addressed by researchers before they can be widely deployed. These rather hard problems include huge amount of data that needs to be analyzed, high false positive rates introduced by the anomaly detection paradigm and the lack of representative training data. This section discusses the individual problems influencing the choice of the solutions in this thesis in more detail.

Huge volume of data

Network security is clearly a Big Data problem. Network hosts generate lots of data every day. Essentially, one terabyte of data can be easily transfered over the perimeter of an enterprise network in one day. Such a large volume practically prevents deep analysis of all the data, such as deep packet inspection[3].

Many researcher therefore focus on the analysis of an aggregated information of the network communication in form of network traffic logs. But still the number of log records can easily outreach 10 billions per day[4], which requires the detection techniques to have very low computational (linear $O(n)$ or linearithmic $O(n \log n)$) and space complexity.

High cost of false positives

The common problem of all anomaly-based detectors is the amount of *false positives*, i.e. a normal or expected behavior that is identified as anomalous or malicious. Each false positive costs scarce analyst time because he needs to do the deep analysis of the incident to disprove the maliciousness which can take up to several days of work. Therefore many researchers have been searching for a way to create more precise anomaly detectors and a lot of various false positive reduction techniques have been introduced (see Section 2.3 for detailed review). The

[3] Deep packet inspection examines both the header and data part of each transferred packet, searching for protocol non-compliance, viruses, spam, intrusions, etc.

[4] CTA detection engine introduced in Chapter 4 processes at the time of writing this thesis more than 10 billion HTTP log records per day.

goal of all these techniques is to find the effective trade-off between false positives and attack detection accuracy (true positive rate).

Class imbalance

Since the anomalies are defined as rare instances in the data, it is natural that the distribution of normal and anomalous class will be skewed. This implies that the supervised techniques used in the field of network anomaly detection should be implemented via the cost-sensitive variation of the classification problem.

Few or no labeled data for training and validation

Obtaining labeled data in security domains and in the field of network intrusion detection especially can be difficult, time consuming, and expensive. Besides, labeled data frequently contains errors in labels of different sorts, for example some alerts might be missed and labeled as legitimate, or even worse, all samples of alerts of certain types might be missed and therefore labeled as legitimate.

There are no high-quality labeled data [52] available for training or validation of the anomaly detection systems. The creation of such a data is very difficult, expensive and time-consuming. Although the anomaly term is well defined, the boundary between the normal and anomalous behavior is not precise and the exact notion of anomaly is different for different application domains and context dependent. Next, the a priory probabilities of the anomalies are environment dependent.

Many researchers were trying to create labeled data for anomaly-based intrusion detection systems training and validation. The most famous labeled dataset created by the DARPA and the MIT Lincoln Laboratory group [72] has been largely criticized because the rare types of behavior were not represented proportionately rendering the dataset unrealistic [104].

Since the real network datasets are rarely publicly available due to the privacy constraints put on the network traffic some authors proposed to simulate the traffic [162]. However, it is really difficult to reproduce the pattern of anomalous or normal behavior in a dynamic environment such as computer network. It is also very difficult to create purely legitimate network traffic data [144]. Modeling behavior of a normal network user is very hard task because even normal user can switch between several distinct profiles of normal behavior.

Anomaly detectors are heterogeneous, with unknown characteristics, precision and error function

The anomaly detectors have typically unknown characteristics that are determined by the current deployment environment — each detector will act differently in different environments, the same anomaly can be scored differently by different anomaly detectors, when performed in different environments. Moreover, the detection accuracy of an anomaly detector varies for various types of anomalies. Detector that provides very good detection rate for one type of malicious behavior may completely fail for another type. The accuracy is also affected by the learning process of the detector and by the environment; the accuracy changes together with the changes of the network characteristics (i.e. network characteristics will be completely different during the day and at night). Some incidents can be efficiently detected only by one detector or by a combination of a subset of the detectors.

The performance of the detector can be negatively influenced by the side-effects of the attack itself. Adversary can even intentionally manipulate detectors learning process — anomaly should be a result of malicious action, but malicious adversaries often adapt themselves to make their actions appear as normal [147].

1.2 Proposed IDS Architecture

The above concerns motivated the architecture of the IDS used in this thesis. In this section, we will provide a brief overview of the architecture in order to be able to later present the specialized techniques of the individual components in the context of the whole system. The architecture is composed of four layers (see Fig. 1.1). Each consecutive layer is designed to reduce the number of false positives introduced by the anomaly detection principle.

- The first layer uses a number of network anomaly detectors that are using various anomaly detection methods, described in detail in Chapters 3 and 4, applied to different subsets of network features to introduce diversity among them.
- The second layer consists of two parallel ensembles of the network anomaly detectors created in both, unsupervised and supervised manner to provide good generalization and at the same time boost the efficacy against the known malicious samples.
- The third layer contains several Local Adaptive Multivariate Smoothing (LAMS) models, described in Chapter 6. These are designed to further reduce the number of false positives of the whole detection system by smoothing the outputs of the ensembles over time and feature space.
- The last layer then aggregates the results of individual models to construct the final output in which each flow is assigned with one anomaly score.

An extended version of the proposed architecture was successfully adopted in two network intrusion detection systems: CAMNEP and CTA. CAMNEP [141, 59] is a network anomaly detection engine, developed at the Czech Technical University and Cognitive Security[5], that processes NetFlow [30] records exported by routers or other network traffic shaping devices. Cognitive Threat Analytics (CTA) [29] engine, developed at Cisco Systems, analyzes HTTP proxy logs (typically produced by web proxy servers located on a network perimeter) to detect infected computers within the network. Although the proxy logs do not contain all the network host traffic (only HTTP(s) requests), the information about each request is richer than the NetFlow. Both the detection systems use similar architecture design that differs only in the anomaly detectors and the features and metrics of the LAMS models. In both systems, the output of the fourth layer is processed by following classification system [14, 53, 16, 15] that selects the top anomalous flows as identified by the proposed anomaly detection engine, clusters them into events and assigns labels and event severities using a set of supervised classifiers. The network analyst is then provided with a list of labeled network incidents sorted by severity which significantly reduces the amount of his work. The description of the classification system is out of the scope of the proposed thesis.

1.2.1 Anomaly Detectors

In the first layer, the captured network traffic data is analyzed by a set of anomaly detectors. Each detector works with a set of selected features that allows to identify the malicious behavior, rather than random, non-malicious fluctuations. As said above, we don't seek to build a single perfect anomaly detector, but rather to design the algorithms (such as those described in Chapter 3 and 4) that can be applied on diverse subsets of traffic features in order to define a multitude of

[5] Cognitive Security, a privately held, 28 person company headquartered in Prague, was acquired by the Cisco Systems in 2013.

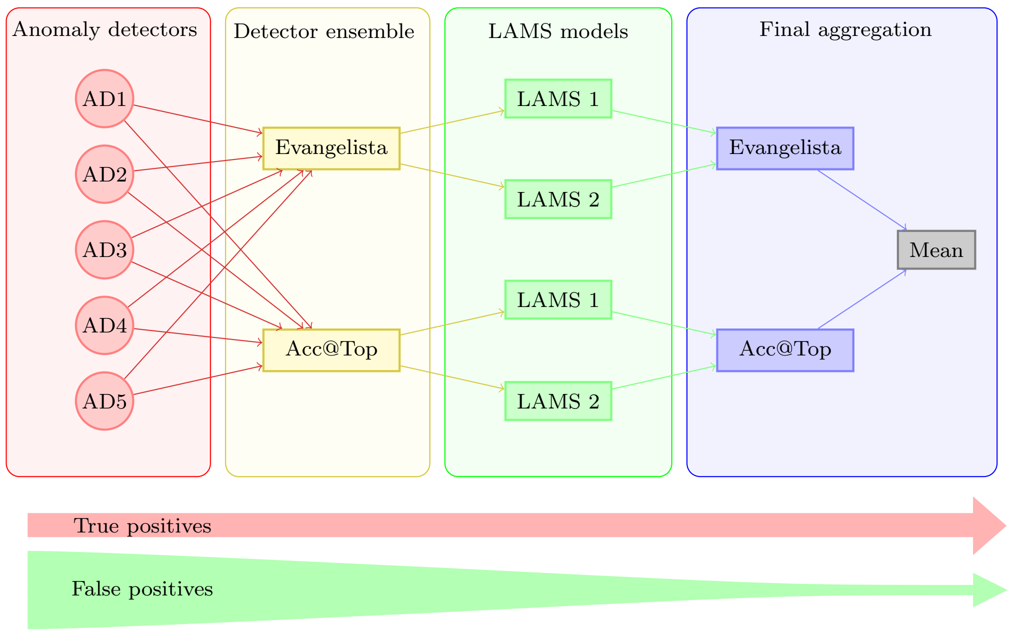

Fig. 1.1: Proposed IDS architecture containing four layers. In the first layer the data is processed by a number of simple network anomaly detectors. Next layer consists of two parallel ensembles that reduce the error of the individual anomaly detectors via aggregation. The false positives are even further reduced by the ensemble of LAMS models in the following layer. Finally, the anomaly scores of the LAMS models are aggregated to produce the final scoring. Each layer reduces the amount of false positives unaffecting the overall recall as visualized by the two arrows.

heterogeneous anomaly detectors. Additionally, the individual anomaly detectors are using low complexity algorithms enabling the near real-time analysis of huge amount of network data.

Each anomaly detector is defined by three components: the algorithm, traffic features, and parameters. The algorithms presently used in CAMNEP or CTA are of two different categories:

- Statistical anomaly detectors are based on a direct application of statistical methods (such as PCA) on a selected subset of traffic features. In our methodology, we typically identify promising approach to statistical anomaly detection, test it on a large number of traffic feature projections and select one or more combinations. The parameters are essentially used only to fine-tune the process from the performance characteristics and their influence is minor.
- The knowledge based detectors use a different approach. The process of their creation is not method-centric, but rather data-centric. The designer typically starts by noticing an exploitable characteristics of network traffic and analyzes whether the characteristics are relevant for the detection of a network attack. In the second stage, the designer identifies a set of features that needs to be measured and selects the simplest possible detection algorithm which is applied on the features. Parameters of the anomaly detector typically play an important role, and some of the methods may initially be based on thresholds or other baselines identified in the design phase. The resulting anomaly detectors are then deployed and tested on wide range of real network attacks captured in test datasets, prior to their deployment.

Both approaches to detector construction complement each other and help to achieve better efficacy. The statistical detectors typically bring more positives in general — their application discovers innovative attacks and unusual behaviors alike. By their nature, they tend to be stable with respect to intentional manipulation on a single host level. On the other hand, the knowledge-based detectors embody a specific knowledge about one or more traffic characteristics as specified by the designer. They are frequently designed as a reaction to a new type of anomalies identified by progressively introduced statistical detectors. In this case, they can bring additional insight that helps the system to suppress some of the false positives introduced by the statistical detectors, while leaving the true positives unaffected.

In this thesis we propose five novel network anomaly detection algorithms of both types described in detail in Chapters 3 and 4. They use various anomaly detection principles as well as network features extracted from either NetFlow of HTTP proxy logs. Each anomaly detection method assigns an anomaly score to each network observation (NetFlow record or HTTP proxy log record), denoted in further text as network *flow*[6] or *sample*. Anomaly score specifies the degree to which a flow is considered anomalous. Network incident analyst may choose either to analyze the top few anomalies or use cut-off threshold to select all network flows exceeding the threshold for further analysis.

1.2.2 Detector Ensemble

The high sensitivity (measured as *recall*) of the individual detectors is a crucial element of any intrusion detection system. However, by design, no individual detector can achieve a high-enough precision to be useful on its own. Therefore, many intrusion detection systems rely on the ensemble approach [62, 154]. They use an ensemble of relatively simple and heterogeneous detectors, where some of them can be specialized to a particular type of intrusions, whereas others can be general anomaly detectors capable of detecting previously unseen attacks at the expense of higher false alarm rates.

As discussed earlier, such a setup has multiple advantages, including faster processing of the data stream, lower complexity of the detectors, simpler inclusion of the domain knowledge into the system. Furthermore, the errors of individual detectors cancels out [41, 128] decreasing the false positive rate in effect. Arithmetic mean or majority vote aggregation functions are preferred if the individual classifiers have roughly the same accuracy [94]. This is not the case of the proposed anomaly detection system because some of the detectors are specialized to detect only some type of malicious behavior. For such a systems Evangelista [47] suggested to use mean of mean and maximum scores of the scores assigned by the individual detectors within the ensemble, as it should be more robust to presence of poor detectors without knowing which ones are poor. This combination function (further called *Evangelista aggregation*) favors the highest anomaly scores and reduces the effect of poor detectors (see Section 2.2.3 for detailed description).

Additionally to the unsupervised Evangelista aggregation, we propose to use a parallel aggregation (denoted on the Figure 1.1 as Acc@Top), created in the supervised manner using the labeled data of currently known malicious samples to improve the recall of the whole system on the known network attacks or malware. This, however, introduces new problem of how to create such an ensemble. Although a vast prior art on the problem exists (see Section 2.2 for detailed review), we believe that particularities of the security domain, namely the highly imbalanced ratio of non-alarm and alarm samples in the data, lack of accurately labeled datasets, and the need of extremely low false positive rates, call for a tailored solution which is presented in Chapter 5.

1.2.3 LAMS Models

The amount of false positives of the ensembles can be further reduced using a Local Adaptive Multivariate Smoothing (LAMS) models introduced in Chapter 6. The models smooth the out-

[6] We will therefore refer to the individual detectors as *flow-based anomaly detectors.*

puts of the ensembles simultaneously over time and feature space to have more robust estimate of the true anomaly. This approach reduces the amount of unstructured false positives caused by stochasticity of the network traffic (see Section 6.1 for definition).

There are multiple LAMS models defined using various subsets of network traffic features exploiting again the properties of ensemble systems. The expectation is that errors of outputs of different LAMS models will be uncorrelated and will, similarly to the anomaly detectors, cancel out in the final aggregation phase.

1.2.4 Final Aggregation

In the final layer, the anomaly scores of each of the ensembles that are adjusted by the LAMS models are first aggregated using the Evangelista aggregation function, motivated by the same reasons as described in the detector ensembles in Section 1.2.2. The anomaly scores of the two sources, Evangelista and Acc@Top ensembles are then aggregated into the final anomaly score using maximum of the individual anomaly scores of each network flow.

1.3 Key Contributions

This thesis makes four main contributions in the field of flow-based network anomaly detection published in international journals, peer-reviewed conferences and workshops.

- **Development of new, more sensitive anomaly detection algorithms [67, 70, 66, 125, 59]** (presented in Chapters 3 and 4). We propose five novel anomaly detection algorithms that use both NetFlow and HTTP access logs to detect network anomalies that can be related to malicious activities. Each anomaly detector is experimentally evaluated on a real network data to prove its effectiveness in practice.
- **Large margin aggregation [68]** (Acc@Top described in Chapter 5) is a novel algorithm for finding a convex combination of anomaly detectors maximizing accuracy at τ-quantile of the returned samples, which is a scenario frequently appearing in the security field. Since the labeled data is in our domain rarely perfect, an emphasis is put on evaluation, involving both HTTP and NetFlow anomaly detectors, eight types of combination functions, 34 different network captures containing more than 20 million samples of behavior under various types of labeling noise.
- **Local adaptive multivariate smoothing [69]**(described in Chapter 6) is a theoretically sound method that is able to decrease the false alarm rate of anomaly-based intrusion detection systems. The technique can reduce large portion of false positives introduced by the anomaly detection by replacing the anomaly detector's output on a network flow with an aggregate of its output on all similar network flows observed in the past.
  Structured and unstructured false positives are mathematically formulated and it is argued why unstructured false positives are more difficult to white-list or remove. We prove under mild assumptions that the proposed method reduces the amount of unstructured false positives caused by stochasticity of the network traffic. The arguments are supported by extensive experimental evaluation on both CAMNEP and CTA anomaly detection engines.
- **Network anomaly evaluation dataset [59]** is a labeled dataset containing network traces of 13 different scenarios of running malware from seven different families. The dataset is captured from the Czech Technical University network, containing network traces of real users together with unmodified, unrestricted malware samples ran inside number of virtual machines. The published dataset is recognized by the community as one of the most valuable datsets for evaluation of the network detection systems [106, 18, 182, 27].

Since each of the proposed algorithms is specialized in detecting specific type of network behavior, the description of the relevant subsets of the dataset is left for the individual experimental sections throughout this thesis (the undivided description of the whole dataset can be found in [59]). Additionally, some of the experiments presented in this thesis are using labeled datasets that are not included in the publicly available set due to the legal issues.

1.4 Outline of the Thesis

This thesis is structured as follows:

Chapter 2 reviews related work in the field of network anomaly detection with emphasis on the methods that are used in one of the proposed anomaly detection engines in Section 2.1. Next, Section 2.2 reviews the ensemble systems in general as well as the specific techniques used in the field of network anomaly detection. Finally, Section 2.3 reviews the state-of-the-art in the field of false positive reduction of the network intrusion detection systems, relevant to the approach proposed in Chapter 6.

Chapter 3 and **Chapter 4** describe the CAMNEP and CTA detection engines that are able to identify malicious network communication from NetFlow and HTTP access logs respectively. We introduce several novel network anomaly detection techniques that enrich the ensembles of the state-of-the-art network anomaly detection methods used in both the detection systems. The detectors are constructed using both statistical and knowledge based approaches and use subset of the network features to detect specific type of malicious behavior. For this reason, each of the detectors has a specific settings of the experimental evaluation that are together with the results presented with the description of the method.

Chapter 5 presents a robust ensemble construction technique, that is using the anomaly detectors described in previous chapters to create an ensemble with much higher accuracy on the labeled samples. The experimental evaluation of the proposed method show that it outperforms the state-of-the-art methods for optimizing the accuracy at top.

Chapter 6 introduces Local Adaptive Multivariate Smoothing (LAMS) model that is using smoothing of the anomaly scores to even further reduce the false positives of the whole system. It is shown that the LAMS models reduce the amount of false positives caused by the stochasticity of the network traffic.

Chapter 7 experimentally evaluates all parts of the system together and shows that each of the additionally introduced layers heavily reduces the amount of false positives. The experiments were conducted on large number of real network datasets containing more than 5,000 malicious samples to prove the effectiveness of the proposed techniques in practice.

Chapter 8 summarizes the contributions of this thesis and provides a list of publications and patents of the author.

Chapter 2
Background and State-of-the-Art

In the previous chapter, we introduced the network anomaly detection and motivated the use of an ensemble of simple network anomaly detectors to create an intrusion detection system with reduced amount of false positives. In this chapter, we review related work in the field of network anomaly detection, multiple classifier systems and false positive reduction.

We start with a general overview of the existing network anomaly detection methods in Section 2.1. Next, Section 2.2 reviews the prior art that deals with multiple classifiers systems with a strong emphasis on the ensemble approach to the network anomaly detection. Finally, Section 2.3 surveys existing approaches for further false positive reduction to cover the related work of the LAMS models introduced in Chapter 6.

2.1 Network Anomaly Detection

A great deal of research effort has gone into developing anomaly detection systems since it was proposed by Dorothy Denning in 1987 [38]. There are several survey papers [46, 121, 26] that summarized number of various detection methods. Patcha *et al.* [121] categorized anomaly detection approaches using statistical or signal processing techniques (e.g., wavelets [10], Kalman filter [158]), concepts from the machine learning domain (e.g., PCA [95], Bayesian networks [161]), or methods applied for data mining tasks (e.g., clustering [177]).

In this section, we focus only on the network anomaly detection methods related in some way to CAMNEP or CTA systems or to one of the newly proposed anomaly detection algorithms introduced in Chapters 3 and 4.

2.1.1 Lakhina Volume

The Principal Components Analysis (PCA) is a popular method in the network anomaly detection. The volume prediction algorithm presented by Lakhina *et al.* [95] uses the PCA algorithm to build a model of traffic volumes from individual sources using the NetFlow information. The observed traffic for each source IP address with a non-negligible volumes of traffic is defined as a three dimensional vector: the number of flows, number of bytes and number of packets transfered from the source IP address. The traffic model is defined as a dynamic and data-defined transformation matrix that is applied to the current traffic vector. The transformation splits the traffic into normal (i.e. modeled) and residual (i.e. anomalous). The method returns the residual amount of flows, packets and bytes for each source IP address. These values define the context (identical for all the flows from the given source). An anomaly is determined by transforming the 3D context into a single value in the $[0, 1]$ interval.

2.1.2 Lakhina Entropy

The entropy prediction algorithm presented by [96] is based on the same PCA-based traffic model as described in previous section, but uses different features. It aggregates the traffic from the individual source IP addresses, but instead of traffic volumes, it predicts the entropies of destination IP addresses, destination ports and source ports over the set of context flows for each source. The context space is therefore three dimensional. An anomaly is determined as the normalized sum of residual entropy over all three dimensions. The metric is simple: a function measures the difference of residual entropies between the flows and aggregates their squares.

2.1.3 KGB

Pevný *et al.* [125] proposed method inspired by Lakhina's detector that uses entropies of IP addresses and ports to model network host's behavior. The main difference to the work of Lakhina is in the aggregation. Lakhina aggregated flows on the level of peering links in the network, while Pevný's method models the individual hosts. This difference allowed finer resolution of the attacks. Furthermore, Pevný reduced the complexity of the algorithm and amount of data needed for training, which allows to perform detection on high-speed networks in near real-time manner.

2.1.4 MINDS

The Minesota Intrusion Detection System (MINDS) [45] builds a context information for each evaluated flow using the following features: the number of flows from the same source IP address as the evaluated flow, the number of flows toward the same destination host, the number of flows towards the same destination host from the same source port, and the number of flows from the same source host towards the same destination port. CAMNEP system uses an extended version of the original MINDS system, which also uses a secondary window defined by the number of connections in order to address slow attacks. The anomaly value for a flow is based on its distance to the normal sample. The metric defined in this four-dimensional context space uses a logarithmic scale on each context dimension, and the marginal distances are combined using the L2 distance. Finally, the variance-adjusted difference between the floating average of past values and the evaluated flow on each of the four context dimensions is used to estimate the anomaly score of the current flow.

2.1.5 Xu

In the algorithm proposed by Xu *et al.* [178], the context of each flow to be evaluated is created with all the flows coming from the same source IP address. For each context group of flows, a three-dimensional feature vector is built with the normalized entropy of the source ports, the normalized entropy of the destination ports, and the normalized entropy of the destination IP addresses. The anomalies are determined by a classification rules that divide the traffic into normal and anomalous. The distance between the contexts of two flows is computed as the difference between the three normalized entropies, combined using L2 distance. The implementation used in CAMNEP system of the algorithm is close to the original, which was further expanded in [177], except for the introduction of new rules defining a horizontal port scan as anomalous.

2.1.6 TAPS

The TAPS method [160] is, contrary to the previous anomaly detectors, a knowledge based detector designed to detect horizontal and vertical network scans (i.e. scanning for running hosts and host's open ports respectively). The algorithm considers only the traffic sources that created at least one single-packet flow during a particular observation period. These preselected sources are then classified using the following three features: number of destination IP addresses, number of destination ports and the entropy of the flow size measured in number of packets. The anomaly value of the source IP address is based on the ratio between the number of unique destination IP addresses and destination ports. When this ratio exceeds a predetermined threshold the source IP address is considered as a scan origin. The original method suffered of an unusually high number of false positives. Therefore, in CAMNEP system, the method was extended by the flow size entropy to achieve better results.

2.1.7 Detection Based on TCP Flags

Each packet transfered over the network has a TCP flag that indicates the current connection state (see Section 3.1.1 for detailed description). Detection of malicious behavior using TCP flags appears mainly in solutions that are using pattern matching approaches. Roesch [143] developed SNORT rules, that include several TCP flag patterns, to detect malicious activities. Similarly, Staniford *et al.* [161] used SNORT TCP flag rules to filter illegal flag combinations in the packets.

Yoo [183] created finite state automata, in which the transitions are the individual TCP Flags and the state represents the current state of the communication, with only one final state that corresponds to the connection closure. The proposed system is able to detect invalid TCP flag sequences, that are not accepted by the defined automata.

Mahoney *et al.* [105] estimated probabilities of various TCP packets header fields (including TCP flags) using previously seen data. The anomaly detection is then based on the assumption that the events that occur with probability p should have anomaly score of $1/p$ making the rare occurrences anomalous.

All the presented solutions are using TCP flag information contained in individual packets and won't be able to process aggregated TCP flag information contained in the NetFlow. In Section 3.1 we present a novel network anomaly detection method that is using the PCA to detect malicious network traffic using the aggregated TCP flag information contained in the NetFlow.

2.1.8 DGA Detection

The Domain Generating Algorithm (DGA) is a technique typically used by malware to search for the Command and Control (C&C) servers by periodically generating a list of domains and trying to contact them (detailed description of the DGA can be found in Section 3.3). The first report on malware using the DGA was made by Stone-Gross *et al.* [34] in 2005 who reverse engineered Torpig malware and found the DGA algorithm.

Antonakakis *et al.* [8] presented a method that leverages the idea that bots from the same botnet generate similar non-existent domains (NXDomain) [12]. They were able to identify and classify botnets using a combination of clustering and supervised learning. Similarly, Guerid *et al.* [136] analyzed DNS traffic to identify and cluster various botnets according to their NXDomain responses.

Zhou *et al.* [190] leveraged the idea that every domain name in the domain group generated by one botnet using DGA is often used for a short period of time and have similar life and query style. They clustered all the requested domains according to the top level domain and IP. Then the clusters were compared to see if there are some clusters with similar lifetime span and similar visit time pattern that were then reported as malicious.

A number of researchers tried to use the fact that algorithmically generated domain names have an alphanumeric distribution different from legitimate domain names, as observed by McGrath and Gupta [108]. Yadav *et al.* [180] used various metrics, such as Kullback–Leibler distance, Edit distance and Jaccard index, to identify patterns inherent to algorithmically generated domains. Mae *et al.* [103] employed statistical learning techniques based on lexical features (domain name length, host name, number of dots in URL, etc.). However, these techniques are ineffective against modern DGAs that use English dictionary words with slight modifications like adding suffixes -able, -hood, -ment, -ship, etc.

The aforementioned methods use information contained in the DNS resolve packets, which means that they need to have a access to a DNS server capable of logging all the requests or capture all the packets between the users and DNS servers. We are not aware of a method that uses only NetFlow information. Furthermore, techniques based on clustering of NXDomains suffer from a great amount of false positives generated from common typos and misconfigured applications, e.g. a common typo in google.com — let us say gooogle.com, can create a cluster of all users that made the same typo that would be labeled as DGA malware.

In Section 3.3 we introduce a novel method for detecting hosts infected with DGA-based malware that uses only the information contained in the NetFlow. The NetFlow does not contain any information about the requested host name or any information about the DNS resolves, so the above described methods are not applicable to the NetFlow data.

2.2 Ensemble Systems

This section reviews the available literature related to combining parallel classifiers or anomaly detectors. Although we will mention some of the combination strategies for labeled (hard decision) output detectors, we want to mainly address the combiners that work with continuous valued detectors as they are the most relevant to the work proposed in this thesis.

During past two decades many methods to combine classifiers have been developed. They can be divided into two general groups: methods for creating ensemble members and methods for combining classifiers in ensembles.

The methods for creating ensemble members operate on classifiers and put more emphasis on a development of the ensemble's structure than the combination process itself [78, 85, 64]. The most popular representatives of this group are boosting and bagging algorithms which both rely on the voting technique for the aggregation. Next, there are works introducing other methods for building ensemble of classifiers such as Bayesian averaging, random forests, etc. We do not consider these as relevant, since we focus on the combination of existing classifiers, not on creating new ones. Furthermore, these techniques require a lot of labeled data for training, and it is unrealistic to obtain enough labeled data in the network security field.

The second group of methods operates on classifiers' outputs searching for the most effective combination. There are two main approaches for combining classifiers in multiple classifier systems: *ensemble* and *modular*. The ensemble is commonly used for combining a set of redundant classifiers. The redundancy occurs because each classifier provides a solution to the same task. This is in contrast to the modular approach in which the task is decomposed into a number of sub-tasks. Each module is concerned with finding a solution for one sub-task. To complete the whole task, each component is expected to contribute.

The combination strategies can be categorized into three groups [94]: unsupervised, semi-supervised, and supervised. The unsupervised combination strategies (also referred to as non-trainable) do not need any labeled data for their construction. The semi-supervised combination strategies are trained using a mixture of unlabeled and small amount of labeled data. The supervised combination strategies need larger amount of labeled data for training. The overview of all the discussed methods together with their relevance to the defined groups can be found in the Table 2.1.

	Section	Score	Label	Supervised	Semi-Supervised	Unsupervised
Arithmetic mean	2.2.1	X	–	–	–	X
Other nonparametric	2.2.2	X	–	–	–	X
Evangelista combination	2.2.3	X	–	–	–	X
Ordered weighted average	2.2.4	X	–	–	–	X
Weighted average	2.2.5	X	–	X	x	–
Voting methods	2.2.6	–	X	x	x	X
Hierarchical mixture of classifiers	2.2.7	X	X	X	–	–
Consensus theory	2.2.9	x	X	–	–	X
Bayes combination	2.2.10	X	–	X	x	–
Dynamic classifier selection	2.2.11	X	X	X	x	–
Dempster-Shafer combination	2.2.12	X	X	-	-	X

Table 2.1: Overview of the discussed combining methods and their categorization.

2.2.1 Arithmetic Mean

The simplest non-parametric combination strategy for the continuous-valued output detectors is the *average* or *arithmetic mean.* Let $\mathcal{H}_m = \{h_1, \ldots, h_m\}$ be a set of m detectors and the output of h_i for sample $\mathbf{x}$ is $h_i(\mathbf{x}) \in \mathbb{R}$, then the average gives output $H(\mathbf{x})$ as

$$H(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} h_i(\mathbf{x}). \tag{2.1}$$

This approach does not need any other interventions, prior knowledge about the detectors or the environment the ensemble system is running on. It is stable, although not always most accurate combiner. Usage of this rule is based on the assumption that different detectors exhibit similar accuracies and make uncorrelated errors. In that case the errors are averaged out. There are many detailed studies that have shown simple averaging to be a very effective combination function, particularly in large complex data sets [179]. Because of its simplicity and very good accuracy [191] it is one of the most popular solutions for the unsupervised classifier combination that can heavily reduce the error of the base classifiers when their errors are uncorrelated as shown in the following theorem.

Theorem 1 *If we assume that errors of individual detectors are uncorrelated and have zero mean than the error of average of m detectors is smaller by factor m than the average error of the individual detectors.*

Proof. The output of each detector can be written as true value $f(\mathbf{x})$ plus an error $\epsilon_i(\mathbf{x})$, i.e.

$$h_i(\mathbf{x}) = f(\mathbf{x}) + \epsilon_i(\mathbf{x}),\ i = 1, \ldots, m.$$

Then the mean squared error of h_i can be written as

$$err(h_i) = \int \left(h_i(\mathbf{x}) - f\left(\mathbf{x}\right)\right)^2 p(\mathbf{x})d\mathbf{x} = \int \epsilon_i(\mathbf{x})^2 p(\mathbf{x})d\mathbf{x}$$

and the average error made by the base detectors is

$$\overline{err}(h) = \frac{1}{m}\sum_{i=1}^{m}\int \epsilon_i(\mathbf{x})^2 p(\mathbf{x})d\mathbf{x}.$$

Similarly, the expected error of the ensemble that uses the arithmetic mean of the base detectors can be derived as

$$err(H) = \int \left(\frac{1}{m}\sum_{i=1}^{m} h_i(\mathbf{x}) - f(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x} = \frac{1}{m^2}\int\left(\sum_{i=1}^{m}\epsilon_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x}.$$

The zero mean of the detector errors gives us

$$\int \epsilon_i(\mathbf{x})p(\mathbf{x})d\mathbf{x} = 0, \forall i \in N.$$

From the assumption that errors of the base detectors are uncorrelated we get

$$\int \epsilon_i(\mathbf{x})\epsilon_j(\mathbf{x})p(\mathbf{x})d\mathbf{x} = 0, \forall i,j \in m, i \neq j.$$

Than the error of the ensemble

$$err(H) = \frac{1}{m^2}\int\left(\sum_{i=1}^{m}\epsilon_i(\mathbf{x})\right)^2 p(\mathbf{x})d\mathbf{x} =$$

$$\frac{1}{m^2}\int \sum_{\substack{i,j\in 1..m\\ i\leq j}} (2-\delta_{ij})\epsilon_i(\mathbf{x})\epsilon_j(\mathbf{x})p(\mathbf{x})d\mathbf{x} =$$

$$\frac{1}{m^2}\int\sum_{i=1}^{m}\epsilon_i(\mathbf{x})^2 p(\mathbf{x})d\mathbf{x} = \frac{1}{m}\overline{err}(h)$$

□

However, it should be noted, that the above derived reduction of error is generally hard to achieve as the assumption that the errors of the base detectors are uncorrelated is almost always violated.

The first explicit use of an ensembles in anomaly detection [97] used the arithmetic mean to aggregate results of a set of anomaly detectors. The method employed feature bagging to create a diverse set of anomaly detectors and their outputs were fused either by the mean of their anomaly scores, or by picking k most anomalous samples from each detector (*breadth-first* strategy).

He *et al.* [77] developed a Subspace Outlier Ensemble using 1-dimensional subspaces (SOE1) for mining anomalies in categorical data. Each subspace is modeled via simple histogram and the final result is aggregated using product or sum of individual anomaly scores.

Similarly, Pevný [123] used histograms trained on a number of random projections. The final anomaly is obtained using average of probability estimates of the individual projection vectors.

2.2.2 Other Simple Non-parametric Combination Strategies

The other non-parametric combinations include minimum, maximum, median, trimmed mean and geometric mean [91]. The minimum combination function is the most pessimistic choice as the anomaly is raised only if it is supported by all the ensemble members. This results in really small false positive rate but the ensemble will miss many positives that can be identified only by one or a subset of the anomaly detectors. At the other extreme, the maximum function will raise an anomaly if at least one detector raises an anomaly. The maximum will result in an ensemble with high recall but usually low precision. The product rule is a good approach if individual classifiers are independent. It fails if the anomalies are zero or very small. The median seems like robust solution for the combination, but in ensemble of heterogeneous detectors with various precisions for various classes can be, similarly to the simple average, really inefficient, when there is only one precise detector for some class of anomalies.

Kuncheva [93] theoretically studied minimum, maximum, average, median and majority vote, together with the single classifier. She gave formulas for the classification error of the presented methods, assuming that the estimates are independent and identically distributed. She showed that for normally distributed errors, the fusion methods had very similar performance, but, for the uniformly distributed error, the methods differed significantly.

2.2.3 Evangelista Aggregation Function

Shanbhang *et al.* [154] experimentally compared several static combination functions, namely *mean*, *median*, *minimum*, *maximum*, and *mean of maximum and mean* in the field of network intrusion detection. According to their results, the mean of maximum and mean, proposed by Evangelista [47], was the most effective as it was more robust to presence of poor detectors without knowing which ones are poor.

This nonparametric function, hereinafter referred to as *Evangelista aggregation function* H_{Evan}, can be formally defined as

$$H_{Evan}(\mathbf{x}) = \frac{1}{2}\left(\frac{1}{m}\sum_{i=1}^{m} h_i(\mathbf{x}) + \max_j h_j(\mathbf{x})\right), \tag{2.2}$$

where $\mathcal{H}_m = \{h_i : \mathcal{X} \mapsto \mathbb{R}\}_{i=1}^{m}$ is a set of m anomaly detectors.

