IMPROVING ANOMALY DETECTION PERFORMANCE USING INFORMATION THEORETIC AND MACHINE LEARNING TOOLS

by

Ayesha Binte Ashfaq

A dissertation submitted to the faculty of
The School of Electrical Engineering and Computer Science (SEECS, NUST)
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Information Technology

Department of Computing

2014

Approved by:

Dr. Syed Ali Khayam

Dr. Zahid Anwar

Dr. Awais Shibli

Dr. Hayder Radha

ABSTRACT

AYESHA BINTE ASHFAQ. Improving Anomaly Detection Performance Using Information Theoretic and Machine Learning Tools. (Under the direction of DR. SYED ALI KHAYAM)

Anomaly detection systems (ADSs) were proposed more than two decades ago and since then considerable research efforts have been vested in designing and evaluating these systems. However, accuracy in terms of detection and false alarm rates, has been a major limiting factor in the widespread deployment of these systems. Hence, in this thesis we (i) Propose and evaluate information theoretic techniques to improve the performance of existing general-purpose anomaly detection systems; (ii) Design and evaluate a novel and specific-purpose machine learning-based anomaly detection solution for bot detection; (iii) Stochastically model general-purpose anomaly detection systems and show that these systems are inherently susceptible to parameter estimation attacks; and (iv) Propose novel design philosophies to combat these attacks.

To improve the performance of current general-purpose anomaly detection systems, we propose (i) a feature space slicing framework; and (ii) a multi-classifier ADS. The feature space slicing framework operates as a pre-processor, that segregates the feature instances at the *input* of an ADS. We provide statistical analysis of mixed traffic highlighting that there are two factors that limit the performance of current ADSs: high volume of benign features; and attack instances that exhibit strong similarity with benign feature instances. To mitigate these accuracy limiting factors, we propose a statistical information theoretic framework that segregates the ADS feature space

into multiple subspaces before anomaly detection. Thorough evaluations on real-world traffic datasets show that considerable performance improvements can be achieved by judiciously segregating feature instances at the input of a general-purpose ADS. The multi-classifier ADS, on the other hand, defines a standard deviation normalized entropy-of-accuracy based post-processor that judiciously combines *outputs* of diverse general-purpose anomaly detection classifiers, thus building on their strengths and mitigating their weaknesses. Evaluations on diverse datasets show that the proposed technique provides significant improvements over existing techniques.

During the course of this research, the threat landscape changed considerably with botnets emerging as the most potent threat. However, existing general-purpose anomaly detection systems are largely ineffective in detecting this evolving threat because botnets are distinctively different from their predecessors. Since botnets follow a somewhat invariant lifecycle, instead of pure behavior-based solutions, current bot detection tools employ the bot lifecycle for detection. However, these specific-purpose tools use rigid rule-based detection logic that falls short of providing acceptable accuracy with evolving botnet behavior [1]. Extending the design philosophy of this thesis, we propose a post-processing detection logic, for specific-purpose bot detection. The proposed post-processor models the high level bot lifecycle as a Bayesian network. Experimental evaluations on diverse real-world botnet traffic datasets show that the use of Bayesian inference based post-processor provides considerable performance improvements over existing approaches.

Lastly, we stochastically model a few existing general-purpose anomaly detection systems and demonstrate that these systems are highly susceptible to parameter es-

timation attacks. Since current day malware is becoming increasingly stealthy and difficult to mine in overwhelming volumes of benign traffic, we argue that anomaly detection systems need to be significantly redesigned to cope with the evolving threat landscape. To this end, we propose *cryptographically-inspired* and *moving target* based ADS design philosophies. The crypto-inspired ADS design aims at randomizing the learnt normal network profile while the moving target-based ADS design randomizes the feature space employed by an ADS for anomaly detection. We provide some preliminary evaluations that show that randomizing ADS parameters greatly improves the robustness of anomaly detection systems against parameter estimation attacks.

To my parents for letting me pursue my dreams for so long,

to Dr. Khayam for being an amazing mentor

&

to my son Aly

ACKNOWLEDGEMENTS

"All the praises and thanks be to Allah, Who has guided us to this, never could we have found guidance, were it not that Allah had guided us!" Al-A'raf (7:43)

I thank Almighty Allah for keeping me steadfast all these years. This thesis has been possible because of His blessings.

I would also like to thank my parents for always believing in me and supporting me, even though sometimes I was more ambitious than I should have been. Thank you for taking such great care of Aly so that I could work. You're the world's best grandparents and we love you.

I would also like to offer my gratitude to Qasim Ali for being a wonderful friend in one of the most testing times of my life. Thank you for being you.

Finally and most importantly I would like to thank my advisor, Dr. Syed Ali Khayam. I consider myself extremely fortunate to have had the chance to work under his supervision. He is one of the most inspiring people I've come across in my life and I've learnt a lot from him. His guidance and supervision provided me understanding of not only how to do great research but also of work-ethics, morality and the value of family. I find myself at a lack of words to offer my gratitude. He was a father when I needed support, a brother when I needed comfort, a friend when I needed a break, a mentor when I needed guidance and a critic when I needed correction. Thank you for being such a wonderful person and such an amazing advisor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Malware, botnets, spam, phishing and denial of service attacks have become continuous and imminent threats for today's networks and hosts [4], [5], and are growing in their number and sophistication. Financial losses due to these malware attacks are in the orders of billions of dollars. For example, the economic losses to recover from the CodeRed worm alone were estimated at $2.6 Billion [6], [7], while that for Conficker are estimated to be as high as $9.1 Billion [8] with 15 million infected machines worldwide [9]. Moreover, self-propagation and/or intelligent social engineering methods (e.g., social networks, blogs, websites, etc.) allow malware to spread at astonishing rates [9], [10], [11], [12], [13], [14]; the slammer worm covered the Internet in minutes [11] infecting 1 million hosts in less than 1 second [15]. Moreover, according to recent study [16], there was a soaring 392% increase in the number of malware infections experienced by commercial organizations in 2012 as compared to 2011. Hence, as the detection methodologies have refined over the years, so have the attackers with stealthy malware propagation [13]. Thus, in addition to the exponentially increasing volume and impact of these new malicious code threats, the stealthiness, sophistication and impact of malware attacks are increasing at an alarming pace.

While the original models for intrusion detection system were proposed more than two decades ago [17], intrusion detection still remains an active area of research as

the attacks continue to adapt, and evade intrusion detection solutions [18]. Intrusion detection systems comprise two types of detection methods: signature-based detection and anomaly detection. Signature-based detection or misuse detection detects anomalies/malware based on predefined signatures that need to be constantly updated [19], [20]. These detectors, while are valuable in detecting known malware, have an inherent limitation of not being able to detect zero-day attacks (i.e. attacks for which no signatures have yet been established [21].). It might take days before a zero-day attack is identified and a signature is developed and distributed [22]. According to a recent research [23], network breaches go undetected for around 140 days before they are detected and consequently defenses are put in place. Anomaly detectors, on the other hand, learn the normal behavior of the network. Any significant variations from this learnt normal behavior is termed anomalous. Hence, these systems are able to detect zero-day attacks by flagging deviations from normality. These anomaly detection systems, however, suffer from low accuracies [24], [25], [26] (low detection rates and high false alarms) and susceptibility to evasion attacks [27], [28] (disseminating malicious traffic in the network while staying below the threshold and hence go undetected). Thus there is always a tradeoff between zero-day attack detection (i.e. anomaly detection) and incurring low false positive rates (i.e. signature-based detection). An ideal anomaly detection system would have both these characteristics.

In this thesis we treat an ADS as a black box and hence do not propose or make any changes to its detection methodology. Rather in this research we propose pre- and post-processing methods that operate on the input/output parameters of an ADS with the aim to improve the performance of current general- and specific-purpose anomaly

detection systems. Towards the end of this thesis, we briefly touch upon the design philosophy of existing general-purpose anomaly detection systems to demonstrate that these systems are inherently susceptible to evasion attacks.

1.1 Thesis Statement

The thesis of this dissertation is that **The accuracy of current anomaly detection systems (ADSs) can be significantly improved by designing intelligent pre/post ADS processors using information theoretic and machine learning tools, such that the ADSs continue to operate as a black box**.

1.2 Motivation

The last decade has witnessed an unprecedented increase in the volume of new or zero-day network attacks [4], [29]. As the economic size of the network security black market – currently estimated at hundreds of billions of dollars per annum [30]– [31] – continues to grow, attacks are becoming alarmingly stealthy and sophisticated. Numerous general- and specific-purpose anomaly detection systems (ADSs) [32]– [33] have been proposed in the last few years to combat these rapidly evolving attacks. However, existing anomaly detection systems still fall short of achieving acceptable performance for commercial deployments. Hence, signature-based detection systems remain the de facto tools for detecting malware.

This motivated us to analyze and identify the accuracy limiting factors in existing general- and specific-purpose anomaly detection systems and subsequently propose

pre- and post-processing methods for accuracy improvements in current ADSs.

1.3 Overview of Contributions

Our aim in this thesis is to:

Objective 1 - Propose pre- and post-processing techniques to improve accuracies of existing general-purpose ADSs;

Objective 2 - Propose a novel specific-purpose ADS post-processor for bot detection; and finally

Objective 3 - Identify new guidelines to redesign general-purpose ADSs.

To achieve Objective-1, we propose techniques to improve the performance of current general-purpose anomaly detection systems. It is pertinent to state that in this work performance refers to the accuracy of an ADS. Hence the terms *performance* and *accuracy* are used synonymously throughout the thesis.

We first present thorough evaluations of representative ADSs to identify their strengths and weaknesses. We then propose two information theoretic techniques to improve the accuracy of current general-purpose ADSs: (i) a feature space slicing framework; and (ii) a multi-classifier ADS. These techniques can be used by any existing general-purpose ADS as a pre- or post- processing module respectively to achieve higher accuracies. It is pertinent to reiterate that our proposed methods do not alter the main functioning and methodology used by the ADS for anomaly detection. Our proposed methods work on the input and output parameters of an ADS, while the detector continues to operate as a black box.

The feature space slicing framework is used as a pre-processor to a general-purpose ADS. It slices the incoming traffic based on a similarity measure to mitigate the accuracy limiting factors of averaging-out (overwhelming volumes of benign features) and noise (attack features similar to benign features). Extensive evaluations on real-world traffic datasets shows that feature space slicing provides an improvement of up to 75% in detection rates and a decrease of up to 99% in false alarms.

The multi-classifier ADS employs a standard deviation normalized entropy-of-accuracy model to combine the outputs of diverse general-purpose anomaly detectors. It is employed as a post-processing module to current ADSs. We argue that it is inportant to consider not only the accuracies of individual ADSs but also the variations in their accuracies during combining. Comparison with other combining schemes shows that multi-classifier ADS provides approximately 12% increase in detection rates and a 3-40% decrease in false alarms.

To achieve Objective-2 we propose a novel ADS post-processor to detect the botnet malware. Botnets evolved as an imminent threat during the course of this research. With the immense resources acquired by these botnets over the last few years (in the form of millions of infected machines worldwide), there has been an unprecedented increase in targeted attacks launched from these botnets [34]. However, considering the havoc that these botnets have caused over the years, there have been very few deployable solutions for the detection of this evolving threat.

General-purpose anomaly detection systems are not well-suited to detect this evolving malware, as these systems are inherently designed to accumulate evidence which is then used to detect deviations with reference to a specific feature space. Botnets,

however, exhibit behavior that transits from one state space (e.g. inbound scan, exploit etc.) to another (bot binary download, attack etc.) and the use of a single feature space cannot ensure bot detection. Though an anomaly detection system may be able to detect a bot in some specific state(s) (i.e. detection of deviation across a specific feature space), more information is required to confirm bot infection/activity. To overcome this shortcoming in pure behavior based detection, current bot detectors employ the bot life cycle for bot detection. The anomaly detector now learns the malicious behavior rather than the normal behavior of the network. Since the detector still operates on the behavior of the malware, which is non-variant across all classes of the malware, hence, we classify these detectors under the umbrella of anomaly detection. Employing the bot behavior for detection reduces the false alarms of the ADS as well. However, current bot detection tools still fall short of providing acceptable accuracies with changing bot behavior.

We identify the key shortcomings in existing bot detectors and mitigate these shortcomings in a Bayesian network-based post-processor for bot detection. We propose Bottleneck, a system that models the lifecycle behavior of the bot as a Bayesian network, which is then used as a signature for bot detection. This has the following significant advantages: (i) bot lifecycle is a high level signature and hence can be used to detect current and evolving botnets; (ii) would yield better accuracy as compared to pure behavior-based detection; and (iii) Bayesian networks are self learning and hence would be able to adapt with changing bot behavior.

We collect diverse real-world traffic datasets to perform comparative analysis of existing bot detectors with Bottleneck. Bottleneck provides comparable performance

to existing rule-based detectors when trained and tested on similar datasets. However, when evaluated on changing bot behavior, Bottleneck provides significantly better performance as compared to existing detectors. For example, the detection rate improved from 30% (provided by a leading bot detector, BotHunter [35]) to 90% (provided by Bottleneck) with only 1 additional false positive, when evaluated on the same set of events.

Finally, we observed that the security of existing general-purpose ADSs rely on the secrecy of their design and/or the features employed for detection. If the features and/or the design principle is known, these systems can be easily by-passed by an intelligent attacker employing stealthy malware. In cryptographic systems, security relies in the secret key and not the algorithm itself. This results in systems that are robust against evasion attacks due to the randomness introduced by the secret key having a large sample space. We argue that if we can incorporate similar randomness in anomaly detection design, the resultant solution would be robust against parameter estimation attacks. Hence to achieve Objective-3, we stochastically model current general-purpose ADSs and highlight that the design of these systems are susceptible to evasion attacks. We propose to redesign general-purpose ADSs and provide some novel ADS design philosophies. We first stochastically model a few representative ADSs and experimentally show that an attacker can estimate the network traffic features and detection thresholds. The attacker can hence induce stealthy malware traffic while staying below the detection threshold and hence passing undetected through an ADS. We propose a cryptographically-inspired ADS design that randomizes the benign distribution to ensure that the attacker cannot correctly estimate the baseline

distribution and hence cannot develop reliable estimates of the network parameters required for successful evasion. We also propose a moving target-based ADS design that randomizes the feature space of an ADS across time. We provide some preliminary evaluations showing that it becomes computationally infeasible for the attacker to launch evasion attacks when the ADS parameters are randomized.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2. provides a brief background and taxonomy of anomaly detection systems. It also provides detailed description of current general-purpose ADSs and datasets used in this thesis. It also presents some preliminary Receiver Operating Curve(ROC)-based evaluations for understanding existing general-purpose ADSs and highlighting their strengths and weaknesses.

Chapter 3. presents a feature space slicing framework that is proposed as a preprocessing method to improve the accuracy of current general-purpose ADSs. The proposed method is evaluated on diverse general-purpose ADSs and datasets.

Chapter 4. provides another information theoretic technique, the multi-classifier ADS design, to improve the accuracy of existing general-purpose ADSs. It also provides comparative evaluation of the multi-classifier ADS with existing combining techniques. Thorough evaluations on multiple diverse datasets are also presented.

Chapter 5. presents our proposed Bayesian network-based bot detection framework. It provides a detailed comparison with existing bot detection tools and shows

that our proposed post-processing bot detection method induces considerable performance improvements with changing bot behavior.

Chapter 6. stochastically models general-purpose anomaly detection systems and list the future directions of our work.

Chapter 7. summarizes the key conclusions of this thesis.

CHAPTER 2: BACKGROUND, ANOMALY DETECTION SYSTEMS, DATASETS & PRELIMINARY EVALUATIONS

In this chapter we give an overview of (a) intrusion detection methods; (b) general-purpose and specific-purpose anomaly detection systems and datasets used throughout this thesis; and (c) some preliminary ROC evaluations.

2.1 IDS Detection Methods

In broad terms, the field of intrusion detection comprises two types of detection methods: misuse detection (also known as signature detection) and anomaly detection. Misuse detection, the predominant detection method employed in today's anti-virus software, requires a signature of an attack to be known before the attack can be detected. While such signature-based detectors can provide 100% detection rates for known attacks, they have an inherent limitation of not being able to detect new or previously-unseen attacks; a 468% increase in previously-unseen attacks was reported over just a six month period in 2007 [4]. Moreover, development and dissemination of attack signatures require human intervention and therefore misuse detectors are finding it difficult to cope with rapidly-evolving network intrusions. On the other end of the intrusion detection spectrum are Network-based Anomaly Detection Sys-

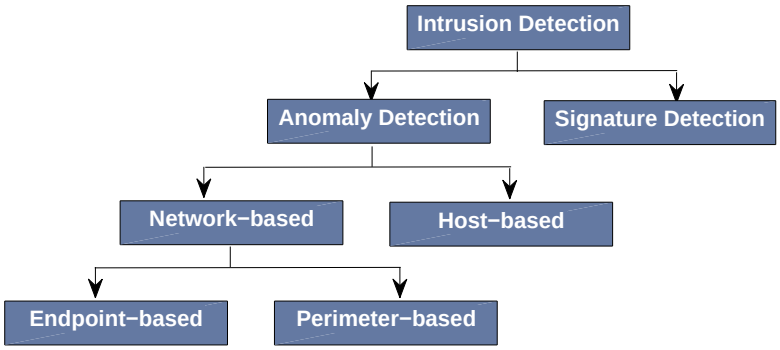


Figure 1: IDS Classification.

tems[1] (ADSs) which model the benign or normal traffic behavior of a network or host and detect significant deviations from this model to identify anomalies in network traffic. Since these ADSs rely on normal traffic behavior for attack detection, they have the capability to detect previously-unknown attacks. Consequently, significant research effort has been focused on development of network-based ADSs in the past few years [36][2].

Anomaly detection systems can further be categorized into either host based systems or network based systems [39]:

- *Network-based ADS:* Network based systems detect anomalies by analyzing unusual network traffic patterns

- *Host-based ADS:* Host-based systems detect anomalies by monitoring an endpoints operating system (OS) behavior, for instance by tracking OS audit logs, processes, command-lines or keystrokes

[1]This thesis explicitly deals with only network-based ADSs. Hence, ADS refers to network-based ADS throughout this thesis.

[2]Interestingly, the promise and advantages of anomaly detectors over signature detectors were identified by the seminal DARPA-funded IDS evaluation studies much before the CodeRed worm [37], [38].

This Intrusion detection classification is shown in Figure 1. These network-based IDSs can be either endpoint or perimeter based depending on the traffic analyzed for anomaly identification. In this research work we mainly focus on ***network-based anomaly detection systems***, however we do briefly touch upon a few interesting host based systems as well in Chapter 3. Hence, throughout this thesis, we refer to the network-based anomaly detection systems as merely anomaly detection systems.

2.1.1 Taxonomy of Anomaly Detection Systems

Figure 2 presents the taxonomy based on [2], which subdivides the evaluated anomaly detection algorithms into different categories based on detection principles employed by the ADS. The taxonomy also clearly lists (and highlights in blue) all anomaly detection systems that we have implemented and used throughout this thesis. Clearly, the ADSs used in this study are quite diverse in their detection frameworks and span across all branches of the ADS taxonomy proposed in [2]. These ADSs range from very simple rule modeling systems like PHAD [40] to very complex and theoretically-inclined self-learning systems like the PCA-based subspace method [3] and the sequential hypothesis testing technique [41]. Other than general-purpose ADSs, the taxonomy also includes specific-purpose anomaly detectors namely BotHunter [35] and BotFlex [42]. These are the only publically-available bot detection techniques.

We have also highlighted (in red), our proposed anomaly detection system *Bottleneck* that we propose for bot detection. It can be observed that it is a self-learning system that learns the normal behavior of a bot for anomaly detection.