The reason for good efficacy of this combining rule is that the max function selects opinion of the most sensitive detector that produces the highest value of an anomaly at any particular instance. Using the max function in conjunction with the arithmetic mean makes such a rule robust in all cases.

Because of these properties the Evangelista aggregation is used in both CTA and CAMNEP systems to aggregate the scores of the individual anomaly detectors forming one of the proposed ensembles (see Section 1.2 for more details).

2.2.4 Ordered Weighted Average

Another simple approach that belongs to the unsupervised group of combination strategies is the *ordered weighted average* [181, 24] (OWA). The ordered weighted average does not attach specific coefficient to a classifier, but the outputs of the classifiers are first sorted in descending order and then a weighted sum is calculated using the coefficient associated with the place in the ordering. OWA operator H^{OWA} is defined as

$$H_{\alpha}^{OWA}(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i h_{\pi(i)}(\mathbf{x}), \tag{2.3}$$

where π is the permutation of the indexes so that the $h_{\pi(j)}(\mathbf{x})$ is the j-th largest value from set of the classifiers' outputs $\{h_1(\mathbf{x}), \dots, h_m(\mathbf{x})\}$ and $(\alpha_1, \dots, \alpha_m)$ is a collection of weights ($\sum_{i=1}^{m} \alpha_i = 1$ and $\alpha_i \geq 0, \forall i$) assigned to a position in the ordered list of anomaly scores. Thus the fundamental aspect of these operators is the re-ordering step.

The ordered weighted average combination was successfully used in the CAMNEP [142] IDS and we showed that there are OWA functions that outperform simple average, when applied to ensemble of network anomaly detectors.

2.2.5 Weighted Average

The *weighted average* (WAVG) uses different weights assigned to different detectors to combine the individual scores. This technique creates new non-trivial problem of choosing the weights, which significantly influences the final performance. The ensemble system can consist of a very large number of detectors making the search for best parameter settings even more difficult.

According to the number of weight parameters the WAVG aggregation functions can be categorized into three groups [94].

- m weights – there is one weight per classifier. The weight of the i-th classifier is typically based on its estimated error rate.
- $c \times m$ weights – the weights are specific for combination of class and classifier. This allows to have different sets of weights for different classes. Linear regression is the commonly used procedure to derive weights for this model.
- $c \times c \times m$ weights – the support for each class is obtained by a linear combination of supports of all detectors for all classes. These can be organized in matrix forming a *decision profile* and used as the intermediate feature space (class indifferent combiners). Decision profile and class indifferent combiners are described in more detail in Section 2.2.8.

Since the anomaly detection is a binary classification problem, we focus mainly on the first group, where only m weights are used. The weighted average of a set of m detectors $\mathcal{H}_m = \{h_i : \mathcal{X} \mapsto \mathbb{R}\}_{i=1}^{m}$ can be then formally defined as

$$H_{\alpha}(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}) = \alpha^T h(\mathbf{x}), \tag{2.4}$$

where the $\alpha = (\alpha_1, \dots, \alpha_m)$ are the weights assigned to individual detectors. Some of the prior art requires the combination to be convex to have the aggregated score of the same scale as the scores of the individual detectors. In that case two additional conditions for the weights, $\mathbf{1}^{\top}\alpha = 1$ and $\alpha_i \geq 0,\ \forall i \in \{1, \dots, m\}$, need to be introduced.

Since the classifiers are typically not of the identical accuracy, it is reasonable to give higher priority to the more accurate detector over the lower accuracy detector. Therefore, the weight of each detector can be set to be proportional to its accuracy on a validation set. This approach, called *performance weighting* [119], has been found to be marginally better than the arithmetic mean [56, 4].

The benefits of taking into account detectors accuracies were further exploited in the work of Ashfaq *et al.* [9], who developed *Standard Deviation normalized Entropy of Accuracy* weighted scheme which uses entropy measure of the accuracy together with standard deviation in detection/false alarm rates.

A hybrid solution proposed in [115] relies on artificial samples generated uniformly at random. The proposed Heterogeneous Detector Ensemble on Random Subspaces (HeDES) framework for unsupervised generation anomaly detector ensemble trains several classifiers to separate the artificial samples from the provided true ones, and then weights of classifiers in the WAVG combination are determined according to their accuracy on the artificial samples.

A necessary condition to combine heterogeneous anomaly detectors is similar range of their outputs. This problem is tackled in [92] by using estimated cumulative distribution functions of detectors' output. The authors show that their approach outperforms other normalization strategies including HeDES, maximum rank [97] or sigmoid mean [58]. The ensemble construction technique, proposed in Chapter 5, uses an adaptation of [92] as described in Section 5.1.

Similarly to HeDES the *ENCORE* [48] combining uses accuracies of detectors to estimate the weights. ENCORE labels a sample as anomalous only if two currently most precise detectors labeled the sample as anomalous and their difference in precisions is smaller than a predefined threshold. This strategy was developed for the label output classifiers but can be easily adapted to the continuous value output detectors.

In general, when using labeled data, the algorithms for learning the weights of the weighted average do not differ much from general algorithms for supervised classification. But, for security applications the algorithms should be designed to handle large disproportions between numbers of samples in positive and negative classes, and achieve extremely low false positive rates. Such algorithms are also needed in information retrieval (although the requirement on low positive rates is not as strict).

One class of such an algorithms maximizes accuracy of ranking in top τ-quantile, which can be viewed as prioritizing the malicious samples over the legitimate ones. These algorithms (optimizing for example *Prec@k* [107] or Normalized Discounted Cumulative Gain [170]) frequently lead to non-convex optimization problems that are difficult to solve efficiently or lead to suboptimal solutions [98]. A notable exception is SVM-*perf* [84] method optimizing a convex upper bound on the number of errors among the top k items, but still the training is computationally intensive due to a large number of constraints of the quadratic program.

Another class of relevant algorithms like RankBoost [54] maximize area under Receiver Operating Characteristic (AUC) [49], which is equivalent to optimizing ranking. There are also methods that focus only at the top quantile of the returned samples. Infinite Push [133] and Top Push [101] concentrate on the higher ranked negatives and try to push them down.

In this thesis we propose a novel method for optimizing the weights of the weighted average of a set of heterogeneous network anomaly detectors, that optimizes the accuracy in the top τ-quantile that outperforms the above reviewed methods. The detailed description together with thorough evaluation can be found in Chapter 5.

2.2.6 Majority Vote

Majority vote is a combination scheme for a labeled output detectors and so we describe it only briefly. There are three common majority schemes, unanimity, simple majority and plurality. The unanimity will result in a decision only if all of the present classifiers will agree on the same class. In simple majority approach more than 50% classifiers must agree with the labeling. Plurality vote scheme will choose the label that most of the classifiers will agree on. There are also several modifications to these schemes that introduce weights (weighted majority vote), thresholding (threshold plurality vote) or other techniques described in [132].

Zhou *et al.* [188, 189] presented weighted majority voting based on diversity. This approach assumes that ensemble with higher diversity results in greater improvements in detection performance over the base anomaly detectors. They presented *mutual information* for measuring the diversity among the base detectors. The authors apply this diversity-based weighted voting scheme together with the joint boundary method which labels current sample as normal only if all of the detectors labeled it as normal. Joint boundary combiner's predictions of normal data can be quite accurate but typically predictions of anomalies suffer from high false alarm rate. Therefore, to decrease the false alarm rate, they applied weighted voting based on the diversity measures only to the data records that are predicted as anomalies by the joint boundary method. The authors showed that this approach outperforms the arithmetic mean and simple majority voting.

Since the majority vote aggregation was designed for fusion of multi-class, labeled-output classifiers, the information about the certainty of the base classifiers is not used. This typically leads to a lower efficacy of the resulting ensemble.

2.2.7 Mixture of Experts

The mixture of experts [83] is a supervised learning technique based on the divide-and-conquer principle. It uses several experts (classifiers) whose outputs are combined using the weighted average. The weights are determined by a gating network that for each sample decides what experts to use. It is typically trained using the expectation maximization (EM) algorithm. This approach is particularly useful when different experts are trained on different parts of the feature space, or when heterogeneous sets of features are available to be used for a data fusion problem.

Several mixture-of-experts models can be further combined to form a hierarchical mixture of experts [85]. From the conceptual point of view, all experts are organized in tree-like structure. Each leaf represents individual expert or expert network that for given input sample tries to solve a local supervised learning problem. The outputs of one node are combined by the gating function and the total output of the node is given by the convex combination as shown on Figure 2.1.

The main disadvantage of this technique is the fact that it is not applicable for a high dimensional data, as increase of the complexity of the tree-like architecture and associated input space subdivision leads to the increased variance and numerical instability.

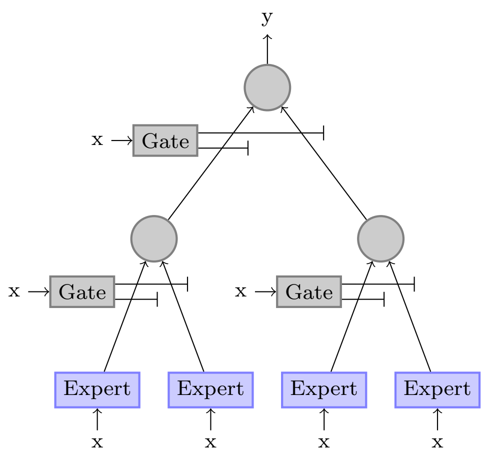

Fig. 2.1: An example hierarchical mixture of experts model with depth two and branching factor of two.

2.2.8 Class-indifferent Combiners

The degrees of support (score values) for given input x_k of each classifier of the ensemble h_i and each class ω_j can be organized in matrix called *decision template* [94]. More formally, if we define the $h_{i,j}(x_k)$ as the support that the classifier h_i gives to the hypothesis that x_k sample comes from class ω_j, the decision profile DP will be defined as

$$DP(x_k) = \begin{pmatrix} h_{1,1}(x_k) & \dots & h_{1,c}(x_k) \\ \vdots & \vdots & \vdots \\ h_{L,1}(x_k) & \vdots & h_{L,c}(x_k) \end{pmatrix}. \tag{2.5}$$

Using the decision profile we can classify the combination methods into two groups [94]. The combination methods that are using one column of the decision profile at a time are called *class-conscious*, the examples of this group are the aforementioned simple and weighted average functions, product and other statistics. Alternatively one can ignore the context of the decision profile and treat the $h_{i,j}$ values as a features in new feature space, called *intermediate feature space*. The combination of the results is made by another classifier that takes the features from the intermediate feature space as an input. This type of combination methods is called *class-indifferent*.

The class-indifferent approach is therefore building a second classification layer on top of the ensemble of simple classifiers. One can even build a system that uses even more classification layers, but it shall be noted that this kind of complexity increase has to be justified by similar gain in the accuracy. Next, training of such a complex system is nontrivial problem. One of the possible solutions for training is to use the Stacked generalization, described in following section, where the combiner is trained on unseen data created by a cross-validation procedure.

2.2.8.1 Stacked Generalization

Stacked generalization is a technique to achieve the highest generalization accuracy [174]. The idea is to first train an ensemble of classifiers using a bootstraped samples of the training data, whose outputs are then used to create a meta-dataset. This dataset is then used to train a meta-classifier that uses the predicted classifications by the classifiers instead of the original input attributes. The target remains as in the original training set. The underlying idea is to learn whether training data has been properly learned. For example, if a particular classifier incorrectly learned a certain region of the feature space, and hence consistently misclassifies instances coming from that region, then the meta-classifier may be able to learn this behavior, and along with the learned behaviors of other classifiers, it can correct such improper training. The same idea has been adapted in regression tasks, where is this approach called stacked regression.

Manhem *et al.* [110] designed a model (called Troika) to address Stacking problems, namely, the poor performance on the multi-class problems. Troika's ensemble scheme, shown in Figure 2.2, may be used to combine any types of classifiers which were trained on any subgroup of possible classes of a problem's domain. Troika uses three layers of combining classifiers, rather than one. The authors showed that Troika is on average better than using the best classifier selected using cross-validation.

Zenko *et al.* [184] evaluated several methods for constructing ensembles of heterogeneous classifiers that use stacking and showed that they can perform comparably to selecting the best classifier. They also proposed a new stacking method that uses multi-response model trees at the meta-level. They showed that it clearly outperforms existing stacking approaches and selecting the best classifier from the ensemble by cross validation.

In the case of anomaly detection we have only one-class classifiers and thus the intermediate feature space contains only one decision value from each detector. Furthermore, learning of such a structure requires a lot of training data and the training process itself is very complex.

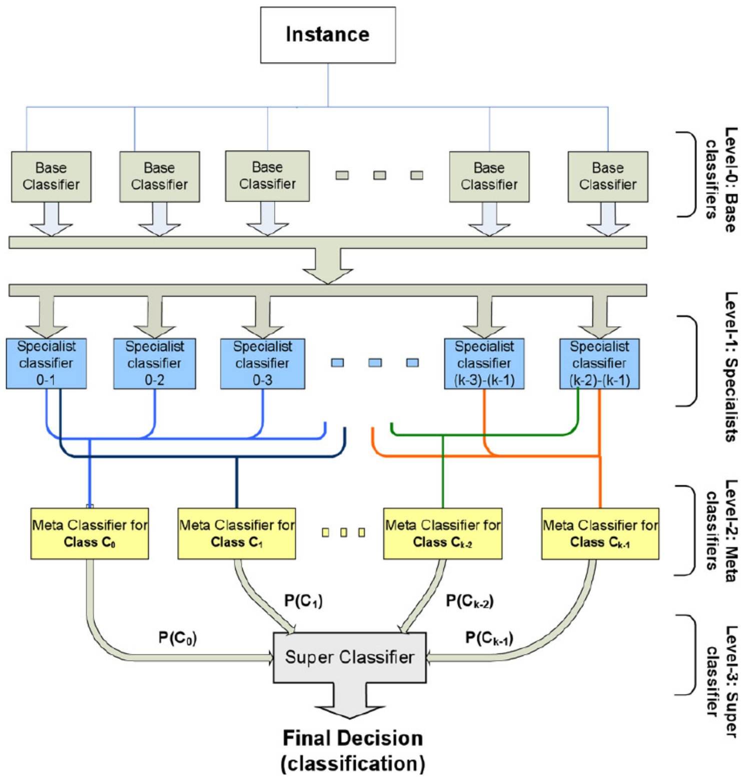
Instance
Base Classifier
Base Classifier
Base Classifier
Base Classifier
Base Classifier
Level-0: Base classifiers
Specialist classifier 0-1
Specialist classifier 0-2
Specialist classifier 0-3
Specialist classifier (k-3)-(k-1)
Specialist classifier (k-2)-(k-1)
Level-1: Specialists
Meta Classifier for Class C0
Meta Classifier for Class C1
Meta Classifier for Class Ck-2
Meta Classifier for Class Ck-1
Level-2: Meta classifiers
P(C1)
P(Ck-2)
P(C0)
P(Ck-1)
Super Classifier
Level-3: Super classifier
Final Decision (classification)

Fig. 2.2: Architecture of stacked generalization method called Troika that has three layers of combining classifiers; the Specialist classifiers combine the outputs of the base-classifiers, Meta-classifiers combine the Specialist classifier outputs, and finally, the Super-classifier, combines all Meta-classifier outputs. The Super-classifier is responsible for the output of Troika's prediction. The figure was borrowed from [110].

2.2.8.2 Decision Templates

Decision template [94] fusion rules are based on the idea to remember the most typical decision profile, called decision template, for each classification class. These decision templates are then compared with current decision profile using a similarity measure. The most similar detector is chosen to label the current sample. The commonly used measures are squared Euclidean distance and symmetric difference [94]. Therefore, the decision template can be also defined as the nearest mean classifier in the intermediate feature space.

Giacinto *et al.* [62] introduced an ensemble system including three groups of classifiers that are trained on three different subsets of features. Then, three simple fusion functions (i.e., majority vote, average, and belief) are employed for aggregation. A subsequent work of the same authors described an ensemble architecture including multiple one-class k-means classifiers [60]. Each classifier is trained on a training subset containing a specific attack type belonging to a specific attack class. The aggregation is based on the Decision Template.

2.2.9 Consensus Theory

The consensus theory [102] deals with the problem of finding an agreement among autonomous entities such as peoples, autonomous software agents or computers in a network. This problem arises for example when multiple sensors observe the same object or control devices compute the same response to a system behavior. Though they may be provided with different inputs, the computing devices have to agree on the same output. The agreement is achieved using the consensus rule that is typically derived to satisfy a set of desirable theoretical properties [17]. In these rules the experts are typically assumed to be capable of assessing their own importance and also the importance of the other participants.

Gao *et al.* [57] proposed unsupervised algorithm that is reaching consensus among multiple base detectors to infer discriminative combination model different and not obtainable using Bayesian model averaging or weighted voting. Method first joins the observation using the detectors's predictions so two clusters are created. They are defined by each of the base detectors, one containing all the samples labeled as normal the other contain the samples labeled as anomalous. The samples and clusters are represented in a graph where each cluster links to all the samples it contains. The algorithm iteratively propagating information between clusters and record nodes to maximize the consensus by promoting smoothness of label assignment over the graph. The whole process is based on the idea that the detectors that agree with the other detectors more often should be weighted higher in the voting. It is assumed that the detectors can make mistakes, but their decisions should not be flipped. They expect that if a detector labels a sample as anomalous with confidence 0.9 and normal with confidence 0.1 it is unlikely to be the other way around. Authors also extended the algorithm to semi-supervised learning and incremental learning cases. Finally, they showed that on three real network datasets, the algorithm improves the overall accuracy by 20% when compared to the standalone detectors.

Volpato *et al.* [172] proposed multi-agent architecture for the network intrusion detection systems. They developed several detection agents, each with its own model that is using various features from the network traffic to detect anomalies. Each agent has its own trust model that assesses the reliability of the agent. Authors propose novel cooperative negotiation protocol between the detection agents. The protocol consists of several steps: first each of the agents receives a sample q_j and computes its offer p_i^t. The $p_i^t(q_i) \in [0, 1]$ represents the partial degree of anomaly from the i-th agent. The detection agent sends its offer together with its trust level to a mediator. The mediator computes a counter offer from the received offers and an agreement a^t using an *agreement function* and sends it back to the detection agents. Each detection agent calculates new offer p_i^{t+1} by using a *negotiation function* $F_i(p_i^t, a^t)$. The negotiation process repeats until a global agreement is reached. The agreement function is weighted average of the offered values, where the weights are trusts of the agents. The negotiation function expresses the measure of disagreement among agents and is calculated as $F_i(p_i^t, a^t) = p_i^t + \alpha_i(a^t - p_i^t)$, where the α_i represents the agreement coefficient which expresses the willingness of detection agent i to propose its offer versus the counter offer from the mediator. The agreement coefficient α_i is computed as value of sigmoid shaped function of the trust value. Authors showed that the proposed mechanism is stable and always reaches agreement.

The consensus techniques are for real-time processing of huge number of samples almost always unusable. Evaluating several steps of the negotiations for each sample is very time demanding and too complex typically without corresponding accuracy gain.

2.2.10 Naive Bayes Combining

Bayesian approach to decision fusion computes probabilities of hypotheses using evidences provided by the individual classifiers. It computes the posterior probabilities using both prior and conditional probabilities provided by the individual classifiers. This scheme assumes that the classifiers are conditionally independent given the class. However, this assumption is nearly always violated, in the case of parallel anomaly detectors severely. But it turns out that the performance is quite robust, even in the case of correlated classifiers [186]. Drawbacks of this approach are the need of a priori probabilities of the classes and the labeled data for training the confusion matrix that is typically used for the posteriori probability estimation. In addition, Bayesian combination scheme does not provide any information about the quality of the computed probability nor the existence of conflicting evidences that can influence Bayes decision criteria.

Scot [151] proposed to use the Bayes rule to combine different intrusion detection methods based on both anomaly detection and signature detection, so as to take into account both user behaviors and attack patterns.

Corona *et al.* [33] designed several multiple classifier architectures for supervised and unsupervised anomaly-based intrusion detection. They proposed to map outputs of used one-class classifiers to probability density functions. In such a way, the outputs of different classifiers can be aggregated using naive Bayes combination rule.

2.2.11 Dynamic Classifier Selection

The *dynamic classifier selection* (DCS) [94] selects for each sample one classifier that is more likely to classify it correctly. This scheme is originally developed for the situation, where each of the classifiers have knowledge about only a part of the feature space and is responsible for the decisions in that part. Therefore, the overall precision can be at most as good as the precision of the best classifier.

Giacinto *et al.* [65, 62, 63] showed that DCS is better in comparison to majority vote, average, naive-Bayes and decision template when applied in the intrusion detection field. It exploits the diversity among the detectors and negatively and positively correlated classifiers. The authors stated that DCS is designed to approximate an ideal oracle, which, for each new pattern selects the classifier that provides the correct label. This allows to effectively handle different accuracies and pair-wise correlations exhibited by the individual classifiers.

The dynamic classifier selection approaches are in its basic version unusable for the anomaly detectors ensemble, because some anomalies can be detected only by a mixture of the base detectors and not only by one base detector. Furthermore, the standalone detectors have a very large number of false positives that are not necessarily reduced by this approach.

The dynamic classifier selection was successfully used in the CAMNEP intrusion detection system [142, 139]. To overcome the above drawbacks the CAMNEP system used the DCS method to selects the best simple non-trainable combination function from static set of predefined functions instead of selecting only one anomaly detector. The selected function was used to aggregate the anomaly scores of a batch of data that contained samples from real network traffic from specific time interval (typically five minutes). This way, a different combination function could be selected in each batch reacting to the current state of the network.

CAMNEP used more than 30 different static aggregation functions including simple average, several weighted average functions (see Section 2.2.5) and few ordered weighted average functions (described in Section 2.2.4). The best aggregation function was selected according to its performance on a set of small artificially added labeled samples called *challenges*. These were mixed in the input of the system so the detectors were not able to differentiate between the simulated samples and the samples from the real network environment. There were challenges containing both, legitimate and malicious behaviors so the performance could be calculated as

$$d = \frac{\bar{x} - \bar{y}}{\sigma_x + \sigma_y} \ , \tag{2.6}$$

where the $\bar{y}$ is the average value and σ_y is the variance of the legitimate challenge anomaly scores and the $\bar{x}$ and σ_x values are the average and variance of the malicious challenge anomalies. The aggregation function with largest d was selected to aggregate the anomaly scores assigned to the data of the current batch.

The experiments presented in [142] showed that the dynamic selection of the optimal aggregation function can significantly reduce the number of false positives and that the targeted insertion of challenges selected according to threat models can influence the system sensitivity to reflect the risks associated with each attack type.

The main disadvantage of this approach is that it relies on the quality of the challenges. Additionally, the challenges need to be regularly updated to reflect the current network threats. This is similar to creating a signature for the signature-based intrusion detection systems. Next, there are only 30 functions that can be used and they have to be adjusted whenever new base anomaly detector is added or removed from the ensemble.

2.2.12 Dempster-Shafer Combination

The Dempster-Shafer (DS) combination rule takes its inspiration from the evidence combination of Dempster-Shafer theory. It strongly emphasizes the agreement between multiple sources and ignores all the conflicting evidence through a normalization factor. One of the computational advantages of the DS framework is that priors and conditionals need not be specified, unlike the Bayesian methods.

The DS theory uses exhaustive and mutually exclusive logical statements Θ about some problem domain that are called propositions or hypotheses A_i, $i = 1, \ldots, M$. To each proposition a belief value from $[0, 1]$ is assigned, based on the presence of evidence e. The value of belief is derived from basic probability assignment (BPA) $m(A_i)$, which defines impact of each evidence item on subsets of all propositions. For the BPA holds $m(\emptyset) = 0$ and $\sum_{A \subseteq \Theta} m(A) = 1$. The summary of $m(B)$ for all subsets $B \subseteq A$ becomes the total belief in A. More formally,

$$bel(A) = \sum_{B \subseteq A} m(B). \tag{2.7}$$

Two independent evidences expressed as two BPAs m_1 and m_2 can be combined using the joint mas combination in following manner [153]:

$$m_{1,2}(A) = (m_1 \oplus m_2)(A) = \begin{cases} 0 & A = \emptyset \\ \frac{1}{1-K} \sum_{B \bigcap C = A} m_1(B) m_2(C) & A \neq \emptyset, \end{cases}$$

where the $K = \sum_{B \bigcap C = \emptyset} m_1(B) m_2(C)$ is the measure of the amount of conflict between two BPAs.

In DS theory, BPA is the degree of the belief of truth induced by a certainty of evidence. In multi-classifier combination the beliefs of truth for outputs from each individual classifier can be evaluated by confidence values or the classifiers performance measure. Additionally, Hakan *et al.* [6] showed that there is no need for the classifiers to be independent to be able to use DS framework for the final combination.

Zhang *et al.* [185] proposed to compute the BPA using the true positive, false positive, true negative and false negative ratios of individual classifiers. They used the Barnet's method [13] to compute the final BPA from all BPA of all classifiers. Finally, the class with the highest believe value is used as the final decision of the ensemble.

The main disadvantage of the DS combination lies in its complexity. The task of finding all pairs B and C such that $B \bigcap C = A$ has an exponential time complexity which renders this technique unusable for the online intrusion detection systems.

2.3 False Positive Reduction

Since the amount of false positives represents the main drawback of the anomaly-based approach to the network intrusion detection, many researchers focused on developing additional techniques to reduce the amount of false positives introduced by the anomaly detection. The prior art uses statistics [159], time series [171], and machine learning algorithms [126] to achieve this goal. Generally, the methods can be categorized into two groups [112]:

- *detection techniques* focus on development of more accurate detection method with lower false positive rate. They frequently rely on the data-mining techniques,
- *alert processing techniques* operate on the network alerts produced by existing intrusion detection systems and identify the false positives to filter them out.

The following sections survey the existing research relevant to both the above defined groups of false positive reduction methods.

2.3.1 Detection Techniques

Hooper [80] introduced an intelligent network quarantine channels technique to get additional information about the suspected hosts. This information is further used to estimate the probability of a host being infected. Although, this technique provides additional information that can reduce the amount of false alarms, it is not always allowed by the organization's security policy to perform any additional host checking.

Xiang *et al.* [176] proposed a multi-level hybrid classifier which combined the supervised tree classifiers and unsupervised Bayesian clustering to detect intrusions. Performance of this new approach showed to have high detection and low false alarm rates. They concluded that keeping the false negative rate as low as possible while maintaining an acceptable level of false positive rate is essential for IDS since the false alarm might bring inconvenience to the administrators.

Elshoush *et al.* [44] reviewed collaborative, intelligent intrusion detection system which is proposed to include both misuse-based and anomaly-based methods. They provide backgrounds on how to use alert correlation to reduce the false positive rate with different system architectures. They suggested fuzzy logic, soft computing and other AI techniques, to be exploited to reduce the rate of false alarms while keeping the detection rate high.

Lee *et al.* [99] developed a framework for fully unsupervised training and online anomaly detection. In the framework, a self-organizing map that is seamlessly combined with K-means clustering was transformed into an adaptive and dynamic algorithm suitable for real-time processing. The performance evaluation of proposed approach showed that it could significantly increase the detection rate while the false alarm rate remained low.

Bolzoni *et al.* [21] proposed to correlate anomalies in incoming and outgoing traffic as successful attack should raise alarms in both directions.

2.3.2 Alert Processing Techniques

The alert processing techniques work with a notion of a network alert, which is a collection of network flows, rather than with individual flows.

Pietraszek [126] created a classifier that is reducing the workload of the human analyst by classifying the alerts into true positives and false positives. The knowledge of how to classify an alert is learned adaptively by observing the analyst. Similarly, Tian *et al.* [167] used the analyst feedback to generate custom-made filtering rules, that automatically discard similar alerts encountered in the future. However, the need of human interaction represent always a bottleneck since the effectiveness of the IDS depends on the human analyst.

Fontugne *et al.* [52] constructed a graph based on the network traffic of each of the alerts. The nodes in the graph represent the individual alerts and the edges with weights correspond to the similarity between the alerts that is based on the containing network. The authors used this technique to be able to filter out the uncorrelated alerts that are typically false positives. The graph is used to cluster the anomaly detectors into communities. The communities are then combined using correspondence analysis algorithm called SCANN [111].

Neural networks and fuzzy logic are used in Alshammari *et al.* [5]. This method requires a lot of labeled alerts for the training in order to be able to reduce false positives.

In Morin *et al.* [113], Chronicles Formalism is used in order to justify alarm relationships. In this way, alarms that do not seem to be part of an attack can be filtered out.

Viinikka and Debar [171] proposed to use Exponentially Weighted Moving Average (EWMA) control charts of the flow of alerts to spot abnormalities. Their experimental results concern the production of less and more qualitative alerts.

2.4 Chapter Summary

This chapter identified the wider body of literature to which the thesis contributes. We first reviewed a number of network anomaly detection methods with a strong emphasis on the algorithms relevant to the newly proposed detectors in Chapters 3 and 4. Since many researchers focus on the ensemble learning when dealing with the network anomaly detection, in Section 2.2 we reviewed ensemble-based methods and their reported issues, from which we are motivated to search for a better solution. Finally, Section 2.3 reviewed the state-of-the-art in the field of false positive reduction of the intrusion detection systems which motivated the work introduced in Chapter 6.

Chapter 3
NetFlow Anomaly Detectors

The network anomaly detection algorithms introduced in this chapter are implemented as a part of CAMNEP[1] [141, 59] system. CAMNEP is an anomaly detection engine that uses only general statistics about the network communication in form of NetFlow [30] data to detect malicious network traffic. The NetFlow format was initially developed by Cisco Systems for IP traffic information collecting. It was followed by the IPFIX [31] and both these formats became standards for the industrial network monitoring.

Many network security researchers focus on the NetFlow data because it captures high level statistics about the network communication which have been shown sufficient for detecting many threats [11]. The NetFlow data is easy to obtain since it is supported by most of routers or other network traffic shaping devices. Collecting the NetFlows introduces only small additional load for the exporting devices allowing to process data from high speed backbone networks. Additionally, the NetFlow data respects the privacy of the users, as they do not contain any information about the content of the network traffic, making their exploitation easier from the legal standpoint.

The NetFlow data is structured in records, each record describes one *flow*. A flow is defined as an unidirectional component of the network connection and contains all packets with the same source IP, destination IP, source and destination port and protocol (TCP, UDP, ICMP, etc.). NetFlow record is created when FIN packet[2] is detected or predefined time interval elapsed. The total number of packets, transfered bytes and bitwise OR of all TCP flags from all the transfered packets are recored in each NetFlow. Complete list of NetFlow fields together with an example of possible values is shown in Table 3.1.

The CAMNEP anomaly detection engine identifies anomalous traffic using a set of anomaly detection algorithms. They work in two stages: (*i*) they extract meaningful features associated with each flow (or group of flows), and (*ii*) they use the values of these features to assign an anomaly score to each flow. The anomaly value may depend on the flow itself, on other flows in the current context, and on the internal traffic model, which is based on the past traffic observed on the network.

The CAMNEP contains number of detectors based on already published anomaly detection methods. Some of them are based on Principal component analysis (Lakhina detectors [95, 96] and KGB detectors [125] described in more detail in Sections 2.1.2,2.1.1,2.1.3), some detect abrupt changes (MINDS [45] detector described in Section 2.1.4), and some even use fixed rules (Xu [177] detector described in Section 2.1.5). Furthermore, there are detectors designed to detect specific type of unwanted behavior like network scans (TAPS [160] detector described in Section 2.1.6).

Below, we revisit three novel detectors proposed by the author. Since each of the detectors is designed to detect a specific type of anomalous network behavior we present the experimental evaluation on the detector specific behaviors together with the method description.

[1] Cooperative Adaptive Mechanism for NEtwork Protection

[2] The packet that has the FIN TCP flag set is used by the network host that wishes to terminate the ongoing communication.

Feature	Example of values
start-time	1440870672
duration	5
protocol	TCP
source ip	192.168.1.2
destination ip	208.80.154.224
source port	1604
destination port	443
TCP flags	.AP.SF
type of service	0
number of packets	1201
number of bytes	1.8 M

Table 3.1: Example of one NetFlow record containing information about both communication participants (source and destination IP and port), time of the communication, protocol, bitwise OR of all TCP flags of the transfered packets, type of service (tos), number of packets and bytes transferred.

3.1 TCP Flags Anomaly Detector

TCP flags anomaly detector focuses solely on the TCP traffic and uses the aggregated TCP flags information contained in the NetFlow data to detect network attacks and other malicious behavior. Similarly to Lakhina-based agents (described in detail in Sections 2.1.2,2.1.1,2.1.3) it uses PCA to detect deviations from the normal behavior. This novel method is able to detect flows with malicious flags combination (introduced in Section 3.1.2) and other types of attacks with specific TCP flags signature deviating from that of the normal traffic.

3.1.1 Description of the TCP Flags

Since Transmission Control Protocol (TCP) is a stateful protocol, we can determine the purpose of each packet. The TCP flags indicate different connection states or information about how the packet should be handled. These states are encoded by nine binary flags shown in Table 3.2.

Flag	Description
NS	ECN-nonce concealment protection
CWR	Congestion Window Reduced [134]
ECE	ECN-Echo [134]
URG	URGent data
ACK	Valid ACKnowledge
PSH	PuSH request
RST	ReSeT the connection
SYN	SYNchronize the sequence number
FIN	FINal data - no more data from sender

Table 3.2: Table of all defined TCP flags together with their meaning.

A TCP connection is established when the initial handshake is complete, after which the actual data exchange can start. The typical TCP connection progresses as follows:

1. Client sends a SYN packet with sequence number j.

2. Server receives the SYN packet and sends its own SYN packet with a sequence number k. At the same time it acknowledges the client's SYN packet by sending an acknowledgment (ACK) packet with a sequence number $j + 1$.
3. The client acknowledges the server's SYN packet by sending an ACK packet with a sequence number $k + 1$. At this point, the connection is established and the client can transmit data to the server.
4. The client and server exchange data, each sending an acknowledgment with increased sequence number when data is received.
5. The server closes the connection by transmitting a FIN packet with sequence number m.
6. The client acknowledges the FIN packet with an ACK whose sequence number is $m + 1$ (assuming no data transmitted). If d bytes of data is transmitted, then the sequence number is $m + 1 + d$. The server may still send more data. The client closes the connection by transmitting its own FIN packet with sequence number n.
7. Finally, the server transmits an ACK packet with sequence number $n + 1$. That terminates the connection. Either party may initiate termination of the connection and the termination may not be as graceful as shown above. The connection may be terminated at any time by resetting it with a TCP RST packet.

The first three TCP flags in Table 3.2 (NS, CWR, and ECE) are not commonly seen in "normal" transmissions or used in abuse situations. Although not ignored by the proposed detection method we will skip those in the further text, since they are not important in order to explain the approach.

3.1.2 TCP Flag Distributions During Malicious Actions

In this section we review the normal and abnormal TCP flag combinations and the flag characteristics of several well known network attacks. First, we start with a description of the normal TCP flags combinations:

- SYN, SYN ACK, and ACK are used during the three-way handshake which establishes a TCP connection (as described in Section 3.1.1).
- Except for the initial SYN packet, every packet in a connection must have the ACK bit set.
- FIN ACK and ACK are used during the graceful teardown of an existing connection.
- PSH FIN ACK may also be seen at the beginning of a graceful teardown.
- RST or RST ACK can be used to immediately terminate an existing connection. Packets during the conversation portion of the connection (after the three-way handshake but before the teardown or termination) contain just an ACK by default. Optionally, they may also contain PSH and/or URG.

This implies that most of the packets emitted by a normal host in the network should contain ACK flag, followed by the PUSH. There should be significantly less SYN and FIN packets, since users are sending more packets in the initialized connections than creating and closing them. Furthermore, the most common teardown of a connection is done using the FIN packets and only way how to start a connection is to use SYN packet, so normally, users should have the same amount of FIN and SYN packets. These claims were experimentally verified and the distribution of TCP flags of normally behaving user is shown in the Figure 3.1.

Packets with any other then above described flag combination can be classified as abnormal. The abnormal combinations of the TCP flags usually occur during malicious activities. Here we briefly introduce the most popular techniques:

- *Scans* are used by an attacker to find hosts with specific service (*horizontal scan*) or to find all running services on specified host (*vertical scan*). These attacks are characterized by lot of SYN/FIN packets from the attacker. Since the SYN/FIN is one of the well known TCP flag illegal combinations and can be easily detected, the attackers are using other variants of

SYN/FIN with additional flag set (SYN FIN PSH, SYN FIN RST, SYN FIN RST PSH) to avoid the detection. Next, the attackers also use the SYN RST packets where the RST is used instead of the FIN. SYN packet is sometimes also used, leaving the handshake unfinished. The responses to this attack are typically SYN ACK when the service is available or SYN RST when the port is filtered.

- *Stealth connection* is commonly used for *fingerprinting* — recognition of operation system (OS). It is usually done by sending SYN packet, waiting for the response and answering with RST packet. Since different OS implement the TCP protocol slightly differently, these differences can be exploited to identify target OS. Moreover, the attacker can hide its identity, because the OS does not log the connections that were not properly established (the ACK packets were not exchanged).
- *SYN Flood* is a a type of *Denial of Service* (DoS) attack in which an attacker tries to consume all the servers resources by initiating many connections, but not finishing the handshake protocol. This is achieved by sending single SYN packet and not answering to ACK packets leaving the server waiting. This attack can be characterized by many packets with the SYN flag, and many responses with the SYN/ACK.
- *Distributed Reflected DoS (DRDoS) attack* is a stealth DoS attack. It is a technique in which the attacker spoofs his IP address for victim's to avoid identification. He sends SYN packets that are carrying the victim's source IP to lot of hosts. Each host receives the spoofed SYN packet and responses with SYN/ACK packet to the victim's IP causing DoS/flood attack. This way the attacker's spoofed SYN packets are being "reflected" of innocent hosts in the network and for the administrators it looks like the hosts used for the "reflection" are performing a DDoS attack.

3.1.3 Detecting TCP Flag Anomalies

Anomaly detection method uses only TCP flag information contained in the NetFlow, which limits the method exclusively to TCP communication (other protocols are skipped). The statistics of TCP flags usage for every host are calculated from NetFlow records in each five–minute time interval. This choice was made on the basis of related works [157, 187, 96] showing five–minute intervals give the best performance.

The network is modeled by distribution of TCP flags of flows aggregated by source or destination IP address. This means that we create two different models for source and destination IP. Since the models differ only in the aggregation, they are explained in following text on the aggregation with respect to source IP address. Next, this fact indicates that we can only identify anomaly on the individual network host level and not the flow level, in other words, we are able to label only anomalous IPs and not the anomalous NetFlows for each five–minute time window. Figure 3.5 shows TCP flags usages during different behaviors of one source IP over five minutes. As can be seen, various behaviors have diverse TCP flag signatures.

For the TCP flags modeling of each IP address, five consecutive time windows (time frame of 25 minutes) gave us consistently the best results and it represents a good trade-off between performance and computational complexity. Denoting quantities calculated at the time step t by the same superscript, the behavior of one source IP is described by the following 45 dimensional vector:

$$x^t(\text{sIP}) = \left\{ \overrightarrow{flags_\tau}(sIP) | \tau \in \{t-4, \ldots, t\} \right\}, \tag{3.1}$$

where $\overrightarrow{flags_\tau}(sIP)$ is 9-dimensional vector of normalized counts of individual TCP flags of all flows from source IP sIP in time window τ. Thus, the vector $x^t(\text{sIP})$ captures behavior of the IP address during time steps $t-4, \ldots, t$, capturing the changes in TCP flags usage over that period of time.

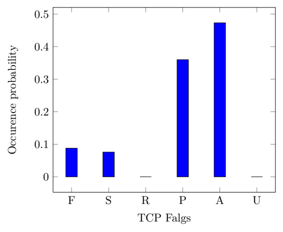

Fig. 3.1: Normal user behavior

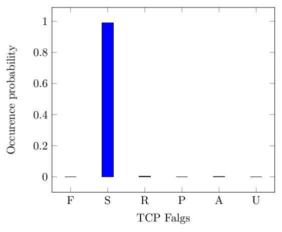

Fig. 3.2: Horizontal scan

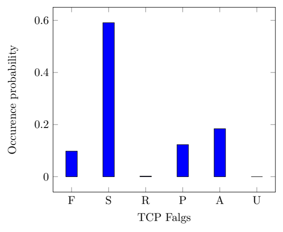

Fig. 3.3: FastFlux malware

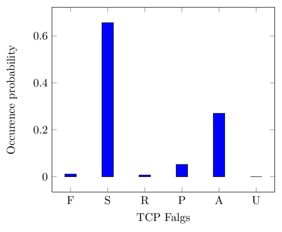

Fig. 3.4: SPAMBot malware

Fig. 3.5: Histograms of usages of TCP flags of various behaviors of one network host.

Rationale behind this feature vectors is that feature vectors of TCP flags of different IP addresses should be similar, thus correlated. Principal Component Analysis (PCA) can identify low-dimensional sub-space where most of the traffic lies. IP addresses behaving differently will have different feature vector, which is not going to be correlated and consequently, it will not lie in the subspace of the normal traffic.

The model of the network is created as follows. The 45 dimensional feature vector (3.1) detailing behavior of hosts at the time step t is calculated from data acquired at time steps $t-4,\ldots,t$. Feature vectors of all IP addresses active during these time steps are arranged in matrix $\mathbf{X}^t \in \mathbb{R}^{n_x,d}$, where d is number of features (in our case $d = 45$) and n_x is number of active IP addresses. Each row of matrix $\mathbf{X}^t$ corresponds to feature vector of one active IP address. The IP address is considered to be active, if it has at least 20 flows in at least one time step $t-4,\ldots,t$. Features in time steps with less then 20 flows are set to zero. IP addresses with less than 20 flows in all time steps are not used in the model. Anomalies for these flows are not calculated because of the lack of the normal behavior.

Similarly to [125], the anomaly detection in the time step t, is done using a PCA on the matrix from the previous step $\mathbf{X}^{t-1}$ to build the model of the traffic, which is used to calculate anomaly values (see Subsection 3.1.3.2) for every IP address active in time step t (each row of matrix $\mathbf{X}^t$). If the anomaly value of a given IP address exceeds the threshold, it is deemed as being anomalous.

This approach allows us to detect two types of anomalies:

1. Since the model captures correlation of feature vectors across IP addresses, if there are some with feature vectors uncorrelated with the majority (their behavior is very different), they are detected as anomalous. By means of this mechanism, it is possible to identify anomalous behavior of individual IP addresses.
2. Since the model used to detect anomalies at the time step t is derived from a traffic captured at time steps $t-5,\ldots,t-1$, abrupt changes of the behavior of large number of IP addresses will be detected as anomalous, even though the behavior is correlated with the majority in the time step t (but not with the majority in the time step $t-1$). This mechanism will detect

the beginning of large-scale, possibly coordinated, attacks. As in the previous case, it also permits to identify IP addresses participating in long and coordinated attacks.

In the rest of this section, we first briefly describe the PCA (Section 3.1.3.1) and then we describe the calculation of anomaly values (Section 3.1.3.2).

3.1.3.1 Principal Component Analysis

The calculation of principal components p_i starts by calculating the sample covariance matrix $\mathbf{W}$. Assuming vectors $x \in \mathbf{X}$ have zero mean ($\frac{1}{n_x}\sum_{j=1}^{n_x} x_j = 0$), the sample covariance matrix $\mathbf{W}$ is calculated as

$$\mathbf{W} = \frac{1}{n_x - 1}\sum_{j=1}^{n_x} x_j^T x_j. \tag{3.2}$$

The projection vectors p_i and corresponding variances λ_i are eigenvectors and eigenvalues of the covariance matrix $\mathbf{W}$. A side note here is that eigenvalues λ_i corresponding to eigenvectors p_i are all real numbers, because the sample covariance matrix $\mathbf{W}$ is symmetric.

Depending on the level of correlation among the columns of $\mathbf{X}$, the covariance matrix $\mathbf{W}$ does not have to have a full rank, which expresses itself in eigenvalues having very low values. For numerical stability, we discard all eigenvectors corresponding to eigenvalues smaller than 10^{-6}.

3.1.3.2 Anomaly Detection Algorithm

To detect anomalies in the data we use *variance on major and minor components*, used for example in [156]. Let's assume that the PCA performed on a data set $\mathbf{X}^{t-1}$ returns r components $\{p_1, \ldots, p_r\}$ corresponding to eigenvalues $\lambda_1 > \lambda_2 > \ldots > \lambda_r \geq 10^{-6}$. The components are divided into two sets: one containing components $\{p_1, \ldots, p_s\}$ with high variances, and other containing components with small variances $\{p_{s+1}, \ldots, p_r\}$. This results in two variants of the anomaly detection algorithm that use either the anomaly measure $f(y)$ or $f^{\perp}(y)$ to assign anomaly score to the vector $y \in \{\mathbf{X}^t_{j\cdot} | j \in \{1, \ldots, n^t_x\}\}$ using either the high or low variance components as

$$f(y) = \sum_{i=1}^{s} \frac{(y^T p_i)^2}{\lambda_i^2}, \tag{3.3}$$

$$f^{\perp}(y) = \sum_{i=s+1}^{r} \frac{(y^T p_i)^2}{\lambda_i^2}. \tag{3.4}$$

The anomaly detection algorithm at time step t works as follows:

1. From data acquired in time steps $t-5, \ldots, t-1$ are used to construct matrix $\mathbf{X}^{t-1}$. The rows of the matrix corresponds to feature vectors (3.1) of IP addresses with at least one TCP flow in one of the time steps $t-5, \ldots, t-1$.
2. After the matrix $\mathbf{X}^{t-1}$ is normalized to have zero mean, the co-variance matrix $\mathbf{W}$ is calculated and the eigenvalue decomposition is performed.
3. The principal components $\{p_1, \ldots, p_r\}$ are divided into two sets: one containing components $\{p_1, \ldots, p_s\}$ with high variances, and other containing components with small variances $\{p_{s+1}, \ldots, p_r\}$.
4. For every active IP address in the time step t, we calculate the feature vector 3.1 and one of the anomaly measures defined in Equation (3.3) or (3.4).

The last thing we did not discussed is the number of components considered to be in the set of high and low variances. We varied this parameter in our preliminary experiments and found value $s = 1$ to give consistently good results. This value is used in experiments presented in Section 3.1.4.

3.1.4 Experimental Evaluation

We experimentally compare the TCP flags anomaly detectors with anomaly measures f (Equation (3.3), denoted in further text by F) and $f^{\perp}$ (Equation (3.4), further referred to as FOG), both with aggregation over source and destination IP (srcIP, dstIP) addresses (4 detectors in total) with other existing anomaly detectors from the prior art described in detail in Section 2.1. Namely, detector of Xu *et al.* [178] with aggregation over source IP addresses (further referred as Xu-srcIP) and destination IP addresses (referred as Xu-dstIP), Lakhina *et al.* [96] detector with entropy features (further referred as Lakh-Entr) and traffic volume features (in tables denoted as Lakh-Vol), Minesota Intrusion Detection System [45] (MINDS), the TAPS scan detection method [160] and the anomaly detectors of Pevny *et al.* [125] (denoted as KGB-F-srcIP, KGB-F-dstIP, KGB-FOG-srcIP, KGB-FOG-dstIP).

To compare these methods we have created the Receiver Operation Characteristics (ROC) of each of the mentioned anomaly detection methods using several partially labeled datasets from the real networks. Next, we have evaluated the ROC's Area Under the Curve (AUC) to be able to easily compare all the presented methods. Because we are interested only in the false positive rates smaller than 1% of all the traffic, we computed the ROC's AUC as an integral of the ROC up to 1% of false positives threshold. This gives us good insight about the detector's performance for small false positive threshold, which is important to the field of network anomaly detection.

3.1.4.1 Experimental Settings

We have performed nine different experiments on six different networks. The datasets were constructed using three different approaches:

- manually performed attacks — attacks that were designed and manually created by one of our security experts to mimic some of the simple network attacks used by attackers (Experiments 3 and 4).
- VM infected by real malware [59] — usage of discovered malware binaries to infect virtual machine and label all communication of this machine as malicious (Experiments 1 and 2).
- manually identified malware/attacks — malicious labels were created using the manual analysis of the network captures made by network security specialists (Experiment 5 to 9).

The latter two cases were further analyzed to check that they mainly use TCP communication to be able to use them for the TCP Flag detectors evaluation.

Experiment 1 and Experiment 2 use the evaluation datasets published in [59] captured on the Czech Technical University (CTU) network with more than 1 500 users and in average over 30 000 NetFlows per one 5-minute batch. Both experiments were using a real malware running inside controlled virtual machine (VM)[3] and cover the time of one day. In Experiment 1 only one VM running Windows XP infected by Neeris malware was used, that generated a lot of traffic. Experiment 2 contains ten different VMs infected by FastFlux botnet. The labels were created so every NetFlow originating from the infected machine/s is labeled as Neeris and FastFlux src and the replies are labeled as Neeris and FastFlux dst respectively. All the NetFlows that are typically created by the idle Windows XP machine were labeled as normal — these typically are Windows connection check, Windows Update checks, NTP checks, etc. For the normal label we used the traffic going from several hosts performing normal activity during the attack — several users that were using these hosts for browsing, sending mails, using instant messaging

[3] The bandwidth of the connection was lowered to 150kb/s to avoid huge usage of the network.

etc. Again, we have manually went through all the NetFlows from those host and label them as legitimate.

Experiment 3 uses a dataset created with an artificial attack scenario that used one of the host in the CTU network with open SSH port as victim of an attack, attacked from outside the CTU network. In the first step of the scenario, we simulated the information gathering by scanning part of the university network for an open SSH port. Next, we performed brute force dictionary password cracking on the victim host. After 1000 trials we used the correct password to simulate the successful break–in. Finally, we downloaded 0.5 GB to simulate data stealth. Corresponding NetFlows in the captured dataset were manually labeled using the IP addresses, ports and time information.

Similarly to Experiment 3, Experiment 4 contains artificial attack that was in this case not created manually, but performed by a Rbot malware, controlled by our network specialist, that we manually installed on one of the CTU network hosts. Since we had the full control over this bot, we were able to perform two simple attacks to one of our victim outside the CTU network — ICMP attack and SYN flood attack. We have labeled both using the IP addresses and time information.

Experiments 5 to 9 contain NetFlows from various networks (each experiment represent different network) of various sizes and types. All the labels were created by manual inspection of the NetFlows. We have been able to identify several malware families (virut [152], pushdo [3], shiz [43], proxybot, ircbot), one regular vertical scan and one stealthy. All the malware families were confirmed using publicly available information about the malware behavior. The legitimate labels were created by labeling the most popular legitimate services (MS update service, google, facebook, youtube, bing, etc.) found in the experimental datasets.

3.1.4.2 Experimental Results

Table 3.3 shows the comparison of all anomaly detectors mentioned in Section 3.1.4. Values are AUC calculated up to the false-positive of 1% of all the traffic. The "-" denotes the fact that the detector did not set a anomaly value for the specific behavior. This could be caused by lack of statistical data or the evaluated behavior is out of the scope of the detector (e.g. host that uses only UDP protocol will not have an anomaly assigned by TCP Flags detectors, because it can assign anomaly only when there is at least 20 TCP connections, see Section 3.1.3).

It is evident from the table that there is no single detector that would outperform others in detection of all presented types of malicious behavior. This can be explained by the fact, that each method utilizes different model of the network, uses different NetFlow features and thus detects slightly different anomalies. But, we can see that the newly introduced detectors outperform the prior art in most of the cases.

TAPS outperformed proposed detectors in the scan-like behaviors, which is not surprising since TAPS was designed to detect scans and floods. The KGB detectors are able to detect dns tunnels and some type of a scans, which is caused by the fact that these behaviors use number of various ports, that heavily affects the analyzed entropies.

Surprisingly both Lakhina's detectors are not able to cope with the proposed Flags detectors nor the KGB detectors. This is interesting, since these methods are based on the PCA as well. These results show that the selection of the features is crucial for the network anomaly detector's performance.

It is also interesting to observe, how detectors with different aggregations complement each other. Detectors aggregating over source IPs detect anomalous requests, while detectors aggregating over destination IPs detect the responses.

	Flags F-dstIP	Flags F-srcIP	Flags FOG-dstIP	Flags FOG-srcIP	KGB F-dstIP	KGB F-srcIP	KGB FOG-dstIP	KGB FOG-srcIP	Lakh Entr	Lakh Vol	MINDS	TAPS	Xu dstIP	Xu srcIP
Exp 1														
Neeris dst	**0.315**	0.071	0.073	0.157	0	0	0	0.036	0	0.033	0	0.288	0.167	0
Neeris src	0.230	0.027	0.334	0.049	0	0	0.005	0.033	0	0	0.048	0	0	**0.641**
Exp 2														
FastFlux dst	**0.689**	0.583	0.197	0.337	0	0.006	0	0.207	0	0.002	0.009	0.149	0.009	0.001
FastFlux src	0.170	0.216	**0.380**	0.140	0	0	0.004	0.004	0.002	0	0.009	0.005	0	0.006
Exp 3														
SSH scan src	0.023	0.013	0.428	**0.876**	0.008	0	0.005	0.113	0	0	0.001	0	0.250	0.005
SSH scan dst	0.162	0.069	0	0.102	0	0	0.744	0	0.001	0	0	0.033	**1**	0
SSH cracking	0.181	0.006	0.279	**0.571**	0.002	0	0.003	0.186	0	0	0.109	0	0.250	0.005
Download dst	-	0	-	0	-	0	-	0	-	**0.668**	-	-	-	
Download src	**0.967**	-	0	-	0	-	0	-	-	-	0.834	-	-	-
Exp 4														
syn flood src	0.027	0.211	0	**0.612**	0	0	0.242	0.574	0	0.572	0.427	0	-	-
syn flood dst	0.391	0.106	0.430	0.303	0	0	0.474	0	0	0	0.426	**1**	0.002	0
Exp 5														
virut dst	0	0.008	0	0.070	0	**0.478**	0	0.094	0.039	0.002	0.106	0.429	0.009	0.028
virut src	0.092	**0.711**	0.086	0.618	0	0	0.028	0	0	0	0.044	0	0	0.188
Exp 6														
pushdo dst	**0.477**	0.177	0.409	0.363	0	0.001	0.006	0.010	0.008	0.271	0.004	0.025	0.010	0.002
pushdo src	0.016	**0.824**	0.247	0.716	0	0	0	0.009	0	0.005	0.004	0	0	0.105
Exp 7														
port scan src	0.008	0	0.003	0.761	0.003	0	0.002	**1**	0	0	0	**1**	0.026	0.685
port scan dst	0.902	0.508	0.971	0.003	0	0.012	**1**	0	0.013	0.004	0.004	0.006	0	-
Exp 8														
shiz dst	0	0.010	0	**0.376**	0.185	0	0.021	0	0.031	0.147	0.013	0.333	0.006	0.184
shiz src	0.006	0.010	0.043	**0.175**	0.008	0	0	0.009	0	0	0.054	-	-	0
dnstunnel src	0	0.571	0	0.291	**1**	**1**	0	**1**	-	-	0.378	-	0	-
dnstunnel dst	0	0	0	0	**1**	**1**	0	0.005	-	-	0.380	-	-	-
proxybot dst	0.001	0.011	0	0.015	0	0.007	0	0.036	0.078	0.130	0	**1**	0.007	0.100
proxybot src	0	0	0	0	0.001	0	0	0	-	-	0	-	-	**0.074**
ircbot dst	0	0.020	0	0.255	0	0	0	0.076	0	0.057	0	**0.750**	0.008	0.128
ircbot src	0	0	0	0	0	0	0	0	-	-	0	-	-	**0.781**
Exp 9														
port scan src	0.006	0.069	0.301	0.648	0	0	0	0.344	0	0	0.229	**0.692**	0	0
port scan dst	0.572	0.282	**0.860**	0.749	0	0	0	0.436	0	0	0.032	0.013	0	0

Table 3.3: Comparison table of the ROC AUCs on FP level of 1% of all the anomaly methods on data acquired from various experiments.

3.2 Request-Response Anomaly Detector

The request-response anomaly detector is a representative of the knowledge based anomaly detectors that is designed to detect reconnaissance attacks (various network scans) and ongoing Denial of Service (DoS) attacks.

The network scans are used by attackers to probe the network systems for available services to take an advantage of their security deficiencies. Network services using the TCP and UDP Internet protocols can be accessed via special ports which are generally known (for example the SSH service is by default assigned to the TCP port 22). Ports that are used by the services are referred to as open, since it is possible to establish a connection with them, whereas unused ports are referred to as closed; every attempt to connect with them will fail. The attacker can therefore use a port scanning tool to find all open ports on one machine via probing all ports on a particular IP address (vertical scan) or find machines in the network that provide a service of interest by probing a specific port on all network IPs (horizontal scan).

Denial of Service (DoS) attack is an attempt to make a network service unavailable to its intended users by flooding the service providing host (server) with huge number of request. Since the server can only process a certain number of requests at once, the legitimate user's request wont be typically processed when the server is under the DoS attack. The Distributed

Denial of Service (DDoS) attack uses more than one attack source, often thousands of unique IP addresses.

Both the above described techniques result in large number of failed TCP requests. The request-response anomaly detector uses this specific property to detect the attackers by matching request-response pairs in order to identify requests without responses that represent failed connections that are indicative of the ongoing attack. If the number of failed connections is higher than normally observed amount over all hosts in the network an anomaly is raised.

Additionally to various scans and DoS/DDoS attacks, the detector is able to detect the C&C[4] search of some malware families that perform the search by contacting a large number of raw IPs. Since most of them are typically unavailable, the search results in a large number of unsuccessful connections.

3.2.1 Request-Response Pair Matching

Pair matching of the NetFlow records is done based on the IP-port-protocol triple. First the flows are sorted by time. Then the first flow is pulled from the list and the matching flow with the same protocol but reverse source and destination IPs and ports is found by iterating over the list. The two are declared to be a request-response pair and removed from the list. If a matching pair is not found the flow is declared to be a request without a response and is also removed from the list. The procedure repeats until the list is empty.

3.2.2 Network Model

The anomaly score is calculated using a simple model known as the *Z-value test* [1] (also known as *z-score*, *standard score* or *sigma score*). Consider a set of 1-dimensional quantitative data observations, denoted by $x_1, \ldots, x_n$, with mean μ and standard deviation σ. The Z-value for the data point x_i is denoted by z_i, and is defined as follows:

$$z_i = \frac{|x_i - \mu|}{\sigma}. \tag{3.5}$$

The Z-value test computes the number of standard deviations by which the data varies from the mean. This definition of anomaly has an implicit assumption that the data is drawn from a normal distribution.

Since we are interest in only large number of failed connection we restrict the Z-value to right tail of the normal distribution. The anomaly score f of a particular network host i is then defined as

$$f(x_i) = \begin{cases} \frac{x_i - \mu}{\sigma} & \text{if } x_i > \mu \\ 0 & \text{otherwise,} \end{cases} \tag{3.6}$$

where the mean μ and standard deviation are calculated using the number of failed connections over all network hosts and x_i is the number of failed connections of host i. The anomaly score is calculated for each host in the network every five minutes using the latest five–minute batch of the NetFlow data similarly to the Flags detection method (see Section 3.1.3).

[4] Command and Control. See Section 3.3 for detailed description.

3.2.3 Experimental Evaluation

Table 3.4 shows the AUC scores of several types of malicious behaviors: malware C&C search, port scans and DoS attacks. All the datasets were captured on the Czech Technical University network. The malware samples were generated using real malware running inside controlled VM [59], the scan and flood attacks were performed by network security specialist attacking our own servers (see Sections 3.1.4.1 for more details about the experimental settings).

The AUC scores show high detection quality for all three types of malicious behaviors. The scan detection quality depends on how successful and extensive the scan is. Typically, a network host has only few open ports and the rest is closed which means that the performed vertical scan against such host, that is probing first 1024 ports, will receive responses only for a couple of the requests while the most of the requests will be without responses leading to higher anomaly score assigned by the proposed method. The horizontal scan, on the other hand, will have much higher success rate as for the popular services (SSH, FTP, HTTP, etc.) there will be more hosts in the scanned network that will respond to the requests, which eventually reduces the anomaly score of the horizontal scan.

Dataset	**AUC value**
Malware C&C search	
Sirefef (src)	0.9648
Pushdo (src)	0.8481
Port scans	
Vertical scan (src)	0.9893
Vertical scan (dst)	1.0000
Horizontal scan for SSH (src)	0.8873
Horizontal scan for SSH (src)	0.8996
DoS	
ICMP attack (src)	0.9196
ICMP attack (dst)	0.9534
Distributed SYN flood (dst)	0.9533

Table 3.4: Results of the request response anomaly detector. AUC values are shown for various types of malicious behaviors, that are known for generating unsuccessful connections.

3.3 Domain Generation Algorithm Detector

Domain Generation Algorithm (DGA) detector is designed to detect the malware-compromised network hosts that are a part of a botnet. Botnets are the root cause of many malicious activities, such as denial of service, spam distribution, click fraud, adware, distributed brute-forcing of remote services, identity and data theft and many more. A typical botnet consists of a number of malware-compromised machines, called bots, that are remotely controlled by a botmaster using a command and control (C&C) channel. There are two main types of botnet C&C structures [35]: peer-to-peer (P2P) and centralized.

In the P2P [145, 165] structure every node can serve as C&C server distributing commands and updates in peer-to-peer manner. This makes the botnet more robust and resilient, hard to identify and to take down. This approach is less popular because it is very hard to implement and maintain. Additionally, the commands take a longer time to reach all the bots because of the latency introduced by the distributed botnet topology. Finally, each newly infected host has to be provided with an initial list of bots to which it may connect.

The centralized structure [35] is the most popular due its simplicity. In this scenario, the bots contact one predefined domain or IP address on which the C&C server is located. The disadvantage of this approach is that the C&C server represents a single point of failure. When taken down, the botmaster loses control over the whole botnet. Network administrators use blacklists of well-known C&C domains to block the communication at the firewall level. Furthermore, Anti-virus companies and OS vendors are working hard to take down these C&C servers and are successful in doing so.

To overcome the disadvantages of the centralized structure, modern malware uses various techniques to hide its C&C server. One of these techniques is fast-flux [79], in which the C&C server is hidden behind a number of proxies that are associated with one domain name and the IP addresses are swapped in and out with extremely high frequency using domain name server (DNS) changes. This way the bots communicate with the C&C using a number of ever changing proxies.

Similarly, malware can use a *domain generation algorithm* (DGA), also referred to as *domain fluxing*. In this scenario, the malware contacts a domain that was generated using a domain generation algorithm with a specific seed in specific time intervals. Whenever the botmaster wants to send a command to his botnet, he needs to register a new domain that he generated using his own copy of DGA with the same seed as the botnet just before the botnet will try to contact it. Botmasters are trying to expose their C&C servers for the minimum amount of time. Domains are registered and DNS configurations are made just a few minutes before the infected bot is supposed to query the domain, and the C&C servers are shut down and removed immediately afterwards, so the whole process takes less than an hour. This renders the detection mechanisms that rely solely on a static domain lists ineffective.

The DGA can be a simple algorithm that uses a seed and the current date and/or time to generate alphanumeric combinations for a new domain. More sophisticated DGAs (e.g. Kraken botnet [7]) can create English-language-like domains with properly matched syllables or combinations of English dictionary words, which makes them undetectable by the means of domain names analysis.

When such a malware is found, it has to be reverse engineered to uncover the underlying domain generation algorithm in order to block all the generated domains on a firewall or register them before the botmaster does. This task can be time-consuming and needs advanced reverse engineering skills. Furthermore, attackers can make this even more difficult by altering the technique in a way that the DGA seed is based on the responses of popular sites like *google.com*, *baidu.com*, *answers.com* (Conficker-C [129]) or even trending topics on twitter (Torpig botnet [163]) that cannot be known in advance, rendering the filtering approach unusable.

The proposed DGA detection algorithm is based on the fact that each host is expected to make a DNS query before contacting any IP, that was not previously visited by the host. Thus, we monitor the amount of DNS requests of each host in the local network together with the amount of unique IP addresses that it contacts. We then calculate the ratio $\rho(a)$ for each host a in the local network, defined as

$$\rho(a) = \frac{\delta(a)}{\pi(a) + 1}, \tag{3.7}$$

where $\delta(a)$ is the number of DNS requests and $\pi(a)$ is the number of unique IP addresses contacted by the host a. There is an addition of one in the denominator to avoid undefined values of $\rho(a)$ when the number of IPs contacted is zero.

It is important for the $\pi(a)$ to be the number of unique IPs that communicated with host a and a was an initiator of such communication. This allows us to detect DGA malware even if it is running on a server which we would not be able to do if we did not know the initiator of communication. Illustration of DGA-infected client and server machines is shown in Figure 3.6. Since the NetFlows are unidirectional, we are using request-response identification described in Section 3.3.1 to identify the initiator of each communication.

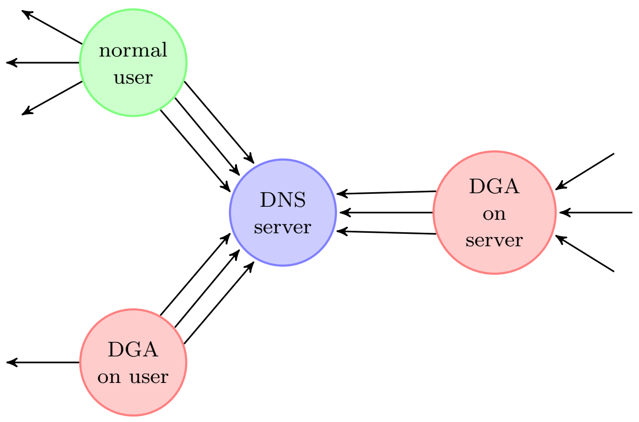

Fig. 3.6: Illustration of DGA behavior. A normal user has a number of DNS requests proportional to number of contacted IPs, whereas DGA on a user host has a high number of DNS queries without contacting any new IPs. Finally, a server with DGA is contacted by a high number of IPs from the network, but still the number of DNS requests it does is disproportional to the number of IPs contacted by the server.

The ratio $\rho(a)$ is typically low for ordinary hosts. A high value of $\rho(a)$ is expected when a host is running DGA. We found that the high value can also indicate a server under some type of brute-force attack as the servers are typically logging all suspicious activities together with the DNS resolve of the IP. An example may be an ssh server that for each unsuccessful login attempt logs DNS resolved information of the client. This means that both DGA and some brute-force activities can be detected using our approach.

Figure 3.7 shows the histogram of $\rho(a)$ for all IPs from a part the Czech Technical University (CTU) network. As can be seen the ratio follows normal distribution with heavy tail on the right. Which supports our above mentioned assumption.

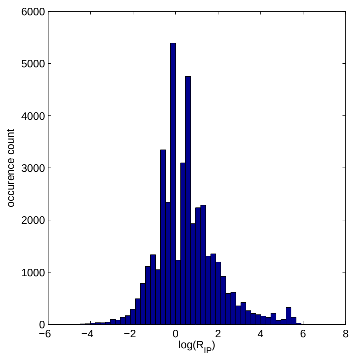

Fig. 3.7: Histogram of $\log \rho(a)$ of all hosts that make at least one DNS request in Czech Technical University network accumulated over several hours.

To detect the hosts infected by DGA malware we use a simple outlier detection technique, described in Section 3.3.3, to identify hosts from the tail of the distribution, that represent hosts with an unusually high ratio when compared to the majority of the hosts in the observed network and mark them as anomalous.

3.3.1 Request-response Identification

In order to be able to correctly calculate the amount of IP addresses that the host contacts and differentiate them from IP addresses that the host is contacted by, we need to know which flows are requests and which are responses. Contrary to the Request-response anomaly detector, introduced in Section 3.2, we need to differentiate the request and response flow (only the pairing is not sufficient).

Our request-response identification is done in two steps. Since the request has to be prior to the response, we first find the request-response pairs based on IP-port-protocol triples the same way as described in Section 3.2 and mark the NetFlow in a pair with smaller starting time as request and the other as response. The unmatched NetFlows are marked as requests without responses. This however suffers from unreliable timestamps generated by some network probes which leads to random labeling of NetFlows as requests and responses. Typically, different network probes have different error distributions of timestamps. Figure 3.8 shows the distribution of timestamp differences for all request-response pairs where requests were made to port 80 for a part of Czech Technical University network. We assume that all the communication to port 80 is with HTTP servers, which enables us to decide which flow is the request and which is response. There is approximately a thousand hosts in the network and all the data capture is done by a single software probe. Figure 3.9 shows the distribution of the same feature for a large corporate network. The network has around 50 thousand hosts and the capture of NetFlows is done by several routers and hardware probes around the network. Request-response pairs with equal timestamps are not shown in the histogram.

In Figure 3.8 most of the timestamp differences are positive which is expected as the responses should have greater timestamps than the requests. In Figure 3.9 the distribution is nearly symmetrical around zero which means that the request has 50 percent probability of having a smaller timestamp than corresponding response.

It is evident that in the university network, the timestamp difference is a strong feature for the request response identification while in the corporate network it leads to random result. This can be due to the fact that the second network is considerably larger and has more than one probe, which means that requests and responses can take different routes and potentially be captured by different probes altogether. Additionally, the probes do not necessarily have synchronized times between them. Even when running time synchronization services, like NTP, the probes can have few to hundred milliseconds difference in time rendering this feature useless in these setups.

Next section introduces a service detection technique that is used to improve the above described method in the environments with high NetFlow timestamp errors. We assume that in client-service communications all flows from client to the service are requests and flows from service to client are responses. We therefore proceed to find all active services and mark NetFlows that are involved in communication with them accordingly.

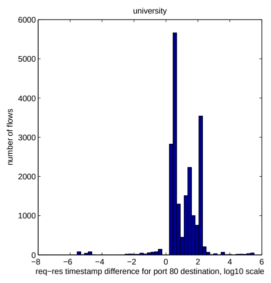

Fig. 3.8: CTU network

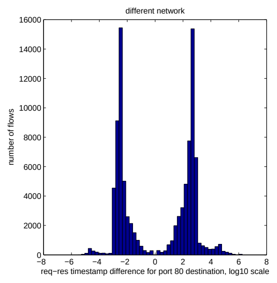

Fig. 3.9: Large corporate network

Fig. 3.10: Timestamps difference histogram.