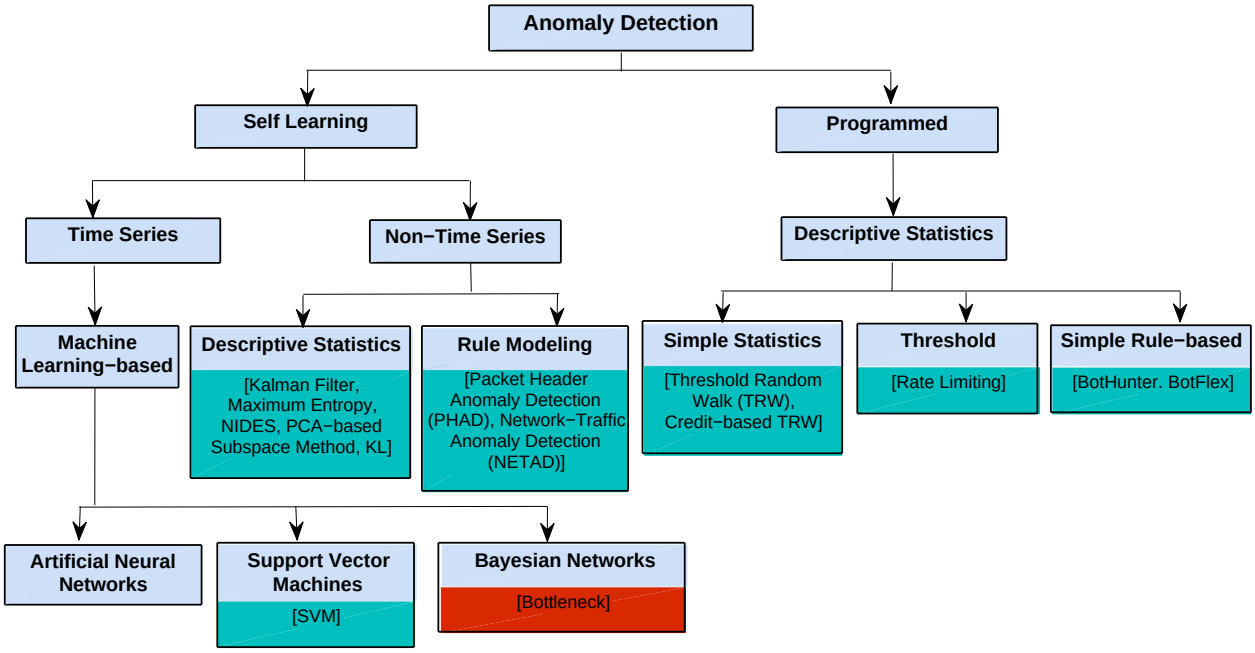


Figure 2: Taxonomy of the anomaly detection systems listing the detectors used in this work [2].

The following sections describe in detail the ADSs that we have used as well as the datasets used for evaluations throughout this thesis.

2.2 Anomaly Detection Systems

We provide brief descriptions of the evaluated algorithms. Primarily we will focus on the algorithm adaptation and parameter tuning for the datasets under consideration. For more details, readers are referred to the original papers: [32], [41], [43], [40], [44], [3], [45], [46], [47], [48], [33], [49], [42] and [35].

Parameters not mentioned in this section are the same as those described in the algorithms' respective papers. Implementations of the anomaly detection systems are available at [50].

2.2.1 Rate Limiting

Rate limiting [32], [51] detects anomalous connection behavior by relying on the premise that an infected host will try to connect to many different machines in a short period of time. Rate limiting detects portscans by putting new connections exceeding a certain threshold in a queue. An alarm is raised when the queue length, η_q, exceeds a threshold. ROCs for endpoints are generated by varying $\eta_q = \mu + k\sigma$, where μ and σ represent the sample mean and sample standard deviation of the connection rates in the training set, and $k = 0, 1, 2, \ldots$ is a positive integer. Large values of k will provide low false alarm and detection rates, while small values will render high false alarm and detection rates. In the LBNL dataset, connection rate variance in the background traffic is more than the variance in the attack traffic. Therefore, to obtain a range of detection and false alarm rates for the LBNL dataset, we use a threshold of $\eta_q = w\mu$, with a varying parameter $0 \geq w \leq 1$, and the queue is varied between 5 and 100 sessions.

2.2.2 Threshold Random Walk (TRW) Algorithm

The TRW algorithm [41] detects incoming portscans by noting that the probability of a connection attempt being a success should be much higher for a benign host than for a scanner. To leverage this observation, TRW uses sequential hypothesis testing (i.e., a likelihood ratio test) to classify whether or not a remote host is a scanner. We plot ROCs for this algorithm by setting different values of false alarm and detection rates and computing the likelihood ratio thresholds, η_0 and η_1, using the method

described in [41].

2.2.3 TRW with Credit-based Rate Limiting (TRW-CB)

A hybrid solution to leverage the complementary strengths of Rate Limiting and TRW was proposed by Schechter et al. [43]. Reverse TRW is an anomaly detector that limits the rate at which new connections are initiated by applying the sequential hypothesis testing in a reverse chronological order. A credit increase/decrease algorithm is used to slow down hosts that are experiencing unsuccessful connections. We plot ROCs for this technique for varying η_0 and η_1 as in the TRW case.

2.2.4 Maximum Entropy Method

This detector estimates the benign traffic distribution using maximum entropy estimation [47]. Training traffic is divided into $2,348$ packet classes and maximum entropy estimation is then used to develop a baseline benign distribution for each packet class. Packet class distributions observed in real-time windows are then compared with the baseline distribution using the Kullback-Leibler (KL) divergence measure. An alarm is raised if a packet class' KL divergence exceeds a threshold, η_k, more than h times in the last W windows of t seconds each. Thus the Maximum Entropy method incurs a detection delay of at least $h \times t$ seconds. ROCs are generated by varying η_k.

2.2.5 Packet Header Anomaly Detection (PHAD)

PHAD learns the normal range of values for all 33 fields in the Ethernet, IP, TCP, UDP and ICMP headers [40]. A score is assigned to each packet header field in the testing phase and the fields' scores are summed to obtain a packet's aggregate anomaly score. We evaluate PHAD-C32 [40] using the following packet header fields: source IP, destination IP, source port, destination port, protocol type and TCP flags. Normal intervals for the six fields are learned from 5 days of training data. In the test data, fields' values not falling in the learned intervals are flagged as suspect. Then the top n packet score values are termed as anomalous. The value of n is varied over a range to obtain ROC curves.

2.2.6 Network Traffic Anomaly Detection (NETAD)

NETAD detects anomalies in network packets [44]. It filters out all outgoing traffic and considers only the first few packets of the incoming server requests. For each packet, NETAD models the first 48 bytes starting with the IP header. Each byte is considered as a nominal attribute with 256 possible values. For packets with less than 48 bytes, the missing attributes are set to 0.

The packets are modeled into 9 different subsets based on 9 IP and TCP packet types such that a packet can belong to more then one subset. For each packet, NETAD computes the following score: $\sum tn_a \frac{(1-\frac{r}{256})}{r} + \frac{t_i}{f_i+\frac{r}{256}}$ where t is the time when the attribute was last anomalous, n_a is the number of training packets from the last anomaly to the end of the training period, r is the number of allowed values (i.e.

256), t_i is the time since the value i was last observed and f_i is the frequency of value i. And summation is performed over $9 \times 4 = 432$ combinations.

2.2.7 PCA-based Subspace Method

The subspace method uses Principal Component Analysis (PCA) to separate a link's traffic measurement space into useful subspaces for analysis, with each subspace representing either benign or anomalous traffic behavior [3]. The authors proposed to apply PCA for domain reduction of the Origin-Destination (OD) flows in three dimensions: number of bytes, packets, IP-level OD flows. The top k eigenvectors represent normal subspaces. It has been shown that most of the variance in a link's traffic is generally captured by 5 principal components [3]. A recent study showed that the detection rate of PCA varies with the level and method of aggregation [52]. It was also concluded in [52] that it may be impractical to run a PCA-based anomaly detector over data aggregated at the level of OD flows. We evaluate the subspace method using the number of TCP flows aggregated in 10 minutes intervals. To generate ROC results, we changed the number of normal subspace as $k = 1, 2, \ldots, 15$. Since the principal components capture maximum variance of the data, as we increase k, the dimension of the residual subspace reduces and fewer observations are available for detection. In other words, as more and more principal components are selected as normal subspaces, the detection and false alarm rates decrease proportionally. Since there is no clear detection threshold, we could not obtain the whole range of ROC values for the subspace method. Nevertheless, we evaluate and report the subspace method's accuracy results for varying number of principal components.

2.2.8 Kalman Filter based Detection

The Kalman filter based detector of [46] first filters out the normal traffic from the aggregate traffic, and then examines the residue for anomalies. In [46], the Kalman Filter operated on SNMP data to detect anomalies traversing multiple links. Since SNMP data was not available to us in either dataset, we model the traffic as a 2-D vector X_t. The first element of X_t is the total number of sessions (in the endpoint dataset) or packets (in the LBNL dataset), while the second element is the total number of distinct remote ports observed in the traffic. We defined a threshold, η_f on the residue value r to obtain ROC curves. Thresholding of r is identical to the rate limiting case. An alarm is raised, if $r < -\eta_f$ or $r > \eta_f$.

2.2.9 Next-Generation Intrusion Detection Expert System

NIDES [48] is a statistical anomaly detector that detects anomalies by comparing a long-term traffic rate profile against a short-term, real-time profile. An anomaly is reported if the Q distribution of the real-time profile deviates considerably from the long-term values. After specific intervals, new value of Q are generated by monitoring the new rates and compared against a predefined threshold, η_s. If $\Pr(Q > q) < \eta_s$, an alarm is raised. We vary η_s over a range of values for ROC evaluation.

2.2.10 KL Detector

The KL detector [45] maps the feature space into different classes based on their numbers to obtain a benign distribution p_X. This distribution is then compared with run-time distribution q_X using the KL divergence. The KL divergence measure

quantifies the extent of dissimilarity between two probability distribution. If the KL score exceeds a fixed threshold, it is classified as anomalous.

2.2.11 Support Vector Machine (SVM) on Bags of System Calls

Kang et al. propose a bag of system calls representation for detection of intrusive system call sequences [33]. During a conversion, the ordering information between system calls is lost and only the frequency of each input sequence is preserved. A feature is defined as an ordered list of the frequency of all the system calls in a given sequence. SVMs are then used for classification [53].

2.2.12 Host-based KL Detector

The KL detector [49] maps system calls into different classes based on their numbers to obtain a benign distribution p_X. This distribution is then compared with run-time distributions q_X using the KL divergence. The KL divergence measure quantifies the extent of dissimilarity between two probability distribution. If the KL score is more than a fixed threshold, it is classified as anomalous.

2.2.13 BotHunter

BotHunter [35] is a decision engine built over the Snort IDS. It modifies the Snort ruleset to detect events possibly indicative of a bot infection, and implements a rule-based correlation layer to evaluate whether a given host is infected. BotHunter is a closed source tool which is built on the model of a typical bot lifecycle to detect bot infections. It monitors the two-way communication between hosts in the monitored

network and the internet. All host activities are mapped against the bot lifecycle model. When enough evidence is acquired such that any one of the BotHunter defined conditions for bot declaration are met, the host is flagged as a bot. Following are the three conditions defined by Bothunter for bot declaration:

Condition 1: Evidence of a local host infection, and evidence of outward malware coordination or attack propagation.

Condition 2: At least two distinct signs of outward bot coordination, attack propagation, or attacker preparation sequences are observed.

Condition 3: Evidence that a local host has attempted to establish communication with a confirmed malware control host or drop site.

BotHunter does not provide any provision for threshold tuning.

2.2.14 BotFlex

BotFlex [42] implements a rule-based decision engine over the Bro IDS. It is built over the bot lifecycle model comprising scanning, host exploit, malicious binary download, Command and Control (C&C) communication and attack phases. BotFlex intercepts host input about various network activities from Bro and correlates the received information to declare botnet infection. The activity of each host is mapped against the defined conditions for bot declaration. The conditions are listed below:

Condition 1: Direct C&C/RBN blacklist match.

Condition 2: (Exploit OR Egg download) AND (C&C OR Attack)

Condition 3: (Egg download AND C&C) OR (C&C AND Attack) OR (Egg Download AND Attack)

Unlike BotHunter, BotFlex is an open source tool for bot detection. However, it also does not provide any provision for threshold tuning.

2.3 Datasets

We wanted to use real, labeled and public background and attack datasets. Real and labeled data allow realistic and repeatable quantification of an anomaly detector's accuracy, which is a main objective of this work.

The following datasets have been used in this work: a) LBNL dataset has been collected at the edge router of the Lawrence Berkeley National Laboratory (LBNL); b) Endpoint-based WiSNet lab dataset has been collected at network endpoints by our research lab; c) an ISP traffic dataset collected at the B-RAS of a leading Pakistani ISP, Nayatel [54]; d)SysNet lab dataset comprises network traffic collected from a well administered lab network; and e) Host-based system call dataset [55] collected at The University of New Mexico (UNM). The LBNL, Endpoint and UNM datasets are used for evaluations in Chapter 3 and 5 while the ISP and the SysNet lab datasets are bot traffic datasets and used for evaluations in Chapter 4.

The following sections provide some description of these datasets.

2.3.1 The ISP Dataset

Pakistan represents an effective botnet traffic source, as increased Internet penetration and low computer security awareness have resulted in Pakistan becoming one of the most infected countries worldwide [56]. The dataset constitutes 500 GB trace col-

lected at the B-RAS of a leading Pakistani ISP, Nayatel [54]. This data was collected at a link with an average data rate of 19 MB per second ($\approx$ 28K packets per second), and with 80/tcp being the most common port, followed by 443/tcp. The trace contained 48,606 unique IP addresses of which 381 IP addresses represented Nayatel's local network, that is they were statically assigned to mid-to-small size enterprises. While we assume that an IP address represents a single machine, we acknowledge that the ISP customers are likely using their public IPs to represent multiple private (NATed) hosts.

We obtained the groundtruth for this dataset using a one-time sample of a reputation list maintained by Team Cymru [57], which maintains sensors distributed around the globe to monitor active C&C servers. We synchronized our data collection for 7 hours on September 18, 2012, to this hourly updated list of active C&C servers. We flag as bots all the hosts in the trace that perform bidirectional communication with an IP in that C&C list.

The trace consists of traffic from 381 static IP addresses, of which 108 we declare as bots according to the groundtruth, while we declare the remaining 273 as benign. As already mentioned, many of these static IP users are hosting multiple machines behind a NAT server. Hence each IP may in fact represent multiple hosts.

The groundtruth provided also tells us the bot family that each C&C server is associated with; Table 1 presents a brief description of each of these families along with the number of hosts infected by each family. It can be seen that the trace contains at least 12 known malware families, comprising both IRC- and HTTP-based botnets, serving a wide range of criminal purposes ranging from data stealing to large

Table 1: Description of bot families in Nayatel dataset and number of machines in the trace infected by each.

Bot Family	Infected IPs	Possible infection vectors	C&C Type	Description
blackenergy	8	Distributed via drive by download or spam (no self propagation)	HTTP-based centralized	DDoS, information stealing and malware distribution capability
dirtjumper	3	Wide variety – exploit, drive by download or distributed by malware	HTTP-based	DDoS toolkit with large number of variants (e.g. Pandora and Di botnet spawned from it)
graybird	2	Various Windows exploits depending on version; e.g. Hupigon uses IE vulnerability when compromised website accessed	IRC-based or custom protocol; centralized	Spyware Trojan
gumblar	59	Various exploits, especially Acrobat or Flash from websites hosting infected Javascript;	Centralized	Information stealing and delivering other malware; uses stolen FTP credentials to infect websites
haxdoor	8	Bundled download through other applications, or various browser exploits triggered from spam or blog links	IRC-based	Keylogging Trojan to steal banking information
hitpop	1	Often downloaded by Trojans as payload bundled with other malware	HTTP-based	DDoS
mpack	15	Varies widely	Varies widely	Malicious toolkit from which a variety of botnets are spawned (e.g. Srizbi botnet; generally hosted on infected websites
optima	4	Drive by download among other possibilities	Centralized and HTTP-based	DDoS
pincher	21	Trojan possibly distributed as bundled download	Runs C&C server on TCP port 81	Information stealing; also possibly used as malware dropper (installs other malicious files)
torpig	1	Drive by download; exploits Adobe/Java vulnerabilities and installs Mebroot rootkit	Centralized HTTP-based	Man in the browser phishing attacks for financial data stealing
tsunami	4	Not associated with any particular infection vector	IRC based	Trojan with DDoS capability
zonebac	27	No particular vector; drive by download possible	Nothing known	Backdoor Trojan; lowers browser security settings; uploads sensitive information about host; could install additional malware

scale DDoS attacks. This dataset therefore represents a comprehensive sample of botnet traffic, covering a variety of infection and attack vectors, merged with benign traffic.

2.3.2 SysNet Lab Dataset

We acknowledge that evaluation on the Nayatel dataset is inherently reliant on the accuracy of the groundtruth. This is an unavoidable limitation on any data that is collected in an uncontrolled environment. To ensure that this limitation does not bias our performance evaluation, we also collected a controlled dataset where we had complete confidence in the groundtruth.

Moreover, the Nayatel trace while large in terms of traffic volume and number of hosts, is temporally rather limited, covering only 8 hrs of traffic. Bot behavior often shows temporal variations which might affect detection accuracy as bots spread out their activity over time to remain inconspicuous. To overcome this limitation, we collected another dataset at SysNet lab with captured traffic spanning over a period of 24 hours. We captured benign user traffic from a small, well-administered lab network over the 24 hour period. This trace was collected at the network edge over the course of a regular working day with 22 users present in the lab and comprised complete incoming and outgoing packets as well as local network traffic.

To generate malicious traffic, we use the following 10 bot binaries shared by ICIR at UC Berkeley [58]: four variants of Pushdo, two variants of Sality, Kolabc, Virut, Dorkbot, and Bobax. The trace, hence, includes HTTP-based, IRC-based and P2P based bots, covering a range of malicious purposes such as spam, DDoS and information stealing. Therefore the dataset makes for a representative sample of bot traffic. A description of each of these bots is given in Table 2. We ran each of the binaries in a separate virtual environment (Windows XP VM) and allowed the infected virtual

Table 2: Description of bot binaries in SysNet Lab dataset.

Bot	**Infection vector**	**C&C type**	**Purpose**
Dorkbot	Spreads via links on social networks/IM, or self-propagates through copying itself to removable drives or sending Skype messages to contacts	IRC-based C&C	Password stealing; limited DDoS ability
Kolabc	Propagates via vulnerability exploit on Windows	Centralized IRC-based	unknown
Virut	Spreads by copying itself to executable files	Centralized; IRC-based; might use DGA for locating C&C	Used for a wide variety of criminal activities:DDoS, spam, information theft etc.
Pushdo	Spam email links lead to drive by download	Centralized employing fast flux	Spam, phishing attacks and information stealing
Sality	Replicates itself across network shares by infecting executable files	P2P-based	Used for spam, proxying and distributing intensive tasks like password cracking
Bobax	Spreads via sending spam links to itself to victim's contacts and outbound scanning for vulnerable ports	Centralized; HTTP-based	Spam

machines to communicate with their C&C servers. The traffic from the VMs was captured by running Wireshark in the host machine. The benign and the malicious traces were then merged and synchronized to generate a single 24hr dataset containing 22 benign and 10 malicious hosts.

2.3.3 The LBNL Dataset

This dataset was obtained from two international network locations at the Lawrence Berkeley National Laboratory (LBNL) in USA. Traffic in this dataset comprises packet-level incoming, outgoing and internally-routed traffic streams at the LBNL edge routers. Traffic was anonymized using the `tcpmkpub` tool; refer to [59] for details of anonymization.

Table 3: Background Traffic Information for the LBNL Dataset

Date	Duration(mins)	LBNL Hosts	Remote Hosts	Backgnd Rate(pkt/sec)	Attack Rate(pkt/sec)
10/4/04	10min	4,767	4,342	8.47	0.41
12/15/04	60min	5,761	10,478	3.5	0.061
12/16/04	60min	5,210	7,138	243.83	72

2.3.3.1 LBNL Background Traffic:

LBNL data used in this study is collected during three distinct time periods. Some pertinent statistics of the background traffic are given in Table 3. The average remote session rate (i.e., sessions from distinct non-LBNL hosts) is approximately 4 sessions per second. The total TCP and UDP background traffic rate in packets per second is shown in column 5 of the table. A large variance can be observed in the background traffic rate at different dates. This variance will have an impact on the performance of volumetric anomaly detectors that rely on detecting bursts of normal and malicious traffic.

The main applications observed in internal and external traffic are Web (HTTP), Email and Name Services. Some other applications like Windows Services, Network File Services and Backup were being used by internal hosts; details of each service, information of each service's packets and other relevant description are provided in [60].

2.3.3.2 LBNL Attack Traffic:

Attack traffic was isolated by identifying scans in the aggregate traffic traces. Scans were identified by flagging those hosts which unsuccessfully probed more than 20 hosts, out of which 16 hosts were probed in ascending or descending order [59]. Mali-

cious traffic mostly comprises failed incoming TCP SYN requests; i.e., TCP portscans targeted towards LBNL hosts. However, there are also some outgoing TCP scans in the dataset. Most of the UDP traffic observed in the data (incoming and outgoing) comprises successful connections; i.e., host replies are received for the UDP flows. Table 3 [column 6] shows the attack rate observed in the LBNL dataset. Clearly, the attack rate is significantly lower than the background traffic rate. Thus these attacks can be considered low rate relative to the background traffic rate. (We show later that background and attack traffic at endpoints exhibit the opposite characteristics.)