3.3.2 Service Detection

The service detection algorithm is based on the median of number of peers difference, described in detail in Section 3.3.2.1. To define the peer, we first need to define an endpoint which is a unique IP-port-protocol triple. The peer of an endpoint e is then defined as an endpoint that communicates with e. The rationale behind the median number of peer difference is that the endpoints representing a service will typically have many peers (many clients that are connecting to that endpoint) and the client endpoints — peers of the service endpoint – will have a smaller amount of peers (clients are typically using a different port for every request creating a unique endpoint for communication with each server).

The services obtained from the peers difference rule are then filtered using two additional heuristics, described in Sections 3.3.2.2 and 3.3.2.3, in order to lower the amount of false positives.

3.3.2.1 Median of Number of Peers Difference

First we calculate the number of peers that each endpoint communicates with. Then the median difference in number of peers d_e between an endpoint e and all its peers is defined as

$$d_e = \text{median}\{|P_e| - |P_i|\}_{i \in P_e},$$

where P_e and P_i are the sets of peers of endpoint e and i respectively.

Figure 3.11 shows a histogram of d_e values for CTU network over the time frame of 15 minutes. Port numbers were used for labeling the endpoints as services or clients in the histogram. Even though we used port numbers for labeling, we did not want to use them as a feature in the algorithm, because many services run on higher ports and many clients (e.g. NTP) communicate from lower ports. This is is also reflected in the histogram where most of the endpoints with low ports and negative or zero values were either NTP, SNMP, etc. that communicate on lower ports on both sides. The endpoints with high port number and positive d_e value were usually legitimate services like HTTP on port 8080.

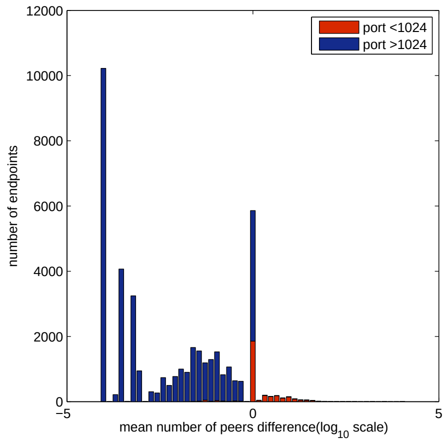

Fig. 3.11: Median of number of peers difference histogram, 15 minutes aggregation.

The final decision if an endpoint e is a service is made by thresholding the d_e values. Services are those endpoints whose median of number of peers difference is positive, the rest are considered to be clients.

3.3.2.2 Unsuccessful Connections

The median of number of peers difference is a strong feature, however, it can lead to some false positives. One is when a scan from a fixed IP port is performed. In this scenario the endpoint performing the scan has a high number of peers and would be classified as a service. This endpoint can be differentiated from a service by the number of unsuccessful connections originated from the endpoint f_e. An ordinary service should have zero or a very low f_e, while f_e for a scan will be usually high.

3.3.2.3 Communication Via Both TCP and UDP on Ports Higher Than 1023

A lot of P2P traffic is also caught by the service detector. This is because P2P often uses a fixed port for communication and contacts a high number of hosts. Therefore, the number of peers for an endpoint involved in P2P can be also really high.

However, P2P communications can be filtered out using a simple rule — communication on a port higher than 1023 using both TCP and UDP protocols which is typical of peer to peer [89].

3.3.3 DGA Anomaly Detector

We expect the majority of the network connections to be legitimate, only a small amount of traffic is expected to be malicious [130]. If we assume that the ratio $\rho(a)$, defined in Equation 3.7, of the legitimate behavior follows a normal distribution, we can use the Z-value [1], defined in Section 3.2.2, to estimate the anomaly score. Since we are interested only in detection of hosts that have more DNS requests than number of visited IPs, which corresponds to higher value of $\rho(a)$, we are restricting the Z-value to the right tail of the normal distribution. The left tail represents behaviors in which the host contacts a lot of IPs without DNS queries, such as horizontal scanning or P2P networks.

The anomaly detection algorithm first calculates the ratio $\rho(a)$ given by Equation 3.7 for every host on the local network. Then for each five-minute batch the mean and standard deviation of the $\rho(a)$ values of all hosts on the local network are estimated. The choice of the time interval was made on the basis of the related works [157, 187] showing 5-minute intervals to give the best performance.

Anomaly scores are obtained using a modified Z-value (see Section 3.2.2 for definition) as

$$f(x) = \begin{cases} \frac{x-\mu}{\sigma} & \text{if } x > \mu \\ 0 & \text{otherwise} \end{cases},$$

where x is the value of the ratio for a given endpoint, μ and σ are the current model values.

3.3.4 DNS Server

DNS servers represent a family of false positives of the proposed DGA detector. For each DNS request they get they make several DNS requests, to resolve the DNS record not present in their cache. They therefore look anomalous from the standpoint of a DGA detector. We filter out DNS resolvers using information from service detection defined in Section 3.3.2.

3.3.5 Experimental Evaluation

3.3.5.1 Controlled Infection

In this section, the experimental evaluation uses NetFlow datasets created using deliberate infection with Shiz malware [2] which is known for using the DGA algorithm. We infected a clean Windows machine connected to the CTU network. The malware was run for several hours on three different days and times to capture the various states of the network resulting in three different experiments. Detection was conducted on the NetFlows generated from the traffic of the whole university network. Collected NetFlows were labeled using the source IP addresses of the infected hosts as malicious and the rest of the traffic was labeled as legitimate. The ROC AUC, given in Table 3.5, show very high detection quality for each of the presented experiments that represent the network in different states.

Day	1	2	3
AUC value	0.9733	0.9923	0.9989

Table 3.5: AUC scores of the proposed detection algorithm for the controlled infection scenario.

Figure 3.12 shows the distribution of anomaly values for one five-minute batch in which the DNS anomaly detector identified traffic generated by Shiz. The traffic from Shiz is labeled red in the histogram, based on the IP address of the machine in which it was running. Only the outgoing traffic is labeled, as it is what we want to detect. Note that some of the background traffic was also labeled as very anomalous. Manual inspection of those NetFlows revealed that the traffic was originating from several IP addresses that really did show some DGA-like behavior. For example, one of the cases was a mail server that performed reverse DNS resolution for every client, resulting in hundreds of DNS requests every five minutes.

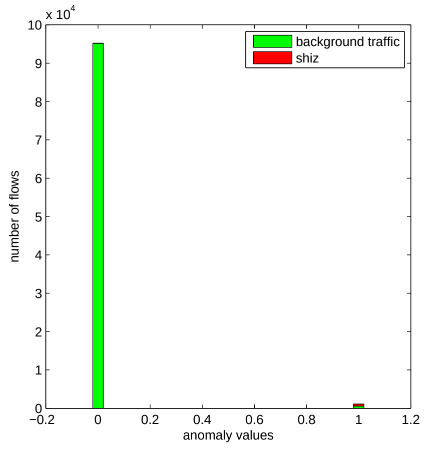

Fig. 3.12: Anomaly values assigned to flows from the CTU network where Shiz malware was run.

3.3.5.2 Mixed — various host types

Another set of experiments was conducted using NetFlow records of the Shiz malware [2]. The sample was run inside a virtual machine for 12 hours and packet captures (pcap) were created. NetFlows were generated from the pcaps and then mixed into background traffic from the CTU network. Three datasets were generated with the same background traffic, starting at the same time. In the first dataset the malware traffic was mixed into a previously inactive IP address, in the second dataset into an IP address of normally behaving network host and in the third dataset into an HTTP server IP address. The AUC values are shown in Table 3.6.

The best results are for the previously inactive IP, with the highest AUC values. This is because it does not have any other traffic except for DGA, therefore its $\rho(a)$ ratio is high. The host with the user has the lowest AUC. This is because the user is active, he contacts websites and other servers, therefore his $\rho(a)$ is lowered by the number of IPs he contacts. The HTTP server's AUC is somewhere in between. This is probably because the server also contacts some IPs, possibly for time synchronization or updates. Its $\rho(a)$ is not affected by the users of the server, because we are able to differentiate between requests and responses. A version of the DGA detector that did not make a distinction between requests and responses achieved the AUC of 0.8229 on the same dataset.

Host type	inactive host	user	server
AUC value	0.9811	0.9179	0.9432

Table 3.6: User-server comparison.

3.3.5.3 Mixed — various DGA malware families

In the last experiment we used ten malware samples of six different families. Again, all the malware samples were run inside a virtual machine for 12 hours and pcaps were captured. NetFlows were generated from the pcaps and mixed into background traffic from a large size company with more than 50 000 users. The malicious traffic was mixed into the traffic of six randomly selected active user IPs and six active servers. Each row of the Table 3.7 shows the minimal and average AUC score over all six hosts of the same type for a specific malware sample. As can be seen from the table the presented technique is able to detect various DGA malware families with high precision when running on the user machine — the minimal AUC is above 0.80 and the average AUC score above 0.89 for all the tested samples. The efficacy is slightly worse for the server hosts, where the minimal AUC score is 0.70 for the Win32-AutoRun malware sample and the average is above 0.86 for all the samples. We should also note that for our experiments we assume that the original traffic that was not altered by mixing is legitimate and does not contain any hosts infected by DGA performing malware. This is not necessarily true, manual inspection of the false positive showed many hosts performing DGA like activities. Unfortunately, due to lack of additional information, it was impossible to definitely prove or disprove an actual infection by malware.

3.4 Chapter Summary

This chapter introduced CAMNEP, an anomaly detection system that uses NetFlow records to detect malicious network communication. We have briefly introduced the existing anomaly detection algorithms used in the CAMNEP system and proposed one statistical and two knowledge

Host type	Malware family	mean(AUC)	min(AUC)
User	Trojan-Generic	0.9926	0.9883
	Variant-Kazy	0.9943	0.9943
	Win32-AutoRun	0.9303	0.8650
	Win32-AutoRun	0.9259	0.7974
	Win32-AutoRun	0.9279	0.8093
	Win32-Waski-A	0.8894	0.8646
	GameOverZeus	0.9837	0.9505
	GameOverZeus	0.9907	0.9843
	GameOverZeus	0.9830	0.9684
	caphaw	0.9920	0.9824
Server	Trojan-Generic	0.9687	0.9156
	Variant-Kazy	0.9953	0.9953
	Win32-AutoRun	0.8877	0.6966
	Win32-AutoRun	0.9116	0.7703
	Win32-AutoRun	0.8977	0.7325
	Win32-Waski-A	0.8586	0.7478
	GameOverZeus	0.9720	0.9355
	GameOverZeus	0.9687	0.9175
	GameOverZeus	0.9767	0.9554
	caphaw	0.9588	0.8857

Table 3.7: AUC scores of the proposed method on various malware samples.

driven network anomaly detection techniques to enrich the existing detector ensemble. Each proposed method was experimentally evaluated using a real network data to prove its effectiveness in practice.

Chapter 4
HTTP(S) Anomaly Detectors

Botnets are one of the most sophisticated and popular type of cybercrime today. They are the root cause of most of the malicious activities in the computer networks, such as denial of service attacks, spam sending, click frauds, adware, distributed brute-forcing of a remote service, identity and data thefts and many more. A typical botnet consists of a number of malware-compromised machines, called bots, that are remotely controlled by a botmaster using command and control (C&C) channels. Exploitation of a machine starts with malware infection from malicious web page, email attachment, etc. As soon as the malware infects a host, it usually tries to establish a connection to one or more C&C servers to download updates, retrieve commands or send private information obtained from the infected host. These callbacks to the C&C servers are usually through the port 80 since it is a commonly open communication channel in the majority of networks as there is only a small amount of networks that prevent its users from accessing the web. The next generation firewalls that are common in present networks can detect if a connection crossing by the port 80 is a standard HTTP connection or not (various tunnels through port 80) and allow or block it accordingly. For this reason, the malware developers are designing malware that is able to communicate via HTTP(s) protocol. They are mimicking the normal HTTP behavior by using existing HTTP fields for malicious purposes to hide their activities within noisy HTTP traffic, which makes the detection of those activities a challenging task.

For these reasons we think that detection of malicious HTTP(s) requests deserves a tailored solution. In this chapter we introduce several novel HTTP anomaly detectors that are currently part of the Cisco Cognitive Threat Analytics (CTA) [29] security solution that analyzes HTTP proxy logs (typically produced by proxy servers located on a network perimeter) to detect infected computers within the network. Although the logs do not contain all host traffic (only the HTTP(s) requests are recorded), the information is richer than the NetFlow as each log entry (hereinafter referred to as HTTP flow or HTTP request) contains the following information extracted from HTTP request: time of the request, source IP address, destination IP address, username[1], URL, MIME type, downloaded and uploaded bytes, User-Agent identifier, etc. The complete list of all the features together with their possible values is shown in Table 4.1.

CTA contains more than 30 different anomaly detectors designed to detect various types of anomalies using:

- empirical estimates of (conditional) probabilities such as probability that any user visit a specific country $P(\text{country})$, specific user visiting a specific domain $P(\text{domain}|\text{user})$, a domain is visited using a specific web browser $P(\text{User-Agent}|\text{second level domain})$, etc.,
- time series models that model user's activity over time, detect sudden changes in the activity, identify periodical requests, etc.,

[1] This chapter uses the username to identify the network entity contrary to the previous chapter that uses the source IP address. The IP address of a particular host may change over time, but the username is more reliable as it always identifies the same network user allowing to create a long term models of his behavior. Therefore, this chapter refers to the modeled network entity as a *network user* instead of the previously used network host.

Feature	Value example
x-timestamp-unix	1440870672
sc-http-status	200
sc-bytes	16671
cs-bytes	0
cs-uri-scheme	https
cs-host	en.wikipedia.org
cs-uri-port	1604
cs-uri-path	/wiki/Anomaly_detection
cs-uri-query	
cs-username	Martin Grill
x-elapsed-time	5
s-ip	208.80.154.224
c-ip	192.168.1.2
Content-Type	text/html; charset=UTF-8
cs(Referer)	https://www.google.com/
cs-method	GET
cs(User-Agent)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36

Table 4.1: Example of HTTP flow. This is one of the HTTP flow created during the download of a wikipedia.org page. Each page download generates more HTTP flows since all the page resources have to be downloaded. To render the page from the example the browser generated additional 20 HTTP flows containing page styles, scripts, pictures, etc. In the example the cs/sc prefixes denote the client to server and server to client communication respectively, so the sc-bytes represent the amount of bytes downloaded by the client and cs-bytes the amount of uploaded bytes to the server. The cs(Referer) field identifies the URL address of the webpage that linked to the resource being currently requested. By checking the referrer, the new webpage can see the origin of the request. The rest of the features is self-explanatory.

- HTTP(s) specific detectors use a domain knowledge to detect the anomalies. These include domain name analysis (n-gram models), analysis of TLS [40] certificates of the HTTPs domains, WHOIS [36] information analysis, etc..

Additionally to the existing HTTP(s) anomaly detectors this thesis proposes two new detectors that are described in detail in the following sections. Similarly to the structure used in Chapter 3 we present the experimental evaluation of each proposed detector together with the algorithm description.

4.1 Long First Touch Anomaly Detector

First of the proposed HTTP anomaly detectors is designed to detect a specific type of C&C communication that is carried over the HTTP protocol. It leverages a simple principle based on the URL complexity to estimate the anomaly score of the individual HTTP requests.

As described above, the malware trying to exfiltrate data or communicate with the C&C server typically tries to mimic normal HTTP requests to avoid being detected by the signature matching detection systems. This can be done by encoding the transfered data in the URL string (URL path and/or URL query). In that case the URL looks similar to the ones of legitimate behavior (see Sections 4.1.3 and 4.1.2 for examples). However, this approach leads to HTTP requests with longer URLs that cannot be directly related to any other legitimate activity of the infected user.

In the legitimate scenario, the user that wishes to visit a specific website types the URL into the address bar of his browser and hits enter. It is unlikely that he would type a long or complex URLs. We expect to see the longer URLs being reached only when an additional resources of the requested site are downloaded or when the user follows a link on some webpage such as in the results of a search engine. In both these cases the cs(Referer) HTTP header field contains the originally requested site. Although the malware might also fill the cs(Referer) to mimic the legitimate requests, the referred site is typically missing in the HTTP flows. There is no need for the malware designers to visit other websites which would eventually increase the chance of being detected because of larger network footprint. Additionally, as discussed earlier (see Section 1.1), generating a set of flows that would simulate a legitimate user's behavior is a complex problem.

4.1.1 Possible URL Lengths

The minimal URL length is given by the fact that URL needs to contain the primary access mechanism. Since we are considering HTTP/HTTPs flows only the mechanism can be either `http://` or `https://`. Additionally, there has to be a location specified, that in its final form can contain only three characters, which gives us the theoretically minimal URL length of 10 characters (e.g. `http://a.b`).

The maximal length, on the other hand, is not limited. The HTTP protocol [50] does not place any a priori limit on the length of a URL and the servers should be able to handle URL of any resource they serve. But in practice, the extremely long URLs are very rare because they will not work in most of the popular browsers and servers. For example, URLs with more than 2,083 characters will not work in the Microsoft Internet Explorer. Therefore seeing an URL with more characters should raise a suspicion as it is not designed to be contacted by a web browser.

Figure 4.1 shows the distribution of URL lengths of more than 160 million URLs that have been observed to be requested by more that 30,000 users over one week. As can be seen, the majority of the URLs has less than 200 characters. There is only a really small amount of URLs of length greater than 1,000 with a small peak around the length of 2,000.

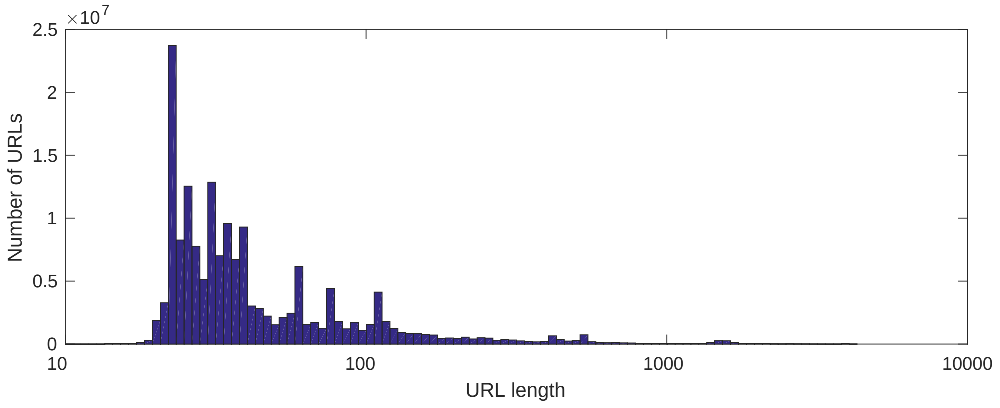

Fig. 4.1: Histogram of URL lengths of an HTTP flows captured in a corporate network with more than 30,000 active users over one week.

4.1.2 Legitimate Long URLs

Typically, the number of characters of a normally behaving user is smaller than 100, but there might be legitimate flows that have URL of much greater length that are summarized in the following bullets:

- Requests for the additional resources of a web site. Web sites typically consist of a large number of elements. Each element of the downloaded site (picture, javascript, font, css style, etc.) needs to be requested via the HTTP so the browser can correctly render the whole site. Therefore the original HTTP request issued by a user (e.g. http://google.com) is followed by a number of HTTP requests that are sent automatically by the browsers. These request may need to pass a number of parameters back to the server that can be encoded in the URL resulting in really long URLs. However, these requests will have the Referer field set to the original requested site (i.e. http://google.com). Therefore, the request that have the Referer field set to an existing requested domain may be skipped by the Long first touch anomaly detector.
- Notification checks. Some web pages have a mechanism that updates its content every predefined time period (e.g. checking of new mail of the web-based email clients) that can also due to a larger number of parameters result in a long URL. Such an HTTP requests do not necessary have the Referer field filled, or it may point to a request that was issued a long time ago.
  The notification checks are also skipped by the proposed detector, using a long term memory that contains all domains requested by the user in the past.
- Links sent by other means. These are the HTTP requests that result from user clicking on a link that was received by other means (through email, instant messaging, etc.). In that case the request can be of extensive length irrelevant to the other flows of the user. Since we are not able to distinguish these cases from the malicious one, this type of behavior represents a family of false positives of the proposed detection algorithm.

4.1.3 Malicious Long URL Example

DNS Changer [169] is an example of a malware that uses URL path and query to communicate with the C&C server. It is a malware capable of stealing user and system information together with being able to execute commands on the device. It bypasses Windows Powershell restrictions to modify the DNS registry entries in Windows devices in order to replace the name servers on the user's machine. This results in the user accessing some sites to be redirected to a phising sites[2]. After the successful infection, the malware informs the C&C about its state together with information about the infected user an the current DNS registry settings. The HTTP request sent to the C&C does not have the Referer field filled, but the URL has 1310 characters that look like this:

```
   http://big4u.org/u/?q=Gn3G2Z9sYR4z6XEiqz8Iue391IA347YihS6uDdcjAi-CAGitqNK7o_HqKD4wD2Cf3iFrjJNNV
C68QduLgaADk-XEjgKaeiefjoeYhnQK8NIxqxwarvcX7Nzy8KdLR4wgGiLvTkSBV6W8vfyFtz0kby0c31SMXE7lgOGt7h1V8v
66V_xbQ-mULrv2X_jH_dnioyTkarp-pvTvcRbraaGzrWcTcujS9X30CZeXljVllogXjxxV5OHaSzOOwv61udS7h2KFA8yLxjg-
wZl7ZmJtAGAcjz2DeNkswsHcX6v7-wtbehFVmAH6H2K3sNyk0xfnTc2xz2dxrQfoBlq_oC0JHyALeFey1uGuqp1ctpXWRvctRq
7uaPS2obaVHM4LSGy2diBTSDX-GTyZPmxIq0zGf8DYVF3CY4PcJWPxyfQZTUpSBFlBQmgTsmRM6eTbAYlGMd6S21dBtiRJQVW
YCwQ0LdzIX5dAVlfrMXCjzPeaHyp6G2Goo7rDDk0NX_-3oOFOJvtDfxSOiCSaJ8DqUk4RVMjcDPtN0gOQhixAoqtejpFahsAeO
LVCD5PT_6BPrDl&c=AZc-E-xVB8zzHM8ItRK6dLwZXFfmW1wt3n2mgH1RK2RVbInVFZFFsHUZ_jG_TjgGnSALg6XZQz7k2tAs2
uu5mCs17BDW_XiCHcRGu2pCUysfdbCubNk8D0pKqC23BEEmqvliSEYSleritZW91lIkFfG_FVnJ5p-ncRQGaPHCQpuTqhbWxXG
sQeO2PdBlzK__oCwSI2s864v9xkttqVxo8k8D1z1_unGui2FGRath1r-jGOSXYOxrnhzHwMH81RoIXh880mzyoiu14fRIdptoM
```

[2] A fake website that tries to obtain sensitive information such as usernames, passwords, and credit card details by masquerading as a legitimate site.

koq1S3jc237TyFU1hFsND-asnFZi9W69lptgYvXBBlNNNhiRxsyGFh4Wp0G9co1bvWqZaLlg4m3XrcfF59K0DvehW3TrjXtnd
t4kCbniPIIem-Aswv0J-8DnPCA7KBii1nye03cGTpsrv0Um1dqaxatY-NRdFN4lG9igtPws5xgXhPgKGX3WrNzK7i3WLcIv7c
28l7h_0C6fRwlYZwue7fcBB00a1Fg7XRkPT1Tcj5PpDfi_Rryle6c04bPL-PwRZMaKTIY3knrgMuFMYEt2VshSuqVxiuuhIo4_
SUu7VVInJq2NpW1GUa8IavWdCTcnCm49stcag6IkKtu4p6_nZgbs8oHKXc4AJVYd6yv9_Vjo4SF48h7tUS-596oqWK6fJvppsy
89tYuLuUsV4Dg0iNha0R&r=1702771575091654957

If we decode the above Base64 encoded query we can see that the HTTP request contains a lot of private information that is being exfiltrated:

http://big4u.org/u/?q={"dns_setter":{"activity_type":16,"args":{"!":"6bf129378af2417c"},"bits":
{"file_type":2,"job_id":"3267496844249690321"},"build":160,"exception_id":0,"hardware_id":"2748707
171507313081","is_admin":true,"major":1,"minor":0,"os_id":603,"register_date":"1456082605","regist
er_dsrc":"1","report_id":"1702771575091654957","service_pack":0,"source_id":"302","status":true,"s
uspect_flags":0,"suspect_group":"","suspect_info":"","user_time":1456064633,"version":16777376,"x6
4":true}}&c={"dns_configuration":{"affiliate":"55","bits_domains":"legco.info/u/;ough.info/u/;heat
o.info/u/;yelts.net/u/;deris.info/u/;big4u.org/u/;listcool.net/u/;listcool.info/u/;monoset.info/u
/","bits_interval":6,"bits_jobs":2,"bits_timeout":90,"dns_list":"82.163.143.171;82.163.142.173","n
et_timeout":45,"pshell_domains":"likerut.info/u/;theget.biz/u/;bootfun.info/u/;sportnew.net/u/;ukj
obmy.com/u/;moonas.info/u/;fasilmy.info/u/;paneljob.info/u/;usafun.info/u/;safesuns.info/u/","psh
ell_interval":8,"pshell_jobs":1,"report_domains":"riyah.info;riyah.net;zambi.info;lenda.info;amous
.net","report_ip":"185.17.184.11","retry_limit":3,"retry_period":10,"session_id":"3153783903249204
410"}}&r=1702771575091654957

The exfiltrated data contains the version of the operating system including the registration status, info about the installed updates, architecture and the rights that the malware was able to obtain (in this case full administrator rights). Additionally, the info about the changed DNS records is sent back to the C&C.

4.1.4 Long First Touch Anomaly Detection Method

The above analysis motivated the creation of the proposed Long first touch anomaly detector. The detector assigns a non-zero anomaly score only if both the following conditions are met:

- the Referer is not valid (it is empty or nonexistent). If there is a valid Referer that points to an URL that was visited by the user in the past it represents an additional resource of a web page or an user clicking on a link on different site that generated the current request. Both these legitimate cases may result in a long URLs that event though not directly typed by the user, do not represent malicious behavior.
- the requested domain was not visited by the user in the past using a short URL (smaller than 40 characters). All such domains are stored for each user using the Bloom filter data structure [20] that allows to store huge amount of domain names that the user visits over time. This filters out various status checks and updates of a sites that are opened in the browser as their Referer is typically empty or cannot be found in the cached flows (only flows from last five minutes are cached).

The anomaly score is then estimated using the length of the URL, as the longer URL are capable of transferring more information, as

$$f(x) = 1 - \exp\left(-\frac{1}{\lambda l(x)}\right), \tag{4.1}$$

where the $l(x)$ is the length (i.e. number of characters) of the URL of the flow x and λ is a normalization parameter of the algorithm. The evaluation on real network data showed that setting $\lambda = 40$ gave consistently good results.

4.1.5 Experiments

We have evaluated the proposed detection algorithm using HTTP proxy logs collected in four different corporate networks of various sizes ranging from 5,000 users up to 40,000 users. Using a signature-based detection, we were able to identify five different malware families that are known to exfiltrate information through the URL. All the signatures were either created by a network intrusion detection analyst or acquired using publicly available signatures including various domain blacklists and security blogs.

The identified malware families include:

- DNS Changer Trojan [169] is a malware capable of stealing sensitive user information that is described in detail in Section 4.1.3.
- Kazy malware [155] has been associated with a variety of different criminal activities, including keyloggers, phishing scams and data theft.
- Zeus malware is known for stealing sensitive user information by using a man-in-the-browser web injection module. The module enables the malware to modify web pages on the client side, modify transaction content, and insert additional transactions without the user knowing.
- Cutwail botnet [164] is a botnet mostly involved in sending spam e-mails. Some of the variants are connecting to the C&C server via HTTP to receive instructions about the emails they should send. After they are done with their task, the bots report back to the owner the exact statistics on the number of emails that were delivered, and on which and how many errors were reported. This is again done via encoding the information into the URL.
- Trojan.Bumat!rts [127] is typically hidden within legitimate executable files belonging to other programs. A hacker may use Trojan.Bumat!rts to infiltrate an infected system and put stored data at risk of being stolen.

The AUC scores, presented in Table 4.2, show high detection quality for all the identified malware types. The only slightly worse efficacy was achieved for the Zeus malware, that uses slightly shorter URLs than the other malware families.

Malware family	AUC
DNS Changer Trojan	0.99
Kazy malware	0.97
Zeus malware	0.93
Cutwail botnet	0.97
Trojan.Bumat!rts	0.98

Table 4.2: Results of the Long first touch anomaly detector. The AUC scores are evaluated for several types of malware that are known for exfiltrating private information using the URL.

4.2 User-Agent Discrepancy Detector

The User-Agent discrepancy detector uses User-Agent [50] HTTP request header field that contains information about the user agent originating the request (i.e. application that created and sent the request). The User-Agent is typically used for statistical purposes (statistics of the overall usage of particular browser or operating system for accessing particular domain), the tracing of protocol violations, and finally for the automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations[3].

[3] The usage of User-Agent field for recognizing of the user agent to know the limitations of that particular application proved to be ineffective because of compatibility problems encountered each time a new version of

Although it is not required, each HTTP request should include User-Agent field. The field can contain multiple product tokens and comments identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.

Since the User-Agent string representing users browser is commonly user specific (we found out that one exact User-Agent is on average used in the network only by two users) and also typically created at runtime using version information of various parts of the operating system, it is not an easy task to obtain the exact same User-Agent as the user normally uses. Because of that, the malware creators are typically using either their own specific non-browser User-Agent string or User-Agent string of one pre-selected browser, hardcoded in the malware binary. This gives us the opportunity to detect such User-Agents that are discrepant with the ones that the user of infected machine typically uses. The presented approach is not able to detect malware that hijacks the User-Agent of the browser that the user is typically using. But we found out that 95% of evaluated malware samples were using hardcoded User-Agents in their HTTP requests.

Furthermore, the User-Agent string can be easily changed by the user. There is a number of browser plugins that allow users to quickly change their User-Agent (these are recently heavily used by user of Windows XP, that are spoofing their User-Agents to look like they use newer version of Windows OS, because of the end of the support of Windows XP and lot of banks are blocking those obsolete operating systems to access their on-line banking system). This could cause a lot of false positives, because of the artificial changes of the User-Agents. But we found out that the users are not switching the User-Agents frequently, when using those tools.

4.2.1 User-Agent Anomaly Detection

Malware-infected user that uses HTTP to communicate with the C&C server needs to create the HTTP header in which the User-Agent field should be filled. The User-Agent, that is set by the malware, can have one of the following forms:

1. **Empty User-Agent field.** Some of the malware is leaving the User-Agent unset to avoid detection or simply because of the fault of the malware creator.
   But, since the User-Agent field is not required and the specification [50] only says that the User-Agent "should" be filled, there are some legitimate applications that can also create HTTP requests with the User-Agent field unset. The portion of such requests is really small compared to others.
2. **Specific.** In this case the User-Agent is a non-browser (does not follow standard browser pattern of the form `Mozilla/X.X (additional information)`). These can represent a bug in the malware source code such as misspelled commonly used User-Agents, programming language default User-Agents (we have encountered malware that used Visual Basic defaults) or it can be set to a specific string on purpose to implement some functionality that identifies the botnet (e.g. handshake for the server to handle malware requests differently than the ones of normal users) or leak some information (e.g. information about the infected user that can contain IP or other user identification, information about installed anti-virus applications, etc.) or bogus string like `HTMLGET 1.0` as used by IKEE.B botnet.
   Again, some of the specific User-Agents may be used by legitimate applications. These are various update clients, toolbars, RSS readers, etc. These User-Agents are typically used by multiple machines in the network since there are more users that are using the same application or more machines of the same type (e.g. printer of one manufacturer) with the same default User-Agent

Internet Explorer was shipped. As a consequence, the logic around the User-Agent string has grown increasingly complicated over the years. The introduction of Compatibility Modes has meant that the browser now has more than one User-Agent string, and legacy extensibility of the string was deprecated after years of abuse.

3. **Spoofed**. Malware sets HTTP User-Agent as the user-specific browser would do. This means that the malware is able to detect which browser is used by the user and extract the User-Agent from that particular browser. This is not a trivial task because the User-Agent is typically constructed at runtime using various parts of the OS. In this case the proposed method can not identify the malware based on User-Agent information alone, since the User-Agent used by the malware would be aligned with the one the user typically uses. Another approach is to hijack the browsers process to perform the request for the malware, in this case the User-Agent will be also the same as the user is typically using. In this case the malware is unrecognizable from the normal user when focusing only on the User-Agent field and the proposed detection algorithm wont be able to detect this type of malicious requests.
4. **Discrepant**. Malware sets HTTP User-Agent to a constant string, representing a browser, not matching the current computer environment. Since the hijacking of the browser is not a trivial task, as described above, there is a number of malware that uses this approach to fill the User-Agent field in the HTTP request. This approach is undetectable by the signature matching techniques as the User-Agent is typically a legitimate one.

We propose three different anomaly detection approaches to detect malware that falls into one of the categories: Empty, Specific or Discrepant. The malware that uses Spoofed User-Agent cannot be detected using the User-Agent feature only, but as shown in Section 4.2.2.4 the spoofed User-Agent is used only by less than 20% of malware.

Each of the anomaly detection algorithms described below first extracts user identifier, IP address, visited domain and User-Agents string from the HTTP flow, then the User-Agent string is assigned to group of empty, well-known browsers or unknown User-Agents. The browser and non-browser recognition is done using UADetector library [146] which uses regexp matching to recognize the well-known User-Agents. This library also provides the browser family type (Internet Explorer, Firefox, Chrome, etc.) that is used by the Discrepancy detector. The rest of the anomaly detection process differs for the three groups: unknown User-Agents are handled by Unknown-Unused User-Agent detector, described in Section 4.2.1.2, anomalous User-Agents of well known browsers are detected using User-Agent Discrepancy Detector described in Section 4.2.1.3. Finally, the flows with missing User-Agent are evaluated using the No User-Agent anomaly detector, described in Section 4.2.1.1.

4.2.1.1 No User-Agent Anomaly Detector

As stated above, the empty User-Agent in non-HTTPS request is created by applications that simply do not fill this field. These are typically various update services or programs that are requesting limited set of services from the Internet (e.g. weather updates, translations, news, etc.). Using this we can simply model which domains are used by a larger set of users and label domains that are visited by minority as anomalous as it is less likely that the majority of network users would be infected with the same malware.

Therefore, in this case, we propose to model the usage of domains. For each domain we count the number of unique visitors, creating a domain usage histogram. Each bin of the histogram represents number of users of one particular domain d. The anomaly f is then equal to normalized surprisal [118], that is computed as:

$$f(d) = -log(P(d)) - H(\{h(y)\}_{y \in D}), \tag{4.2}$$

where the D represents a set of all observed domains accessed with no User-Agent, $H(\{h(y)\}_{y \in D})$ is the entropy of the bins and the $P(d)$ is the probability of the domain d estimated using the histogram as $P(d) = \frac{d}{\sum_{y \in D} h(y)}$.

4.2.1.2 Unknown Unused User-Agent Detector

The unknown, non-browser User-Agents typically represents special application that is using HTTP for its communication. The reason for such behavior is that these applications can either work with the web content or resources (RSS reader, Java, Adobe Flash, Weather toolbar, etc.) or are similarly to the malware using a HTTP to avoid blocking of their communication on the firewalls (Skype, Windows Update Service, etc.).

For the unknown, non-browser User-Agents we propose to model User-Agent frequency among all the users. The detection algorithm start with the removal of version number and additional information from User-Agent string to discard permissible differences in user node configurations (i.e. `Dropbox 2.4.3 (WinXp, x64)` will become `Dropbox`). The key assumption is that the frequency of each User-Agent string in this group over all network users is roughly comparable for each string representing benign traffic. Thus we calculate the average frequency of each User-Agent string among all network users and mark as anomalous those User-Agent strings that are in the left tail of the distribution.

The final anomaly score is therefore calculated as follows:

$$f(x) = \begin{cases} 0 & \text{if } x \leq \tau_1 \\ \frac{h(x)-\tau_1}{(\tau_2-\tau_1)} & \text{if } \tau_1 < x < \tau_2 \\ 1 & \text{if } x \geq \tau_2 \end{cases},$$

where $h(x)$ is number of occurrences of the User-Agent string x and τ_1 and τ_2 are thresholds, set to 10 and 30 percentile of the distribution of the frequencies over all User-Agents.

4.2.1.3 User-Agent Discrepancy Detector

Detection of anomalous well-known, browser User-Agents is based on the fact that one user uses only one version of a web browser on one machine. Almost no one has a two different versions of the same browser — for some browsers it is not even possible.

The anomaly detection algorithm, that identifies discrepant User-Agents, works as follows. We collect all the User-Agent strings for each user and browser family. Thus we know the exact version of all the browsers that the user uses on his machine. If we find a User-Agent of the same family that differs from the commonly used one, we first check if the user updated his browser (version of some of the components increased and the user is using the newer User-Agent string from now on) otherwise the User-Agent is labeled as malicious.

Additionally, there are several additional rules that solve some of the special cases like Internet Explorer's compatibility mode, that can change the versions of some components, version change caused by usage of both the 32 and 64-bit versions of Internet Explorer on x64 machine, usage of the chromeframe inside Internet Explorer, etc.

The anomaly detection algorithm models all the versions of all the browsers used by one user using a histogram. Each bin in the histogram represents the number of occurrences of one of the browser version string. The final anomaly is calculated as follows:

$$f(x) = 1 - \frac{h(x)}{\sum_i (h(i))}, \tag{4.3}$$

where the $h(x)$ is the number of occurrences of the User-Agent x.

4.2.2 Experiments

The experiments were performed using the HTTP access logs collected over the period of 14 days from mid-size company with more than 6,000 active users, that generated over 700 million flows.

4.2.2.1 No User-Agent Detector

We have manually analyzed portion of the HTTP requests with missing User-Agent field. Each analyzed request was evaluated using various publicly available detection and reputation systems and labeled as malicious or legitimate. Using this process we have labeled 1.5 million HTTP flows as legitimate and 4.5 thousand as malicious from the total of 4 million, having 35% label coverage.

Using this approach, we have been able to identify only one HTTP domain, that was reported to communicate with a number of malware families. The domain was mainly used as a C&C of a Trojan.Dropper malware. The AUC value for the No User-Agent detector on this malware sample is shown in Table 4.3.

Malware	AUC
Trojan.Dropper	0.97160

Table 4.3: AUC of the No User-Agent detector on one of the identified malware samples.

4.2.2.2 Unknown-Unused User-Agent Detector

Similarly to above, we have manually labeled all the HTTP requests with non-browser User-Agents. From the total number of 42 million HTTP flows we have been able to identify 0.5 million malicious flows and label 39 million flows as legitimate resulting in 92% coverage.

All the malicious flows were generated by several users that were infected by malware of one of the following families:

- H-worm [74] is a malware that tunnels information about the infected machine through the User-Agent field. The User-Agent is of the form, `ECAB5BF2<|> NCPD312722<|> vazquezp<|> Microsoft Windows 7 Enterprise<|> underworld final<|>McAfee VirusScan Enterprise .<|>false`, containing private information like username, installed antivirus, computer id and main drive id.
- Win32/Sality.3 generates flows with User-Agent set to `MyApplication 1.0`. In this case the malware creator used default setting of his development environment when implementing the HTTP communication. This is a 11-years old malware used to send SPAM and perform DDoS attacks that has been recently updated to run also on routers to spread itself among the infected router's users.
- Win32.QQPass uses User-Agent of the form `GetWeb` to access several subdomains of the qq.com, a dynamic DNS service, that were all reported by number of web analyzers as malicious.
- QQ_hider is a malware that was unknown at the time of the experiment evaluation (no other security solution was able to detect it). It again uses the qq.com service for contacting the C&C server. User-Agent of this malware is set to `QQ Bug Report Tool`, but there is no such an application provided by qq.com. Furthermore, it contacts the bugreportv2.qq.com

subdomain, that is one of the regular dynamic DNS domains. Therefore, it seems that this is really nice camouflage of some malware.

The Table 4.4 shows the evaluation results of the Unknown-Unused detector on the whole dataset. As can be seen the detector performed very well on all the manually identified malware samples, described above. The best detection performance achieved is for H-worm malware. This is caused by the fact that the malware uses different User-Agent string for every user, so from the Unknown-Unused detectors perspective these are the most anomalous User-Agents. The others have slightly worse AUC value, because there were more users that were infected by the same malware.

Malware	AUC
H-worm	0.96996
Win32/Sality.3	0.95403
Win32.QQPass	0.94982
QQ_hider	0.94950

Table 4.4: AUC of the Unknown Unused User-Agent detector on the manually labeled malware samples.

4.2.2.3 User-Agent Discrepancy Detector

Since there is a great number of HTTP flows with a web browser-like User-Agents (more than 600 million flows with lot of them being unique), we were not able to manually analyze such a portion of the data to have significant coverage that could be used for the evaluation. Instead of the manual labeling of all the browser-like flows, we have just analyzed the ones, labeled by the Discrepancy detector as the most anomalous and assumed the rest to be legitimate. For this we have made an analysis of all flows that had anomaly value greater than 0.8. This way we can correctly estimate the precision of the detector but ignore the recall.

We have successfully identified only two malicious users infected by malware of two different families:

- Win32/Alman was using discrepant User-Agent of a form `Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)` that is a common IE User-Agent.
- Trojan.Win32.Dropper.aa used `Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.5072; .NET CLR 3.0.04506.30; InfoPath2)` which again is a legitimate User-Agent but in our case discrepant with the one the user was using.

Since we were able to identify only two malware families in the network, we have artificially added one more to the weblog dataset. We have extracted the weblog from the network traffic of the Flame/SkyWiper malware binary and mixed it into the data. The malware flows were changed to have an IP and user identifier of one of the legitimate users from the network, that uses MSIE of a version 6.0 on a Windows OS.

The Flame/SkyWiper was selected because of its analysis made by [82]. This malware uses specific User-Agent that looks like: `Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.2150)`. This User-Agent was artificially created, since the .NET of version 1.1.2150 was never released by Microsoft.

The Table 4.5 shows the AUC evaluation of the User-Agent discrepancy detector for the both malware families, identified in the network and the simulated one, that was evaluated in another round of the evaluation. Only the Trojan.Win32.Dropper.aa has a AUC value less than 0.9. This was caused by the fact that this malware was sending a lot of requests from a user, that was using the HTTP rarely, having comparable usage to the user's legitimate User-Agent.

Malware	AUC
Win32/Alman	0.91564
Trojan.Win32.Dropper.aa	0.87745
Flame	0.94855

Table 4.5: AUC of the User-Agent Discrepancy detector on two real malware families identified in the network traffic and one artificially added to the same background traffic.

4.2.2.4 User-Agent Usage by Malware Samples

To demonstrate usefulness of the proposed method we have analyzed a big portion of malware from the Totalhash malware database. This publicly available database contains huge number of malware samples together with a analysis made by their sandboxing solution. Since all the malware is run inside specific sandbox using one OS with one default User-Agent (`Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727)`) it is easy to compute simple statistic to know how various malware samples are setting the User-Agent.

We were able to acquire information about the behavior of 181 762 malware samples that were using HTTP for the communication. From this number only 34 488 malware samples were using a User-Agent aligned with the default User-Agent of the sandboxed OS. This is the upper bound, since there can be malware samples, that are using hardcoded User-Agent aligned with the one of the sandbox VM only by coincidence. 77 348 samples were using browser–like User-Agent, discrepant with the default one. There were also 43 840 samples with empty User-Agent fields. The remaining 26 086 were non-browser User-Agent strings. These were further investigated to see if the malware is using popular non-browser User-Agents to reduce the probability of detection. Surprisingly, only 5 samples were using Java User-Agents, 7 were using User-Agents of Adobe Flash and only 8 used User-Agent of TeamViewer. Most of the User-Agents were of the form `Downloader X.XX`, `UniversalUserAgent(winHTTP)`, `httpget`, `ABC`, etc. The numerical proportions of individual categories described above are illustrated on Figure 4.2.

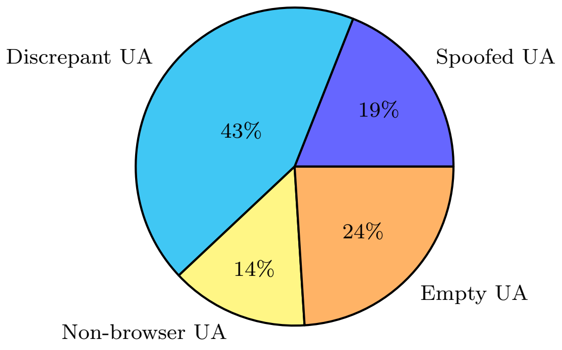

Fig. 4.2: Proportions of various User-Agent categories observed on 181 762 malware samples in the Totalhash database. Almost half of the observed samples were using discrepant browser User-Agents. 24% of the malware samples were sending HTTP requests with empty User-Agent field. There were only 19% samples with spoofed User-Agent. Finally, only 14% samples were using Non-browser (unknown) User-Agents.

This proves presented approach to be usable in practice, since more than 81% of the malware from the totalhash database, that use HTTP for communication would be detectable by one of the proposed anomaly detection mechanism.

4.3 Chapter Summary

We have introduced the CTA detection engine that uses more than 30 various anomaly detectors that are specialized in detecting anomalous HTTP(s) requests collected by the HTTP proxies. Two novel knowledge driven detection methods were proposed and their efficacy was experimentally evaluated using a real network data.

Chapter 5
Large Margin Ensemble Optimization

This chapter introduces a novel supervised method to learn a combination of a number of heterogeneous anomaly detectors, where some of them can be specialized to particular types of malicious behavior, whereas others can be general anomaly detectors capable of detecting previously unseen attacks at the expense of higher false alarm rates. Such a setup has multiple advantages, including faster processing of the data stream, lower complexity of the detectors, and simpler inclusion of domain knowledge into the system (see Chapter 1 for detailed discussion). It can be argued that using a supervised method to learn the combination may bring the expense of lower generalization, but according to our experience completely unsupervised approaches rarely have false positive rate low enough to be usable in practice. Moreover, anomaly detectors and their features are usually selected based on the experience of the designer, which is a kind of proxy for labels and surely not guaranteed to be complete.

Although a vast prior art on the problem exists (see Section 2.2), we believe that peculiarities of the security domain, namely a highly imbalanced ratio of non-alarm and alarm samples in the data, lack of accurately labeled datasets, and the need of extremely low false positive rates, call for a tailored solution. We propose a method of finding a convex combination of outputs of a fixed set of anomaly detectors maximizing the number of true alarms in τ-fraction of most anomalous flows and an experimental study of the effect of different types of label noise in the training data on the accuracy of combinations obtained by different methods to better understand their advantages and drawbacks.

If the proposed method requires labeled data, one can ask why not use them to train a classifier and sidestep the use of anomaly detectors? The most important reason to favor their use is that network traffic discussed in this chapter is very non-stationary and anomaly detectors are good at coping with this aspect, as they can constantly update their models (see [123, 32, 23] for a review).

This chapter is organized as follows: We first introduce a method for scaling the outputs of anomaly detectors to guarantee scores of the same scale to be able to easily combine them. Section 5.2 formally defines the combination problem and presents the proposed solution. The experimental Section 5.3 compares the proposed solution with existing methods using sets of anomaly detectors of both CAMNEP and CTA network intrusion detection systems.

5.1 Scaling Outputs of Anomaly Detectors

Generally, individual anomaly detectors need not generate anomaly scores of the same scale. This causes problems during the combination process, since one or more detectors could be inadvertently favored. Therefore, the anomaly scores generated by both the CAMNEP and CTA anomaly detectors are normalized using the *gaussian scaling* proposed by Kriegel *et al.* [92]:

$$\tilde{h}(x) = \max\left\{0, \mathit{erf}\left(\frac{h(x) - \mu_h}{\sigma_h\sqrt{2}}\right)\right\}, \tag{5.1}$$

where the $\tilde{h}(x)$ is the normalization of the anomaly score $h(x)$ assigned to the flow $x \in \mathcal{X}$ by anomaly detector h. The used Gaussian Error Function *erf*() is monotone and thus ranking stable. The μ_h and σ_h are the mean and the standard deviation of the anomaly scores returned by the anomaly detector h. These can be estimated from the data using $\hat{\mu}_h = E(h)$ and $\hat{\sigma}_h = \sqrt{E(h^2) - E(h)^2}$ in single run or adaptively adjusted when running in an on-line scenario. This transforms the anomaly scores of individual anomaly detectors into probability estimates, where the probability of zero represent normal flow, aligned with the predictive model, whereas one indicates highly anomalous flow. These are therefore directly comparable and can be aggregated using a number of combination techniques [92].

5.2 Ensemble Construction Method

We assume that the network operator observers network flows (samples) from an unknown distribution $P_o = \pi P_a + (1 - \pi)P_b$ with P_a / P_b being distributions of malicious / background samples and $\pi \in [0, 1]$. The network operator uses set of m anomaly detectors on samples $\mathcal{H}_m = \{h_k : \mathcal{X} \mapsto [0, 1]\}_{k=1}^m$ (w.l.o.g. it is assumed that zero means the sample is legitimate and one means the sample is malicious) and wishes to have a convex combination of anomaly detectors $\alpha = (\alpha_1, \ldots, \alpha_m)$ that would maximize the number of alarms in top τ quantile of the distribution of the combined anomaly scores. For purposes of this chapter it is safe to assume that each flow (sample) is described by m-dimensional vector (an output of m anomaly detectors), which implies that distributions P_o, P_a, and P_b are defined on the m-dimensional Euclidean space.The requirements on detectors having their image in the interval $[0, 1]$ and learning a convex combination instead of a linear one are to improve interpretability of the results as discussed in [92], but can be dropped.

With respect to the above, networks operator's goal can be written as

$$\arg\min_{\alpha\in\mathbb{R}^m} R(H_\alpha) = \underbrace{\mathbb{E}_{x\sim P_b}\left[\mathbb{1}(\alpha^{\mathrm{T}}h(x) \geq q_{\alpha,\tau})\right]}_{R^{\mathrm{fp}}(H_\alpha)} + \underbrace{\mathbb{E}_{x\sim P_a}\left[\mathbb{1}(\alpha^{\mathrm{T}}h(x) < q_{\alpha,\tau})\right]}_{R^{\mathrm{fn}}(H_\alpha)}, \tag{5.2}$$

subject to

$$\begin{aligned} H_\alpha(x) &= \sum_{k=1}^{m} \alpha_k h_k(x) = \alpha^{\mathrm{T}} h(x), \\ \mathbf{1}^\top \alpha &= 1, \\ \alpha_i &\geq 0,\ \forall i \in \{1, \ldots, m\}, \end{aligned} \tag{5.3}$$

where the first term in (5.2) is the false alarm rate, the second term is the false negative rate, and finally $q_{\alpha,\tau}$ is a τ-quantile of observed distribution of ensemble's output $\{\alpha^{\mathrm{T}}h(x)|x \in P_o\}$. The minimized term (5.2) captures the accuracy of a particular convex combination in top τ-quantile of its distribution, which is the goal.

In theory it would be sufficient if (5.2) minimizes either only false positive rate R^{fp} or only false negative rate R^{fn}, because each of them together with constraints (5.3) implies minimization of the other. But the experiments suggest that including both terms increases the robustness with respect to noise on labels, since the error and its gradient are estimated from larger number of samples implying their better estimates. This is demonstrated in Section 5.3.4, where the combination of anomaly detectors was found by optimizing either only false positive rate or only false negative rate under constraints (5.3). The experiments have confirmed that optimizing the proposed (5.2) is indeed more robust to error in labels, which are almost inevitable in security domains. In the rest of this section we show, how to find a good solution in practice using adaptation of the method of Boyd et al. [22].

First, the true loss function (5.2) cannot be used in practice, since the true probability distributions P_a and P_b are not known. Therefore the expectations are replaced by their empirical estimates calculated from some labeled data used for learning the weight vector α. Below the $\mathcal{S} = \mathcal{S}_a \cup \mathcal{S}_b$ denotes the set of available samples with $\mathcal{S}_b$ being the set of background (legitimate) samples and $\mathcal{S}_a$ the set of malicious samples. The empirical estimate of (5.2) is therefore

$$\hat{R}(H_\alpha) = \frac{1}{|\mathcal{S}_b|}\sum_{x\in\mathcal{S}_b} \mathbb{1}\left[\alpha^{\mathrm{T}}h(x) \geq \hat{q}_{\alpha,\tau}\right] + \frac{1}{|\mathcal{S}_a|}\sum_{x\in\mathcal{S}_a} \mathbb{1}\left[\alpha^{\mathrm{T}}h(x) < \hat{q}_{\alpha,\tau}\right], \tag{5.4}$$

where $\hat{q}_{\alpha,\tau}$ is an empirical estimate of the true quantile $q_{\alpha,\tau}$ defined as

$$\hat{q}_{\alpha,\tau} = \arg\max_{\omega} \frac{1}{|\mathcal{S}|}\sum_{x\in\mathcal{S}} \left[\mathbb{1}(\alpha^{\mathrm{T}}h(x) \leq \omega)\right] \leq \tau. \tag{5.5}$$

Since the empirical loss function (5.4) is neither convex nor smooth, finding the optimal solution is an NP-complete problem. A usual approach is to replace indicator function $\mathbb{1}$ with a convex surrogate, for example an exponential used in this work[1]. This substitution leads to the following optimization problem

$$\begin{aligned} \arg\min_{\alpha} \quad & \frac{1}{|\mathcal{S}_b|}\sum_{x\in\mathcal{S}_b} \exp\left(\alpha^{\mathrm{T}}h(x) - \hat{q}_{\alpha,\tau}\right) + \frac{1}{|\mathcal{S}_a|}\sum_{x\in\mathcal{S}_a} \exp\left(\hat{q}_{\alpha,\tau} - \alpha^{\mathrm{T}}h(x)\right) \\ \text{subject to} \quad & \mathbf{1}^{\top}\alpha = 1, \\ & \alpha_i \geq 0,\ \forall i \in \{1,\dots,l\}, \\ & \hat{q}_{\alpha,\tau} \text{ is a } \tau\text{-quantile defined in (5.5).} \end{aligned} \tag{5.6}$$

where the optimized term (further denoted as $\hat{R}_{\exp}(H_{\boldsymbol{\alpha}})$) is an upper bound of the empirical loss function $\hat{R}(H_{\boldsymbol{\alpha}})$ defined in Equation (5.4).

Nevertheless the last problem is still hard to solve, as it is not convex. Boyd et al. [22] showed how to find a good solution in polynomial time using series of convex problems. However his algorithm does not guarantee to find the global minimum, and the computational complexity prevents it to be used on problems with millions of samples. We therefore propose to solve (5.6) by a simple gradient algorithm summarized in Algorithm 1, which albeit not reaching the global minimum performs well, according to our experiments. In each step the current solution $\boldsymbol{\alpha_k}$ is updated by subtracting a small multiple of the gradient of (5.6), which is decreasing in each step to ensure convergence. The $\boldsymbol{\alpha_k}$ is then truncated to satisfy the constraints, and finally the estimate of the quantile $\hat{q}_{\alpha,\tau}$ is updated. The algorithm may find sub-optimal solutions but the experiments in Section 5.3 show that the found solutions are in most of the cases better than the ones of the state-of-the-art methods.

The combination of detectors found by the above algorithm is optimized with respect to the *known* malware, by which we understand the malware whose samples are present in the training set and most of them are correctly labeled. We believe that it is very hard to draw any conclusions about the accuracy of the algorithm on malware that has never been observed. If the unknown malware is similar to the known one (e.g. using similar components or having similar behavior), then it is likely that the above optimization will help. In order to get insight to this phenomenon on real data, the experimental section compares accuracy of several algorithms on training sets with errors on labels of different types. We believe this study will help to select the right algorithms for practice.

[1] The chosen convex surrogate does not have a significant impact on the solution and can be replaced by the reader's favorite choice, e.g. logistic, hinge, truncated square, etc.

```
Data: Set of labeled samples $x_1, \ldots, x_l \in \mathcal{S}$,
      set of anomaly detectors $\mathcal{H}_m$
      and $\delta_{min}$.
Result: weights $\boldsymbol{\alpha} \in \mathbb{R}^m$
Start with equal weights $\boldsymbol{\alpha_1} = \mathbf{1}/m$;
repeat
    Set $q_{H_{\boldsymbol{\alpha}}}(\tau)$ to be $\tau$-quantile of the distribution of $H_{\alpha_k}$;
    Update the step size as $\gamma_k = \frac{1}{\sqrt{k}}$ ;
    $\boldsymbol{\alpha_{k+1}} = \boldsymbol{\alpha_k} - \gamma_k \frac{\partial}{\partial \alpha} \hat{R}_{\exp}(H_{\boldsymbol{\alpha_k}})$;
until $|R(H_{\boldsymbol{\alpha_k}}) - R(H_{\boldsymbol{\alpha_{k-1}}})| < \delta_{min}$;
```

Algorithm 1: The algorithm used to solve the optimization problem (5.6).

5.3 Experimental Evaluation

The proposed combination technique was evaluated and compared to prior art using the anomaly detectors of CAMNEP and CTA intrusion detection systems. The first one, described in Section 5.3.2, uses NetFlow [30] records, while the second one, described in Section 5.3.3, uses logs from HTTP proxy servers.

In this chapter we use a measures from information retrieval to compare the algorithms, namely *precision* and *recall*. Assuming that malware samples have positive labels, precision is the fraction of the number of malware samples classified as positive and the total number of samples classified as positives, and recall is the fraction of malware samples classified as positives and the total number of malware samples. To highlight that the detection threshold is set to 1% of the most anomalous samples, we abbreviate both measures as (*Prec@1%*) and recall (*Rec@1%*). When evaluating the accuracy at top, the use of precision and recall is preferred over the popular area under the Receiver Operating Characteristic curve (AUC ROC) [49], because the latter compares the algorithms in areas which are outside the region of the interest (top 1% anomalies).

The use of machine learning methods in security is frequently hindered by the lack of fully labeled dataset. While samples labeled as malicious are most of the time connected to some malicious behavior, it can frequently happen that some background samples are actually malicious, but the labeling oracle (analyst) has failed to recognize them. Experiments described below aim to simulate three types of noise in labels (and of course the noise-less case denoted as Non.) to investigate their effect on the learning of the combination function. The types of considered label noise are:

- Samples of all types of malicious activities are within the training data, but 50% of samples of each activity type were not recognized as malicious by the oracle (human), and therefore they are labeled as a background. This case is denoted below as anomaly label noise (ALN).
- Samples of some (50%) types of malicious activities are completely missing in the training data, but they are present in the testing data. Samples of remaining types of malicious activities are present in the training set, but as in the previous case the labeling oracle did not recognize 50% of their samples. This case is denoted as missing anomaly types (MAT).
- Samples of all types of malicious activities are present in the training data, but the oracle did not know 50% of types, and labeled them as background. On samples from remaining 50% of types of malicious activities present in the training set the oracle again made a mistake and has labeled them as background. This type of noise is further denoted as anomaly label noise with type mislabeling (MLT).

The testing set was always noiseless to allow for fair comparison and evaluate the effects.

Datasets for each intrusion detection engine are described in corresponding subsection. The available data was split so that 50% of samples were used to learn the combination of anomaly detectors and the rest for testing. This split has been repeated five times to account for the variance of the estimate.

5.3.1 Compared Algorithms

The experimental comparison involves four unsupervised combination rules (*mean*, *max*, *rank BFS* [97], *mean rank* [97]), and four combination rules trained by supervised methods (SVM-*perf* [84], TopPush [101], RankBoost [54], and the proposed method). All the methods are described in detail in Section 2.2.

SVM-*perf* used L1-slack algorithm with constraint cache setting, so that 1% of positive examples was used as value of k for *Prec@k*. Regularization constant in TopPush was set to one. The proposed method (*Acc@Top*) was set to optimize the accuracy in top 1% of most anomalous samples, which means $\tau = 0.99$.

Algorithms chosen for comparison enabled comparing unsupervised methods among themselves (repeating the experiment in [154]), relevant supervised methods among themselves, and also the gain one can expect when using supervised methods even though the labels are not perfect.

5.3.2 Evaluation on NetFlow Anomaly Detection

The NetFlow anomaly detection engine [141, 59], introduced in Chapter 3, processes NetFlow [30] records exported by routers or other network traffic shaping devices. The anomaly detection engine identifies anomalous traffic using an ensemble of anomaly detection algorithms. Some of them are based on Principal component analysis [95, 96, 125], others detect abrupt changes in the behavior [45] or even use fixed rules [177]. Furthermore, there are detectors designed to detect specific type of unwanted behavior like network scans [160] or malware with domain generating algorithm [67]. In total the NetFlow anomaly detection engine uses 16 anomaly detectors. Thus the goal is to find a linear combination of these 16 anomaly detectors maximizing the accuracy in top 1% quantile.

The evaluation used several datasets from a traffic captured on the network of Czech Technical University (CTU) in Prague. The datasets and labels especially were created by three different approaches: manual labeling, infecting virtual machines, and performing real attacks against our computers within the network. In manual labeling, experienced network operator was able to successfully identify malicious activities that generated almost 10% of the total number of the flows (samples). In datasets with manually infected virtual machines[2] all their connections were labeled as malicious, whereas the rest was labeled as background. In the final dataset a network specialist run several attacks against one computer in the network. The attack vector consisted of a horizontal scan to discover open SSH ports, followed by SSH brute-force attack to break the password, and finished by SSH login and data download simulating data theft.

Figure 5.1 shows precision-recall curves for eight compared algorithms. The graphs demonstrate that the combination found by the proposed method (*Acc@Top*) most of the time dominates all other methods and fixed combination rules. Notable exceptions are cases when some types of malicious activities are completely missing in the training data (MAT) or they are incorrectly labeled (MLT). In these cases unsupervised *mean rank* combination is better on the lower recall part of the curve. This behavior suggests that different anomaly detectors detect different types of malicious activities and the supervised combination has slightly overfitted. In practical applications combining supervised and unsupervised combination rules should be used to ensure good accuracy on known malicious activities and simultaneously some generalization on unknown alerts, where the precision will be substantially smaller. Also notice that the proposed algorithm is the most robust with respect to noise from all supervised ones. SVM-*perf* is good in the noiseless case, but poor when noise of any kind is present. The TopPush is slightly more robust, but still it performed poorly with noise of MAT and MLT types, both of which are also the hardest cases. Unsupervised combination function **mean rank** performed the best

[2] Neeris, FastFlux and RBot were used to infect the machines [59].

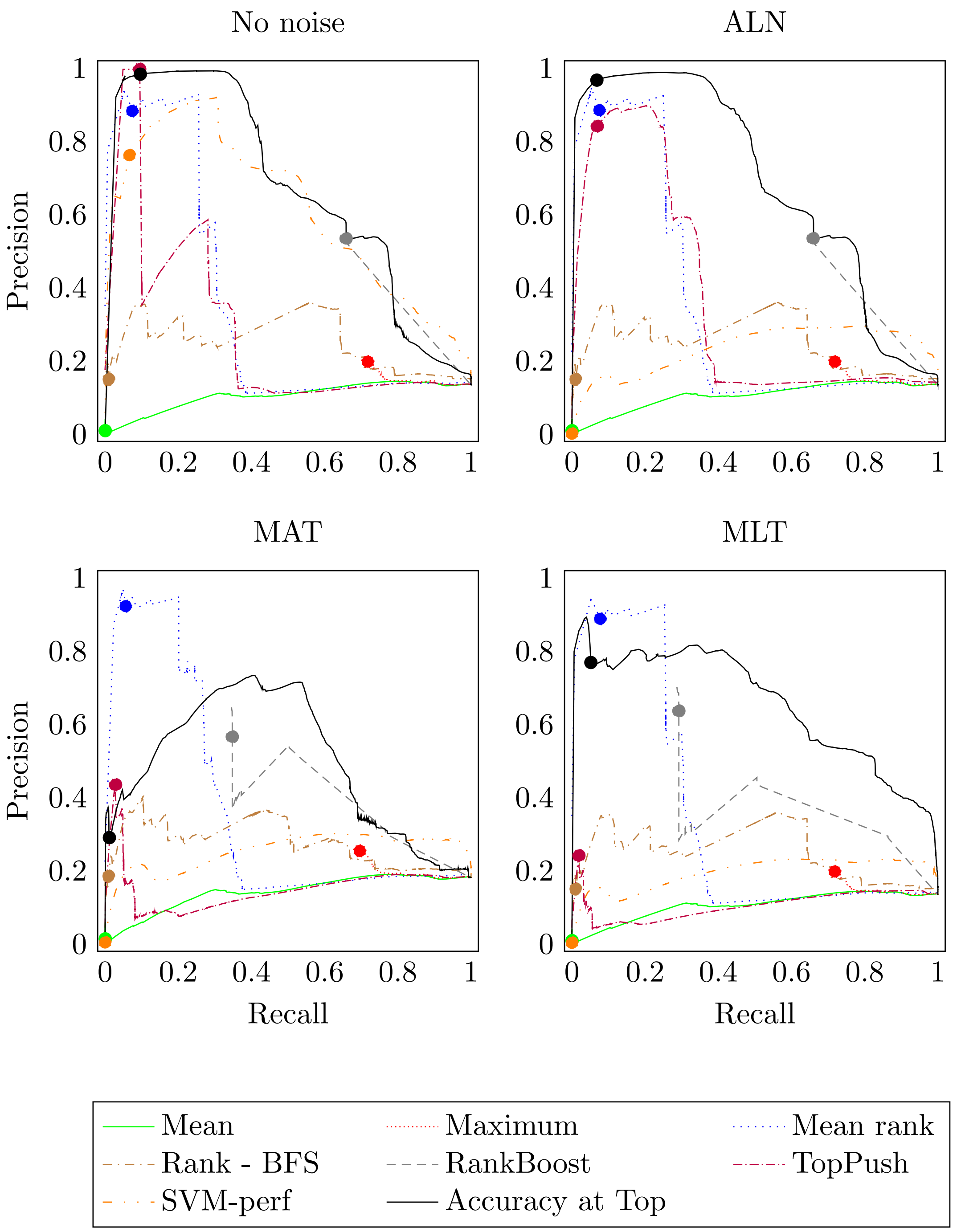

Fig. 5.1: PR curve comparison of algorithms with different types of label noise (described in Section 5.3) using the NetFlow anomaly detectors. Curves represent precision and recall values for all possible thresholds. The threshold corresponding to the 1% quantile is marked on each line with a filled circle.

among unsupervised combination functions and it was surprisingly close to supervised ones at low recall.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Mean	0.9	0.9	1.4	0.9	0.1	0.1	0.1	0.1
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
Maximum	19.8	19.8	25.4	19.8	**71.8**	**71.8**	**69.6**	**71.8**
	(48.8%)	(48.7%)	(49.8%)	(48.8%)	(48.8%)	(48.7%)	(49.8%)	(48.8%)
Mean rank	88.3	88.5	**92.3**	**88.8**	7.4	7.5	5.7	7.8
	(1.4%)	(1.5%)	(1.4%)	(1.5%)	(1.4%)	(1.5%)	(1.4%)	(1.5%)
Rank BFS	15.0	15.0	18.5	15.0	1.0	1.0	1.0	1.0
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
RankBoost	53.4	53.5	56.6	63.7	65.9	65.9	34.8	29.2
	(16.6%)	(16.6%)	(9.8%)	(7.3%)	(16.6%)	(16.6%)	(9.8%)	(7.3%)
TopPush	**99.7**	84.1	43.5	24.2	9.5	6.9	2.9	1.9
	(1.3%)	(1.3%)	(1.4%)	(1.7%)	(1.3%)	(1.3%)	(1.4%)	(1.7%)
SVM-*perf*	76.2	0.1	0.4	0.3	6.7	0.0	0.0	0.0
	(1.2%)	(1.0%)	(1.0%)	(1.0%)	(1.2%)	(1.0%)	(1.0%)	(1.0%)
Acc@Top	98.3	**96.7**	29.1	76.9	9.7	6.8	1.2	5.2
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)

Table 5.1: Comparison of various combination techniques as applied to the NetFlow anomaly detectors described in Section 5.3.2. Each column represents precision or recall in percent for various types of noise defined at the beginning of Section 5.3. Best-scoring algorithm is boldfaced. Small numbers in braces below rates show the fraction of samples returned in top 1% quantile of anomaly scores. Technically this value should be equal to one, but if many samples have the same value, the algorithm returns all of them, which can results to values significantly higher than 1.0%.

Precision and recall in top 1% quantile are shown in Table 5.1. It shows that the presented algorithm has the best or close to the best precision if we compare the supervised combination rules. As discussed above, the unsupervised *mean rank* is better in the presence of severe noise. The low recall of all algorithms except unsupervised *maximum* is caused by the high volume of malicious activities which have amounted up to 10% of the total volume of the traffic. This means that they cannot all fit into the top 1% quantile.

At the first sight RankBoost achieves the best recall of all algorithms, but a closer inspection reveals that it returns 20% of samples as those that belong in top 1%. This highly undesired behavior is caused by assigning the same score to multiple samples. The same phenomenon can be observed in the case of Maximum aggregation function.

5.3.3 Evaluation on HTTP Network Anomaly Detection

Cisco Cognitive Threat Analytics (CTA) [29] engine, introduced in Chapter 4, analyzes HTTP proxy logs (typically produced by proxy servers located on a network perimeter) to detect infected computers within the network. Although the logs do not contain all host traffic (only HTTP(S) requests), the information is richer than the NetFlow as each entry contains the following information extracted from HTTP headers: time of the request, source IP address, destination IP address, url, MIME type, downloaded and uploaded bytes, User-Agent identifier, etc. CTA contains more than 30 different anomaly detectors detecting anomalies according to empirical estimates of (conditional) probabilities such as P(country), P(domain|host), P(User-Agent|second level domain), etc.), time series analyses (models of user activity over time, detection of sudden changes in activity, identification of periodical requests, etc.), and HTTP specific detectors (e.g., discrepancy in HTTP User-Agent field [70]).