Since most of the anomaly detectors used in this study operate on TCP, UDP and/or IP packet features, to maintain fairness we filtered the background data to retain only TCP and UDP traffic. Moreover, since most of the scanners were located outside the LBNL network, to remove any bias we filter out internally-routed traffic. After filtering the datasets, we merged all the background traffic data at different days and ports. Synchronized malicious data chunks were then inserted in the merged background traffic.

2.3.4 Endpoint Dataset

Since no publicly-available endpoint traffic set was available, we spent up to 14 months in collecting our own dataset on a diverse set of 13 endpoints. Complexity and privacy were two main reservations of the participants of the endpoint data collection study. To address these reservations, we developed a custom tool for endpoint data collection. This tool was a multi-threaded MS Windows application developed using the `Winpcap` API [61]. (Implementation of the tool is available at [50].) To reduce

Table 4: Background Traffic Information for Four Endpoints with High and Low Rates

Endpoint ID	Endpoint Type	Duration(months)	Total Sessions	Mean Session Rate(/sec)
3	Home	3	373,009	1.92
4	Home	2	444,345	5.28
6	Univ	9	60,979	0.19
10	Univ	13	152,048	0.21

the packet logging complexity at the endpoints, we only logged some very elementary session-level information of TCP and UDP packets. Here a *session* corresponds to a bidirectional communication between two IP addresses; communication between the same IP address on different ports is considered part of the same network session. To ensure user privacy, the source IP address (which was fixed/static for a given host) is not logged, and each session entry is indexed by a one-way hash of the destination IP with the hostname. Most of the detectors evaluated in this work can operate with this level of data granularity.

Statistics of the two highest rate and the two lowest rate endpoints are listed in Table 4[3]. As can be intuitively argued, the traffic rates observed at the endpoints are much lower than those at the LBNL router. In the endpoint context, we observed that home computers generate significantly higher traffic volumes than office and university computers because: 1) they are generally shared between multiple users, and 2) they run peer-to-peer and multimedia applications. The large traffic volumes of home computers are also evident from their high mean number of sessions per second. For this study, we use 6 weeks of endpoint traffic data for training and testing. Results for longer time periods were qualitatively similar.

[3]The mean session rates in Table 4 are computed using time-windows containing one or more new sessions. Therefore, dividing total sessions by the duration does not yield the session rate of column 5.

To generate attack traffic, we infected VMs on the endpoints by the following malware: `Zotob.G`, `Forbot-FU`, `Sdbot-AFR`, `Dloader-NY`, `SoBig.E@mm`, `MyDoom.A@mm`, `Blaster`, `Rbot-AQJ`, and `RBOT.CCC`; details of the malware can be found at [62]. These malware have diverse scanning rates and attack ports/applications. Table 5 shows statistics of the highest and lowest scan rate worms; `Dloader-NY` has the highest scan rate of 46.84 scans per second (sps), while `MyDoom-A` has the lowest scan rate of 0.14 sps, respectively. For completeness, we also simulated three additional worms that are somewhat different from the ones described above, namely `Witty`, `CodeRedv2` and a fictitious TCP worm with a fixed and unusual source port. `Witty` and `CodeRedv2` were simulated using the scan rates, pseudocode and parameters given in research and commercial literature [62], [63].

2.3.4.1 Endpoint Background Traffic:

The users of these endpoints included home users, research students, and technical/administrative staff. Some endpoints, in particular home computers, were shared among multiple users. The endpoints used in this study were running different types of applications, including peer-to-peer file sharing software, online multimedia applications, network games, SQL/SAS clients etc.

2.3.4.2 Endpoint Attack Traffic:

The attack traffic logged at the endpoints mostly comprises outgoing portscans. Note that this is the opposite of the LBNL dataset, in which most of the attack traffic is inbound. Moreover, the attack traffic rates (Table 5) in the endpoint case are

Table 5: Endpoint Attack Traffic for Two High- and Two Low-rate Worms

Malware	Release Date	Avg. Scan Rate(/sec)	Port(s) Used
`Dloader-NY`	Jul 2005	46.84 sps	TCP 135,139
`Forbot-FU`	Sept 2005	32.53 sps	TCP 445
`MyDoom-A`	Jan 2006	0.14 sps	TCP 3127 – 3198
`Rbot-AQJ`	Oct 2005	0.68 sps	TCP 139,769

Table 6: UNM Intrusion Instances

sunsendmailcp (sscp)	decode	forwarding loops
3	2	5

generally much higher than the background traffic rates (Table 4). This characteristic is also the opposite of what was observed in the LBNL dataset. This diversity in attack direction and rates provides us a sound basis for performance comparison of the anomaly detectors evaluated in this study [41], [43].

For each malware, attack traffic of 15 minutes duration was inserted in the background traffic of each endpoint at a random time instance. This operation was repeated to insert 100 non-overlapping attacks of each worm inside each endpoint's background traffic.

2.3.5 Host-based System Calls Datasets

The University of New Mexico (UNM) dataset [55] provides system call traces for various processes. Forrest et al. [64] argue that monitoring the behavior of a process might not cover the full spectrum of normal behavior as some processes behave in quite a varied manner. Therefore, they artificially generated system call sequences. We used the synthetic sendmail traces for our experiments. These traces were generated by enumerating potential sources of variation for normal sendmail operations. Trace files contained process IDs and their respective system calls. Table 6 shows the intrusions

and their instances used; see [55] for details.

2.4 Preliminary Evaluations

In this section, we provide some preliminary evaluations of general- and specific-purpose anomaly detectors described in Section 2.2 on the Endpoint, Router and ISP datasets.

2.4.1 Averaged ROCs for the Endpoint Dataset

Figure 3 provides the averaged ROC analysis of numerous anomaly detection schemes used extensively in research literature. Clearly, the Maximum Entropy detector provides the highest accuracy by achieving near 100% detection rate at a very low false alarm rate of approximately 5 alarms/day. The Maximum Entropy detector is followed closely by the credit-based TRW approach. TRW-CB achieves nearly 90% detection rate at a reasonable false alarm rate of approximately 5 alarms/day. The original TRW algorithm, however, provides very low detection rates for the endpoint dataset. Results of these three schemes are shown more clearly in Figure 4(a). Based on these results, the Maximum Entropy algorithm provides the best accuracy on endpoints, while TRW provides the best detection on LBNL dataset.

The Kalman Filter approach is also quite accurate as it provides up to 85% detection rates at a reasonably low false alarm cost. Rate Limiting, although designed to detect outgoing scanning attacks, provides very poor performance. This result substantiates the results of [24] where very high false positive rates for high detec-

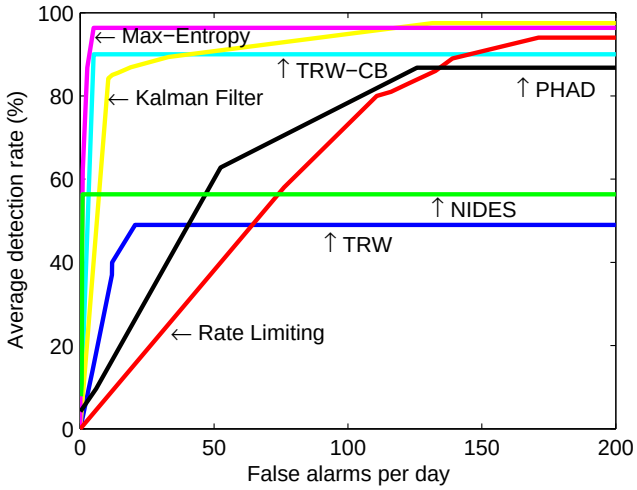
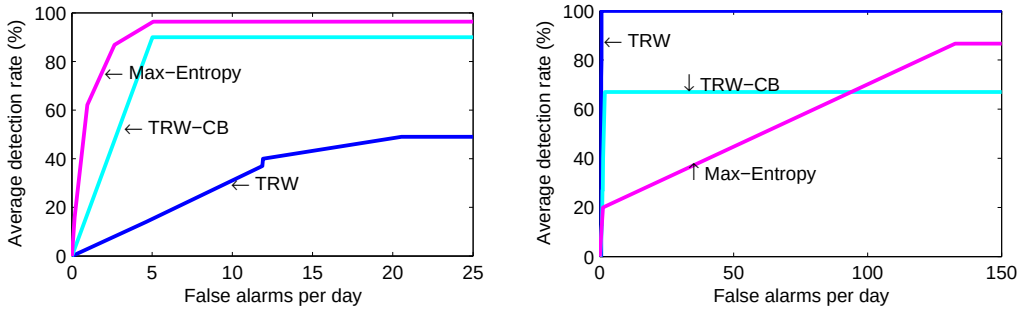


Figure 3: ROC analysis on the endpoint dataset; each ROC is averaged over 13 endpoints with 12 attacks per endpoint and 100 instances per attack.



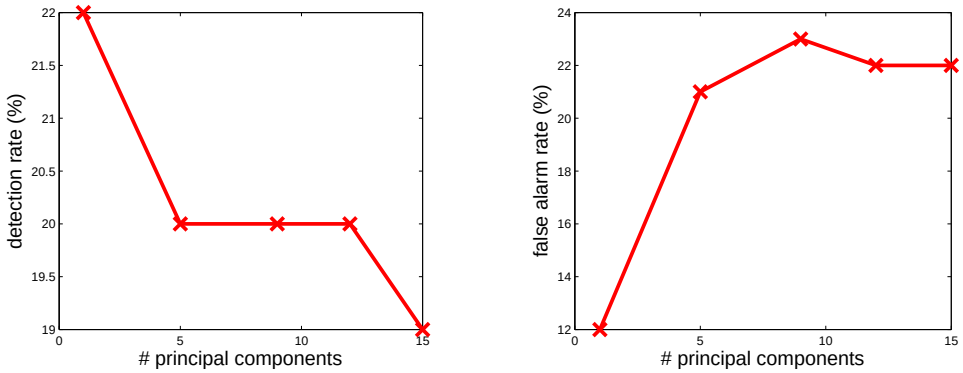
(a) Endpoint dataset

(b) LBNL dataset

Figure 4: Comparison of the Maximum Entropy, TRW and TRW-CB algorithms.

tion rates were reported for classical rate limiting. Hence, we also deduce that rate limiting is ineffective for portscan detection at endpoints.

PHAD does not perform well on the endpoint data set. The detection is accompanied with very high false alarm rates. NIDES achieve reasonable detection rates at very low false alarm rates, but is unable to substantially improve its detection rates afterwards. PHAD relies on previously seen values in the training dataset for anomaly detection. Therefore, if a scanner attacks a commonly-used port/IP then PHAD is unable to detect it. On similar grounds, if the malicious traffic is not bursty



(a) Detection rate (b) False alarm rate

Figure 5: Detection and false alarm rates for the subspace method [3].

enough as compared to background traffic then NIDES will not detect it, irrespective of how much the detection threshold is tuned.

Due to the thresholding difficulties for the subspace method explained in Section 2.2, in Figure 14 we report results for this technique for varying values of selected principal components. The highest detection rate of 22% is observed at $k = 2$ principal components. This already low detection rate decreases further at $k = 5$ and drops to 0% at $k = 15$. False alarm rates show the opposite trend. Thus the subspace method fails to give acceptable accuracy on the endpoint dataset.

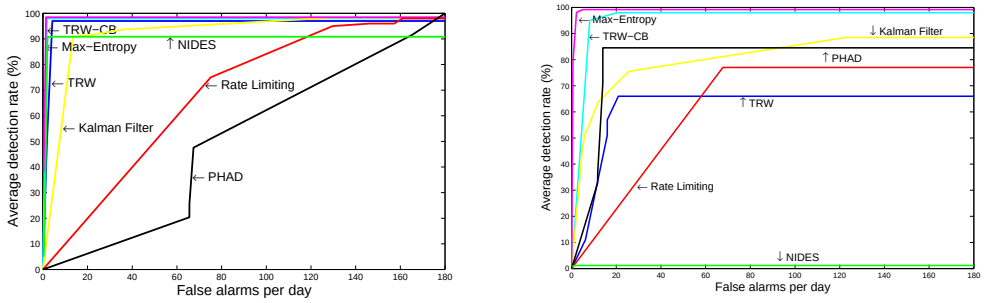
The ROC results for the endpoint dataset are somewhat surprising because two of the top three detectors are general-purpose anomaly detectors (Maximum Entropy and Kalman Filter), but still outperform other detectors designed specifically for portscan detection, such as the TRW and the Rate Limiting detectors. We, however, note that this analysis is not entirely fair to the TRW algorithm because TRW was designed to detect incoming portscans, whereas our endpoint attack traffic contains mostly outgoing scan packets. The credit-based variant of TRW achieves high ac-

curacy because it leverages outgoing scans for portscan detection. Thus TRW-CB combines the complementary strengths of rate limiting and TRW to provide a practical and accurate portscan detector for endpoints. This result agrees with earlier results in [24].

2.4.1.1 ROCs for Low- and High-Rate Endpoint Attacks

To evaluate the scalability of the ADSs under high- and low-rate attack scenarios, Figure 6 plots the ROCs for the highest rate (`Dloader-NY`) and lowest rate (`MyDoom-A`) attacks in the endpoint dataset. It can be observed that for the high-rate attack [Figure 6(a)] Maximum Entropy, TRW, TRW-CB and Kalman Filter techniques provide excellent accuracy by achieving 100% or near-100% detection rates with few false alarms. NIDES' performance also improves as it achieves approximately 90% detection rate at very low false alarm rates. This is because the high-rate attack packets form bursts of malicious traffic that NIDES is tuned to detect. Rate Limiting and PHAD do not perform well even under high attack rate scenarios.

Figure 6(b) shows that the accuracies of all detectors except PHAD and Maximum Entropy degrade under a low-rate attack scenario. Maximum Entropy achieves 100% detection rate with false alarm rate of 4-5 alarms/day. TRW-CB recovers quickly and achieves a near-100% detection rate for a daily false alarm rate around 10 alarms/day. NIDES, however, shows the biggest degradation in accuracy as its detection rate drops by approximately 90%. This is because low-rate attack traffic when mixed with normal traffic does not result in long attack bursts. TRW's accuracy is also affected significantly as its detection rate drops by about 35% as compared to the



(a) `Dloader-NY`, high scan rate (b) `MyDoom-A`, low scan rate

Figure 6: ROC curves for the lowest and highest rate attack in the endpoint dataset; results averaged over 12 endpoints with 100 instances of each attack.

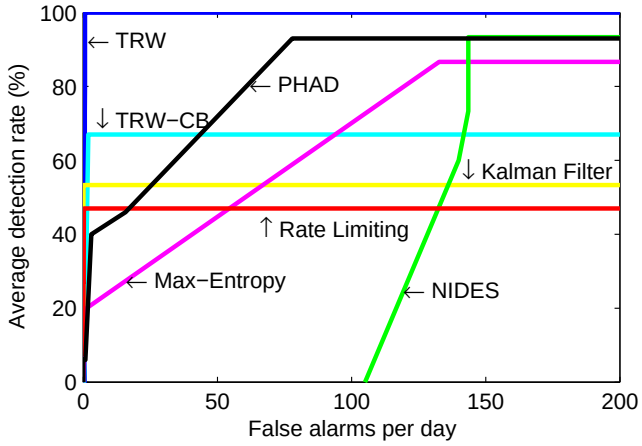


Figure 7: ROC analysis on the LBNL dataset.

high-rate attack. PHAD does not rely on traffic rate for detection, and hence its accuracy is only dependent on the header values observed during training.

2.4.2 Averaged ROCs for the LBNL Dataset

Figure 7 shows the ROCs for the LBNL dataset. Comparison with Figure 4 (a) and (b) reveals that the Maximum Entropy detector is unable to maintain its high accuracy on the LBNL dataset; i.e., the Maximum Entropy algorithm cannot scale to different points of network deployment. TRW's performance improves significantly as it provides a 100% detection rate at a negligible false alarm cost. TRW-CB, on

the other hand, achieves a detection rate of approximately 70%. Thus contrary to the endpoint dataset, the original TRW algorithm easily outperforms the TRW-CB algorithm on LBNL traces. As explained in Section 2.3, the LBNL attack traffic mostly comprises failed incoming TCP connection requests. TRW's forward sequential hypothesis based portscan detection algorithm is designed to detect such failed incoming connections, and therefore it provides high detection rates. Thus on an edge router, TRW represents a viable deployment option.

Kalman Filter detector's accuracy drops as it is unable to achieve a detection rate above 60%. PHAD provides very high detection rates, albeit at an unacceptable false alarm rate. Other detectors' results are similar to the endpoint case. (Results for the subspace method were similar to those reported earlier and are skipped for brevity.) It can be observed from Figure 7 that all algorithms except TRW fail to achieve 100% detection rates on the LBNL dataset. This is because these algorithms inherently rely on the high burstiness and volumes of attack traffic. In the LBNL dataset, the attack traffic rate is much lower than the background traffic rate. Consequently, the attack traffic is distributed across multiple time windows, with each window containing very few attack packets. Such low density of attack traffic in the evaluated time-windows remains undetected regardless of how much the detection thresholds are decreased.

2.4.3 Delay Comparison

Table 7 provides the detection delay for each anomaly detector. On the endpoint dataset, delay is reported for the highest and the lowest rate attacks, while on the LBNL dataset this delay is computed for the first attack that is detected by an

Table 7: Detection Delay of the Anomaly Detectors

	Rate Limiting	TRW	TRW-CB	Max Entropy	NIDES	PHAD	Subspace Method	Kalman Filter
`MyDoom` (msec)	310	510	40	215000	∞	900	79	377
`Dloader-NY` (msec)	140	320	20	56000	0.086	990	23	417
LBNL (msec)	660	660	290	86000	330	330	∞	800

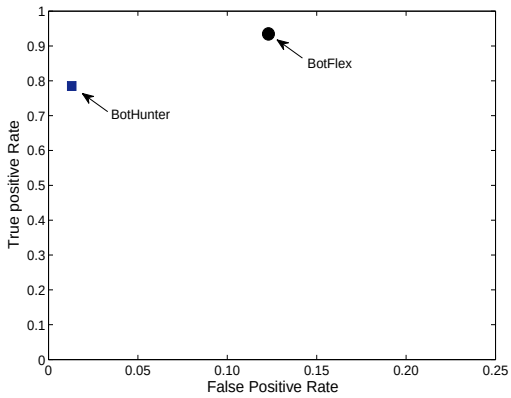


Figure 8: Accuracy evaluation of BotHunter and BotFlex on the Nayatel ISP dataset.

anomaly detector. A delay value of ∞ is listed if an attack is not detected altogether. It can be observed that detection delay is reasonable (less than 1 second) for all the anomaly detectors except the Maximum Entropy detector which incurs very high detection delays. High delays are observed for the Maximum Entropy detector because it waits for perturbations in multiple time windows before raising an alarm. Among other viable alternatives, TRW-CB provides the lowest detection delays for all three experiments. Detection delay for the TRW is also reasonably low.

2.4.4 Averaged ROCs for the Nayatel Dataset

We evaluate the overall accuracy of BotHunter and BotFlex over the Nayatel dataset. Figure 8 provides ROC-based accuracy evaluation. Since BotHunter and BotFlex do not provide any provision for threshold tuning, accuracy results of these

rule-based systems are points in the ROC space. It can be observed that BotHunter and Botflex provide good accuracies ($> 75\%$ TP and $5-6.5\%$ FP) on the ISP dataset. This is mainly because the bot traffic in the dataset triggered a large number of bot lifecycle events, which facilitated the decision engines of BotHunter and BotFlex.

2.5 Chapter Summary

So far, we have provided a detailed taxonomy and description of the general- and specific-purpose anomaly detection systems and datasets used throughout this work. The ADS taxonomy highlighted the diversity of the anomaly detection systems evaluated in this research. We also provided ROC-based performance evaluations of the ADSs, used throughout this thesis, to develop a better understanding of their strengths and weaknesses. The evaluations showed that general-purpose Maximum Entropy and Kalman Filter ADSs provide acceptable accuracies on the endpoint dataset, while TRW provides good accuracy on the router dataset. Moreover, under changing attack rates, PHAD and Maximum Entropy ADSs are able to maintain good accuracies. The evaluations of specific-purpose ADSs showed both BotHunter and BotFlex provide good accuracies on the ISP dataset.

In the next chapter we present an information theoretic pre-processing method to segregate features at the input of general-purpose ADSs. We present evaluations on a diverse set of five network-based and two host-based ADSs. We will show that slicing features at the input of an ADS not only improves accuracy, but also has immense impact on the space and time requirement of the ADS as well.

CHAPTER 3: A FEATURE SPACE SLICING BASED PRE-PROCESSOR TO IMPROVE THE ACCURACY OF EXISTING GENERAL-PURPOSE ANOMALY DETECTION SYSTEMS

In 2003, Gartner Inc. reported Anomaly Detection Systems (ADSs) as a market failure and predicted that "ADSs will be obsolete by 2005" [65,66]. Ten years later, we know this prediction to be overly pessimistic. Nevertheless, there is widespread consensus that ADS technology has not been as disruptive as was originally anticipated. Despite its pessimistic projections, Gartner did rightly predict that the following reasons (among others) will limit ADSs' commercial appeal: a) low accuracies (low detection rates, high false alarm rates), and b) inability to monitor high-speed traffic. These two factors represent an interesting tradeoff because a general-purpose ADS is supposed to build a highly robust traffic model (to achieve high accuracy), but is expected to do so at an extremely low computational cost (to allow real-time deployment). Commercial general-purpose ADSs try to find the right balance of accuracy and computational complexity, but generally end up leaning too heavily towards one end of this spectrum.[4]

In this work, we ask and address the following question: *Can the accuracy of a general-purpose ADS be improved if we slice ADS feature space at the cost of higher*

[4]For instance, some ADSs run offline but claim to have near 0% false alarm rates [67], while other products run at wirespeeds but do not provide accuracy guarantees [68–70].

computational resource utilization? We first observe that existing ADSs are not designed to exploit better computational platforms to achieve higher accuracies. To mitigate this problem, we identify the fundamental accuracy limiting factors for statistical network and host-based ADSs. We then show that these bottlenecks can be alleviated by our proposed feature space slicing framework. Our framework, operating as a pre-processor, slices a statistical ADS' feature space into multiple disjoint subspaces and then performs anomaly detection separately on each subspace by utilizing more computational resources. We propose generic information theoretic methods for feature space slicing and for determining the appropriate number of subspaces for *any* statistical ADS. Performance evaluation on three independently-collected attack datasets and multiple ID algorithms shows that the enhanced ADSs are able to achieve dramatic improvements in detection (up to 75%) and false alarm (up to 99%) rates.

3.1 Motivation

Intrusion detection accuracy has been a serious limitation in commercial ADS deployments. A main reason for this limitation is the expectation that an ADS should achieve very high accuracy while having extremely low computational complexity. The constraint of low computational cost has recently been relaxed with the emergence of cheap high-performance platforms (e.g., multi-core, GPU, SCC, etc.). Moreover, current ADSs perform anomaly detection on aggregate feature spaces, with large volumes of benign and close-to-benign feature instances that overwhelm the feature

space and hence yield low accuracies. However, if the feature space at the input of an ADS is sliced such that similar features are aggregated into the same subspace, this would result in anomalous feature instances falling into only a few subspaces. Hence since the attack instances are no longer scattered across the feature space, the resultant anomaly classifier would detect the anomaly.

3.2 Challenges

Many cheap high-performance platforms–ranging from low-end multi-core processors [71, 72] to massively parallel Graphical Processing Units (GPUs) [73, 74] and Single-chip Cloud Computers (SCCs) [75]–have emerged to cater for the speed and power requirements of compute-intensive applications. These high-performance platforms are now being proposed to speed up network security devices [76–79]. As these off-the-shelf high-performance platforms become pervasive, the computational resource constraint on ADSs is becoming increasingly less critical. However, the inherent design of current ADSs does not allow them to exploit the parallelism available in emerging hardware.

If multiple instances of an ADS (treated as a black box) are deployed on multiple processing cores (one ADS per core) for anomaly detection, it can in turn enable ADSs to exploit the strengths of off-the-shelf high performance platforms. Consequently, an important question arises: Can the accuracy of an existing ADS be improved if more computational resources are available for its deployment? If the answer to this question is in affirmative then a subsequent question follows: Can a*generic* method

be developed that can allow an existing statistical ADS to enhance its accuracy at the expense of more computational resources?

In this work, we address the above questions in the context of statistical ADSs. Hence, we identify and mitigate accuracy limiting factors in statistical ADSs at the expense of higher computational resources.

3.3 Dataset and ADS

For the evaluation of the current problem, we used the endpoint-based WiSNetlab dataset [80], the router-based LBNL dataset [81]; and the host-based system call dataset [55] (see Chapter 2 for details). Moreover, we evaluated our proposed feature-space slicing technique on five network-based statistical ID algorithms [3, 43, 45–48] and two host-based ID algorithms [33, 49] (see Chapter 2 for details). The main rationales for choosing these ADSs were: 1) Acceptability in Community; 2) Diversity in Accuracies; and 3) Diversity in Detection Principles and Features. Description pertaining to the datasets used in this study and the ADS threshold tuning is given in Chapter 2.

3.4 Technical Approach

A statistical ADS flags traffic anomalies by measuring perturbations in a probabilistic model of network or host traffic. Our performance evaluation in [25] (reiterated in Chapter 2) showed that statistical ADSs offer the most promising accuracies among existing research ADSs e.g rule-based ADSs, volumetric ADSs etc. We identify two

fundamental factors that limit the accuracy of a statistical ADS: 1) Averaging out: in which high volumes of benign feature instances inundate malicious perturbations; and 2) Noise: which is introduced by close-to-benign malicious feature instances (e.g., successful portscans, attacks on common ports/servers, etc.) To mitigate these factors, we propose that an ADS *slices* statistically similar feature instances into disjoint subspaces at its input and then performs anomaly detection on each subspace separately. Such a method localizes the averaging out and noise artifacts to a few subspaces while ADS accuracy is enhanced in the remaining subspaces, albeit at the cost of higher computational resources.

To achieve accurate feature slicing, we propose a novel method which quantifies each feature instance based on its information content. We also propose a conditional entropy based method to determine the appropriate number of subspaces that a feature space should be sliced into. Quantified feature instances and number of subspaces are input to an unsupervised clustering algorithm which assigns a subspace to each feature instance. Intrusion detection algorithms are then applied separately to each subspace.

We demonstrate accuracy improvements by incorporating the proposed feature space slicer as a pre-processing module into five network-based ADSs, namely Maximum Entropy [47], TRW-CB [43], NIDES [48], Kullback-Leibler (K-L) detector [45], and Kalman Filter (KF) detector [46]) and two host-based ADSs, SVM [33] and host-KL [49]. It is important to note that *each ADS is evaluated on its own defined set of feature(s) employed by the ADS for detection of traffic anomalies.* For instance, Maximum Entropy ADS [47] uses destination ports and transport protocol features

for detection of anomalies and hence these features will be used by the feature space slicer to slice traffic for Maximum Entropy. For any other ADS, the traffic will be sliced according to the feature(s) employed by that ADS for detection. We do not judge or propose any changes to the ADSs statistical feature space.

We also compare the accuracy dividends and computational requirements of our proposed method with an existing ADS which uses principal component analysis to define traffic subspaces [3, 82, 83]. These ADSs are evaluated on public, labeled and independently collected attack datasets. Our experiments show that significant and consistent accuracy improvements of 1-75% in detection rate and a dramatic decrease of up to 40-99% in false alarms can be achieved by slicing the input feature space into multiple subspaces.

Among the evaluated detectors, Maximum Entropy [47] and PCA-based Subspace [3, 82, 83] methods are of particular relevance to this work because they also segregate the feature space into multiple subspaces before anomaly detection. To maintain a logical flow of thought, detailed empirical insights into the performance and limitations of these methods will be provided in the following sections.

3.5 Terminology

Before presenting the core concepts of this work, we deem it necessary to define a few terms for developing a better understanding of concepts presented henceforth.

Feature space: It constitutes the features that the ADS uses for anomaly detection. A feature space can comprise a single feature class e.g. connection status in TRW

and connection rate in ratelimiting or it can comprise multiple feature classes e.g. Maximum Entropy comprises 2348 feature classes in its feature space. Each feature class is a random variable.

Feature subspace: Our proposed feature space slicer, slices the overall feature space into multiple slices that we call feature subspaces. These feature subspace are then re-normalized to convert them to a valid conditional distribution.

Feature instance: Each feature class, being a random variable, has an associated probability distribution of the values in the image of the feature. These values are the feature instances.

3.6 Preliminary Analysis

Many well-designed systems offer a performance-complexity tradeoff to cater for diverse system deployments. These systems have the provision to improve key performance metrics at the cost of higher complexity. The key performance metric for an ADS is accuracy which is defined in terms of two competing parameters: detection rate and false alarm rate. Contemporary ADSs do not provide any provision to improve detection or false alarm rates at the cost of higher computational complexity.

Accuracy in an ADS is generally controlled through a thresholding parameter; lower threshold values improve detection rate at the cost of higher false alarm rates, and vice versa. This thresholding logic is applied after an ADS' processing has been completed, therefore thresholding does not provide better (worse) accuracy if more (less) computational resources are available for system deployment. This lack of con-

trol over an ADS' accuracy becomes increasingly constraining as ADSs are now being deployed on parallelized high-performance hardware, such as GPUs and multi-core processors. It is thus important to understand the factors that limit the accuracies of statistical anomaly detectors and, subsequently, investigate whether these factors can be alleviated at the cost of higher computational complexity.

In this section, we introduce a notion of *accuracy scalability* in statistical anomaly detection. This section performs analyses on statistical ADSs with an aim to understand their accuracy limitations and explore potential solutions to mitigate these limitations. It is important to mention here that the analysis pertains to tweaking only the input parameters of an ADS, which is treated as a black box, while the main functioning and methodology used by the detector for anomaly detection, is not altered. For example we do not make/propose any changes to the features used by the ADS for detection and the detection principles employed. Specifically, we raise the following question: Is it possible to improve (or scale) the accuracy of a statistical ADS if we slice the ADS's feature space so as to utilize the additional complexity available for ADS deployment? Answering this question requires one to address the following two questions sequentially: 1) What are the fundamental accuracy limiting factors in statistical ADS design? 2) How can the accuracy limiting factors be mitigated by slicing an ADS's feature space? To address this second question, we only tweak the number of classes the feature space is segregated into for Maximum Entropy ADS, the number of benign vectors for PCA-based subspace method and observe the resultant ADS accuracy.

The rest of this section performs analyses on statistical ADSs to answer these

questions.

3.6.1 What are the fundamental accuracy limiting factors in statistical ADS design?

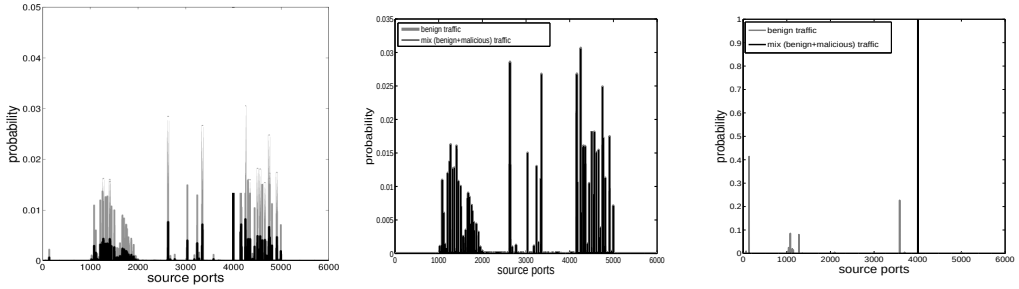
3.6.1.1 Averaging Out

Based on our empirical observations with different ADSs and attack datasets, we argue that the foremost accuracy limiting factor in statistical ADS design is a disparity in the rates of benign and malicious feature instances. Due to the unprecedented volumes and diversity of networked applications and services, traffic observed at endpoints and downstream routers is a mix of many different types of end-user applications and OS programs/processes. Attacks, on the other hand, are becoming increasingly stealthy having rates that are considerably lower than those of benign (p2p, multimedia, gaming, etc.) applications [84, 85].[5]

Due to the disparity between rates of benign and malicious feature instances, intrusions in a statistical feature distribution get *averaged out* by overwhelming volumes of benign feature instances. This causes the feature distributions to be biased towards the frequently-occurring benign feature instances thereby suppressing perturbations caused by malicious feature instances. As a consequence of this averaging out effect, the attack is not detected by an ADS.

Figure 9 provides an illustrative example of this accuracy limiting factor on the endpoint attack traffic dataset. In this figure, a home user's traffic is merged with

[5]While high-rate attacks (e.g., DOS attacks) are also prevalent, these attacks can be easily detected using volumetric approaches and the problem shifts from attack detection to attack prevention and mitigation.

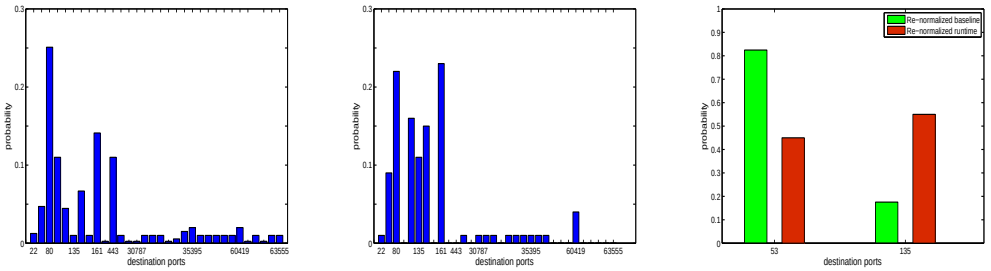


(a) Aggregate traffic (b) Sliced TCP only traffic (c) Sliced UDP only traffic

Figure 9: Example of averaging out: Probability distributions of source ports on aggregate, TCP only, and UDP only traffic; traffic comprises a home user's 20 second traffic window infected by `Witty` worm's UDP-based portscans with a fixed source port of 4000.

traffic from the `Witty` worm. Since `Witty` used a fixed source port 4000 for infection, source port distributions derived from a 20 second time-window pertaining to the TCP and UDP traffic in the aggregate are shown in Figure 9(a). It can be seen that, despite the high average portscan rate (357 scans per sec) of the `Witty` worm, the probability of attack port 4000 does not stand out in the distribution. The main reason for this is that the user regularly generated and downloaded p2p content and the frequency of packets generated by the p2p application were significantly higher than the worm's traffic rate. As a consequence of averaging out, the normalized frequency of the attack port does not stand out in the aggregate distribution, and ADSs are unable to achieve good accuracy on the aggregate distribution. On the other hand, if we slice the traffic into TCP and UDP streams [Figures 9(b) and (c)] and then generate separate distributions for the two streams, pronounced perturbations are observed in the UDP traffic distribution. Such perturbations can be easily detected by an anomaly detector.

Fig. 10 shows normalized probability plots that provide yet another example of this



(a) Baseline distribution (b) Runtime distribution (c) Re-normalized conditional distribution

Figure 10: (a) Probability distribution of destination ports in the baseline distribution; (b) Runtime probability distribution for a 20 seconds time window; traffic comprises a home computer infected by `Blaster` which is generating outgoing portscans on port 135; (c) Re-normalized conditional distribution of the feature subspace constituting the attack port.

accuracy limiting factor on the endpoint attack traffic dataset. In this figure, a home user's traffic is mixed with traffic from the `Blaster` worm which propagates on destination port 135. Fig. 10 (a) presents the aggregate benign distribution learnt during ADS training and used as a baseline for malware detection. Fig. 10 (b) presents the infected runtime distribution for a 20 seconds window. While `Blaster`'s average attack rate in this example is reasonably high (10.5 scans per sec), the probability of attack port 135 is suppressed by huge volumes of benign feature instances. This is primarily due to the huge volumes of benign content generated by endpoints today. As a consequence, the normalized frequency of the attack port was averaged out in the feature distribution (Fig. 10 (b)). It is difficult for an ADS to mine and flag an attack from such an averaged out statistical feature distribution.

When ADSs design feature spaces, the attack class is not known in advance[6]. Hence ADS decision logics operate by summing up feature instances and thresholding the

[6]We consider statistical ADSs in which there is no interdependence between feature classes (Section 3.9).

resultant aggregate. Hence attack perturbations as shown in Fig. 10 (b) get summed up and averaged out by huge volumes of benign feature instances in the feature space.

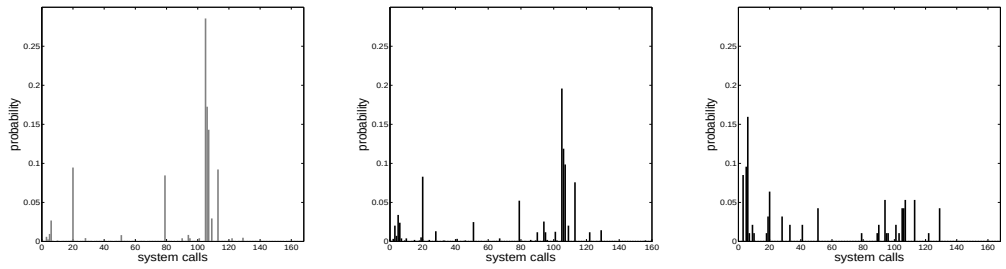
Averaging out can be mitigated if we can somehow slice the traffic in such a way that the high rate benign feature instances are analyzed separately from the aggregate traffic. In this case, we can generate a re-normalized distribution for the remaining feature instances which should characterize the attack-induced perturbations. Hence, the subtle attack perturbations get pronounced in the re-normalized conditional distribution. Fig. 10(c) shows that slicing the aggregate feature space by aggregating similar looking[7] feature instances together mitigates averaging out by making the probability of attack port 135 stand out.

3.6.1.2 Noise

In addition to the averaging-out effect, another accuracy limiting factor in statistical ADSs is introduced by malicious feature instances which are similar to benign feature ones, e.g., successful portscans, common system calls etc. These close-to-benign feature instances also need to be separated from the purely malicious feature instances since they act as *noise* during the anomaly detection process (Section 3.7.2).

Figures 11(a) and (b) show the purely benign probability distribution and purely malicious probability distribution for the `sendmail` intrusion instances, respectively. Note that a large portion of the host-based feature trace generated by the malware overlaps with the benign feature instances. For instance, a system call trace issued by a benign process might be: {`exit`, `fork`, `open`, `read`, `write`, `close`, `wait4`, `creat`,

[7]We evaluate multiple similarity measures in the paper and details are provided in Section 3.7.2.



(a) Benign (b) Aggregate Malicious only (c) Sliced Malicious only

Figure 11: Example of noise: Probability distributions of benign and malicious (aggregate and sliced) system call traces; malicious system calls comprise `sendmail` traces.

`link`, `execv`, `chdir`, `time`, ...}. An infected process inevitably issues many common system calls resembling the benign trace; e.g., `open`, `close`, `exit`, `read`, `write` etc. Figure 11(b) shows that these overlapping malicious feature values act as noise due to their statistical similarity with the benign feature instances and consequently prevent the detection of an intrusion. If, however, the feature space is intelligently sliced with instances having a higher malicious index grouped together, the resultant distribution tends to vary significantly from the benign distribution, as shown in Figure 11(c); the exact slicing method used here will be described in the following section. Similar noise trends (e.g., noise from successful portscans in a worm attack, from p2p traffic during a DoS attack, etc.) are also observed in network anomaly detectors.

Figure 12 provides another example of noise. It shows output of TRW-CB [43] for detecting portscans. This method applies a likelihood ratio test to successful (S) and failed (F) connection attempts to determine if local hosts are scanners. Detection relies on the premise that benign hosts tend to communicate with known hosts and, hence, their probability of connection failure should be lower as compared to a

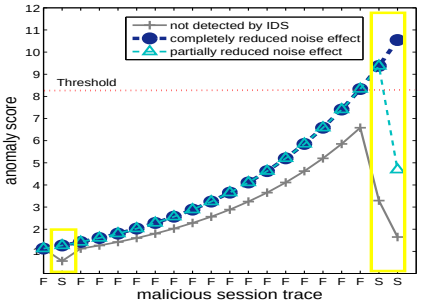


Figure 12: Example of noise– Malware portscans evaluated by reverse hypothesis testing; F and S on x-axis represent failed and successful connection requests, respectively.

portscanner.