Evaluation data was collected from networks of 30 different companies of various sizes and types with collection period ranging from six days to two weeks. The data contains more than seven billion HTTP connections, in which Cisco analysts identified 2 666 infected users with 825 different families of malware. In total the number of HTTP connections created by the malware has reached more than 129 million. Malware connections usually represent less than 2% of the network total traffic, with a notable exception of networks with hosts infected by ZeroAccess malware [175]. ZeroAccess creates many HTTP connections that can easily reach 20% of the volume of network traffic. The other most present malware families were: Cycbot, QBot, SpyEye, BitCoinMiner, and Zbot. Malware connections were identified using multiple approaches starting with an analysis of the most anomalous HTTP flows as reported by the anomaly detection engine, malware reported by the individual network administrators, matching blacklists and other public feeds or third-party software. The rest of the logs remain unlabeled, though we are almost certain there are malware connections that have been missed.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Mean	17.9	18.4	22.0	17.1	7.6	7.8	6.2	7.3
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
Maximum	10.6	10.6	15.8	10.6	**100.0**	**100.0**	**100.0**	**100.0**
	(20.0%)	(19.9%)	(20.9%)	(19.9%)	(20.0%)	(19.9%)	(20.9%)	(19.9%)
Mean rank	0.6	0.5	0.9	0.5	0.2	0.2	0.2	0.2
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
Rank BFS	20.9	21.3	24.7	20.7	9.1	9.3	7.1	9.1
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
RankBoost	81.7	81.6	87.2	81.5	**100.0**	**100.0**	**100.0**	**100.0**
	(2.6%)	(2.6%)	(3.8%)	(2.6%)	(2.6%)	(2.6%)	(3.8%)	(2.6%)
TopPush	**100.0**	93.8	96.7	95.6	42.6	39.9	27.3	40.7
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
SVM-*perf*	2.9	0.0	0.0	0.0	1.3	0.0	0.0	0.0
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
Acc@Top	**100.0**	**99.6**	**99.7**	**97.6**	42.6	42.4	28.2	41.5
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)

Table 5.2: Comparison of various combination techniques as applied to the HTTP anomaly detectors described in Section 5.3.3. Best-scoring algorithm is boldfaced. Small numbers in braces below rates show the fraction of samples returned in top 1% quantile of anomaly scores. Although this value should be equal to one, if many samples share the same value, the algorithm returns all of them, which can results to values significantly higher than 1.0%.

As before, we show PR-curves of all evaluated detector combinations and types of noise in Figure 5.2. We observe that the proposed *Acc@Top* method outperforms all other techniques in all cases of studied noise. Contrary to the above experiments with NetFlow analytic engine, noise does not have significant impact on supervised methods. This indicates that malicious behaviors of different types are similar in the space induced by the CTA HTTP(S) anomaly detectors. This is probably caused by the fact that all labeled malicious behaviors were in some sense connected to malware activity, for which CTA engine is designed. The curve of the *Acc@Top* suggest that even if less or more samples than 1% are requested by the operator, the precision will remain high in all scenarios. In contrast, the curve of the RankBoost method starts at high recall with lower precision suggesting that almost all malicious samples and around 20% legitimate were scored with the same maximal value. Also notice that the *mean rank* unsupervised combination function, dominating in the previous section, was in this experimental scenario superseded by simple *mean*.

Precision and recall at top 1% are shown in Table 5.2, and as in the previous section RankBoost and *maximum* achieve the best recall. The causes are the same. RankBoost and maximum have returned 2.6% and 20% of samples, respectively, which is far away from the required 1%. This is again caused by assigning the same value to many samples. Contrary, the proposed *Acc@Top* meets the 1% requirement with really high precision. Its seemingly low recall is partially caused by the fraction of malicious samples being 2% of the total number of all samples. This means that the best achievable recall while meeting the requirements on returning 1% of the total number of sample is 50%.

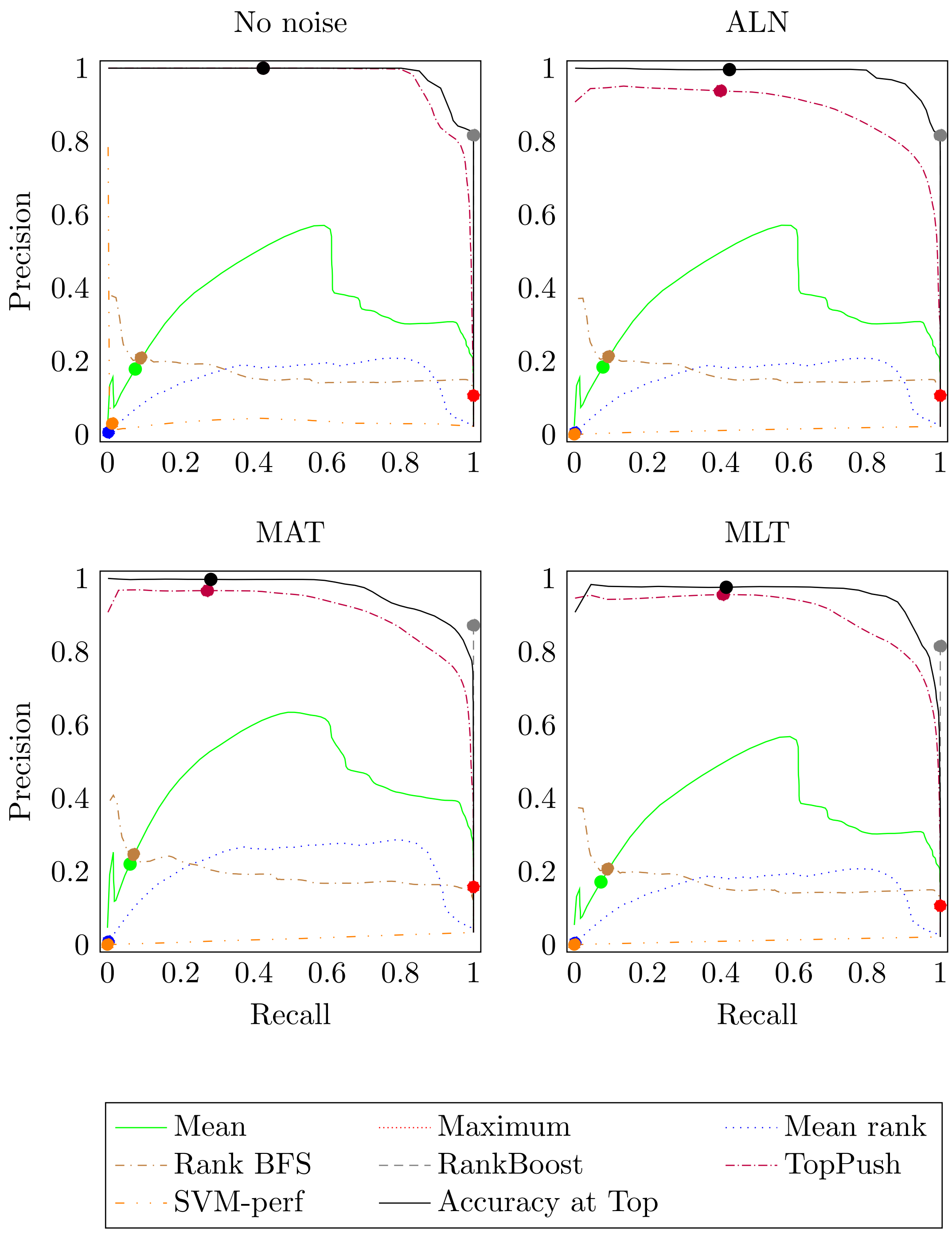

Fig. 5.2: PR curve comparison of various algorithms with various label noise using the HTTP anomaly detectors. Again, threshold corresponding to the 1% quantile is marked on each line with the dot.

5.3.4 Optimizing Only False Positives or False Negatives

To demonstrate the advantage of minimizing both false positive and false negative rates in the objective function $\hat{R}_{\exp}(H_{\boldsymbol{\alpha}})$ (5.6), we have evaluated two additional variants of the objective function with only the false negative part $\hat{R}^{fn}_{\exp}$ (*Acc@Top-FN*) and false positive part $\hat{R}^{fp}_{\exp}$ (*Acc@Top-FP*) using both NetFlow (Table 5.3) and CTA (Table 5.4) anomaly detection systems. As can be seen in the Table 5.3, using only one part of the criteria results in highly decreased efficacy in the NetFlow scenario. Additionally, the false positive variant (*Acc@Top-FP*) results in all the samples having the same, zero value anomaly score. The results on the CTA anomaly detection engine (Table 5.4) are slightly better, but still the proposed *Acc@Top* outperforms both of the variants in all label noise scenarios.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Acc@Top	**98.3**	**96.7**	**29.1**	**76.9**	9.7	6.8	1.2	5.2
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
Acc@Top-FP	13.4	13.4	18.1	13.4	**100**	**100**	**100**	**100**
	(100%)	(100%)	(100%)	(100%)	(100%)	(100%)	(100%)	(100%)
Acc@Top-FN	0.1	0.1	15.7	21.7	0.0	0.0	0.7	1.6
	(1.9%)	(1.9%)	(1.4%)	(2.4%)	(1.9%)	(1.9%)	(1.4%)	(2.4%)

Table 5.3: Comparison of the three variants of the proposed criteria each creating an ensemble for the NetFlow anomaly detection system. Small numbers in braces below rates show the fraction of samples returned in top 1% quantile of anomaly scores. Although this value should be equal to one, if many samples share the same value, the algorithm returns all of them, which can results to values significantly higher than 1.0%.

	Prec@1%				*Rec@1%*			
Method	Non.	ALN	MAT	MLT	Non.	ALN	MAT	MLT
Acc@Top	**100**	**99.6**	**99.7**	**97.6**	**42.6**	**42.4**	28.2	**41.5**
	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
Acc@Top-FP	98.1	1.2	52.7	0.6	41.7	11.2	19.3	25.4
	(1.0%)	(11.7%)	(7.8%)	(26.2%)	(1.0%)	(11.7%)	(7.8%)	(26.2%)
Acc@Top-FN	87.2	87.5	88.5	86.0	37.1	37.2	**29.2**	38.6
	(1.0%)	(1.0%)	(1.2%)	(1.1%)	(1.0%)	(1.0%)	(1.2%)	(1.1%)

Table 5.4: Similarly to above, the table presents a comparison of the three variants of the proposed criteria using the CTA anomaly detection system.

5.4 Chapter Summary

We proposed a new algorithm for finding a convex combination of anomaly detectors maximizing accuracy at τ-quantile of returned samples, which is a scenario frequently appearing in the security field. The algorithm assumes labeled data, which is difficult to obtain and rarely perfect in security domains. Therefore, an emphasis was put on the experimental study, involving two different types of intrusion detection systems, eight types of combination functions, 34 different network captures containing more than 20 million of samples of behavior of different algorithms under different types of noise.

The experimental results showed that the proposed method is more accurate than prior art in finding a good combination of detectors with high accuracy in returned samples. The results also showed that supervised methods can easily overfit if some type of malicious behavior is completely missing in the training data or is incorrectly labeled (mistake of labeling oracle). The severity of the overfitting depends on how much different types of malicious behavior are similar to each other. The comparison of unsupervised combination functions did not have a clear winner, since in one experimental setting *mean rank* was the best while in the second one it was *mean*.

Chapter 6
False Positive Reduction

This chapter focuses on additional reduction of false positives introduced by the anomaly-based intrusion detection systems. Rehák [138] divides the false positives into two classes: *unstructured* false positives are essentially a random noise caused by the stochasticity of the network traffic and *structured* false positives are caused by a persistent but very different behavior of a small number of network hosts, for example mail or DNS servers (the precise definition is left to Section 6.1).

We propose a Local Adaptive Multivariate Smoothing (LAMS) method to decrease the rate of unstructured false positives by smoothing anomaly values with respect to time. This causes similar anomalies[1] occurring at different times to receive similar anomaly score. The rate of structured false positives is either decreased or remains the same, which depends on circumstances discussed in detail in Section 6.2. The method is inspired by the trust models [149, 148, 135, 25] which are specialized knowledge structures designed to maintain information about the trustworthiness of a communication partners, either acquired from interactions, observed from the interactions of others, or received from others by means of reputation mechanism [86]. The design of trust models emphasizes features such as fast learning, robustness in response to false reputation information [81], and robustness with respect to environmental noise. Extended trust models (ETM) introduced by Rehák *et al.* [140] enhance the original trust models with context representations in order to reflect the context of trusting situation, putting it on par with the identity of trusting parties. The paper [139] took the work further and placed the ETMs in the context of network security problem and use them to aggregate the anomaly scores provided by several network anomaly detectors. Each of the detectors had its own ETM model with a feature space defined by identity and detector-specific context, serving mainly as a long term memory of past anomalous events. The proposed LAMS model is an extension of the original ETM model, described in [139]. Unlike its predecessor, it has been simplified by omitting the fuzzy number representation of the anomaly value. Furthermore, we have discarded the identity, focusing only on the context of the observed network events. These simplifications allowed us to re-formulate the problem more concisely as LAMS, and generalize its application in context of classification of a stream of events.

This chapter is organized as follows. The next section properly defines the classes of false positives. Section 6.2 describes the proposed solution and mathematically proofs its correctness under mild assumptions. The same section also discusses the modification to efficiently process data-streams. Section 6.3 shows, how the method was deployed in both CTA and CAMNEP intrusion detection systems, in both cases improving their accuracy.

[1] Similarity can be arbitrary function $k : \mathcal{X} \times \mathcal{X} \mapsto [0, 1]$.

6.1 Classification of False Positives

Hereafter it is assumed that the network anomaly detection system (AD) observes a stream of network flows (e.g., NetFlow [28], HTTP connections, etc.) produced by a set of hosts within the network. Anomaly detection system maintains internal model(s) to assign a score in range $[0, 1]$ to each observed flow with zero indicating the normal flow and one indicating possible attack, as it is assumed that malicious activities have statistical characteristics different from the normal ones [38, 100, 45] making them rare. As mentioned in the introduction the anomaly detection systems produce false alarms since an overwhelming majority of rare flows are not caused by any attack. Rehak [138] divides these false positives into following two classes:

- *Unstructured false positives* are short-term anomalies distributed uniformly over all the network hosts proportionally to the traffic volume. They are typically triggered by widespread, uniformly distributed behaviors (such as web browsing) and we model them as white noise (zero mean and finite variance) added to the anomaly detector's output. Therefore, the observed anomaly score y_i of an flow x_i is equal to

$$y_i = g(x_i) + \eta_i, \tag{6.1}$$

  where $g(x_i)$ is the true anomaly score on flow x_i and η_i is the additive white noise. The η_i therefore hides the true value $g(x_i)$.
- *Structured false positives* are caused by a (long-term) legitimate behaviors of a small number of network hosts. These behaviors are different from the background, and because they are found only at a very small portion of network hosts, they are reported as anomalies. Examples are rare applications performing software update, regular calls of unusual network APIs, etc. Since this type of false positive is typically limited to a small number of network hosts and its behavior is very regular, it can be quickly identified and filtered out using white-lists. Nevertheless, these white-lists are specific for a given network and are difficult to create before deployment. We define the structured false positives to be generated by a mixture of distributions

$$x_{sfp} \sim \frac{\sum_{j=1}^{m} \beta_j \epsilon_j(x_i)}{\sum_{j=1}^{m} \beta_j}, \tag{6.2}$$

  where ϵ_j represent structured false positive with weight β_j. Each component ϵ_j has small variance comparing to that of the unstructured false positives, but means of the components are typically far from each other.

6.2 Proposed Method

The idea behind the proposed local adaptive multivariate smoothing (LAMS) method is to replace the output of the anomaly detector by average anomaly score of similar flows observed in the past, where the similarity between two flows is defined as $K_h : \mathcal{X} \times \mathcal{X} \mapsto [0, 1]$ (further also called *context*). This effectively smooths the output of the anomaly detector and therefore reduces unstructured false positives. This smoothing can be mathematically formulated as

$$\hat{g}_{\mathrm{nw}}(x) = \frac{\sum_{i=1}^{n} K_h(x, x_i) y_i}{\sum_{i=1}^{n} K_h(x, x_i)}, \tag{6.3}$$

where $\{x_i\}_{i=1}^{n}$ is the set of observed network flows, $\hat{g}_{\mathrm{nw}}(x)$ is the expected anomaly of flow x, and $\{y_i\}_{i=1}^{n}$ is the set of corresponding AD outputs. The space $\mathcal{X}$ on which it is defined can be arbitrary (e.g., space of all strings, graphs, etc.), but Gaussian kernel $K_h(x, x') = \exp\left(-\frac{\|x-x'\|^2}{h}\right)$ on Euclidean space is the most common combination (h parameterizes the width of the kernel).

Estimator (6.3) is known as a Nadaraya-Watson estimator [114, 173] which is a non-parametric estimator of the conditional expectation of a random variable.

How does the above smoothing remove false positives according to the division introduced in Section 6.1?

- *Unstructured false positives* are reduced by the Nadaraya-Watson estimator that performs local averaging of the flows. Since the unstructured false positives are of the form $y_i = g(x_i) + \eta_i$, it is proven by Devroye *et al.* [39] that it converges to the true value $g(x_i)$ under fairly general assumptions. More formally, Devroye have shown

$$E|\hat{g}_{\mathrm{nw}}(x) - g(x)| \leq \frac{c}{\sqrt{nh}}, \tag{6.4}$$

  assuming h is chosen in terms of n, with $h \to 0$, $nh \to \infty$ as $n \to \infty$; $c > 0$ is a constant; the kernel K is absolutely continuous and differentiable, with $K' \in L^1$; g is differentiable and the $\mathrm{Var}(y_i)$ is bounded. Since we are using a Gaussian kernel and the anomaly scores y_i are bounded to $[0, 1]$ the estimate converges to the underlying true anomaly score $g(x)$ with a rate of $(nh)^{-1/2}$, when only unstructured false positives are present. However, setting the $h \to 0$ requires infinite computation and memory resources, therefore we restrict the estimator to fixed h.

  The smoothing effect is shown in the Figure 6.1 where the left figure shows the input to LAMS and right figure the output. It is apparent that the left figure is noisier, as points with different colors are close to each other.

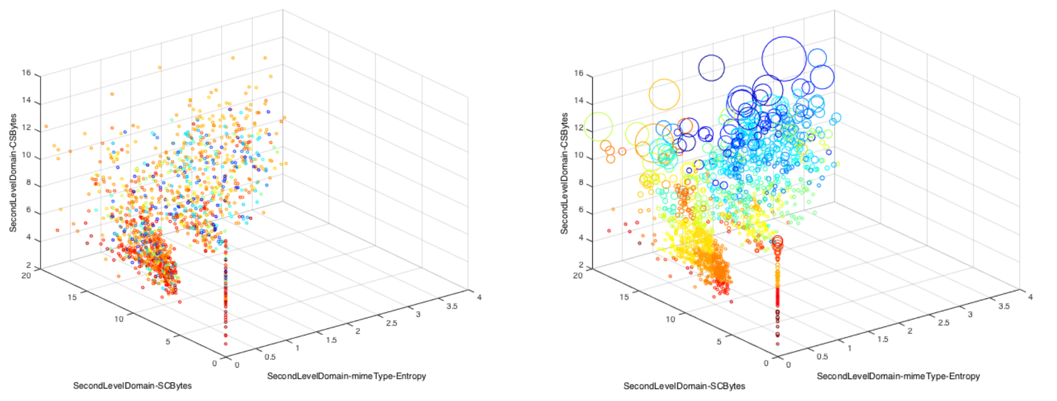

Fig. 6.1: Visualization of score provided by HTTP anomaly detection engine (described in Section 6.3.2) without (left) and with (right) LAMS models in LAMS 3 context space (see Table 6.4 for details). Each point represents one HTTP request observed in the network, where its position is defined by the context features (defined in Table 6.4) and the color denotes the anomaly score, red being the most anomalous and blue the less. The larger the circle the more flows share the same context position and anomaly score. The anomaly scores of the anomaly detection engine are shown in the left figure. As can be seen there are regions of the context space where most of the points have high anomaly scores and regions of low anomaly scores with additional noise. The right figure shows the same points but the anomaly score is smoothed by the LAMS model effectively reducing the unstructured false positives via smoothing.

- *Structured false positives* are long-term flows confined to subset of network hosts without direct relationship to the background in the sense that AD's output on them does not change with the time. LAMS can remove them in following situations.

  If LAMS' similarity measure of alerts K_h is different from that used in the anomaly detector, large number of flows with normal AD score can be similar to structured false positives, which

decreases the scores output by LAMS on false positives. Example of this type of behavior can be seen on the Figure 6.2, where we see that the amplitude of LAMS input is higher than that of the output.
If LAMS aggregates outputs of the same anomaly detector deployed on many networks, structured false positives can be normal in other networks. Then, LAMS aggregation eliminates the false positives by aggregating them with these normal activities.
In other situations, structured false positives are confined and there are no similar flows receiving lower anomaly score. In this case LAMS fails to remove them, but does not increase their number.

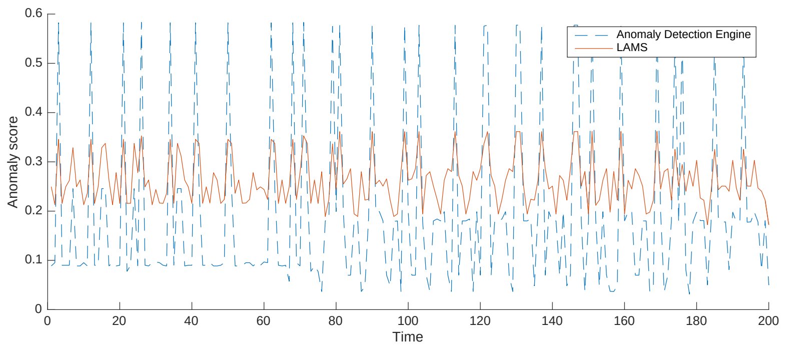

Fig. 6.2: Anomaly score of the HTTP request to google web API in time. There are sudden spikes in AD's anomaly score (blue dashed line) that are reduced by the LAMS model (red solid line).

6.2.1 Complexity Considerations

The complexity of the smoothing of AD's output described by Equation (6.3) is linear with respect to the number of observed flows. This, together with the fact that all observed flows need to be stored, makes LAMS useless for practical deployment, since number of network flows can be as high as millions per second. We therefore resort to common approach to approximate (6.3) from values maintained in a set of pivots $\varPhi = \{\phi_j\}_{j=1}^J$, $\phi_j \in \mathcal{X}$ as

$$\hat{g}_{\text{lams}}(x) = \frac{\sum_{j=1}^J K_h(x,\phi_j)\hat{y}_j}{\sum_{j=1}^J K_h(x,\phi_j)}, \tag{6.5}$$

where $\{\hat{y}_j\}_{j=1}^J$ are estimates of $\hat{g}_{\text{nw}}(\phi_j)$ calculated according to (6.3). This reduces the computational complexity of the estimate in arbitrary x to $O(J)$, which is linear with the number of pivots and independent of the number of observed alerts[2]. The same holds for space complexity, because only positions ϕ_j of finite number of pivots and relevant estimates $\hat{y}_k$ needs to be kept. The set of pivots $\varPhi$ is updated using the modified Leader-Follower algorithm [42], where new pivot is added to the set $\varPhi$ when a flow not similar to any pivot in $\varPhi$ is received. The update process is described in more detail in Section 6.2.2.Contrary to the standard definition of Leader-Follower [42, 131] pivots are not allowed to move, because moving pivots towards areas of higher density causes forgetting of rare flows, which we want to remember.

[2] This statement holds only partially, since with increasing number of observed samples the number of pivots will increase as well [137].

6.2.2 Incremental Update of the LAMS Model

Keeping LAMS model up to date on data-streams requires maintaining estimates $\{\hat{y}_j\}_{j=1}^J$ in $\{\phi_j\}_{j=1}^J$ and alternatively adding new pivot if needed. Upon arrival of a new observation of a network flow (y_t, x_t), estimates $\hat{y}_j$ stored in pivots ϕ_j should be updated using the Equation (6.3) as

$$\hat{y}_j^{\text{new}} = \frac{\sum_{i=1}^n K_h(\phi_j, x_i) y_i + K_h(\phi_j, x_t) y_t}{\sum_{i=1}^n K_h(\phi_j, x_i) + K_h(\phi_j, x_t)}. \tag{6.6}$$

If we denote

$$w_j^{\text{old}} = \sum_{i=1}^n K_h(\phi_j, x_i), \tag{6.7}$$

$$\hat{y}_j^{\text{old}} = \frac{\sum_{i=1}^n K_h(\phi_j, x_i) y_i}{\sum_{i=1}^n K_h(\phi_j, x_i)}, \tag{6.8}$$

the Equation (6.6) can be rewritten as

$$\hat{y}_j^{\text{new}} = \frac{w_j^{\text{old}} \hat{y}_j^{\text{old}} + K_h(x_t, \phi_j) y_t}{w_j^{\text{old}} + K_h(x_t, \phi_j) y_t}. \tag{6.9}$$

Therefore, for efficient incremental update of the model only $\hat{y}_j$ and w_j need to be stored in each ϕ_j. The update can than be done using the following.

$$w_j^{\text{new}} = w_j^{\text{old}} + K_h(x_t, \phi_j) y_t, \tag{6.10}$$

$$\hat{y}_j^{\text{new}} = \frac{1}{w_j^{\text{new}}} \left[w_j^{\text{old}} \hat{y}_j^{\text{old}} + K_h(x_t, \phi_j) y_t \right], \tag{6.11}$$

The above can be further simplified to update only pivots close to x_t, which we define as pivots with similarity $K_h(x_t, \phi_j)$ greater than $K_h^{\min}$. This reduces the complexity of the update process, because only limited number of nearest pivots needs to be updated with each new flow.

The update is outlined in Algorithm 2. First, a set of all pivots in $\varepsilon(x_i)$ in the $K_h(x_t, \phi_j)$ vicinity of a new flow (y_t, x_t) is found and their estimates $\hat{y}$ are updated as defined in Equation (6.11). The γ parameter controls the distance (threshold on the similarity) upon which a new pivot is created with $\phi_{J+1} = x_t$, $\hat{y}_{J+1} = y_t$ and $w_{J+1} = 1$.

```
Data: Stream of flows $(y_i, x_i), i = 1 \ldots$
Result: Set of pivots $\Phi = \{\phi_j = (x_j, \hat{y}_j)\}_{j=1}^J$
Start with an empty set $\Phi = \varnothing$;
while there is a new flow $(y_i, x_i)$ do
    $\varepsilon(x_i) = \{\phi_k \in \Phi \mid K_h(x_i, \phi_k) > K_h^{\min}\}$;
    for $\phi_k \in \varepsilon(x_i)$ do
        Update $\hat{y}_k$ relevant to pivot $\phi_k$ using Equation (6.10) and (6.11);
    end
    Find the most similar pivot $\phi_N$ to the $x_i$: $\phi_N := \arg\max_{\phi \in \Phi} K_h(x_i, \phi)$;
    if $K_h(x_i, \phi_N) < \gamma$ then
        Create new pivot $\phi_{J+1} := x_i$;
        $w_{J+1} := 1$;
        $\hat{y}_{J+1} := y_i$;
        $\Phi = \Phi \cup \{\phi_{J+1}\}$;
    end
end
```

Algorithm 2: LAMS model update algorithm.

The algorithm is controlled by several parameters allowing to trade high locality (precision) for generalization and efficiency. The most important parameter is γ, which influences the total number of pivots in LAMS and hence the cost of the algorithm. When this parameter is set to $\max_{x,x' \in \mathcal{X}} K_h(x, x')$, the number of pivots is identical to the number of unique flows. Departing from this extreme case the number of pivots decreases with decreasing its value with a possible impact (both positive or negative) on the accuracy of estimates if local variations are strong. An example of its effect is shown in Figure 6.3. The minimal update weight K_h^{min} is not overly important and it is there more for computational speed-up, as sample's neighborhood are more controlled by the similarity function $K_h(x, x')$.

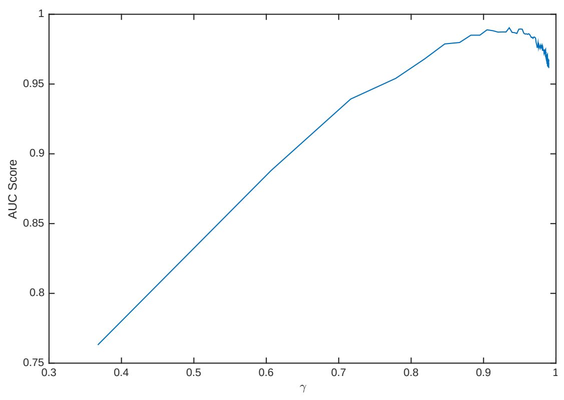

Fig. 6.3: The graph shows effects of γ parameter on the average AUC score (see Section 6.3 for definition). Too large γ reduces effect of the smoothing causing smaller efficacy. On the other hand, too small γ decreases model's efficacy, since the model cannot capture local variations in the flows.

6.2.3 Querying the LAMS Model

The query of the model follows the Equation (6.5) with the difference that the estimate is computed only over similar pivots, controlled by the same parameter γ as used in the update. Formally the estimate $\hat{y}_t$ is calculated as

$$\hat{y}_t = \frac{\sum_{\phi_j \in \varepsilon(x_t)} K_h(x_t, \phi_j) y_j}{\sum_{\phi_j \in \varepsilon(x_t)} K_h(x_t, \phi_j)}, \tag{6.12}$$

where $\varepsilon(x_t) = \{\phi_j \in \Phi \,|\, K_h(x_i, \phi_k) > K_h^{\min}\}$. Restricting the estimate to be calculated from the neighborhood $\varepsilon(x_t)$ is mainly to be consistent with the update procedure. In practice, $K_h^{\min}$ is so small that the effect of points outside $\varepsilon(x_t)$ on the points would be negligible.

When LAMS is deployed, it always performs an update before the query, which means that there is always at least one pivot from which the estimate can be calculated.

6.2.4 LAMS Model Feature Selection

As in most machine learning algorithms the biggest influence on LAMS's accuracy has the selection of similarity measure and features defining it. The number of features that can be extracted from network traffic flows can be high and defining the context with respect to all of them would lead to poor generalization — a phenomenon known as curse of dimensionality [55]. The usual approach is to reduce dimension by feature extraction algorithms [51] or by feature selection. In experiment described in the next section we have chosen the latter approach.

The features should be selected such that malicious network flows are confined in a rather limited part of the space covered by only few pivots. This guarantees that such model will improve the accuracy, since if malicious flows fall into a specific region of the feature space the long term anomaly estimate in this region will be consistently high. To select the features in practice, one either has to have examples of network attacks and use feature selection algorithm [71], or use a domain knowledge provided by an expert.

6.3 Experimental Evaluation

We have evaluated the effectiveness of on both CAMNEP and CTA detection systems. Both use a ensemble of simple yet diverse anomaly detectors allowing to efficiently process large data-streams while providing good detection accuracy.

The accuracy is measured by the area under Receiver Operating Characteristic (AUC) [49], which is frequently used when the operating point of the detector is not known.[3] AUC is equal to the probability that the anomaly score of a randomly chosen malicious sample is higher than a randomly chosen legitimate one. The AUC score one means that the detector is always correct, zero means it is always wrong, and 0.5 means that its accuracy is equal to random guessing.

6.3.1 NetFlow Anomaly Detection

For the purpose of the LAMS model evaluation we use only the Evangelista aggregated output of the 16 anomaly detectors of the CAMNEP [141, 59] anomaly detection engine described in Chapter 3. The combined output is then used as an input in five different LAMS models with different contexts. The rationale behind using more than one LAMS model is again the ensemble principle, as the expectation is that errors of outputs of different LAMS models will be uncorrelated and they will cancel out. Contexts (features) of individual models are shown in Table 6.1 and they are named according to the publications which have used them in anomaly detection. Since all features are real numbers, the similarity is defined using the Gaussian kernel introduced in Section 6.2.

6.3.1.1 Dataset of NetFlow

To evaluate the effects of LAMS model on the NetFlow Anomaly detection engine we have created several datasets from a traffic captured on the network of Czech Technical University (CTU) in Prague. We have used three different approaches to create labels: manual labeling, infecting virtual machines, and attacking our computers within the network by us.

Manual Labeling

A week long capture from the university contains 41 517 828 flows between 19 261 different IP addresses. An experienced network operator has identified 10% as anomalous and 11% as legitimate. The most prevalent malicious traffic was ssh scan (55%) followed by p2p traffic (36%), horizontal scan (6.8%), and vertical scan (1.5%).

[3] By an operating point it is understood an upper bound on false positives / false negatives or their costs.

Name	Context features
Xu-source [177]	entropy of source ports for the source IP entropy of destination ports for the source IP number of destination IP addresses for the source IP
Xu-destination [177]	entropy of source ports for the source IP entropy of destination ports for the source IP number of source IP addresses for the source IP
MINDS [45]	number of flows originating from the source IP number of flows originating from the destination IP number of different destination ports for the source IP
Lakhina [95]	number of flows originating from the source IP number of packets transferred from the source IP number of bytes transferred from the source IP
TAPS [160]	number of different destination IPs of the source IP number of unique ports of the source IP entropy of packets size of the source IP

Table 6.1: Features defining similarity in NetFlow LAMS models.

Malware Infection

The second dataset was created by executing real malware inside controlled virtual machine (VM)[4][59]. VMs were infected using malicious binaries captured during 24 hours of their traffic together with the traffic of the whole CTU network. The labels were created such that every NetFlow originating from the infected machines was labeled as malicious, divided into requests and responses according to the direction. NetFlows corresponding to normal traffic of Windows XP operating system running on the VMs, such as connection check, Windows Update checks, NTP checks, etc., were also labeled as normal.

In the first capture we have infected only one VM running Windows XP with Neeris [59] malware that has generated a lot of traffic (malware generated on average three Netflows per second). Second, we have infected ten different VMs by FastFlux botnet [59]. Finally, we have infected ten VMs by rbot we could control, and instruct it to perform ICMP flood attack against one of our servers.

Manual Attacks

Third dataset was created by network specialist attacking one computer with an open SSH port inside the university network. She has started her attack by horizontal scan of university network searching for a computer with an open SSH port. Then, she performed brute force cracking of an ssh password with a help of dictionary of 1000 passwords with the last one being correct. She has tried all passwords within 5-minute long window. Finally, she has downloaded 0.5 GB from the attacked computer to simulate stealing the data. Corresponding NetFlows in the dataset were labeled on the basis of known attacker and attacked IP addresses and ports and time of the attack information.

[4] The bandwidth of the connection was lowered to 150kb/s to prevent generating huge amount of traffic

6.3.1.2 Experimental Results

Table 6.2 shows AUC scores of the anomaly detection engine after the first aggregation before using multiple LAMS models (denoted without LAMS), and after the second aggregation (denoted with LAMS) on individual datasets and attacks described previously. The results reveals that LAMS models improves the accuracy in detecting attack requests, but decreases the accuracy of responses. To investigate this, we have plot in Figure 6.4 distribution of points corresponding to flows related to the horizontal scan in the context defined by Lakhina's entropy based features. The figure confirms the assumption that requests are located in a very small and distinct part of the space and therefore LAMS models can well estimate the anomaly score, whereas responses are scattered all over the space mixed with other mostly legitimate traffic. According to Table 6.1 Lakhina's entropy features are calculated from a set of flows with the same source IP address receiving the same entropy and also anomaly values. During horizontal scan, each attacked network host will probably have only one response flow to the attack, which will be well hidden among host's own legitimate traffic. This causes (a) the responses to have different entropies and locations in the space and (b) receiving low anomaly values. Contrary, requests of the attack stands out, since they have the same source IP address and the same values of entropies. Moreover, this entropy would be very different from that of other hosts in the network leading to high anomaly. This explanation is supported by the fact that LAMS models do not decrease anomaly values of response to attacks targeting a single host (e.g., vertical scan). Finally note that it is important to detect at least one part of the attack, either the attack, or the responds.