As the number of failed connection attempts of a host increases, the likelihood ratio test increases towards the detection threshold as shown in Figure 12. However, successful scans from the scanner makes the likelihood score move away from the threshold. Figure 12 shows that these close-to-benign malicious feature instances act as noise due to their statistical similarity with the benign feature instances and consequently prevent the detection of an intrusion. If, however, the feature space is intelligently sliced with close-to-benign feature instances removed completely [circular markers in Figure 12] or partially [triangular markers in Figure 12] from the local-host feature trace, the anomaly score crosses the threshold and the attack is detected.

3.6.1.3 Discussion

Both the accuracy limiting factors described in this section can be mitigated if we slice traffic so as to aggregate similar features together in the same subspace. We provide detailed analyses of similarity measures in Section 3.7.2. Such slicing will ensure that similar feature instances get aggregated together and anomaly detection

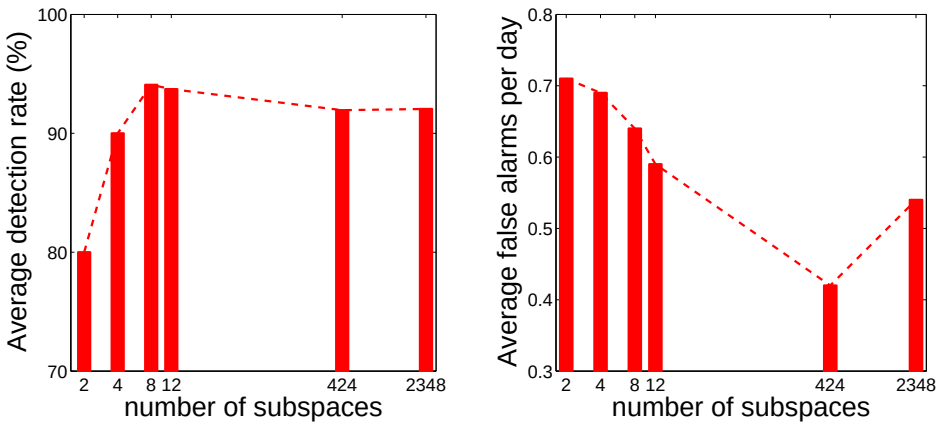
is performed separately on each subspace. While this method will require more computational resources, we hypothesize that it will be able to provide good accuracy. The following section ascertains whether or not averaging out and noise artifacts can be mitigated by feature space slicing.

3.6.2 How can the accuracy limiting factors be mitigated by slicing an ADS's feature space?

As a preliminary attempt to tradeoff computational resources in favor of higher ADS accuracy, this section evaluates two existing feature slicing techniques: a naive method used by the Maximum Entropy detector [47], and the PCA-based method proposed in [82], [3].

3.6.2.1 Maximum Entropy Detector's Feature Slicer

The Maximum Entropy detector's feature space is defined using destination ports (65536 feature values) and transport protocol features (TCP/UDP, SYN/FIN). While the full feature space of the detector contains $65536 \times 4 = 262,144$ feature instances, a method is proposed in [47] to slice this feature space into 2348 packet classes by aggregating less-known ports together, while keeping well-known ports in separate classes. To maintain consistency in notation, henceforth we will refer to these classes as subspaces. Each packet subspace is represented by a random variable X_i, $i = 1, 2, 3, \ldots, 2348$. An associated baseline Probability Mass Function (PMF), p_{X_i}, is estimated from benign training data using maximum entropy estimation [47]; the image of p_{X_i} comprises a set ω_i of all possible destination port values in subspace i.



(a) Detection rate (b) False alarm rate

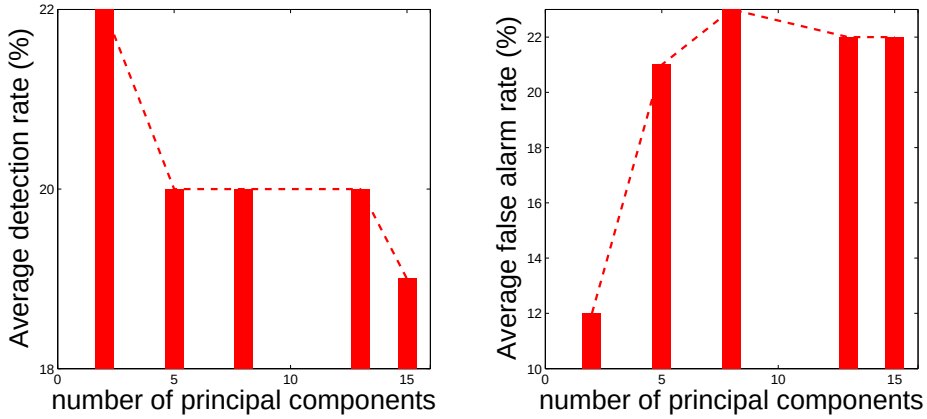
Figure 13: Accuracy of the Maximum Entropy detector with varying number of subspaces; results are generated for the Endpoint dataset and are averaged over all endpoints and attacks.

At run-time, a real-time PMF q_{X_i} is computed by generating a normalized histogram of port usage in subspace i in the last time-window. Difference between the baseline and real-time PMFs of each subspace is quantified using the information theoretic Kullback-Leibler divergence measure [86]:

$$D(q_{X_i}||p_{X_i}) = \sum_{j \in \omega_i} q_{X_i=j} \log_2 \left(\frac{q_{X_i=j}}{p_{X_i=j}} \right). \quad (1)$$

Significant perturbations–i.e., divergence values exceeding a threshold–are flagged as anomalous.

Figure 13 shows the accuracy gain on the endpoint dataset achieved by the Maximum Entropy ADS as the feature subspaces are increased from 2 to 2348 using the detector's slicing technique progressively. Figure 13(a) shows that detection rate improvements of up to 15% can be achieved as we increase the number of subspaces from 1 to 12. While the detection rate reduction saturates at 12 subspaces, the false posi-



(a) Detection rate (b) False alarm rate

Figure 14: Accuracy of the PCA-based subspace method with varying number of principal components; results are generated for the Endpoint dataset and are averaged over all endpoints and attacks.

tive rate decreases by about 57% as we go from 2 subspaces to 424 subspaces. After this point, the accuracy of this naive slicing method starts degrading. Hence, instead of slicing the traffic into 2348 static subspaces, the method could have achieved higher detection rates with 8 subpaces or could have considerably lowered false alarms by operating on 424 subspaces. Reducing the number of subspaces (from 2348 to 8, or even 424) would obviously also lower the memory and runtime resource requirements of the ADS.

3.6.2.2 PCA-based Subspace Method's Feature Slicer

Similar to what is being proposed in our present work, the method of [82], [83] divides the feature space into two disjoint subspaces: a normal subspace S and an anomalous subspace $\tilde{S}$. These subspaces are defined based on typical variations in the aggregate correlated feature space. Let m denote the feature under analysis by the ADS, which can be a flow-, packet- or byte-level feature, and let t denote the number

of successive time intervals of interest. Lakhina's method formulates a $t \times m$ measurement matrix Y and uses Principal Component Analysis (PCA) [82] to transform Y into m eigenvectors that point in the direction of maximum variance. These m principal components are then separated into two disjoint subspaces that correspond to normal and anomalous variations in the feature space. Once the aggregate space is divided into two subspaces, statistical tests like Q-statistic [82] can be used to detect abnormal changes in each constituent feature of $\tilde{S}$.

Figure 14 shows the accuracy of the PCA-based subspace method on the Endpoint dataset using varying number of principal axes for the normal subspace S. Surprisingly, rather than yielding accuracy improvements, an increase in the number of principle components induces a proportional decrease in the detection accuracy of the system. This is because as more and more components from Y are added to the normal subspace S (some of which might be anomalous feature instances), less and less components remain in $\tilde{S}$ for detection of anomalies. Similarly, adding low variance components into S causes the false alarm rates to increase. This is because the low-variance components might present anomalies and adding these components to S can cause the benign components in the residual subspace to be classified as anomalous. This also results in an increase in the false alarm rate, which has also been characterized by other independent studies [52].

3.6.2.3 Discussion

The preliminary results of this section suggest that using feature space slicing for accuracy improvements is a double edged sword. The right slicer can potentially

mitigate averaging out and noise, but accuracy improvements are intricately and sensitively dependent on three factors: a) the feature that the ADS is operating upon; b) the feature slicing method; and c) the number of subspaces into which the feature is sliced. As shown by Figures 13 and 14, an inaccurate choice of any of these factors can in fact lead to a decrease in accuracy.

In the context of the above three factors, we argue that a practical feature slicing technique should be general enough to be deployable with *any* statistical ADS. To achieve such generality it is pertinent that the feature space slicer be agnostic to the features used by an ADS for detection since the feature set varies from one ADS to another. The main reasoning behind this argument is that the coverage of a feature-specific slicer is very limited; for instance, the maximum entropy slicing method is specific to the ADS' feature set and cannot be adapted to improve the accuracies of other statistical ADSs. *Thus we do not judge or propose any changes to an ADS' statistical feature space.* Instead, in the following section we focus on devising methods to cater for the other two factors and our aim is to devise generic methods to identify the *right* number and type of subspaces.

We understand that accuracy improvements provided by such a general feature space slicer would not equal the accuracy improvements that a feature-specific slicer would be able to achieve. We show in the following section that even such a general feature slicer is able to achieve considerable accuracy improvements in a diverse set of ADSs.

3.7 Information Theoretic Feature Space Slicing

We advocate that, in order to separate the malicious feature instances from large volumes of benign and close-to-benign feature instances, the feature space of a statistical ADS should be sliced into multiple subspaces before anomaly detection is performed. The main premise of our proposed feature slicing framework is the following: If an ADS can slice statistically similar feature instances into different subspaces and then perform detection on each subspace separately, then averaging-out and noise artifacts can be localized to a few subspaces while higher accuracies can be achieved in the remaining subspaces.

In this section, we first outline the design constraints and high-level system blocks of the proposed slicing methodology. We then propose information theoretic methods that can address open questions in the design of a feature slicer.

3.7.1 System-Level Design of a Feature Space Slicer

We enforce the following constraints on a practical feature space slicer:

- *Accuracy*: The slicer should localize averaging out and noise artifacts to improve the accuracy of an ADS.
- *Generality*: It should be generic so that it can be incorporated into *any* statistical ADS.
- *Adaptability*: It should be able to automatically adapt the subspaces in accordance with the feature distribution observed in a particular deployment.

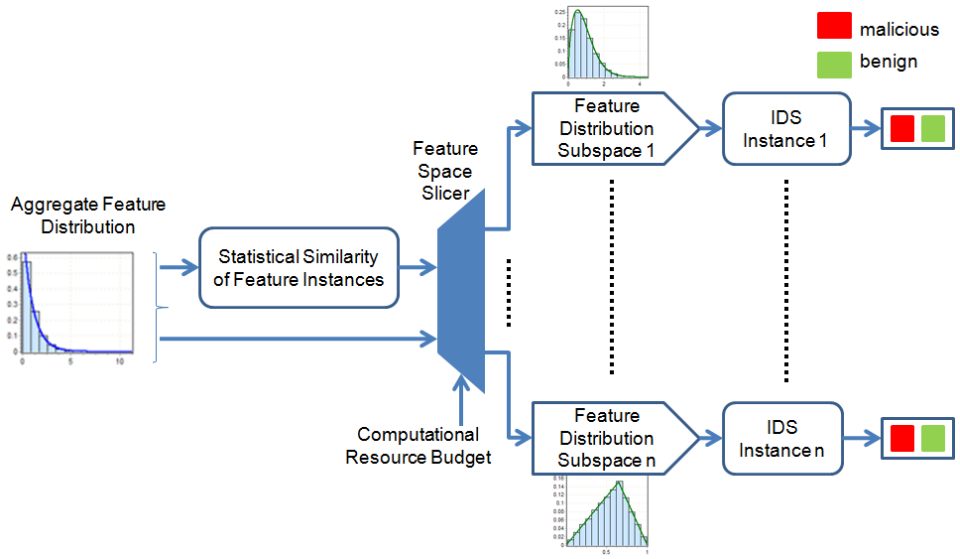


Figure 15: System-level diagram of an ADS employing the proposed feature space slicing principle.

- *Automation*[8]: A practical technique should be fully automated with very few, preferably only one (computational resource budget), user-defined parameters. The number and composition of sliced feature subspaces should be determined automatically in accordance with the computational resource budget and feature distribution.

- *Low Complexity*: While trading off computational resources for higher accuracies, the slicing technique itself should not have very high complexity.

It can be observed that the slicing techniques used by existing Maximum Entropy and PCA-based subspace methods fall short of fulfilling the above constraints. Maximum Entropy detector's slicing technique lacks generality and adaptability, while fulfilling the automation and low-complexity criteria. The PCA-based subspace method, on the other hand, fulfills the generality constraint but fails to provide acceptable ac-

[8]Gartner also rightly predicted that lack of automation will have serious adverse implications in commercial ADS deployments [65].

curacy improvements and has very high complexity.

It is also important to highlight that our proposed feature space slicing framework is different from ensemble based approaches that employ multi-class classifiers for anomaly detection [87]- [88]. Our proposed framework employs multiple instances of the same ADS for anomaly detection on disjoint subspaces. Moreover, similar attack instances perturb similar features and hence are clustered in the same subspace; nullifying the need for cross correlation for dimensionality reduction. This is explained in detail in the subsequent sections. It is also important to mention that the proposed feature space slicer when operating with an ADS, takes the feature space of the ADS as input. This feature space can comprise of a single feature class or multiple classes. The slicer, slices the traffic in accordance with the features used by the ADS.

Figure 15 gives a system-level view of our proposed ADS employing the feature space slicing principle. The ADS first defines the statistical similarity between feature instances in the feature space. Statistically similar feature instances are then clustered together in the same feature subspace, where the number of subspaces is chosen within a user-specified computational resource budget hence fulfilling the automation constraint. Intrusion detection is then performed on each subspace separately. In this accuracy enhancing design, the ADS continues to operate as a black box without any modifications, thereby satisfying the generality constraint imposed above. Moreover, an accurate feature space slicer will cluster and localize the misleading feature instances to a few subspaces, while allowing accurate detection in the remaining subspaces hence fulfilling the accuracy constraint (Table 8, 9, 10). This statistical similarity will obviously be defined based on the feature probabil-

ity distribution, thereby satisfying the adaptability constraint. Hence, the proposed accuracy-scalability framework can be viewed as an intermediate layer between the ADS and the transport layer. We also show that the proposed slicing technique incurs very low additional complexity (Table 11).

The remainder of this chapter answers two related questions:

1. How should statistical similarity between feature instances be defined?

2. How many subspaces should a statistical feature's space be sliced into?

In the following, we try to answer these questions using empirical results from the ADSs under consideration.

3.7.2 How should statistical similarity be defined?

We argue that instead of using a static distribution-oblivious slicer (like the Maximum Entropy one used in the last section), statistical similarity of different feature instances should be based on the probability values observed in a distribution. Let X denote the random variable representing the feature being used by a statistical ADS. Let ω be the set of all possible values that the feature X can take, generally known as the image of X. Let p_{X_i} represent the probability of value i in the image of X. For instance, in the case of Maximum Entropy method of [47], X will represent the destination port feature with $\omega = 0, 1, ...65535$. The probability of each port $p_{X_i}, i \in \omega$ is assigned by generating a frequency histogram of port usage in a time window and then normalizing the histogram. The objective of a feature space slicer is to group similar p_{X_i} together in a subspace.

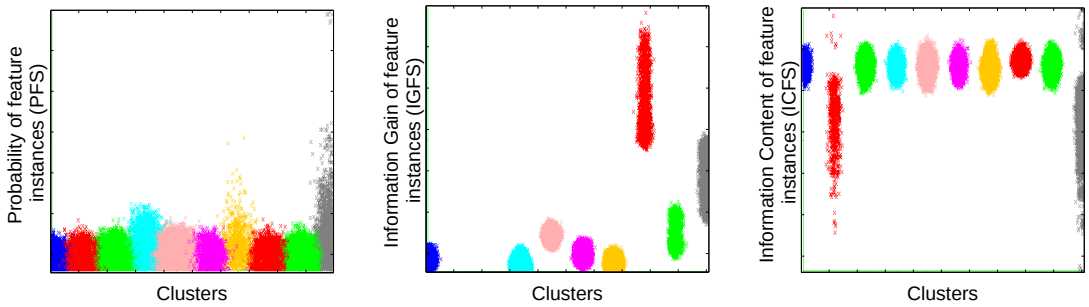
In this section, we evaluate and compare three methods to quantify statistical similarity. Before proceeding, we acknowledge that similarity measures other than the ones used in this work can be tailored to the present problem. However, we observed that the simple measures used here are sufficient to quantify the feature similarities of the ADSs used in this study.

3.7.2.1 Probability and Information Gain-based Feature Space Slicing (PFS, IGFS)

A straightforward measure of distribution-based statistical similarity between values in the image of a feature is the probability, p_{X_i}. Comparing p_{X_i} with p_{X_j} can provide a measure of the similarity between the occurrence probabilities of values i and j of a feature X. We, however, observed that a clustering algorithm operating on probability values directly is unable to provide good demarcation of feature subspace boundaries because it looks at the relative values of probabilities and creates multiple subspaces even in the low probability regions. This scattered clustering is shown pictorially in Figure 16(a) for the source port feature on the LBNL traces.

To mitigate the clustering spread in probability-based clustering, statistical similarity can also be defined on a relative scale. Information gain is a divergence measure that estimates the degree of similarity between the values in the image of a random variable. The information gain [89] between two feature instances i and j of a random variable X can be represented as:

$$I(p_{X_i}||p_{X_j}) = p_{X_i} \log_2 \left(\frac{p_{X_j}}{p_{X_i}} \right), \tag{2}$$



(a) PFS (b) IGFS (c) ICFS

Figure 16: Expectation-Maximization clustering applied to probability, information gain, and information content based feature quantifications in the LBNL dataset; each cluster represents one feature subspace.

where the similarity is measured relative to feature i.

Figure 16(b) shows the IGFS feature space slicing. It can be seen that IGFS takes care of the 'scattering' problem encountered in PFS. However, we noticed that IGFS clusters do not span the entire range of available clusters. For example, in Figure 16(b) clusters 2 and 3 contain only 1% of the feature instances (not visible in the plot). Moreover, IGFS uses a preconfigured threshold to define similarity between subspaces which is against the automation objective that we set for a practical accuracy improvement framework.

3.7.2.2 Information Content based Feature Space Slicing (ICFS)

To mitigate the limitations of relative information theoretic measures for the present problem, instead of IGFS, we can compare the information content of individual values in the image of a feature. Information Content (IC) of value i of a feature X is defined as [86]:

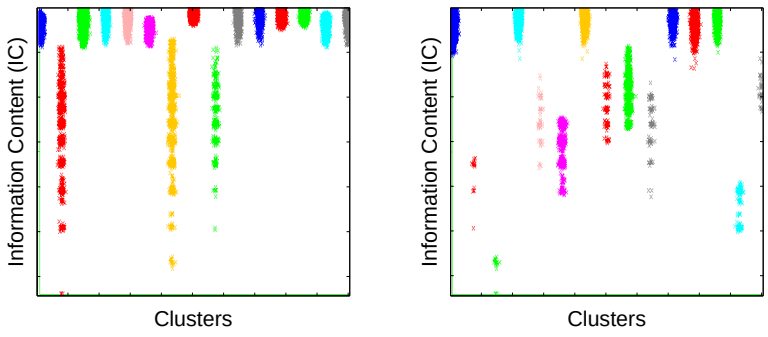
$$H(p_{X_i}) = k \log_2 \left(\frac{1}{p_{X_i}} \right), \tag{3}$$

where k is a constant which is generally set equal to 1. IC quantifies the amount of information present in a symbol of a distribution. Specifically, with a base-2 logarithm, IC represents the minimum number of bits that can be used to source encode a symbol of a probability distribution. Highly likely symbols are more predictable and therefore have low information content. Less likely symbols, on the other hand, have higher information content.

Hence, for the present problem, information content of low and high probabilities will be close to each other and therefore clustering will be more accurate. For instance, where probability-based clustering will assign two separate subspaces to $p_{X_i} = 0.001$ and $p_{X_j} = 0.0001$ because of their relative scales, IC-based clustering will map these probabilities to 13.36 and 9.96, respectively, and group them in a single subspace. Moreover, lower the probability of occurrence of a feature instance, higher is the information content, which results in the clearly demarcated clusters shown in Figure 16(c).

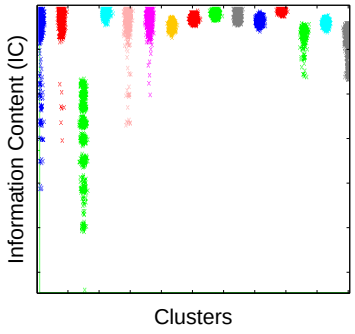
3.7.2.3 Discussion

Once an appropriate quantification of the feature space is achieved, we can slice varying feature instances into disjoint subspaces. This can be achieved by applying an unsupervised clustering algorithm to the quantified probability values; i.e., information content values in the present context. We experimented with four prominent unsupervised clustering algorithms: Expectation-Maximization (EM), Simple k-Means, X-Means and Farthest First algorithms. Figure 17 illustrates IC-based clustering on the LBNL dataset using the clustering algorithm implementations in

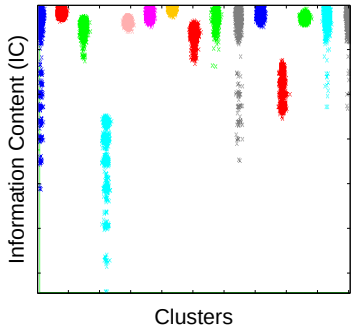


(a) EM

(b) Farthest First



(c) Simple k-Means



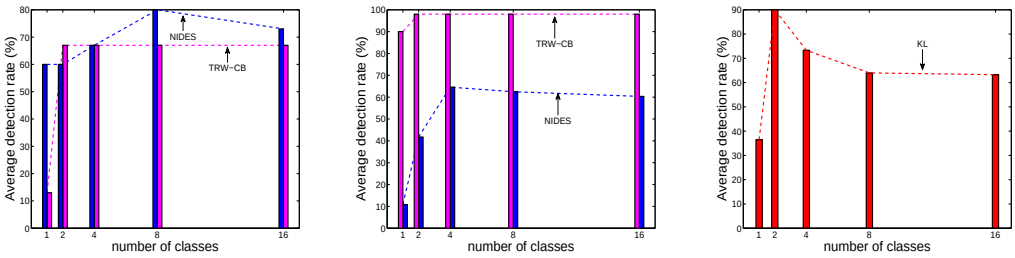
(d) X-Means

Figure 17: Feature space clustering based on information content (IC) of the feature values in the LBNL portscan dataset; each cluster represents one feature class.

the `WEKA` library [90]. We observed that all of the clustering algorithms perform reasonably well in clustering similar feature instances[9]. We hence deduce that the choice of clustering algorithm does not have a significant impact on the accuracy of feature space slicing; rather, the choice of similarity measure is the main accuracy boosting factor. Overall, the EM algorithm provides the best performance in clustering similar IC values.

A limitation of unsupervised clustering algorithms is that they classify feature instances into a user-defined subspaces. This is against our automation and accuracy objectives which require that the appropriate number of subspaces within a compu-

[9]Similar clustering results were observed for the Endpoint and the UNM datasets as well



(a) LBNL dataset (b) Endpoint dataset (c) UNM dataset

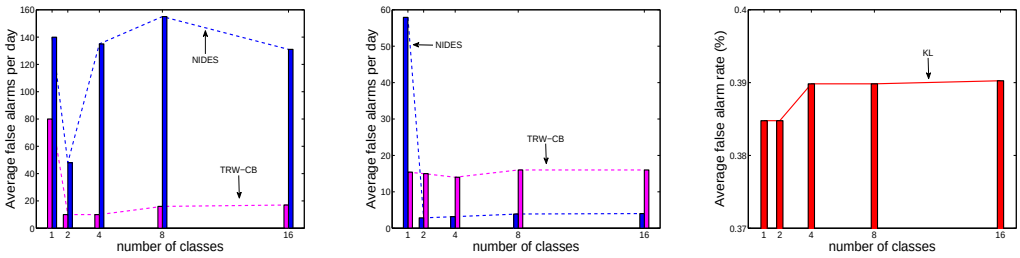
Figure 18: Detection rates of the ADSs with increasing number of classes.

tational resource budget are identified automatically. Moreover, the choice of the number of subspaces is critical because it can degrade ADS accuracy. The following section discusses different approaches to automatically determine the number of subspaces that a feature distribution should be sliced into.

3.7.3 How many subspaces should a feature space be sliced into?

As mentioned earlier, to account for time- and deployment-dependent behavior of ADS features in a usable manner, we treat automation and accuracy as important factors in ADS design. To satisfy the automation and the accuracy constraints, the number of subspaces that a feature space should be segregated into should be determined at deployment time based on a computational resource budget and the learned feature distribution, without actually running the ADS. The number of subspaces should then be periodically updated to cater for time-varying characteristics of a feature distribution.

Figures 18 and 19 illustrate the accuracy variations observed for the network- and host-based ADSs. Figure 18 shows the ADS' detection rate trends as the number of classes is increased. Note that, for most ADSs, the detection rate increases with



(a) LBNL dataset (b) Endpoint dataset (c) UNM dataset

Figure 19: False alarm rates of the ADSs with increasing number of classes.

the number of classes, reaches its maximum accuracy, and then starts to decrease. This reiterates the fact that *increasing the number of sliced feature classes does not necessarily induce an increase in accuracy*. Similar trends are also observed for all the ADSs on the Endpoint and the UNM datasets as illustrated in Figures 18(b) and (c).

Figure 19 demonstrates the ADS trends on the LBNL, Endpoint and UNM datasets in terms of the number of false alarms per day. False alarms tend to show a pattern for most of the ADSs. For example, on the LBNL dataset, the TRW-CB false alarms first decrease with an increase in the number of classes and then increase slightly. Similar false alarm trends can be observed for NIDES and TRW-CB on endpoint dataset. However, for the remaining ADSs (NIDES on LBNL and KL on UNM), no specific false alarm trends can be observed. Thus, while the proposed IC based clustering tends to improve ADS performance in terms of increase in the detection rates, in many cases a proportional decrease in false alarms is also observed. These false alarm variations depend on the traffic characteristics and on the detection methodology employed by the ADS. Hence, it is important for the ADSs to first identify the number of classes that provides acceptable accuracy dividends before the ADS

detection process.

3.7.3.1 Existing Methods

The problem of slicing an aggregate feature space into disjoint subspaces is commonly encountered in data clustering. We evaluated two existing techniques to determine an appropriate number of disjoint feature subspaces before actual clustering is performed: 1) An experimental Elbow method [91] which uses variance as a function of the number of clusters, and 2) Quality Threshold (QT) testing algorithm [92] which was proposed for the analysis and clustering of co-compressed genes. We observed serious practical limitations in employing either of these methods in a real system. For instance, the experimental Elbow method requires rigorous evaluation of data on all possible number of clusters to define a variance pattern. QT is computationally exhaustive since each feature instance is evaluated for correlation.

In view of these limitations, we resorted to developing our own method to ascertain the number of subspaces that a feature space should be clustered into. This method, detailed in the rest of this section, overcomes the shortcomings of the existing schemes as it operates on the aggregate feature distribution instead of standalone feature instances. Nevertheless, we also report results for the Elbow and QT methods in order to benchmark the performance of our proposed method.

3.7.3.2 Conditional Entropy based Method

Conditional entropy [86], $H(Y|X)$, of two random variables X and Y characterizes the information remaining in Y when X is already known. Phrased differently,

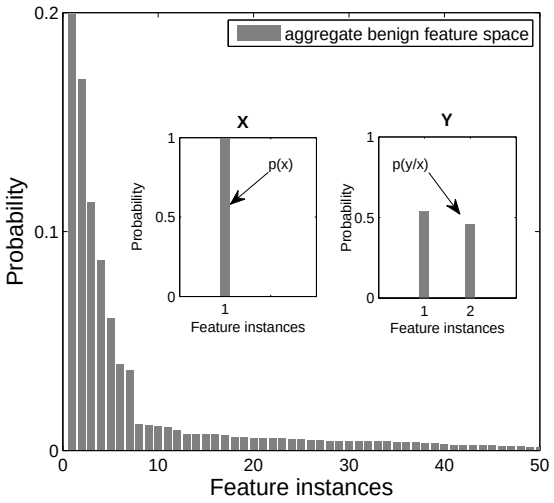


Figure 20: Probability distribution of the aggregate feature space.

conditional entropy is information about Y not given by X. If X and Y are highly correlated, most of the information about Y is communicated through X and $H(Y|X)$ is small. On the other hand, if p_X and p_Y (which respectively represent the probability mass functions of X and Y) are quite different then $H(Y|X)$ assumes a high value.[10]

Let us have a sorted probability distribution of the aggregate benign feature space denoted by p_Z. An illustration of p_Z is shown in Figure 20. From this aggregate distribution, let us define two distributions: a) p_X representing the re-normalized distribution of the most probable feature value in the aggregate feature space distribution; and b) p_Y representing the re-normalized distribution of the union of X and the second most probable feature in the aggregate feature distribution. Here, conditional entropy $H(Y|X)$, which is the amount of information about Y, not given

[10]In the limiting cases, $H(Y|X) = 0$ when $X = Y$, while $H(Y|X) = H(Y)$ when X and Y are independent.

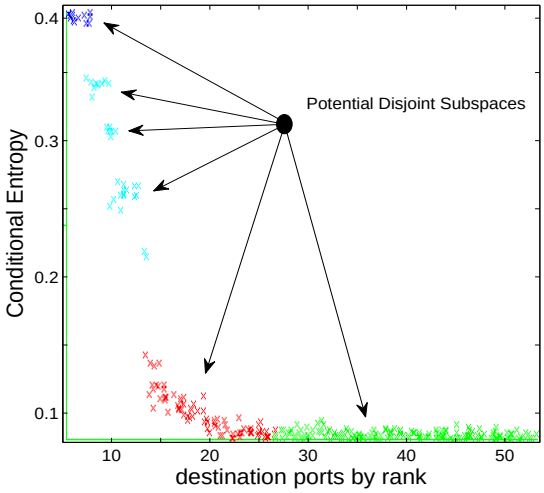


Figure 21: Conditional Entropy based probabilistic analysis of a feature; maximum number of subspaces, within the user-defined computational resource budget should be chosen.

by X is given by:

$$H(Y|X) = -\sum_{\forall x, \forall y} p_{X=x,Y=y} \log_2 \left(p_{Y=y|X=x} \right), \tag{4}$$

where $p_{X=x,Y=y}$ is the joint probability distribution for X and Y. This can also be represented as $p_{X=x} p_{Y|X=x}$. Hence,

$$H(Y|X) = -\sum_{\forall x, \forall y} p_{X=x} p_{Y=y|X=x} \log_2 \left(p_{Y=y|X=x} \right). \tag{5}$$

The conditional entropy [86] value from (5) provides the information gap between distributions X and Y. After noting this gap, the next feature's gap is calculated by setting $X = Y$, and the distribution of Y is re-computed after adding the next most probable feature instance to it. This process is recursively repeated by progressively adding a new feature to Y and noting the information gap. This process is shown in Figure 20. Figure 21 illustrates the gaps in the conditional entropy (CE) values

for the LBNL benign feature instances. Note that larger CE values correspond to a wider information gap and and vice versa. Hence, in order to cluster data judiciously within the computational resource budget, an appropriate value of clusters k, can be chosen from the distribution in Figure 21 and an appropriate number of subspaces is $k+2$ ($k+1$ benign subspaces and a subspace for malicious feature instances).

3.8 Performance Evaluation

In this section we evaluate the accuracy as well as the resource utilization of the proposed feature space slicing technique. For all the three datasets, we sliced the traffic into varying number of classes (as described above) using the `WEKA` implementation [90] of the EM algorithm. Feature distributions used to define clusters are: a) destination port distribution for the Maximum Entropy; number of classes were proportionally divided between TCP and UDP; b) distribution of remote IPs with unsuccessful connections for TRW-CB; c) distribution of packet rates for NIDES; and d) system call frequency distribution for KL and SVM detectors.

3.8.1 Accuracy Evaluation

We evaluate the ADSs on the number of disjoint subspaces obtained from the method discussed above (Elbow, QT and CE). Each method outputs the number of subspaces that the aggregate feature space should be clustered into for a specific ADS; for example, for Maximum Entropy feature space, Elbow method output 8 clusters, QT 3 clusters and CE 10 clusters on the Endpoint dataset. These number of subspaces

Table 8: Accuracy Comparison of Elbow Method, QT testing and Conditional Entropy Feature Space Slicing of Network-based ADSs on the LBNL Dataset

	Maximum Entropy	TRW-CB	NIDES	K-L	Kalman Filter
No Clustering	85%, 144 FPs/day	15%, 80 FPs/day	60%, 140 FPs/day	100%, 11 FPs/day	56%, 10 FPs/day
Elbow Method	87%, 1 FPs/day	68%, 7 FPs/day	60%, 50 FPs/day	100%, 0 FPs/day	48%, 0 FPs/day
QT Testing	80%, 1 FPs/day	68%, 14 FPs/day	80%, 158 FPs/day	100%, 0 FPs/day	57%, 0 FPs/day
CE-based slicing	86%, 1 FPs/day	90%, 10 FPs/day	98%, 4 FPs/day	100%, 0 FPs/day	60%, 0 FPs/day

Table 9: Accuracy Comparison of Elbow Method, QT testing and Conditional Entropy Feature Space Slicing of Network-based ADSs on Endpoint Dataset

	Maximum Entropy	TRW-CB	NIDES	K-L	Kalman Filter
No Clustering	98%, 5 FPs/day	89%, 15 FPs/day	10%, 58 FPs/day	80%, 3 FPs/day	82%, 21 FPs/day
Elbow Method	83%, 7 FPs/day	98%, 15 FPs/day	40%, 4 FPs/day	84.%, 0 FPs/day	80%, 10 FPs/day
QT Testing	95%, 12 FPs/day	98%, 15 FPs/day	50%, 6 FPs/day	81%, 0 FPs/day	80%, 11 FPs/day
CE-based slicing	100%, 3 FPs/day	98%, 11 FPs/day	70%, 4 FPs/day	85%, 0 FPs/day	92%, 1 FPs/day

(for each ADS on each dataset) are then input to the Expectation Maximization clustering algorithm, which assigns each feature instance to one of the $k+2$ subspaces.

Table 8 and 9[11] show the accuracy (detection rate and false alarms per day) of the evaluated network ADSs on the LBNL and the Endpoint datasets respectively. From Table 8 and 9, it is clear that feature space slicing yields considerable false alarm reductions. Both Elbow method and QT testing based EM clustering results show significant reduction in false alarms of the evaluated ADSs. These techniques, however, do not have a substantial impact on detection rates of the ADSs. On the other hand, our proposed Conditional Entropy based method provides dramatic improvements in both false alarm and detection rates.

For the Maximum Entropy detector, the Conditional Entropy method remarkably reduces the number of false alarms/day from 144 to 1 on the LBNL dataset, while increasing the detection rate by 1%. On the Endpoint dataset, Maximum Entropy detection rate is increased by 2%, accompanied by a 40% reduction in false alarms (reducing from 5 to 3). For the TRW-CB detector, a dramatic 75% increase in

[11]The accuracy results for the PCA-based method have already been presented in Section. 3.6.2

Table 10: Accuracy Comparison of Conditional Entropy Feature Space Slicing of Host-based ADSs

	host-KL	SVM
No Clustering	60%, 50 FPs/day	80%, 10 FPs/day
CE-based slicing	90%, 2 FPs/day	84%, 9 FPs/day

detection rate and an 8 times decrease in false alarm rate is observed on the LBNL dataset. On the Endpoint dataset, TRW-CB experiences a 9% increase in detection rate and a 26% decrease in false alarms. NIDES detection rate on LBNL dataset improves by 38% while the false alarms/day decrease from 140 to 4. On the Endpoint dataset, NIDES has an outstanding improvement of 60% on the detection rate and a 93% decrease in false alarm rate. Similar accuracy improvements are observed for KL and Kalman Filter detectors as well.

On the UNM system call dataset as well, significant performance improvements are observed. Feature sliced host-based KL detector's accuracy is improved from approximately 60% detection rate with 50 false alarms/day to approximately 90% detection rate with 2 false alarms/day; a 30% improvement in detection rate and 95–96% reduction in false alarms. Traffic-sliced SVM detector provided excellent performance with up to 84% detection rate. However, in high detection rate regions, there was a false alarm degradation for feature sliced SVM detector. This is because slicing of a process' system calls causes a loss of the temporal correlation (context) between calls. This correlation loss introduces some degradations in high accuracy regions.

Table 11: Computational Resources Utilization of the Intrusion Detection Algorithms and the Proposed Feature Slicer

	Training Time (sec)	Runtime (sec)	Training Memory(MB)	Runtime Memory(MB)
TRW-CB	23185.11	18226.38	25.1	67.54
Max Entropy	50.06	22.79	57.22	66.93
NIDES	2.01	5.4	35.9	40.13
KL detector	22.74	28.33	12.85	13
Kalman Filter	16.59	12.58	34.5	60.3
PCA-based Method	34111.89	15939.27	68.8	16.68
Host-based KL	19.95	22.96	4997	5203
SVM	407.26	797.5	2628.96	2637.82
Feature Slicer	487.92	5.1	0.51	70.26

3.8.2 Evaluation of the Computational Resources Utilization

While we can afford complexity during anomaly detection, we would like the complexity of the slicing technique to be reasonably low. Table 11 lists the training and classification times taken by the anomaly detectors as well as their training and run-time memory requirements. The memory and run-time computational complexity for the clustering module is also listed in the table.[12] Note that the training complexity incurred by the feature space slicing module is somewhat higher than the Maximum Entropy and NIDES algorithms. However, this complexity is incurred only once at the time of training and therefore is not critical. The run-time complexity of the slicing algorithm is 1 to 4 orders of magnitude lesser than all ADS algorithms, except NIDES which has a comparable runtime complexity. For the relatively complex, yet accurate, ADSs (e.g., Maximum Entropy and TRW), the complexity of the slicing module is negligible as compared to the complexity of the underlying ADS algorithms. Memory requirements of the slicing algorithm are comparable to the anomaly detectors. Thus the proposed feature space slicing technique, while being generic, accurate

[12]These numbers are computed using the `hprof` and `VMPerformance Analyzer` tools on a dual-core 2.2GHz Intel machine by running the algorithms over one endpoint's entire traffic dataset.

and automated, is also real-time deployable.

3.9 Limitations

In this section, we enumerate some limitations of the proposed feature space slicing framework and propose countermeasures to mitigate these limitations.

3.9.1 Higher Computation Resource Requirement

An obvious limitation of the proposed method is the additional resources required for running multiple simultaneous instances of the ADSs. However, the proposed feature slicing framework is very amenable for deployment on high performance platforms because the disjoint subspaces identified by the proposed method can be evaluated independently on each processor.

3.9.2 Accuracy

Feature space slicing, if not performed accurately, may spread the attack feature instances over multiple subspaces, thereby allowing the attack to go undetected. In such a case, feature slicing may introduce accuracy degradations instead of improvements. Therefore, care must be exercised in designing an accurate slicer which understands and leverages the inherent properties of the feature space.

3.9.3 Evasion

If the accuracy scalability method is known to an attacker, he/she may evade it by tuning the feature rates such that they are similar to benign traffic. While such

evasion is possible in theory, we argue that it is quite difficult to realize in practice. To evade the detector, the attacker should first know the dynamics of the benign traffic feature instances observed in the network. Even if the benign feature distributions are known, the attacker would have to match the rates of his/her attack features with benign low rate feature instances which would severely limit the efficacy of the attack.

3.9.4 Non-Statistical ADSs

Since we only developed an accuracy scalability framework for statistical ADSs, rule-based systems will not be able to benefit from it. We are currently investigating accuracy scalability methods for non-statistical ADSs.

3.9.5 ADSs with dependence across feature classes

The proposed slicer operates by slicing across the feature space of the ADS. Hence ADSs in which the feature space exhibit dependence across the feature classes for anomaly detection, are not suited for use with the slicer. Since the slicer slices across the feature space, it might not preserve the dependence between feature classes and hence might not yield optimal results.

3.10 Chapter Summary

In this chapter, we proposed an information theoretic technique to improve the performance of current general-purpose anomaly detection systems.

We presented our feature space slicing framework to show that the accuracy of existing statistical ADSs *can* definitely be improved if more computational resources

are available for their deployment. The key idea is to segregate benign and malicious instances into separate subspaces to mitigate averaging out and noise artifacts. We tested our pre-processing feature space slicing framework on a diverse set of network and host-based ADSs. Our experimental evaluation shows that, if feature quantification and number of subspaces is determined judiciously, an ADS operating simultaneously on the reduced subspaces can achieve dramatic improvements in detection and false alarm rates.