	AUC	
Manual labeling	Without LAMS	With LAMS
horizontal scan request	0.82	**0.83**
horizontal scan response	**0.77**	0.66
p2p request	0.87	**0.91**
p2p response	**0.89**	0.75
scan sql	**1**	**1**
scan sql response	**0.92**	0.91
ssh cracking request	0.80	**0.83**
ssh cracking response	**0.84**	0.71
vertical scan	**1**	**1**
vertical scan response	**0.98**	0.96
Malware infection		
Neris bot	0.69	**0.87**
FastFlux botnet	0.81	**0.88**
RBot ICMP flood	0.91	**0.94**
Artificial Attack		
SSH scan	0.93	**1**
SSH scan response	**0.84**	0.77
SSH cracking	**0.83**	**0.83**
SSH cracking response	0.77	**0.78**
Anomalous SSH download	0.95	**0.97**
Anomalous SSH download request	**0.95**	**0.95**

Table 6.2: Comparison of AUC Scores without and with LAMS model.

To demonstrate the decrease in false positive rate on structured false positives, we have used the dataset described in Section 6.3.1.1 and analyzed the traffic identified as the most anomalous traffic by the first part of the detection system (combined output of the anomaly detectors without LAMS models). Within we have identified a legitimate traffic meeting the criteria of

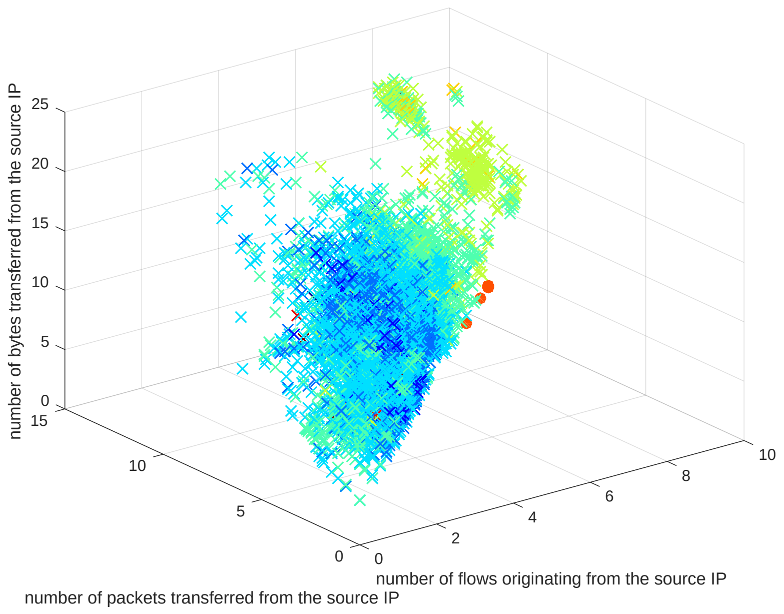

Fig. 6.4: Visualization of the anomaly scores assigned by the Lakhina LAMS model (see Table 6.1) to the horizontal scan and corresponding responses contained in the manually labeled dataset described in Section 6.3.1.1. Each point represent one scan request (cross) or scan response (dot). The color corresponds to obtained anomaly score with red being the most anomalous and blue being the least. Visualization of all the other LAMS models defined in Table 6.1 can be see in the Appendix in Figure A.5.

the structured false positives (being explainable and corresponding to rare flows). The complete list is in Table 6.3 and includes responses of NTP servers, software licenses servers, downloads from local data and database servers, etc. The decrease has been again measured by the AUC score, where the false positives were treated as negative samples and all attacks identified in the dataset 6.3.1.1 as positive samples. AUCs of identified structured false positives are shown in Table 6.3. According to it LAMS models almost always reduces the false positive rate and increases the accuracy of detection.

The above investigation of false positives also revealed some unstructured false positives corresponding to excessive web and DNS traffic of some users, which anomaly detectors flagged as suspicious. LAMS models have decreased their anomaly values as is shown in the second part of the Table 6.3.

6.3.2 HTTP Anomaly Detection

Similarly to the CAMNEP system, we use 30 anomaly detectors of the Cisco Cognitive Threat Analytics (CTA) [29] described in detail in Chapter 4, whose output is combined by Evangelista's fusion function to obtain one anomaly score value for each HTTP flow. This output is used as an input to the second block with several LAMS models with context features listed in Table 6.4 and Gaussian similarity function defined in Section 6.2. Outputs of four LAMS models are again combined by Evangelista's fusion function to the final output.

Structured false positive	AUC without LAMS	AUC with LAMS
NTP server response	0.98	**0.99**
ICQ servers	0.98	**0.99**
Google crawler	0.99	**1**
Local data server downloads	0.9	**0.93**
Local database server requests	**0.98**	**0.98**
Local database server responses	0.96	**0.98**
Local software license server	0.19	**0.89**
Subnetwork gateway requests	0.83	**0.90**
Subnetwork gateway responses	**0.30**	0.24
HTTP proxy requests	0.75	**0.77**
HTTP proxy responses	**0.76**	0.73

Unstructured false positive	AUC without LAMS	AUC with LAMS
DNS request	0.63	**0.70**
DNS response	0.45	**0.71**
Web browsing user 1	0.99	**1**
Web browsing user 2	**0.99**	**0.99**
Web browsing user 3	0.92	**0.94**
Gmail.com servers	0.98	**0.99**

Table 6.3: AUC Scores of structured and unstructured false positives in the dataset introduced in Subsection 6.3.1.1 when LAMS models are used (right column) and not used (middle column). Higher AUC is better.

Name	Context features
LAMS 1	number of unique referrers of the visited domain
	number of unique operating systems used to visit domain
	number of unique MIME types hosted on the domain
LAMS 2	number of unique referrers of the visited domain
	number of unique HTTP status responses of the visited domain
	amount of bytes downloaded using the User-Agent
LAMS 3	number of unique User-Agents used to visit the domain
	entropy of the MIME types hosted on the domain
	number of unique referrers of the visited domain
LAMS 4	URL length
	number of unique
	number of unique operating systems used to visit domain
	number of unique referrers of the visited domain

Table 6.4: Network communication features used by the HTTP LAMS models.

6.3.2.1 Dataset of HTTP Proxy Logs

The database of HTTP flows was collected from networks of 30 different companies of various sizes and types with collection period ranging from six days to two weeks. There are more than seven billion HTTP flows in the database. A small team of security researchers has identified in them 2 666 users infected with a malware of 825 various families that have generated in total more than 129 millions HTTP flows. HTTP flows corresponding to malware requests represent less than 2% of the total traffic of the individual networks except for few cases, where the networks were infected by ZeroAccess malware [175]. ZeroAccess is very noisy, generating many HTTP flows which has reached up to 21% of the total number of flows. The other malware families worth to mention were: Cycbot, QBot, SpyEye, BitCoinMiner, Zbot and many others. All the malware was identified using several approaches starting with an analysis of the most anomalous HTTP flows as reported by the CTA, malware reported by the individual network administrators, using blacklists and other public feeds or third-party software. The rest of the unlabeled flows were considered to be legitimate traffic. We are aware that within them there might be another proxy flows corresponding to malware traffic, but we are not aware about any additional method providing the ground truth. Moreover since the most anomalous traffic was always investigated for a malware HTTP flows, the most important part corresponding to low false positive rate was always labeled. This labeling of the most anomalous part has been done for the detection engine using only anomaly detection part and using both parts including LAMS models.

6.3.3 Experimental Results

To demonstrate the advantage of LAMS models we have calculated AUC score on sub-datasets with HTTP flows of each malware family considered as positives and all unlabeled HTTP flows as negatives. Since there are 825 different malware families in the dataset the table showing the AUC improvement would be huge. Therefore we have decided to show the effect of the LAMS model in a figure. The Figure 6.5 shows a scatter plot where x-coordinate of each point is AUC score of output of the first part of the detection engine using only anomaly detectors and the y-coordinate is AUC score of the output of the second part with LAMS models. This means that if the point is above the diagonal, LAMS models improve the detection accuracy and vice versa. We can observe that LAMS models generally improve the accuracy of detection since majority (75%) of the points is above the diagonal. Particularly noticeable are points within the green rectangle that would not be detectable without LAMS models, but LAMS models significantly increase their anomaly values.

Figures 6.6 and 6.7 show the distribution of anomaly scores of the anomaly detection engine without (left) and with (right) LAMS models. We can see that outputs of an anomaly detection engine without LAMS models on malware HTTP flows are more equally spread across the entire range of values. Contrary, when the anomaly detection engine uses LAMS models, the scores of malware's HTTP flows are located in the most anomalous part of the distribution.

6.4 Chapter Summary

We presented a method designed to reduce false alarm rate of anomaly detection-based intrusion detection systems. The technique smooths detectors' output simultaneously over time and space, which improves the estimate of true anomaly score.

We proved under mild assumptions that the method reduces unstructured false positives caused by stochasticity of the network traffic. The method was evaluated using a large number

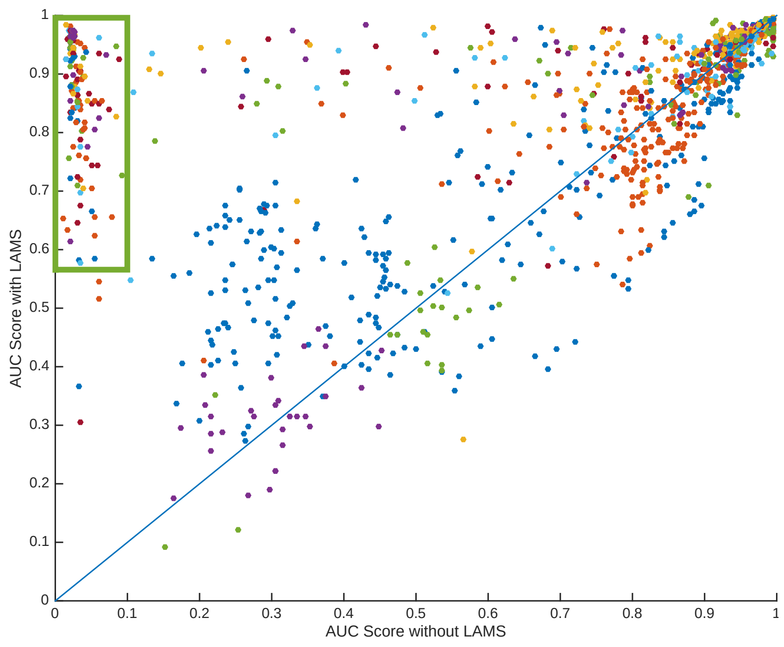

Fig. 6.5: Scatter plot of the AUC scores when running the anomaly detection engine without and with the LAMS model. Each point represent one malware sample and different colors denote different malware families to illustrate variety of the samples. Points that are above the blue line represent an AUC improvement when LAMS is used, whereas point below represent cases where LAMS decreases the efficacy. As can be seen the LAMS model significantly improves the AUC score of the samples with AUC smaller than 0.2, as depicted by the green bar in the figure.

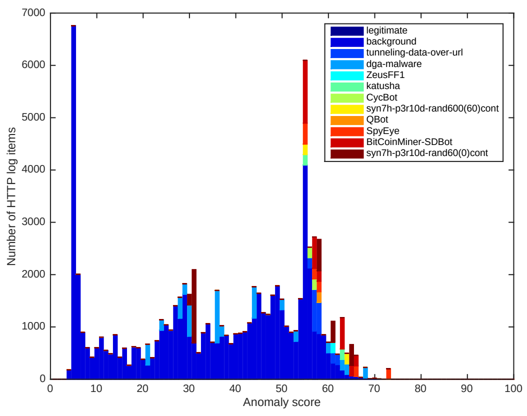

Fig. 6.6: Distributions of anomaly values of HTTP proxy flows without LAMS models.

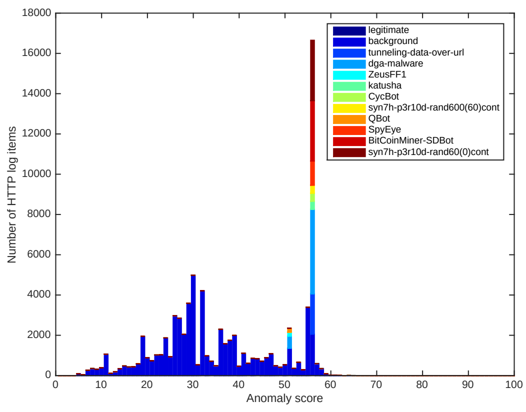

Fig. 6.7: Distributions of anomaly values of HTTP proxy flows with LAMS models.

of samples from two domains with diverse sets of anomaly detectors. The experiments showed reduction of the unstructured false positives, while not having major negative effect in the remaining cases.

Chapter 7
Experimental Evaluation

This chapter experimentally evaluates all the layers of the proposed architecture of the anomaly-based network intrusion detection systems as proposed in Section 1.2. We use both CAMNEP and CTA detection engines to evaluate the efficacy improvements achieved in each of the proposed layers.

The evaluation data was acquired on a number of diverse real enterprise networks of various types and sizes to illustrate the efficacy of the system that can be achieved in practice.

The accuracy is measured by the area under Receiver Operating Characteristic (AUC) [49], which is frequently used when the operating point of the detector is not known in advance. The accuracy at top measures, used in Chapter 5, were discarded because of the poor performance of the individual anomaly detectors whose performance is not measurable using these high precision measures.

7.1 CTA

The Cognitive Threat Analytics (CTA) anomaly detection system uses 32 various HTTP-based network anomaly detectors in the first layer as described in Chapter 4. The second layer contains the Evangelista ensemble and the Acc@Top supervised ensemble created using the method described in Chapter 5. Next, the anomaly scores of both the ensembles are adjusted using four LAMS models as described in Chapter 6. The final anomaly score of each of the HTTP flow is an aggregate of the outputs of the individual LAMS models as described in Section 1.2.

7.1.1 HTTP Proxy Logs Dataset

The evaluation data contains HTTP flows collected from 44 different enterprise networks of various sizes ranging from 3,000 up to 60,000 users. In all the enterprises the data was collected during 10 days resulting in dataset that contains more than six billion flows in total. This evaluation dataset is different from the one described in Section 6.3.2.1 and Section 5.3.3, that was collected during the years 2013 and 2014. The final evaluation uses data collected in 2016 in different enterprise networks.

Similarly to the dataset described in Section 6.3.2.1, a small team of network security analysts was able to identify 19,218 users featuring some type of malicious behavior. The analysts used number of publicly available blacklists, malware reports and blogs, third party-party software or performed their own deep analysis to confirm the assigned labels that can be categorized into three groups: malware, adware and PUA. Users infected with malware represent 12% of all the labels including ransomware[1], banking trojans, data stealers, worms, etc.. Next, 67% of the labels represent various families of adware[2]. Finally, 21% of the labels were related to a Potentially Unwanted Applications (PUA). PUAs are applications that, while not necessarily malicious, are generally considered unwanted in business networks as they may install adware, toolbars, or contain other unsafe program features with unclear objectives. The rest of the unlabeled HTTP flows were considered to be legitimate traffic. We are aware that within them there might be more malicious HTTP flows, but we do not know about any additional method providing the ground truth.

7.1.2 Experimental Results

To observe the effect of the learned combination function, we have evaluated two variants of the CTA intrusion detection system denoted as Evangelista only and Evangelista & Acc@Top. The first uses the Evangelista aggregated ensemble only, completely discarding the supervised ensemble introduced in Chapter 5, whereas the second uses all the components of the architecture as described in Section 1.2.

The supervised Acc@Top ensemble was constructed as described in Chapter 5 using the training data as specified in Section 5.3.3. Therefore, the training and testing data contain HTTP flows collected in different networks and time period with different labels. The telemetry collection time difference between the training and testing data is more than two years allowing us to evaluate the robustness of the system against shifts in the threat landscape.

Table 7.1 shows the mean and standard deviation of the AUC scores calculated over all malicious samples and members of each layer for both the above described variants of the proposed architecture. As can be seen the mean of the AUC score is gradually increasing with decreasing standard deviation in each consecutive layer of both the architecture variants. However, the variant that contained the trained ensemble performed much better, according to generally higher AUC scores with lower deviations.

	AUC			
Evangelista only	Anomaly	Ensemble	LAMS	Final
Malware	0.74 (±0.28)	0.92 (±0.17)	**0.95** (±0.11)	**0.95** (±0.10)
Adware	0.70 (±0.26)	0.85 (±0.22)	0.90 (±0.13)	**0.91** (±0.12)
PUA	0.70 (±0.27)	0.86 (±0.22)	0.91 (±0.13)	**0.92** (±0.12)
Evangelista & Acc@Top				
Malware	0.74 (±0.28)	0.93 (±0.19)	0.96 (±0.08)	**0.98** (±0.05)
Adware	0.70 (±0.26)	0.90 (±0.14)	0.91 (±0.10)	**0.92** (±0.09)
PUA	0.70 (±0.27)	0.90 (±0.15)	0.92 (±0.10)	**0.93** (±0.09)

Table 7.1: AUC improvements introduced by the individual layers of two variants of the architecture of the CTA intrusion detection system. The presented AUC score is a mean of the AUCs of all malicious samples and the members of the individual layers. The number in brackets under the mean AUC score is the standard deviation of the individual AUCs. The highest AUC score in each row is boldfaced. The Evangelista only variant is missing the parallel Acc@Top ensemble created using the supervised method introduced in Chapter 5 to evaluate its effect on the overall efficacy of the detection system.

[1] Malware that covertly encrypts user's private data and demands a ransom payment to restore it.

[2] Malicious applications that generate revenue for its author using advertisement.

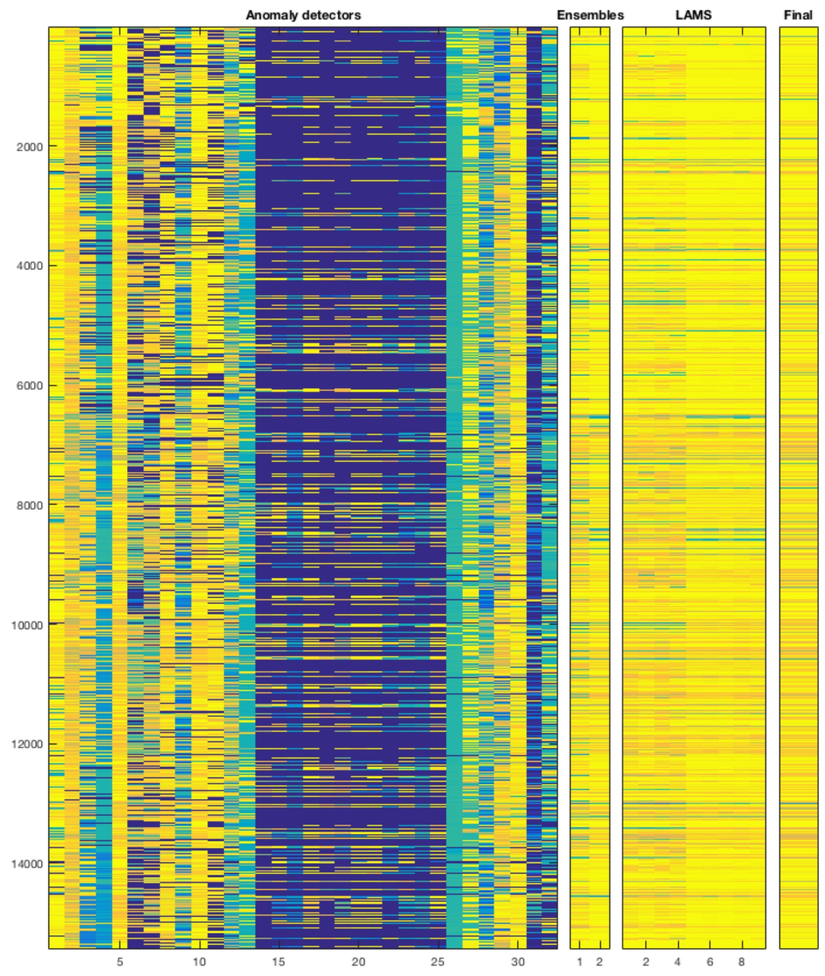

Fig. 7.1: Visualization of the AUC score for all 19,218 malicious samples. Each row of the matrix represents one malicious sample. The matrix is horizontally split according to the system layers into anomaly detectors, detector ensembles, LAMS models and final aggregation, each containing columns that represent individual members of each layer. As can be seen there are 32 anomaly detectors, two ensembles (Evangelista and Acc@Top) and eight LAMS models (four for each ensemble). The cell color representing the AUC score (brighter is higher) show gradual increase in the Ensemble and LAMS layers up to the final AUC scores in the last column.

Figure 7.1 shows a visualization of the above described experimental evaluation. There are 32 anomaly detectors in the first layer of the CTA system some of them are highly specialized in detecting only specific anomalous behaviors (i.e. columns 14 to 25 with low AUC score for most of the malicious samples) whereas the rest are more general anomaly detectors that perform well on the majority of the malicious samples. However, there is no single anomaly detector consistently better in detecting all threats than the rest, which supports the use of the ensemble approach. In the next layer, the AUC scores of both the ensembles are more consistent. But still, there are cases where the Acc@Top ensemble performs well but Evangelista fails and vice versa.

This stretches the need of having both the ensembles implemented. Next layer containing eight LAMS models further increases the AUC score up to the final aggregated score visualized in the last column. The samples with low AUC score achieved by the final layer have consistently low AUC in all the layers. These represent samples that are not detected by any anomaly detector in the lowest layer and thus cannot be improved by the additional layers.

7.2 CAMNEP

The CAMNEP anomaly detection system uses 16 network anomaly detection algorithms designed to detect network anomalies using the NetFlow data. The individual anomaly detectors, described in more detail in Chapter 3, are aggregated using both Evangelista and Acc@Top. The anomaly scores of both the ensembles are then smoothed using five LAMS models, defined in Section 6.3.1, that are then again aggregated to obtain the final anomaly score.

Since we were not able to collect and generate labels for a new NetFlow dataset we have decided to use the same dataset as described in Section 6.3.1.1. To evaluate the effect of the supervised ensemble on the overall system efficacy, we have again evaluated the AUC scores of the system containing only the Evangelista ensemble, completely discarding the Acc@Top ensemble and the complete system.

Table 7.2 shows the AUC scores of the individual system layers for the variant containing the Evangelista aggregation only. As can be seen the AUC score is gradually increasing with decreasing standard deviation after each layer. There are several exceptions in the LAMS models layer that improves the accuracy in detecting attack requests, but decreases the accuracy of the responses. This is, as discussed in detail in Section 6.3.1.2, caused by the fact that the attack responses are typically scattered over the LAMS feature space mixed with other mostly legitimate traffic which results in reduced anomaly score. However, it should be noted that it is important to detect at least one part of the attack, either the attack requests or the responses, with the first being more informative.

To evaluate the effect of the supervised ensemble we have decided to use a leave-one-out cross-validation approach. The AUC score of each of the malicious behaviors was evaluated using the Evangelista ensemble and the Acc@Top ensemble trained on the remaining samples. For the cases when both request and response of an attack are evaluated, we removed both the request and response from the training data. This resulted in 11 separate evaluations with different Acc@Top ensembles. Table 7.3 shows the AUCs of all the evaluations. As can be seen, the system achieves higher AUC scores when compared to the Table 7.2. The only exception is the p2p request and response. The p2p behavior is very different from the behavior of rest of the malicious samples present in the dataset. Therefore the Acc@Top trained on a dataset that missed these samples resulted in slightly worse efficacy of the whole system.

	AUC			
Manual labeling	Anomaly	Ensemble	LAMS	Final
horizontal scan request	0.56 (±0.27)	0.77 (±0.00)	0.78 (±0.05)	**0.83** (±0.00)
horizontal scan response	0.54 (±0.28)	**0.77** (±0.00)	0.64 (±0.16)	0.66 (±0.00)
p2p request	0.55 (±0.26)	0.87 (±0.00)	0.88 (±0.08)	**0.91** (±0.00)
p2p response	0.56 (±0.28)	**0.89** (±0.00)	0.79 (±0.13)	0.75 (±0.00)
scan sql	0.64 (±0.41)	**1.00** (±0.00)	0.97 (±0.07)	**1.00** (±0.00)
scan sql response	0.61 (±0.38)	**0.92** (±0.00)	0.81 (±0.14)	0.91 (±0.00)
ssh cracking request	0.53 (±0.35)	0.80 (±0.00)	0.63 (±0.14)	**0.83** (±0.00)
ssh cracking response	0.46 (±0.33)	**0.84** (±0.00)	0.70 (±0.15)	0.71 (±0.00)
vertical scan	0.74 (±0.30)	**1.00** (±0.00)	**1.00** (±0.00)	**1.00** (±0.00)
vertical scan response	0.61 (±0.35)	**0.98** (±0.00)	0.88 (±0.09)	0.96 (±0.00)
Malware infection				
Neris bot	0.56 (±0.24)	0.69 (±0.00)	0.81 (±0.11)	**0.87** (±0.00)
FastFlux botnet	0.61 (±0.27)	0.81 (±0.00)	0.86 (±0.07)	**0.88** (±0.00)
RBot ICMP flood	0.52 (±0.32)	0.91 (±0.00)	0.91 (±0.03)	**0.94** (±0.00)
Artificial Attack				
SSH scan	0.50 (±0.37)	0.93 (±0.00)	0.96 (±0.13)	**1.00** (±0.00)
SSH scan response	0.49 (±0.34)	**0.84** (±0.00)	0.78 (±0.23)	0.77 (±0.00)
SSH cracking	0.53 (±0.35)	**0.83** (±0.00)	**0.83** (±0.14)	**0.83** (±0.00)
SSH cracking response	0.46 (±0.33)	0.77 (±0.00)	0.75 (±0.15)	**0.78** (±0.00)
Anomalous SSH download request	0.68 (±0.36)	0.95 (±0.00)	0.89 (±0.20)	**0.97** (±0.00)
Anomalous SSH download response	0.61 (±0.37)	**0.95** (±0.00)	0.63 (±0.49)	**0.95** (±0.00)

Table 7.2: AUCs of the individual layers in the case when only the Evangelista ensemble is present. Similarly to Table 7.1, the presented AUC score is a mean of the AUCs of all the members of the individual layers. The number in brackets under the mean AUC score is the standard deviation of the individual AUCs. The standard deviation of the Ensemble and Final layers are equal to zero for all the evaluated malicious behaviors as the layers contain only the Evangelista aggregated ensemble. The highest AUC score in each row is boldfaced.

	AUC			
Manual labeling	Anomaly	Ensemble	LAMS	Final
horizontal scan	0.56 (±0.27)	0.85 (±0.09)	0.83 (±0.07)	**0.88** (±0.00)
horizontal scan response	0.54 (±0.28)	**0.89** (±0.12)	0.75 (±0.17)	**0.89** (±0.00)
p2p request	0.55 (±0.26)	0.63 (±0.37)	0.75 (±0.16)	**0.88** (±0.00)
p2p response	0.56 (±0.28)	0.67 (±0.31)	0.60 (±0.22)	**0.72** (±0.00)
scan sql	0.64 (±0.41)	**1.00** (±0.00)	0.97 (±0.06)	**1.00** (±0.00)
scan sql response	0.61 (±0.38)	0.96 (±0.05)	0.86 (±0.13)	**0.98** (±0.00)
ssh cracking request	0.53 (±0.35)	**0.85** (±0.11)	0.73 (±0.16)	0.84 (±0.00)
ssh cracking response	0.46 (±0.33)	**0.82** (±0.15)	0.64 (±0.20)	0.65 (±0.00)
vertical scan	0.74 (±0.30)	**1.00** (±0.00)	**1.00** (±0.00)	**1.00** (±0.00)
vertical scan response	0.61 (±0.35)	0.91 (±0.06)	0.89 (±0.07)	**0.98** (±0.00)
Malware infection				
Neris bot	0.56 (±0.24)	0.73 (±0.09)	0.84 (±0.15)	**0.92** (±0.00)
FastFlux botnet	0.61 (±0.27)	0.84 (±0.02)	0.86 (±0.05)	**0.93** (±0.00)
RBot ICMP flood	0.52 (±0.32)	0.92 (±0.14)	0.92 (±0.09)	**0.97** (±0.00)
Artificial Attack				
SSH scan	0.50 (±0.37)	**1.00** (±0.00)	0.95 (±0.11)	**1.00** (±0.00)
SSH scan response	0.49 (±0.34)	**1.00** (±0.00)	0.89 (±0.12)	**1.00** (±0.00)
SSH cracking	0.53 (±0.35)	**1.00** (±0.00)	0.80 (±0.27)	0.98 (±0.00)
SSH cracking response	0.46 (±0.33)	**1.00** (±0.00)	0.92 (±0.11)	**1.00** (±0.00)
Anomalous SSH download request	0.68 (±0.36)	0.98 (±0.01)	0.92 (±0.14)	**1.00** (±0.00)
Anomalous SSH download response	0.61 (±0.37)	0.99 (±0.01)	0.69 (±0.41)	**1.00** (±0.00)

Table 7.3: AUC improvements by the individual architectural layers in the case of both the Evangelista and the Acc@Top ensembles are present. Similarly to Table 7.2, the presented AUC score is a mean of the AUCs of all the members of the individual layers. The number in brackets under the mean AUC score is the standard deviation of the individual AUCs. The highest AUC score in each row is boldfaced.

Chapter 8
Conclusions

The main goal of this thesis was to develop a set of techniques to reduce the amount of false positives naturally present in an anomaly-based network intrusion detection system so it could be applied in practice. In Section 1.2 we proposed an architecture of an intrusion detection system, that consists of four layers. The first layer contains a number of simple network anomaly detectors that are able to identify network anomalies related to a malicious behavior. Each consecutive layer of the proposed IDS was designed to reduce the amount of false positives with small to zero effect on the recall. The ensemble layer combines the outputs of the anomaly detectors by constructing two ensembles, Evangelista and Acc@Top, to combine the good properties of both unsupervised and supervised combination functions. Following layer of LAMS models adjusts the anomaly scores by a long term estimates in several feature subspaces to reduce the amount of false positives introduced by the stochasticity of the network traffic. Finally, last layer of aggregation combines the outputs of individual LAMS models resulting in one final anomaly score for each network flow.

The proposed solution was evaluated using a real network data containing huge number of malicious behaviors of various types, severities and network footprint sizes. These included lot of malware, adware, possibly unwanted applications, manually constructed attacks and many more. In Chapter 7 we have shown that each added layer significantly improves the overall detection efficacy of the system.

Additionally, the anomaly detectors presented in Chapter 4, the ensemble construction method and the false positive reduction technique represent a critical components of two existing intrusion detection systems CAMNEP and CTA [29]. The CTA is an on-line malware detection security-as-a-service product delivered by Cisco Systems, which at the time of writing this thesis daily analyzed more than 10 billion web requests generated by millions of users from a number of large enterprise networks. The system finds daily tens of thousands network threats that evaded previously installed security measures positioning the system as the last line of the network defense. This proves the proposed techniques to be sufficient to be able to successfully deploy an anomaly-based intrusion detection system for day-to-day security management of enterprise networks of various types and sizes.

8.1 Main Thesis Achievements

In this section, we will summarize the contributions of this thesis to the state-of-the-art in the field of network anomaly detection. This work improves the existing methods by introducing novel network anomaly detectors, large margin aggregation and local adaptive multivariate smoothing methods all discussed in more detail in following sections.

Novel Anomaly Detection Methods

We presented five novel network anomaly detectors that are using either NetFlow or HTTP access log telemetry data to detect network anomalies related to malicious network behavior.

- TCP flags anomaly detector (Section 3.1) exploits spatial and temporal correlations of TCP flag distribution of source and destination IP addresses. The model uses Principal Component Analysis to model legitimate traffic in a low-dimensional space by removing aforementioned correlations of TCP flags. By using the model, four versions of the anomaly detector were presented, each version detecting different types of anomalies.
- Request-response anomaly detector (Section 3.2) detects various scans, DOS and some type of malware using the request without response identification. This method has low computational complexity and can be easily used on high speed networks.
- DGA detector (Section 3.3) can effectively detect hosts compromised with DGA-performing malware. The detector computes the ratio between the number of DNS requests and newly visited IPs for every host in the local network. Deviations from the mean of these ratios are labeled as anomalous and typically correspond to DGA-based malware.
- Long first touch detector (Section 4.1) is designed to detect malware that tries to mimic the common HTTP requests to hide its communication with the C&C servers. Anomaly is raised when there is an HTTP request with a very long URL that cannot be related to any current user's activity. Experiments shown that the method is able to detect several malware families that are known for such a behavior.
- User-Agent discrepancy detector (Section 4.2) uses User-Agent information contained in the HTTP requests to find the malware-infected hosts in the network. Malware's User-Agents can be browser-like, but in this case they are with high probability discrepant with the one that the infected user actually uses, which is detectable by the proposed User-Agent Discrepancy detector. Next, malware can use its own, specific User-Agent that can be detected by Unused Unknown User-Agent detector. When malware does not fill the User-Agent field at all, we can detect it using No User-Agent detector.

All the presented network anomaly detection techniques were experimentally evaluated using real network data to prove their effectiveness in practice. We have shown that each of the anomaly detectors is able to detect different type of anomalies that can be related to malicious network behavior supporting the use of the ensemble approach.

Large Margin Aggregation

In Chapter 5 we proposed new algorithm for finding a convex combination of anomaly detectors maximizing the accuracy at τ-quantile of returned samples. The algorithm assumes labeled data, which is difficult to obtain and rarely perfect in the security domains. Therefore, an emphasis was put on the experimental study, involving both CAMNEP and CTA intrusion detection systems, eight types of combination functions, 34 different network captures containing more than 20 million samples of behaviors under different types of labeling noise. The experimental results show that the proposed method is not only better than the state-of-the-art, but also more robust with respect to various kinds of noise in labels we can expect in intrusion detection domains. The results also shown that supervised methods can easily overfit if some type of malicious behavior is completely missing in the training data or is incorrectly labeled (mistake of labeling oracle). The severity of the overfitting depends on how much different types of malicious behavior are similar to each other. This motivated us to use both supervised and unsupervised combination functions in the proposed architecture.

Local Adaptive Multivariate Smoothing

Chapter 6 categorizes the false positives of the anomaly-based intrusion detection systems into two classes: unstructured and structured. While the structured can be easily filtered out using white list, the unstructured false positive are much harder to eliminate.

We proposed to smooth the outputs of anomaly detectors by online Local Adaptive Multivariate Smoothing (LAMS) models to reduce the amount of unstructured false positives of the anomaly-based intrusion detection systems. The technique smooths detectors output simultaneously over time and feature space, which improves the estimate of true anomaly score.

We proved under mild assumptions that the method reduces the amount of unstructured false positives caused by stochasticity of the network traffic. This is also supported by an extensive experimental evaluation involving both CTA and CAMNEP intrusion detection systems that use a diverse set of network anomaly detectors applied to different network telemetry data. Finally, we show how the proposed solution can be efficiently implemented to process large streams of non-stationary data.

8.2 Author's Publications

This section summarizes the author's publications related to the content of this thesis.