In the next chapter, we present another information theoretic combining method. However, unlike the method presented, it is a post-processing multi-classifier ADS design. It combines the outputs of multiple diverse ADSs to exploit their strengths and mitigate their weaknesses. We also provide comparative evaluations with other existing combining techniques and show that our proposed method outperforms other techniques on diverse datasets.

CHAPTER 4: A MULTI-CLASSIFIER BASED POST-PROCESSOR TO IMPROVE THE ACCURACY OF EXISTING GENERAL-PURPOSE ANOMALY DETECTION SYSTEMS

The seminal DARPA IDS evaluation of 1999 emphasized and catalyzed a shift in focus from signature-based intrusion detection to anomaly detection which can detect zero-day (previously-unknown) attacks [37]. After more than a decade of sustained research in the anomaly detection domain, contemporary general-purpose Anomaly Detection Systems (ADSs) still fall short of achieving acceptable accuracies at different deployment points and under different types/rates of attacks [25], [24]. The root cause of this problem is that contemporary anomaly detectors-especially non-proprietary ones available in research literature[13] are designed for specific traffic features which are classified using an algorithm customized for these features. Consequently, an anomaly classifier which is highly accurate for certain attacks and/or deployments fails miserably as benign or attack traffic conditions change as was highlighted in Chapter 2 (Section 2.4).

In this work, we propose a novel information-theoretic combining method operating as a post-processor, which caters for the individual classifiers accuracies in a multi-classifier ADS. We first show that existing combining schemes designed for or adapted

[13]Commercial NADSs without exception use multiple simultaneously-operating anomaly classifiers.

to the problem of multi-classifier ADS combining do not provide good accuracies because they do not use individual classifiers detection and false alarm rates in the combining process. Furthermore, we reveal that an accurate multi-classifier ADS, in addition to catering for the mean accuracy rates, must also consider the classifiers variances during combining. Therefore, we propose a Standard Deviation normalized Entropy of Accuracy (SDnEA) method for classifier combining. Using 9 prominent classifiers operating on two publicly-available traffic datasets, we show that around 3%-10% increase in detection rate and a 40% decrease in false alarm rate over existing combining techniques can be provided by the proposed information-theoretic ADS combining technique.

4.1 Motivation

Originally anomaly detection systems constituted standalone anomaly classifiers due to their inherent computational complexity. However, with the advent of multicore processors and high-end machines, this constraint has been lifted. Moreover, recent studies have shown that standalone anomaly classifiers used by network anomaly detectors are unable to provide acceptable accuracies in real-world deployments. Hence, now an ADS can constitute multiple classifiers. These classifiers are designed to detect a particular class of attacks. Combining multiple diverse classifiers into a single ADS enables the detector to detect a wide variety of these attack classes. Hence, recently a trend of combining classifiers has been witnessed. However, judicious methods of combining these classifiers outputs are largely unexplored.

To achieve higher accuracies, ADSs now use multiple classifiers whose outputs are combined to formulate an aggregate anomaly score. However, current combining techniques do not take into account the individual accuracies and the variance in accuracies of the constituent classifiers. We propose an information theoretic combining technique (SDnEA) that combines multiple diverse classifiers into an ADS. An ROC-based comparative performance evaluation shows that SDnEA outperforms all other techniques and yields very high accuracy.

4.2 Challenges

It is now well-accepted that an ADS should use multiple anomaly classifiers to improve its accuracy (for assorted attacks) and scalability (at different deployment points and under varying traffic volumes). However, accurate and judicious methods that can be used to combine the outputs of multiple anomaly classifiers in an ADS have received little attention in research literature [93, 94] [14]. The limited literature on ADS combining is either host-based [94] or relies on learning separate classifiers for different traffic classes [93]. Generic combining techniques that can combine outputs of any given set of ADSs are not well investigated. Hence, the main challenge is to devise a generic mechanism that can combine any existing set of classifier outputs, hence improving the overall accuracy of the resultant ADS.

[14] Some prior research efforts [95], [96], [97] have, however, proposed combining methods for signature detectors.

4.3 Technical Approach

We first adapt and evaluate existing techniques that can generically combine multiple ADSs' anomaly scores. These generic techniques include: 1) Simple voting-based combining (single instance, all instances, majority vote) [97]; 2) two variants (sum and median rules) of the Bayesian pattern recognition combining method [98]; and 3) the ENCORE fusion logic from the character recognition domain [99, 100]. To quantify the improvements provided by these combining methods, we evaluate progressive combinations of prominent NADS classifiers [48], [47], [41], [82], [40], [44], [43], [46], [32] on two publicly-available portscan attack datasets [25].

Our experimental evaluation shows that the accuracies of existing combining techniques have a significant room for improvement. We also reveal that, in addition to mean accuracies (detection and false alarm rates) of the ADSs, we must cater for the variance of each detector during ADS combining. Using mean and variance of accuracy values from the initial supervised learning phase of a ADS, we propose a novel information-theoretic ADS combining logic referred to as the Standard Deviation normalized Entropy of Accuracy (SDnEA) method. We show that SDnEA's performance consistently and considerably surpasses the the accuracy of the best (ENCORE) existing combining technique. Specifically, SDnEA provides a 3%-10% increase in detection rates and a 40% decrease in false alarm rates over ENCORE. We also show that an increase in the number of anomaly classifiers does not always induce a proportional increase in system accuracy. Therefore, a few judiciously selected classifiers can provide better system-level accuracy than many diverse classifiers.

4.4 Dataset and ADSs

We used two network-based datasets: router-based LBNL dataset [81] and the endpoint based WiSNet Lab dataset [80]. For accuracy evaluation, we progressively combined the following prominent and diverse network anomaly classifers: 1) Rate Limiting [32], 2) Threshold Random Walk (TRW) [41], 3) Credit-based Threshold Random Walk (TRW-CB) [43], 4) Packet Header Anomaly Detector (PHAD) [40], 5) Network Traffic Anomaly Detector (NETAD) [44], 6) Subspace Method [82], 7) Kalman Filter [46], 8) Maximum Entropy Detector [47], and 9) Next-Generation Intrusion Detector (NIDES) [48].

4.5 Performance Evaluation of Existing Combining Techniques

We empirically evaluate the accuracies of existing combining methods. To maintain a logical flow of thought, description and adaptation of existing combiners to the present traffic anomaly detection problem are provided inline. In addition to gauging the accuracies of existing combining methods, we expect the results of this section to reveal insights that can be used to develop an accurate combiner.

4.5.1 Existing Combining Schemes

4.5.1.1 Voting-based Combining:

Without loss of generality, let us treat each constituent anomaly classifier as a black-box that takes the traffic z_t as input at discrete time instance t and then outputs a binary label, [1: anomaly, 0: benign]. Let the total number of classifiers to be

combined be N and let $\Lambda = \{1, 2, ...N\}$ denote a set of indices for these classifiers. Since each classifier has two possible outcomes, it can be regarded as a binary variable X_i. These variables are combined into a single anomaly score using a weighted sum random variable:

$$S_N = w_1X_1 + w_2X_2 + ... + w_NX_N = \sum_{i\in\Lambda} w_iX_i \tag{6}$$

A threshold τ is then applied to this sum in order to classify a given input observation z_t as benign or anomalous.

Under a voting logic, a weight of 1 is applied to the constituent classifiers' binary scores which are then summed up: $S_t = \sum_{i\in\Lambda} X_t[i]$, where $X_t[i]$ is the output of the i-th detector for z_t. A final decision is made as follows:

$$f_{z_t}(S_t) = \begin{cases} 1 & if \quad S_t \geq \tau_V \\ 0 & otherwise \end{cases} \tag{7}$$

In this study we evaluate three variants of the voting principle, namely single instance ($\tau_V = 1$), majority vote ($\tau_V = N/2$), and all instance voting ($\tau_V = N$). Clearly, single instance combining will have good detection rates but poor false alarm rates. All instance voting is on the other extreme with low detection rates with low false alarm rates. Majority vote strikes a balance between these two limiting cases.

4.5.1.2 ENCORE Combining [100], [101]:

ENCORE implements a decision consensus approach based on the known accuracies of the classifiers that flag an observation. In the present context, assume that after analyzing the input z_t, k out of the total N anomaly classifiers flag z_t as anomalous.

ENCORE makes its combining decision based on the accuracies of the top 2 of these k classifiers. Let $p_{\alpha_t[i]}$ and $p_{\alpha_t[j]}$ respectively denote the accuracies of the top 2 classifiers, where $\alpha_t \in \{d : detection, f : falsealarm\}$ can either represent detection rates or false alarm rates. The combined anomaly score is decided according to the following rule:

$$f_{z_t}(p_{\alpha_t[i]}, p_{\alpha_t[j]}) = \begin{cases} 1 & if \quad p_{\alpha_t[i]} - p_{\alpha_t[j]} \leq \tau_E \\ 0 & otherwise \end{cases} \tag{8}$$

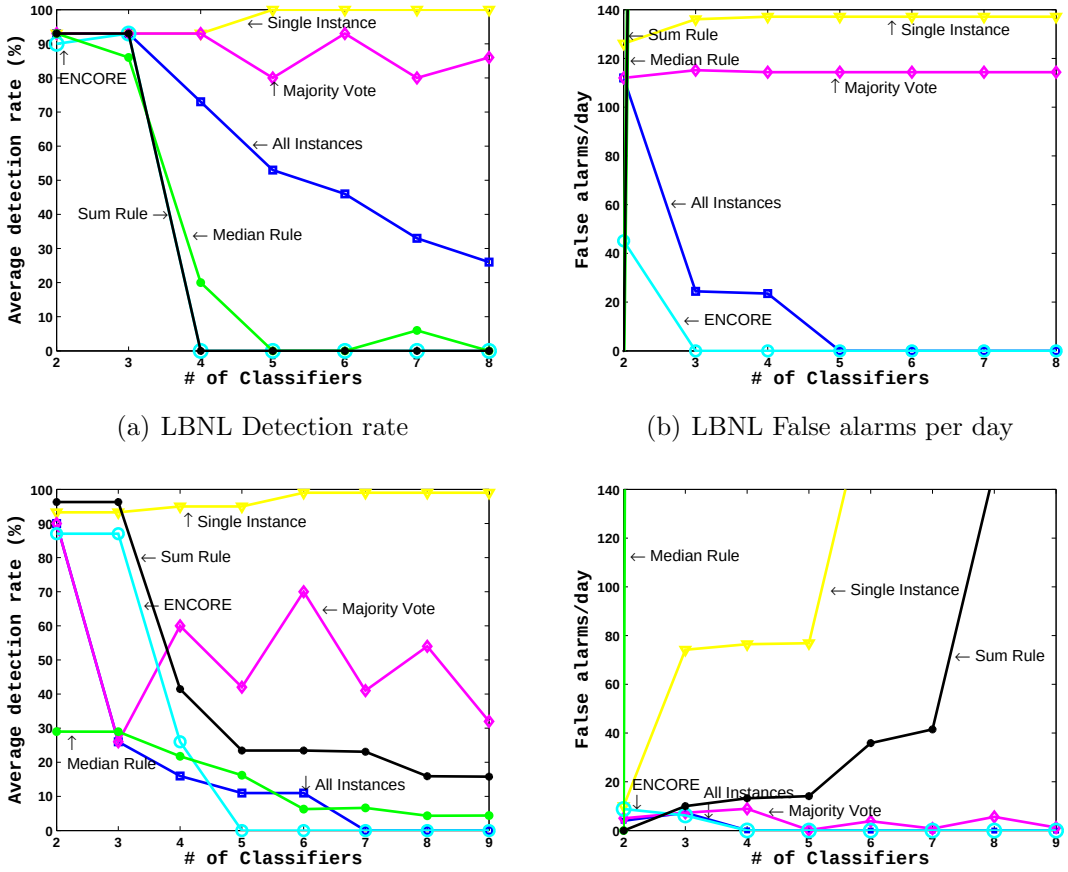
In other words, even an anomalous observation is classified as benign if the most accurate classifiers' accuracies are inconsistent. Note that, unlike voting-based combining, ENCORE takes the accuracies of the individual classifiers into account during the combining process. According to [10], Enhanced Majority Vote ENCORE provides the best accuracy dividends for IDS combining; details can be found in [97], [99], [100].

4.5.1.3 Bayesian Network based Combining:

The authors in [98] combine multiple IDS classifiers, where each classifier uses its own representation of the input pattern. A Bayesian decision rule is used to identify the classifier with the maximum a-posteriori probability to flag an input pattern z_t as anomalous. Assuming reasonably close a-priori and a-posteriori probabilities, the Bayesian detector defines two different classification rules. Under the sum rule, a sum random variable A_t is computed as a function of the individual classifiers' accuracies as follows:

$$A_t = \frac{1}{k} \sum_{j=1}^{k} p_{\alpha_t[j]}, \tag{9}$$

where k is the number of classifiers which flag the input as anomalous and $\alpha_t \in$



(a) LBNL Detection rate

(b) LBNL False alarms per day

(c) Endpoint Detection rate

(d) Endpoint False alarms per day

Figure 22: Accuracies of existing combining methods on the LBNL and Endpoint datasets.

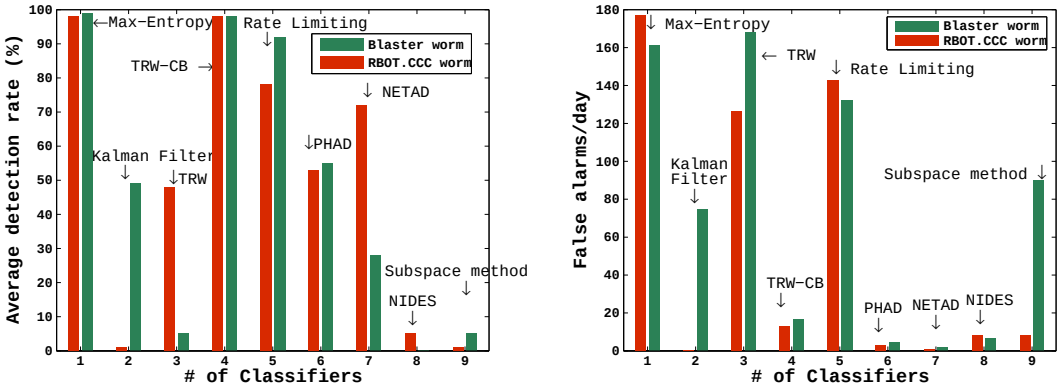
$\{d : \text{det}\,ection, f : falsealarm\}$. Under the median rule, M_t for an input window z_t is the median of the individual classifies' accuracies:

$$M_t = median_{j=1,\ldots,k}[p_{\alpha_t[j]}]. \tag{10}$$

A_t and M_t can be thresholded ($\geq \tau_d$ in case of $\alpha_t = d$ and $\leq \tau_f$ for $\alpha_t = f$) to obtain a combined anomaly score. Similar to ENCORE, the Bayesian combiner accounts for individual classifiers' accuracies in the combining process.

4.5.2 Experimental Results

The experimental results for the voting principle, ENCORE, Bayesian network based median and sum rules are shown in Figure 22. We connect the classifiers in order of the highest performance operating points on the ROC curves. As expected, for single instance combining, as more classifiers are connected to the system, the system accuracy increases in terms of the detection rates but the false alarms also increase proportionally. The abnormal fluctuations in the behavior of the majority voting scheme is due to the non-identical nature of the classifiers that are progressively connected to the system. Detection rates of the system as well as the false alarms vary in no defined pattern as independent and non-identical classifiers are connected that vary in their detection and false alarm rates. For the all instance voting principle, the detection rate of the system decreases as more classifiers are connected since all of them must flag the input as malicious for the system to classify it as malicious. Although this principle has a decreasing trend in the detection rates of the system, it has the same trend in its false alarms as well. ENCORE provides acceptable detection rates since it considers each classifier's overall accuracy when combining ADSs. However, the detection rate goes down to 0 as more and more ADSs are connected to the ensemble. The same trend is observed in the false alarm rates as well. The median rule offers close to 90% detection rate on the LBNL dataset, but fails to maintain this accuracy on the Endpoint dataset. The sum rule provides high detection rates on both the datasets. However, both the techniques suffer from very high false alarm rates (2000 to 7000) on both datasets.



(a) Detection rate

(b) False alarms per day

Figure 23: Variance in the detection and false alarm rates of the classifiers on Blaster and RBOT.CCC worms.

4.5.3 Deductions

4.5.3.1 Classifier Accuracies:

One drawback of the voting principle based ADS combining logic is that it combines the outputs from non-identical classifiers without regard to the accuracies of the constituent classifiers. Intuitively, a classifier with higher accuracy should be given priority over a lower accuracy classifier. Hence a judicious approach is to define the weight (w_i) in Eq. 6, of each constituent classifier in accordance with its accuracy and incorporate it in combining the outputs of multiple classifiers. ENCORE, considers the mean accuracies [101] of the classifiers during ADS combining and consequently offers considerably better detection rates on both datasets.

4.5.3.2 Variance in Accuracies:

During our performance evaluation we observed that a classifier providing very good detection rate for one attack may fail completely for another attack [25]. Similarly,

some classifiers' false alarms varied significantly during different periods of benign activity. Figure 23 (a) and (b) show the detection and the false alarm rates for two specific worms, RBOT.CCC and Blaster. From these results, it is evident that the classifiers vary significantly in not only their detection rates but also in terms of their false alarms.

Hence in order to achieve good $system - level$ accuracy, a ADS combiner, in addition to catering for the mean detection and false alarm rates, should also cater for the variance in the accuracies of the individual classifiers. In the following section, we propose an information-theoretic combiner which caters for the mean and variances of the classifiers during the combining process.

4.6 An Information-Theoretic Combining Method

In this section we propose a novel ADS combining scheme based on the insights of the preceding section. Accuracy of the proposed combiner is compared with the best performing ENCORE method.

4.6.1 Combining Model

As stated in Section 4.5-C, the weights assigned to the outputs of each classifier should characterize the accuracy of the classifier. Thus higher (lower) weights should be assigned to more (less) accurate classifiers. To this end, we propose the use of the information-theoretic Entropy measure to define the weight of a classifier. We

compute a classifier's weight using its accuracy's entropy as follows:

$$\begin{aligned} w_{d_i} &= 1 + p_{d_i}\log_2(p_{d_i}) + (1-p_{d_i})\log_2(1-p_{d_i}), or \\ w_{f_i} &= -p_{f_i}\log_2(p_{f_i}) - (1-p_{f_i})\log_2(1-p_{f_i}). \end{aligned} \tag{11}$$

Based on whether we desire bounds on detections or false alarms, one of the above weights can be used in Eq. 6 to combine multiple anomaly classifiers. Based on the above weights, higher the detection rate for a classifier, lower is the entropy value for it and higher is the weight assigned to its output. Similarly, higher are the false alarms for a classifier, lower is the entropy and a smaller weight is assigned to it.