Articles in Journals with Impact Factor (4)

1. **Martin Grill**, Tomáš Pevný. Learning combination of anomaly detectors for security domain. *Computer Networks*, Elsevier (In press), 2016. Impact factor 1.45. (**80%**)
2. **Martin Grill**, Tomáš Pevný, and Martin Rehák. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences*, Elsevier (In press), 2016. Impact factor 1.58. (**70%**)
3. Sebastian Garcia, **Martin Grill**, Jan Stiborek, Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, Volume 45, September 2014, pages 100-123. Impact factor 1.03. (**30%**)
4. Martin Rehák, Michal Pěchouček, **Martin Grill**, Jan Stiborek, Karel Bartoš, Pavel Čeleda. Adaptive multiagent system for network traffic monitoring. *IEEE Intelligent Systems*, vol. 24, no. 3, pages 16–25, May-June 2009. Impact factor 3.14 (**16%**)

Peer-reviewed Journal Articles (3)

1. Jan Stiborek, **Martin Grill**, Martin Rehák, Karel Bartoš and Jan Jusko. Game Theoretical Model for Adaptive Intrusion Detection System. *Lecture Notes in Computer Science: Transactions on Computational Collective Intelligence*, vol. 15, no. 0, pages 133 – 163, 2014. (**30%**)
2. Martin Rehák, Michal Pěchouček, Karel Bartoš, **Martin Grill**, Pavel Čeleda, Vojtěch Krmíček. CAMNEP: An intrusion detection system for high speed networks. In *Progress in Informatics*, volume 4, pages 65–74, 2008. (**10%**)
3. Martin Rehák, Michal Pěchouček, **Martin Grill**, Karel Bartoš, Vojtěch Krmíček and Pavel Čeleda. Collaborative Approach to Network Behavior Analysis Based on Hardware-Accelerated FlowMon Probes. *International Journal of Electronic Security and Digital Forensics*, vol. 2, no. 1, pages 35 – 48, 2009. (**12%**)

Patents (2)

1. **Martin Grill** and Ivan Nikolaev. Detecting DGA-Based malicious software using network flow information. US Patent 20,160,036,836, 2016. (Published, pending approval)
2. Ivan Nikolaev, **Martin Grill** and Jan Jusko. Detecting network services based on network flow data. US Patent 20,160,080,236, 2016. (Published, pending approval)

In ISI Proceedings (10)

1. **Martin Grill** and Martin Rehák. Malware detection using HTTP user-agent discrepancy identification. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, Atlanta, GA, pages 221-226, IEEE, 2014. (**90%**)

2. Tomáš Pevný, **Martin Grill** and Martin Rehák. Detecting anomalous network hosts by means of PCA. In *Proceedings of IEEE Workshop on Informations Forensics and Security*, pages 103–108, IEEE, 2012. (**33%**)
3. Jan Stiborek, **Martin Grill**, Martin Rehák, Karel Bartoš and Jan Jusko. Game Theoretical Adaptation Model for Intrusion Detection System. In *Proceedings of the 10th Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, pages 201–210, Springer Berlin, 2012. (**20%**)
4. Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, **Martin Grill**, Jan Stiborek, Karel Bartoš and Thomas Engel. Threat-model-driven runtime adaptation and evaluation of intrusion detection system. In *Proceedings of the 6th International Conference on Autonomic Computing and Communications*, pages 65–66, ACM Press, 2009. (**14%**)
5. Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, **Martin Grill**, Jan Stiborek, Karel Bartoš and Thomas Engel. Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 61–80, Springer Heidelberg, 2009. (**14%**)
6. Martin Rehák, Michal Pěchouček, **Martin Grill**, Karel Bartoš, Pavel Čeleda, Vojtěch Krmíček. Collaborative approach to Network Behavior Analysis. In *Proceedings of the 4th International Conference on Global E-Security*, pages 153–160, Springer Heidelberg, 2008. (**15%**)
7. Karel Bartoš, **Martin Grill**, Vojtěch Krmíček, Martin Rehák and Pavel Čeleda. Flow Based Network Intrusion Detection System using Hardware-Accelerated NetFlow Probes. In *Proceedings of CESNET Conference - Security, middleware, and virtualization – glue of future networks*, pages 49–56, CESNET, 2008. (**20%**)
8. Martin Rehák, Michal Pěchouček, **Martin Grill** and Karel Bartoš. Trust Based Classifier Combination for Network Anomaly Detection. In *Proceedings of the 12th International Conference on Cooperative Information Agents (CIA)*, pages 116–130, Springer Heidelberg, 2008. (**15%**)
9. Martin Rehák, Michal Pěchouček, Karel Bartoš, **Martin Grill**, Pavel Čeleda, Vojtěch Krmíček. Improving Anomaly Detection Error Rate by Collective Trust Modeling. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 398–399, Springer Heidelberg, 2008. (**10%**)
10. Martin Rehák, Michal Pěchouček, Karel Bartoš, **Martin Grill** and Pavel Čeleda. Network Intrusion Detection by Means of Community of Trusting Agents. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pages 498–504, IEEE Computer Society, 2007. (**10%**)

In Other Proceedings (8)

1. **Martin Grill**, Ivan Nikolaev, Veronica Valeros and Martin Rehák. Detecting DGA malware using NetFlow. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, pages 1304–1309, IEEE, 2015. (**43%**)
2. **Martin Grill**, Martin Rehák, and Jan Stiborek. Strategic self-organization methods for intrusion detection systems. In *Proceedings of the Congerence Security and Protection of Information*, University of Defence, Brno, 2011. (**33%**)
3. Martin Rehák, Michal Pěchouček, **Martin Grill**, Jan Stiborek and Karel Bartoš. Game Theoretical Adaptation Model for Intrusion Detection System, In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1123–1124, 2011. (**20%**)
4. Martin Rehák, Jan Stiborek and **Martin Grill**. Intelligence, not Integration:Distributed Regret Minimization for IDS Control. In *IEEE Symposium on Computational Intelligence in Cyber Security*, pages 217–224, IEEE, 2011. (**33%**)

5. Martin Rehák, **Martin Grill** and Jan Stiborek. On the Value of Coordination in Distributed Self-Adaptation of Intrusion Detection System. In *Proceedings Web Intelligence and Intelligent Agent Technology WI-IAT11*, pages 196–203, IEEE, 2011. (**33%**)
6. Martin Rehák, Karel Bartoš, **Martin Grill**, Jan Stiborek and Michal Svoboda. Monitoring sití pomocí NetFlow dat - od paketů ke strategiím. In *Proceedings of Česká společnost uživatelů otevřených systémů EurOpen.cz*, pages 75–81, 2009. (**20%**)
7. Martin Rehák, Eugen Staab, Michal Pěchouček, Jan Stiborek, **Martin Grill** and Karel Bartoš. Dynamic Information Source Selection for Intrusion Detection Systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1009–1016, ACM Press, 2009. (**15%**)
8. Martin Rehák, Michal Pěchouček, Pavel Čeleda, Vojtěch Krmíček, **Martin Grill**, Karel Bartoš. Multi agent approach to network intrusion detection (demo paper). In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1695–1696, ACM Press, 2008. **(16%)**

References

1. Charu C Aggarwal. Outlier ensembles: position paper. *ACM SIGKDD Explorations Newsletter*, 14(2):49–58, 2013.
2. Lavasoft Alexander Adamov. Backdoor.Win32.Shiz. `http://www.lavasoft.com/mylavasoft/malware-descriptions/blog/backdoorwin32shiz`, 2012.
3. Loucif Kharouni Alice Decker, David Sancho, Max Goncharov, and Robert McArdle. Pushdo / cutwail botnet. `http://www.trendmicro.com/cloud-content/us/pdfs/business/white-papers/wp_study-of-pushdo-cutwail-botnet.pdf`, 2009.
4. Sultan Aljahdali. An effective intrusion detection method using optimal hybrid model of classifiers. *Journal of Computational Methods in Science and Engineering*, 10:51–60, 2010.
5. Riyad Alshammari, Sumalee Sonamthiang, Mohsen Teimouri, and Denis Riordan. Using neuro-fuzzy approach to reduce false positive alerts. In *Fifth Annual Conference on Communication Networks and Services Research (CNSR'07)*, pages 345–349. IEEE, 2007.
6. Hakan Altınçay. On the independence requirement in dempster-shafer theory for combining classifiers providing statistical evidence. *Applied Intelligence*, 25:73–90, 2006.
7. Pedram Amini and C Pierce. Kraken botnet infiltration, 2008.
8. Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX Security Symposium*, 2012.
9. Ayesha Binte Ashfaq, Mobin Javed, Syed Ali Khayam, and Hayder Radha. An information-theoretic combining method for multi-classifier anomaly detection systems. In *Communications (ICC), 2010 IEEE international conference on*, pages 1–5. IEEE, 2010.
10. Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 71–82. ACM, 2002.
11. Paul Barford and David Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 69–73. ACM, 2001.
12. Paul Barford and Vinod Yegneswaran. An inside look at botnets. In *Malware Detection*, pages 171–191. Springer, 2007.
13. Jeffrey A. Barnett. Computational methods for a mathematical theory of evidence. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, pages 868–875. Morgan Kaufmann Publishers Inc., 1981.
14. Karel Bartoš and Michal Sofka. Robust representation for domain adaptation in network security. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 116–132. Springer, 2015.
15. Karel Bartoš, Michal Sofka, and Vojtěch Franc. Learning invariant representation for malicious network traffic detection. In *22th European Conference on Artificial Intelligence (ECAI)*. IOS Press, 2016.
16. Karel Bartoš, Michal Sofka, and Vojtěch Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *J25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016.
17. Jon Atli Benediktsson and Philip H Swain. Consensus theoretic classification methods. *IEEE transactions on Systems, Man, and Cybernetics*, 22(4):688–704, 1992.
18. Przemysław Bereziński, Bartosz Jasiul, and Marcin Szpyrka. An entropy-based network anomaly detection method. *Entropy*, 17(4):2367–2408, 2015.
19. Battista Biggio, Giorgio Fumera, and Fabio Roli. Multiple classifier systems under attack. In *International Workshop on Multiple Classifier Systems*, pages 74–83. Springer, 2010.
20. James Blustein and Amal El-Maazawi. Bloom filters. a tutorial, analysis, and survey. *Halifax, NS: Dalhousie University*, pages 1–31, 2002.

21. Damiano Bolzoni and Sandro Etalle. Aphrodite: An anomaly-based architecture for false positive reduction. *arXiv preprint cs/0604026*, 2006.
22. Stephen Boyd, Corinna Cortes, Mehryar Mohri, and Ana Radovanovic. Accuracy at the top. In *Advances in neural information processing systems*, pages 953–961, 2012.
23. Christian Callegari, Angelo Coluccia, Alessandro D'Alconzo, Wendy Ellens, Stefano Giordano, Michel Mandjes, Michele Pagano, Teresa Pepe, Fabio Ricciato, and Piotr Zuraniewski. A methodological overview on anomaly detection. In *Data Traffic Monitoring and Analysis*, pages 148–183. Springer, 2013.
24. Christer Carlsson and Robert Fuller. *Fuzzy reasoning in decision making and optimization*, volume 82. Physica, 2012.
25. Cristiano Castelfranchi and Rino Falcone. Principles of trust for mas: Cognitive anatomy, social importance, and quantification. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 72–79. IEEE, 1998.
26. Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.
27. Youksamay Chanthakoummane, Saiyan Saiyod, and N Khamphakdee. Evaluation snort-ids rules for botnets detection. In *National Conference on Infomation Technology*, 2015.
28. Cisco Systems. Cisco IOS NetFlow. http://www.cisco.com/go/netflow, 2007.
29. Cisco Systems. CTA Cisco Cognitive Threat Analytics on Cisco Cloud Web Security. `https://cognitive.cisco.com/`, 2014–2015.
30. Benoit Claise. Cisco systems netflow services export version 9, 2004.
31. Benoit Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008.
32. Angelo Coluccia, Alessandro D'Alconzo, and Fabio Ricciato. Distribution-based anomaly detection via generalized likelihood ratio test: A general maximum entropy approach. *Computer Networks*, 57(17):3446–3462, 2013.
33. Igino Corona, Giorgio Giacinto, and Fabio Roli. Intrusion Detection in Computer Systems Using Multiple Classifier Systems. *Electronic Engineering*, 113:91–113, 2008.
34. David Dagon. Botnet detection and response. In *OARC workshop*, volume 2005, 2005.
35. David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee. A taxonomy of botnet structures. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 325–339. IEEE, 2007.
36. Leslie Daigle. WHOIS Protocol Specification. RFC 3912, March 2013.
37. Belur V. Dasarathy. *Decision Fusion.* IEEE Computer Society Press, 1994.
38. Dorothy E Denning. An intrusion-detection model. In *Security and Privacy, 1986 IEEE Symposium on*, pages 118–118. IEEE, 1986.
39. Luc Devroye and Adam Krzyzak. An equivalence theorem for l1 convergence of the kernel regression estimate. *Journal of Statistical Planning and Inference*, 23(1):71 – 82, 1989.
40. Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, October 2015.
41. Thomas G. Dietterich. Ensemble methods in machine learning. In *MULTIPLE CLASSIFIER SYSTEMS, LBCS-1857*, pages 1–15. Springer, 2000.
42. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley & Sons, New York, 2nd edition, 2001.
43. Jeff Edwards. Shiz and rohimafo: Malware cousins. `http://www.arbornetworks.com/asert/2010/09/shiz-and-rohimafo-malware-cousins/`, 2010.
44. Huwaida Tagelsir Elshoush and Izzeldin Mohamed Osman. Alert correlation in collaborative intelligent intrusion detection systems—a survey. *Applied Soft Computing*, 11(7):4349–4365, 2011.
45. Levent Ertoz, Eric Eilertson, Aleksandar Lazarevic, Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava, and Paul Dokas. Minds-minnesota intrusion detection system. *Next Generation Data Mining*, pages 199–218, 2004.
46. Juan M Estevez-Tapiador, Pedro Garcia-Teodoro, and Jesus E Diaz-Verdejo. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, 27(16):1569–1584, 2004.
47. Paul F Evangelista, Mark J Embrechts, and Boleslaw K Szymanski. Data fusion for outlier detection through pseudo-roc curves and rank distributions. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 2166–2173. IEEE, 2006.
48. MC Fairhurst and Ahmad Fuad Rezaur Rahman. Enhancing consensus in multiple expert decision fusion. *IEE Proceedings-Vision, Image and Signal Processing*, 147(1):39–46, 2000.
49. Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.
50. Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Rfc 2616, hypertext transfer protocol–http/1.1, 1999, 2009.
51. Imola K Fodor. A survey of dimension reduction techniques, 2002.
52. Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 8:1–8:12. ACM, 2010.

53. Vojtěch Franc, Michal Sofka, and Karel Bartoš. Learning detector of malicious network traffic from weak labels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 85–99. Springer, 2015.
54. Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.
55. Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.
56. Giorgio Fumera and Fabio Roli. Performance analysis and comparison of linear combiners for classifier fusion. In *Proceedings of IAPR International Workshop on Statistical Pattern Recognition (SPR 2002)*, pages 424–432. Springer, 2002.
57. Jing Gao, Wei Fan, Deepak Turaga, Olivier Verscheure, Xiaoqiao Meng, Lu Su, and Jiawei Han. Consensus extraction from heterogeneous detectors to improve performance over network traffic anomaly detection. In *INFOCOM, 2011 Proceedings IEEE*, pages 181–185. IEEE, 2011.
58. Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 212–221. IEEE, 2006.
59. Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.
60. Giorgio Giacinto, Roberto Perdisci, and Fabio Roli. Network intrusion detection by combining one-class classifiers. In *Image Analysis and Processing – ICIAP 2005*, volume 3617 of *Lecture Notes in Computer Science*, pages 58–65. Springer Berlin Heidelberg, 2005.
61. Giorgio Giacinto and Fabio Roli. Dynamic classifier selection based on multiple classifier behaviour. *Pattern Recognition*, 34:1879–1881, 2001.
62. Giorgio Giacinto and Fabio Roli. Intrusion detection in computer networks by multiple classifier systems. In *In Proc. of the 16th International Conference on Pattern Recognition (ICPR), Volume 2*, pages 390–393. IEEE press, 2002.
63. Giorgio Giacinto, Fabio Roli, and Luca Didaci. Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern Recogn. Lett.*, 24:1795–1803, 2003.
64. Giorgio Giacinto, Fabio Roli, and Giorgio Fumera. Design of effective multiple classifier systems by clustering of classifiers. In *Proc. of ICPR2000, 15th Int. Conference on Pattern Recognition*, pages 3–8, 2000.
65. Giorgio Giacinto, Fabio Roli, and Giorgio Fumera. Selection of image classifiers. *Electronics Letters*, 36(5):1, 2000.
66. Martin Grill and Ivan Nikolaev. Detecting dga-based malicious software using network flow information, February 4 2016. US Patent 20,160,036,836.
67. Martin Grill, Ivan Nikolaev, Veronica Valeros, and Martin Rehák. Detecting dga malware using netflow. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1304–1309, May 2015.
68. Martin Grill and Tomáš Pevný. Learning combination of anomaly detectors for security domain. *Computer Networks*, pages –, 2016.
69. Martin Grill, Tomáš Pevný, and Martin Rehák. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences*, pages –, 2016.
70. Martin Grill and Martin Rehák. Malware detection using http user-agent discrepancy identification. In *Information Forensics and Security (WIFS), 2014 IEEE International Workshop on*, pages 221–226, Dec 2014.
71. Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.
72. Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. In *Computer Networks*. Citeseer, 2000.
73. Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.
74. Thoufique Haq. Now you see me – h-worm by houdini. `http://www.fireeye.com/blog/technical/threat-intelligence/2013/09/now-you-see-me-h-worm-by-houdini.html`, 2013.
75. M Hasan, Mohammed Nasser, Biprodip Pal, and Shamim Ahmad. Intrusion detection using combination of various kernels based support vector machine. *International Journal of Scientific & Engineering Research*, 4:1454–1463, 2013.
76. Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.
77. Zengyou He, Shengchun Deng, and Xiaofei Xu. A unified subspace outlier ensemble framework for outlier detection. In *Advances in Web-Age Information Management*, pages 632–637. Springer, 2005.
78. Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE transactions on pattern analysis and machine intelligence*, 16(1):66–75, 1994.
79. Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.
80. Emmanuel Hooper. An intelligent detection and response strategy to false positives and network attacks. In *Fourth IEEE International Workshop on Information Assurance (IWIA'06)*, pages 20–pp. IEEE, 2006.

81. Trung Dong Huynh, Nicholas R Jennings, and Nigel R Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.
82. Ali Islam. More flame/skywiper cnc behavior uncovered. `http://www.fireeye.com/blog/technical/malware-research/2012/06/flame-skywiper-cnc-update.html`, 2012.
83. Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.
84. Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, pages 377–384. ACM, 2005.
85. Michael I. Jordan. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214, 1994.
86. Audun Josang, Elizabeth Gray, and Michael Kinateder. Simplification and analysis of transitive trust networks. *Web Intelligence and Agent Systems*, 4(2):139–162, 2006.
87. Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, November 2003.
88. Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375. ACM, 2002.
89. Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM, 2004.
90. Josef Kittler, Mohamad Hatef, Robert PW Duin, and Jiri Matas. On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence*, 20(3):226–239, 1998.
91. Josef Kittler, Mohamad Hatef, Robert PW Duin, and Jiri Matas. On combining classifiers. *IEEE transactions on pattern analysis and machine intelligence*, 20(3):226–239, 1998.
92. Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, volume 42. SIAM, 2011.
93. Ludmila I. Kuncheva. A theoretical study on six classifier fusion strategies. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:281–286, 2002.
94. Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms.* Wiley, 2004.
95. Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.
96. Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining Anomalies using Traffic Feature Distributions. In *ACM SIGCOMM, Philadelphia, PA, August 2005*, pages 217–228, New York, NY, USA, 2005. ACM Press.
97. Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.
98. Quoc Le and Alexander Smola. Direct optimization of ranking measures. *arXiv preprint arXiv:0704.3359*, 2007.
99. Seungmin Lee, Gisung Kim, and Sehun Kim. Self-adaptive and dynamic clustering for online anomaly detection. *Expert Systems with Applications*, 38(12):14891–14898, 2011.
100. Wenke Lee, Salvatore J Stolfo, Philip K Chan, Eleazar Eskin, Wei Fan, Matthew Miller, Shlomo Hershkop, and Junxin Zhang. Real time data mining-based intrusion detection. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 89–100. IEEE, 2001.
101. Nan Li, Rong Jin, and Zhi-Hua Zhou. Top rank optimization in linear time. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2014.
102. Nancy A Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.
103. Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.
104. Matthew Mahoney and Philip Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection*, volume 2820 of *Lecture Notes in Computer Science*, pages 220–237. Springer Berlin / Heidelberg, 2003.
105. Matthew V. Mahoney and Philip K. Chan. Phad: Packet header anomaly detection for identifying hostile network traffic. Technical report, 2001.
106. Marek Małowidzki, P Berezinski, and Michał Mazur. Network intrusion detection: Half a kingdom for a good dataset. In *Proceedings of NATO STO SAS-139 Workshop, Portugal*, 2015.
107. Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 775–782, 2010.
108. D Kevin McGrath and Minaxi Gupta. Behind phishing: An examination of phisher modi operandi. *LEET*, 8:1–8, 2008.
109. Kevin McNamee. Malware analysis report: Zeroaccess. Technical report, Sirefef. Technical Report by Kindsight Security Labs, 2012.

110. Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis*, 53(4):1483–1494, 2009.
111. Christopher J. Merz. Using Correspondence Analysis to Combine Classifiers. *Machine Learning*, 36(1-2):33–58, jul 1999.
112. Asieh Mokarian, Ahmad Faraahi, and Arash Ghorbannia Delavar. False positives reduction techniques in intrusion detection systems-a review. *International Journal of Computer Science and Network Security (IJCSNS)*, 13(10):128, 2013.
113. Benjamin Morin and Hervé Debar. Correlation of intrusion symptoms: an application of chronicles. In *International Workshop on Recent Advances in Intrusion Detection*, pages 94–112. Springer, 2003.
114. Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.
115. Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. Mining outliers with ensemble of heterogeneous detectors on random subspaces. In *Database Systems for Advanced Applications*, pages 368–383. Springer, 2010.
116. Peng Ning and Sushil Jajodia. Intrusion detection techniques. *The Internet Encyclopedia*, 2003.
117. Stephen Northcutt, Lenny Zeltser, Scott Winters, Karen Kent, and Ronald W. Ritchey. *Inside Network Perimeter Security (2nd Edition) (Inside)*. Sams, Indianapolis, IN, USA, 2005.
118. Keith Noto, Carla Brodley, and Donna Slonim. Frac: a feature-modeling approach for semi-supervised and unsupervised anomaly detection. *Data mining and knowledge discovery*, 25(1):109–133, 2012.
119. David W. Opitz and Jude W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Advances in Neural Information Processing Systems*, pages 535–541. MIT Press, 1996.
120. Nikunj C. Oza and Kagan Tumer. Classifier ensembles: Select real-world applications, 2008.
121. Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470, 2007.
122. Sandhya Peddabachigari, Ajith Abraham, Crina Grosan, and Johnson Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114 – 132, 2007.
123. Tomáš Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, pages 1–30, 2015.
124. Tomáš Pevný, Martin Komoň, and Martin Rehák. Attacking the ids learning processes. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8687–8691. IEEE, 2013.
125. Tomáš Pevný, Martin Rehák, and Martin Grill. Detecting anomalous network hosts by means of pca. In *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 103–108, Dec 2012.
126. Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection*, pages 102–124. Springer, 2004.
127. Stelian Pilici. Remove trojan:win32/bumat!rts (virus removal guide). `https://malwaretips.com/blogs/trojan-win32-bumat-rts-removal/`, 2013.
128. Robi Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, 2006.
129. Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. Conficker c analysis. *SRI International*, 2009.
130. Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, pages 5–8, 2001.
131. Federico Montesino Pouzols and Amaury Lendasse. Adaptive kernel smoothing regression using vector quantization. In *Evolving and Adaptive Intelligent Systems (EAIS), 2011 IEEE Workshop on*, pages 85–92. IEEE, 2011.
132. Ahmad Fuad Rezaur Rahman, H. Alam, and Michael C. Fairhurst. Multiple classifier combination for character recognition: Revisiting the majority voting system and its variations. In *Proceedings of the 5th International Workshop on Document Analysis Systems V*, DAS '02, pages 167–178. Springer-Verlag, 2002.
133. Alain Rakotomamonjy. Sparse support vector infinite push. *arXiv preprint arXiv:1206.6432*, 2012.
134. Kadangode Ramakrishnan, Sally Floyd, and David Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040.
135. Sarvapali Ramchurn, Nicholas Jennings, Carles Sierra, and Lluis Godo. Devising a trust model for multi-agent interactions using confidence and reputation. *Applied Artificial Intelligence*, 18(9-10):833 – 852, 2004.
136. Jason Reed, Adam J Aviv, Daniel Wagner, Andreas Haeberlen, Benjamin C Pierce, and Jonathan M Smith. Differential privacy for collaborative security. In *Proceedings of the Third European Workshop on System Security*, pages 1–7. ACM, 2010.
137. Martin Rehák. Multiagent trust modeling for open network environments. *PhD dissertation*, 2008.
138. Martin Rehák and Martin Grill. Categorisation of False Positives: Not All Network Anomalies are Born Equal. Unpublished Lecture, 2014.

139. Martin Rehák, Michal Pěchouček, Martin Grill, and Karel Bartoš. Trust-based classifier combination for network anomaly detection. In *Cooperative Information Agents XII*, volume 5180 of *Lecture Notes in Computer Science*, pages 116–130. Springer Berlin / Heidelberg, 2008.
140. Martin Rehák and Michal Pěchouček. Trust modeling with context representation and generalized identities. In *Cooperative Information Agents XI*, number 4676 in LNAI/LNCS. Springer-Verlag, 2007.
141. Martin Rehák, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Pavel Čeleda. Adaptive multiagent system for network traffic monitoring. *IEEE Intelligent Systems*, (3):16–25, 2009.
142. Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Thomas Engel. Runtime monitoring and dynamic reconfiguration for intrusion detection systems. In *Recent Advances in Intrusion Detection*, volume 5758 of *Lecture Notes in Computer Science*, pages 61–80. Springer Berlin / Heidelberg, 2009.
143. Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.
144. Lee M Rossey, Robert K Cunningham, David J Fried, Jesse C Rabek, Richard P Lippmann, Joshua W Haines, and Marc A Zissman. Lariat: Lincoln adaptable real-time information assurance testbed. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 6, pages 6–2671. IEEE, 2002.
145. Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J Dietrich, and Herbert Bos. Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 97–111. IEEE, 2013.
146. André Rouél. UADetector library. `http://uadetector.sourceforge.net/`, 2011–2014.
147. Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 1–14. ACM, 2009.
148. Jordi Sabater and Carles Sierra. Reputation and social network analysis in multi-agent systems. In *Proceedings of AAMAS '02*, pages 475–482, Bologna, Italy, July 2002.
149. Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24(1):33–60, 2005.
150. Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). Technical Report 800-94, NIST, US Dept. of Commerce, 2007.
151. Steven L Scott. A bayesian paradigm for designing intrusion detection systems. *Computational statistics & data analysis*, 45(1):69–83, 2004.
152. Liviu Serban. Analysis of w32/virut.ce. `http://techblog.avira.com/2011/03/11/polymorphic-virut-malware/en/`, 2011.
153. Glenn Shafer. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton, NJ, 1976.
154. Shashank Shanbhag and Tilman Wolf. Accurate anomaly detection through parallelism. *Network, IEEE*, 23(1):22–28, 2009.
155. Prasha Shrestha, Suraj Maharjan, Gabriela Ramírez de la Rosa, Alan Sprague, Thamar Solorio, and Gary Warner. *Using String Information for Malware Family Identification*, pages 686–697. Springer International Publishing, Cham, 2014.
156. Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. In *in Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with the Third IEEE International Conference on Data Mining (ICDM'03*, pages 172–179, 2003.
157. Fernando Silveira, Christophe Diot, Nina Taft, and Ramesh Govindan. Astute: Detecting a different class of traffic anomalies. *ACM SIGCOMM Computer Communication Review*, 40(4):267–278, 2010.
158. Augustin Soule, Kavé Salamatian, and Nina Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 31–31. USENIX Association, 2005.
159. Georgios P Spathoulas and Sokratis K Katsikas. Reducing false positives in intrusion detection systems. *computers & security*, 29(1):35–44, 2010.
160. Avinash Sridharan, Tao Ye, and Supratik Bhattacharyya. Connectionless port scan detection on the backbone. In *Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International*, pages 10–pp. IEEE, 2006.
161. Stuart Staniford, James A Hoagland, and Joseph M McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1):105–136, 2002.
162. Jan Stiborek, Martin Rehák, and Tomáš Pevný. Towards scalable network host simulation. In *Seconds International Workshop on Agents and Cybersecurity*, page 27, 2015.
163. Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009.

164. Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. In *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats*, LEET'11, pages 4–4, Berkeley, CA, USA, 2011. USENIX Association.
165. Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the storm and nugache trojans: P2p is here. *USENIX; login*, 32(6):18–27, 2007.
166. James Surowiecki. *The wisdom of crowds*. Anchor Books. Anchor Books, 2005.
167. Zhihong Tian, Weizhe Zhang, Jianwei Ye, Xiangzhan Yu, and Hongli Zhang. Reduction of false positives in intrusion detection via adaptive alert classifier. In *Information and Automation, 2008. ICIA 2008. International Conference on*, pages 1599–1602, June 2008.
168. Kagan Tumer and Nikunj C Oza. Decimated input ensembles for improved generalization. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 5, pages 3069 –3074 vol.5, 1999.
169. Veronica Valeros. Dnschanger outbreak linked to adware install base. `http://blogs.cisco.com/security/dnschanger-outbreak-linked-to-adware-install-base`, 2016.
170. Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. Learning to rank by optimizing ndcg measure. In *Advances in neural information processing systems*, pages 1883–1891, 2009.
171. Jouni Viinikka, Hervé Debar, Ludovic Mé, Anssi Lehikoinen, and Mika Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, 10(4):312 – 324, 2009. Special Issue on Information Fusion in Computer Security.
172. Alberto Volpatto, Federico Maggi, and Stefano Zanero. Effective multimodel anomaly detection using cooperative negotiation. In *Proceedings of the First international conference on Decision and game theory for security*, GameSec'10, pages 180–191. Springer-Verlag, 2010.
173. Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964.
174. David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
175. James Wyke. The zeroaccess botnet–mining and fraud for massive financial gain. *Sophos Technical Paper*, 2012.
176. Cheng Xiang, Png Chin Yong, and Lim Swee Meng. Design of multiple-level hybrid classifier for intrusion detection system using bayesian clustering and decision trees. *Pattern Recognition Letters*, 29(7):918–924, 2008.
177. Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 169–180. ACM, 2005.
178. Kuai Xu, ZL Zhang, and Supratik Bhattacharyya. Reducing unwanted traffic in a backbone network. In *SRUTI 05: Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 9–15, 2005.
179. Lei Xu, Adam Krzyzak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE transactions on systems, man, and cybernetics*, 22(3):418–435, 1992.
180. Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *IEEE/ACM Transactions on Networking (TON)*, 20(5):1663–1677, 2012.
181. Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, 1988.
182. Moosa Yahyazadeh and Mahdi Abadi. Botgrab: A negative reputation system for botnet detection. *Computers & Electrical Engineering*, 41:68 – 85, 2015.
183. InSeon Yoo. Protocol anomaly detection and verification. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 74–81, June 2004.
184. Bernard Zenko. Is combining classifiers better than selecting the best one. In *Machine Learning*, pages 255–273. Morgan Kaufmann, 2004.
185. Boyun Zhang, Jianping Yin, Jingbo Hao, Dingxing Zhang, and Shulin Wang. Malicious codes detection based on ensemble learning. In *Autonomic and Trusted Computing*, volume 4610 of *Lecture Notes in Computer Science*, pages 468–477. Springer Berlin / Heidelberg, 2007.
186. Harry Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004.
187. Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network anomography. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, page 30, Berkeley, CA, USA, 2005. USENIX Association.
188. Junlin Zhou, Deng Jun, Yan Fu, and Yue Wu. Distributed anomaly detection by model sharing. In *Apperceiving Computing and Intelligence Analysis, 2009. ICACIA 2009. International Conference on*, pages 297 –300, 2009.
189. Xuchuan Zhou, Yong Zhong, and Liping Cai. Anomaly detection from distributed flight record data for aircraft health management. In *Computational and Information Sciences (ICCIS), 2010 International Conference on*, pages 156 –159, 2010.
190. Yong-lin Zhou, Qing-shan Li, Qidi Miao, and Kangbin Yim. Dga-based botnet detection using dns traffic. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):116–123, 2013.
191. Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

Appendix A
Visualizations of all LAMS models used in the CAMNEP detection engine

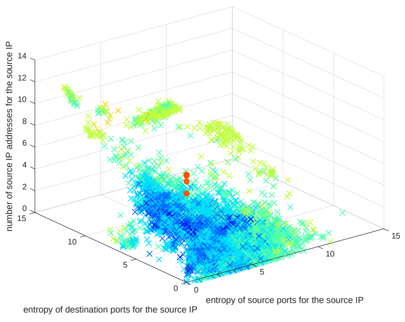

Fig. A.1: Xu-source

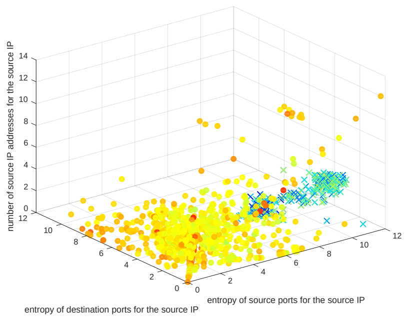

Fig. A.2: Xu-destination

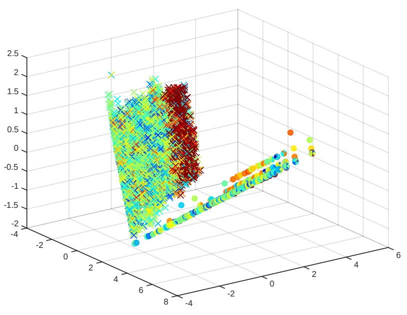

Fig. A.3: MINDS

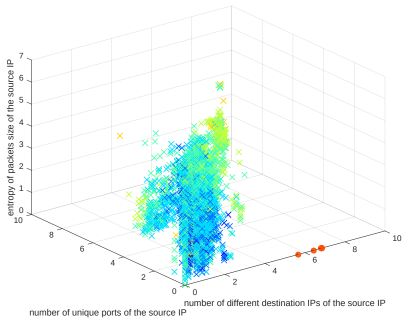

Fig. A.4: TAPS

Fig. A.5: Additional visualization to the Figure 6.4 in Chapter 6 of the horizontal scan and corresponding responses contained in the manually labeled dataset described in Section 6.3.1.1 in the context space of the rest of the LAMS models defined in Table 6.1. Each point on the individual scatter plots represent one scan request (cross) or scan response (dot). The color corresponds to obtained anomaly score with red being the most anomalous and blue being the least. Since the MINDS model uses four features to define its context we have used the Multidimensional scaling to be able to show the data in three dimensional plot (therefore there are no axis labels). The figures show that similarly to the Lakhina model (Figure 6.4), Xu-source and TAPS models have the responses spread across a region of low anomaly score and the requests limited in a small region of high anomaly. For the Xu-destination the requests are spread in the context space but still maintaining high anomaly score. For this particular behavior the MINDS model as the only one increases anomaly score of a part of the responses and reduces the anomaly score of the requests.