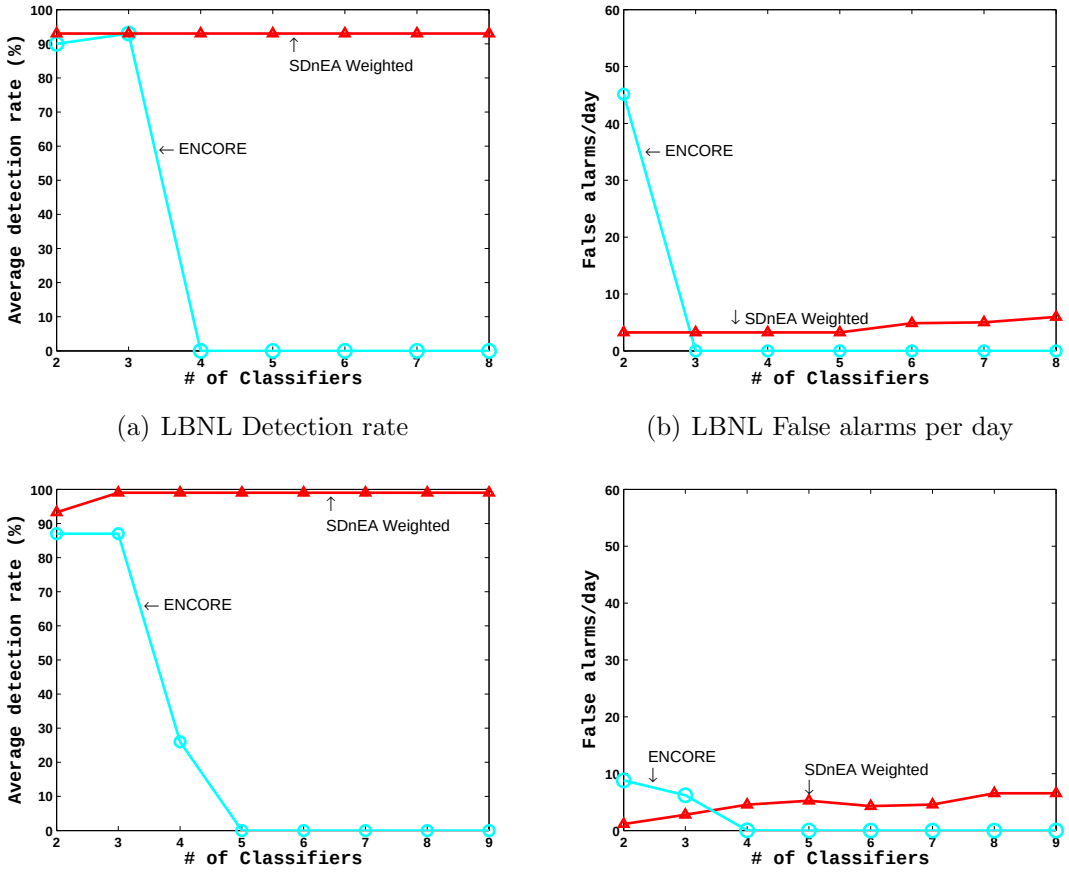
Section 4.5-C also revealed that connecting classifiers having low variance and acceptable detection rates can allow a system to achieve higher system-level accuracy. To incorporate classifiers' variances in the weighting process, we extend the entropy-based weighted averaging as follows:

$$\begin{aligned} w_{d_i} &= \frac{1+p_{d_i}\log_2(p_{d_i})+(1-p_{d_i})\log_2(1-p_{d_i})}{\sigma_{d_i}}, or \\ w_{f_i} &= \frac{-p_{f_i}\log_2(p_{f_i})-(1-p_{f_i})\log_2(1-p_{f_i})}{\sigma_{f_i}}, \end{aligned} \tag{12}$$

where σ_{α_i} is the classifier's standard deviation in detection/false alarm rates. We refer to this weighting scheme as the Standard Deviation normalized Entropy of Accuracy ($SDnEA$) combining scheme.

4.6.2 Performance Evaluation

Figure 24 compares the proposed SDnEA weighted scheme with ENCORE, which provided the best accuracy results in Section 4.5. Clearly, the proposed SDnEA weighted scheme outperforms ENCORE on both LBNL and Endpoint datasets. Ini-



(a) LBNL Detection rate

(b) LBNL False alarms per day

(c) Endpoint Detection rate

(d) Endpoint False alarms per day

Figure 24: SDnEA and ENCORE accuracy comparison on the LBNL and Endpoint datasets.

tially, with two to three classifier combinations, ENCORE and SDnEA provide comparable detection rates on the LBNL dataset. However, with increasing number of classifiers, SDnEA provides a 40% decrease in false alarms over ENCORE. On the endpoint dataset, SDnEA provides a 12% increase in detection rate and a 3% decrease in false alarms for the same classifier combination. These results show that, while combining multiple classifiers to achieve acceptable system performance, variance of the constituent classifiers should also be considered since it has considerable impact on the overall system accuracy.

We also note from Figure 24 that increasing the number of classifiers in the system N does not necessarily induce a proportional increase in accuracy. Hence, increasing system complexity does not guarantee a similar increase in system accuracy. Hence, instead of relying on the number of classifiers to increase system accuracy, one should employ few classifiers having high accuracies and low variances.

4.7 Chapter Summary

In this chapter we proposed an information theoretic multi-classifier ADS design that combines the outputs of multiple diverse classifiers. We showed that our technique outperforms other existing techniques and yields high accuracy.

We presented a Standard Deviation normalized Entropy of Accuracy (SDnEA) combining method for a combination of N parallel connected anomaly classifiers. SDnEA provided consistent and considerable accuracy (detection rates and false alarm) improvements over existing combiners. We also showed that increasing the number of classifiers does not induce a proportional increase in system accuracy. Therefore, a few judiciously selected classifiers can provide better system-level accuracy than many diverse classifiers.

In the next chapter we design, develop and evaluate a post-processing statistical anomaly detection method for detection of bots. We compare our specific-purpose ADS method with existing botnet detection tools, and show that our method is able to provide better accuracy on changing bot behavior, in addition to the added benefits of a stochastic over a rule-based solution.

CHAPTER 5: BAYESIAN INFERENCE BASED POST-PROCESSOR FOR A SPECIFIC-PURPOSE ANOMALY DETECTION SYSTEM

Botnets have proven to be the most evasive threat to the security of networked systems. Botmasters, driven by economic and other incentives (like espionage [102, 103] or even sabotage [104]), continuously evolve their attack and evasion vectors, thereby bypassing most signature-based detection mechanisms. General-purpose anomaly detectors, which classify deviations from normality, are theoretically the best approach to detect this evolving threat. However, since these systems learn the normal behavior to detect activities that are abnormal, they have an inherent limitation of incurring high false positive rates [105].

In this work we present Bottleneck, our prototype implementation, that builds confidence in bot infections based on the causal bot lifecycle encoded in a Bayesian network. We evaluate Bottleneck by applying it as a post-processing decision engine on lifecycle events generated by two existing bot detectors (BotHunter and BotFlex) on two independently-collected datasets. Our experimental results show that Bottleneck consistently achieves comparable or better accuracy than the existing rule-based detectors when the training and test data are similar. For different training and test data, Bottleneck, due to its automated learning and inference models, easily surpasses the accuracies of rule-based systems. Moreover, Bottleneck's stochastic nature allows

its accuracy to be tuned with respect to organizational needs. Lastly, Bottleneck is able to cope with missing training data, without significantly compromising the bot detection accuracy.

5.1 Motivation

Anomaly detection systems are perfectly-suited to detect the evolving botnet threats, but generally suffer from unacceptably high false positive rates. Prior research has used the bot lifecycle as a signature to build bot detection systems. These systems, however, use rule-based decision engines which lack automated adaptability and learning, accuracy tunability and the ability to cope with gaps in training data. To counter these limitations, we propose to replace the rigid decision engines in contemporary bot detectors with a more formal Bayesian inference engine.

Diagnosis of bot infections using Bayesian inference offers many advantages, which we describe next:

Adaptation with threat evolution: Rule-based detection results in rigid, and quite often heuristic (expert opinion based), choices of decision rules and the weights of each lifecycle event. This limits the ability of rule-based decisions to adapt with the evolving threat landscape. Bayesian networks are designed to avoid this pitfall by learning and reasoning about event causalities on a stochastic graph. In fact, Baysesian networks can seamlessly incorporate new evidence in the graph as and when it becomes available.

Accuracy tuning using soft decision confidence: The output of a rule-based decision

engine is a binary (*bot* or $\neg bot$) flag. Consequently, rule-based systems cannot be tuned in accordance with organizational needs–e.g. one organization might want to have very low false positive rates even if that translates into fewer true positives, while another organization might have the opposite preference. Bayesian networks provide a stochastic confidence in the inference which can be thresholded to overcome this shortcoming.

Ability to extrapolate missing data: Training of a rule-based decision system is dependent on the completeness of the training data. For the botnet phenomenon, such data (e.g. observance of a full bot lifecycle) is hardly ever available in real-life scenarios, thus impacting the accuracy of rule-based bot decision engines.[15] A Bayesian inference engine does not suffer from this limitation as the causal relationship embedded in its graph allows it to handle missing data by extrapolating probabilities.

Evidence window determination: An important parameter that impacts the accuracy of a botnet decision engine is the time window for evidence accumulation. While rule-based systems use heuristics to define these parameters, a Bayesian formulation provides a formal notion of fading the belief of a network between successive evaluation windows [106].

5.2 Challenges

Our objective in this research is to understand the challenges faced by an anomaly-based botnet detector, and to explore how these challenges can be overcome by a prac-

[15]While some rule-based engines soften the impact of these fundamental problems by using regression-based weight assignment and soft timers [35], these schemes lack a formal rigor and remain susceptible to data overfitting.

tical and a more formal botnet detector. It is apparent that a viable anomaly-based botnet detector must be able to substantively address the following four shortcomings identified by Sommer and Paxon [105]:

Training: Can the bot detection solution learn and detect the same class of traffic? Existing anomaly detectors learn normal behavior of the network to detect outliers; i.e. network behavior that does not correspond to the learnt normal model. However, quoting from [105]: *Fundamentally, machine learning algorithms excel much better at finding similarities than at identifying activity that does not belong there: the classic machine learning application is a classification problem, rather than discovering meaningful outliers as required by anomaly detection systems.*

Accuracy: Can the detection solution provide accuracies (false positives and false negatives) acceptable for real-world deployments? Most anomaly-based botnet detectors fail to provide reasonable false positives/negatives across different types of botnets. It should be emphasized that even a very small number of false positives can compromise real-life NIDS deployment, as false positives must be resolved by a system admin by mining traffic logs.

Missing Data: Can an anomaly detection solution be robust to missing data? Datasets used for learning and evaluation are never complete. Hence, when these datasets are used for learning, the learnt model is biased towards the statistical patterns present in data while remaining oblivious of missing patterns. Missing data is unavoidable because: a) since data collection is generally time-bound, there is an inherent disparity between data values of interest and those actually observed; 2) The vantage point of dataset collection is typically sub-optimal–e.g., in case of

a distributed traffic phenomenon (like botnets), data values observed at a gateway router/switch are considerably richer in semantics than those observed at an end-point. Clearly, when such datasets are used for bot detection, the behavior of the resultant detection model is static as compared to bot behavior which continues to evolve.

Extensibility: Can an anomaly detection system detect evolving malware? Botnets continue to evolve to bypass security systems put in place for their detection. Existing solutions rely heavily on bot specific characteristics for bot detection and hence are easily evaded by evolving botnets which have immense resources at their disposal [107].

5.3 Technical Approach

Since botnets continuously evolve their attack and evasion vectors, they have proven to be a moving target for detection and mitigation mechanisms. Figure 25 shows a high-level behavior representing a bot's infection-to-attack lifecycle. We observe that this structure has been invariant for the past 6-7 years [35, 108]—essentially the lifetime of the botnet phenomena. Since this invariant characteristic can be used as a signature, we see several commercial and research solutions using rule based decision engines over these lifecycle events [23, 35].

Hence, publically-available techniques [35] [42] detect bots through custom event generation engines that flag infection, C&C communication and attack events, while employing rule-based decision engines as post-processors which correlate these events

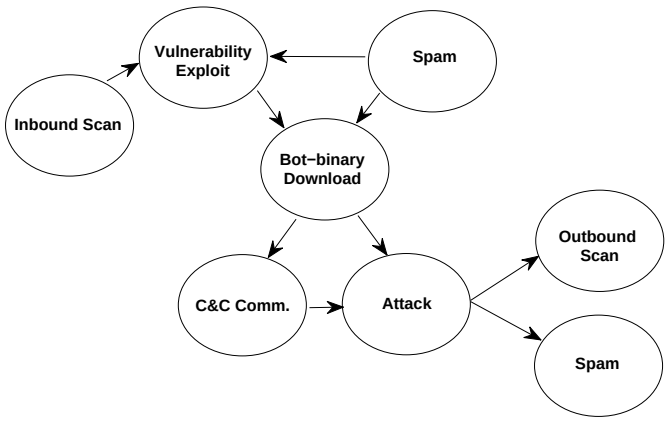


Figure 25: A time-invariant signature of a bot comprising its lifecycle events.

to build an infection score. We advocate a different and a more formal approach by observing that a bot decision engine is perfectly-suited to be postulated as a Bayesian inference problem. A bot's lifecycle events can be directly mapped to a Bayesian network, while stochastic inference on the network can be used to generate the infection score. In this work, we propose Bottleneck: a Bayesian framework that can solve the above shortcomings by *replacing* the heuristic based decision engine with the formal inference of a Bayesian network. Bottleneck builds a Bayesian inference network to model and reason about the causal relationships between bot lifecycle events.

The rest of this chapter evaluates our proposed Bayesian bot decision engine, called *Bottleneck*. For performance evaluation, we replace the rule-based decision engines of the only two publically-available bot detection tools ([35, 42]) with Bottleneck. We perform evaluations on two independently-collected traffic datasets, with the rule-based decision engines forming the comparative baseline of our evaluations. Our experiments show that Bottleneck can consistently match or outperform rule-based

decision engines in accuracy, adaptability, parameter tuning and detection delay.

5.4 Bottleneck: A Bayesian Framework to Infer a Bot Infection

In this section, we present Bottleneck as a Bayesian decision framework that encodes the bot lifecycle. It is pertinent to note that our system can run both in offline (batch) or online modes as it has inherent support for churning decisions after windowing the evidence. We first describe the bot lifecycle in detail. We then highlight the suitability of Bayesian networks in modeling and detecting rapidly-evolving bot malware. This is followed by a brief description of Bayesian networks and the parameter learning and inference algorithms used. Finally, we present an operational and architectural view of our system.

5.4.1 Bot Lifecycle

Infection of a benign host within a network triggers a sequence of events, that we have analyzed to be, consistent across a large corpus of botnets. These events constitute the botnet lifecycle and can be employed for their detection. Bothunter [35] also detects the lifecycle events in their *state-based infection sequence model*, for botnet detection. However, BotHunter employs its own thresholding logic to detect and declare bot infected hosts within the network. Being a closed source tool, little can be extracted from such a thresholding process. It is not clear how the thresholds are selected and what requirements need to be satisfied for bot declaration. Moreover, with the evolution of social media, the social networking sites are extensively being

used as sources for botnet infection. Unlike earlier studies [35] [108], we incorporate such events as well in our proposed lifecycle model.

Figure 25 presents different events constituting a botnet lifecycle. All these events in the lifecycle pertain to the evidence that can be gathered from the network-level flows. Hence, an inbound scan pertains to the inbound network flows recorded at the network perimeter and bound towards (potentially susceptible) network hosts, performing vulnerability scanning. Similarly, an Inbound infection pertains to an end host exploit e.g. bufferoverflow or remote code execution etc. that triggers the bot binary download. The host is now effectively a member of a botnet and hence initiates bidirectional communication with the C&C (Command and Control) server. The bot can then launch attacks as instructed by the botmaster and also infect other hosts within the constituent network.

As for example, Conficker selectively scans the network to find vulnerable hosts. It creates an HTTP server on the source host, and opens a random port between 1024 and 10000. Vulnerable host is exploited using bufferoverflow vulnerability in the MS server RPC handling service to run shell code on the target host. If exploited successfully, the host connects back to the HTTP server and downloads the bot binary in DLL-form[16]. It then starts communication with the C&C server and scans the network for more vulnerable hosts to infect. The botnet is also known to steal personal information like credit card details.

[16]A few variants of the Conficker botnet copy these DLL-form bot binaries to removable media, but since these events cannot be extracts from the network trace, we donot include such events in our lifecycle. Similarly the bot also performs password cracking to copy itself to the ADMIN$ folder, update registry values and performs other configuration changes, which we also not consider as part of our lifecycle events owing to the inability to extract them from the network trace.

Storm is another famous botnet that spread via spam containing infected link that results in drive by download exploit. Once exploited, the host contacts the C&C servers and sends out more spam to infect other hosts within the network.

5.4.2 Why Bayesian inference for Bot detection?

Our initial motivation to employ Bayesian inference in detecting bot infections stems from the rich history of the use of this technique in medical domain for diagnosing diseases. We argue that, much like a disease can be identified by its characteristic symptoms, a bot infection is distinctly identifiable by its infection lifecycle. Beyond this intuitive motivation, our choice stems from background research exploring the challenges faced by an anomaly-based botnet detector as laid out succinctly by Sommer and Paxson [105]. We summarize these challenges and discuss why we believe Bayesian networks are perfectly-suited to overcome these challenges for the present problem of bot detection.

Sommer and Paxson point out that most anomlay detectors *train* their machine learning algorithms to learn normal behavior, but subsequently use these systems to identify outliers [105]. Since these algorithms are tuned to find similarities rather than differences, they yield *lower accuracies* as they misclassify untrained, yet benign, network traffic as malicious.

A Bayesian network that models the bot lifecycle addresses these two related problems of lower accuracy and data training. Instead of modeling normality, such a Bayesian network models the high-level bot lifecycle which remains largely time-invariant. Thus it can be trained to identify the existence of bot and, more impor-

tantly, to not be impacted by presence of unknown but benign traffic.

Another problem with anomaly detection system is that their training sets invariably have *missing data* [105]. This leads to a bias toward patterns present in the dataset while not capturing any missing patterns. A Bayesian network, however, can be built using expert opinion about the high-level bot lifecycle. Thus such a solution is not forced to learn the *structure* of bot infection from (potentially incomplete) data. It is important to note that data is still used to learn the conditional probabilities between the lifecycle events. However, the Bayesian network has the innate ability to extrapolate missing data (using, for instance, EM [109] algorithm) provided the structure is generic and correctly encodes the causality in the system.

A final problem identified by Sommer and Vern relates to the "semantic-gap" between the detection signal and the translation into an actionable item for a network administrator. This translation requires presenting more than a binary detection alert (as its currently generated) to make informed decision. Furthermore, it needs to handle different policy-based responses to similar levels and forms of alerts. A Bayesian network bridges this gap in two ways. First, its detection is accompanied by a belief value that allow different response policies to be implemented. Second, influence diagrams, formed by adding a decision node within the original Bayesian network, provide administrators with a simple way to encode policies in the form of utility tables [110].

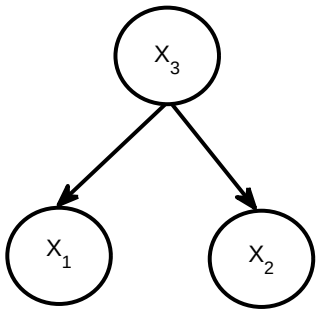


Figure 26: A Bayesian Network.

5.4.3 Bayesian Networks

Bayesian network based models have been used extensively to model uncertainty. A Bayesian network is a collection of nodes (or random variables), all connected together in a directed acyclic graph (DAG) structure. The nodes represent events of interest i.e. information we want to track or infer e.g. communication of a monitored host with the C&C server, bot-binary being downloaded to a local host etc. The directed links between the nodes represent the direction of causality and the information dependence between nodes. A node is independent of all non-descendent nodes given the parent node(s) according to the *directed Markov property* [111].

Bayesian networks, in essence, represent a joint probability distribution function which can be factored depending upon the causal relationships in the network structure. Hence, if a network comprises nodes X_1, X_2 and X_3, with nodes X_1 and X_2 causally related to node X_3 as shown in Figure 26, then the Bayesian network can be represented as the joint probability distribution $P(X_1, X_2, X_3)$, which can be further

factored as follows:

$$P(X_1, X_2, X_3) = P(X_1/X_3)P(X_2/X_3)P(X_3). \tag{13}$$

More generally, with a network comprising n nodes i.e. $\mathbf{X} = X_1, X_2, \ldots, X_n$, the joint probability distribution for any Bayesian network can be represented using recursive factorization as follows:

$$P(\mathbf{X}) = \prod P(X_i/parents(X_i)). \tag{14}$$

The conditional probability distribution reflects the causality in the Bayesian network and entails which random variable is conditioned on other variables.

Learning a Bayesian belief network includes: a) learning bayesian network structure; and b) learning Bayesian parameters. Structure learning in Bayesian networks entails the learning of dependence and independence relationships between the domain variables. These are represented by uni-directional arrows leading from the cause to the effect variable. Parameter learning, on the other hand, is learning the conditional probability tables (CPTs) at each node. These parameters represent the conditional probability values for each state of the variable given each value of the parent node. For nodes with no parents, only prior-probabilities are specified. Once the network is learnt and parameters are set, it is then used for inference. Following sections give a detailed overview of the Bayesian learning and inference techniques employed for our bot detection solution.

5.4.3.1 Bayesian Structure Learning

Since we had a large amount of ground-truthed data we deemed it pertinent to learn the Bayesian network structure as well as performing Bayesian learning using data. There are generally two approaches used for bayesian structure learning: a) search and scoring based approaches; and b) dependency-analysis based approaches [112]. We use the latter for our structure learning since these are known to be efficient and provide better prediction accuracy [113], also due to the lack of prior knowledge pertaining to the causal network parameters required by the former [112].

We employ the Chow and Liu's [114] algorithm for Bayesian structure learning since it is shown to be highly consistent [115]. A Bayesian network with a variable set $\mathbf{X}$ of n random variables is represented by a joint probability distribution:

$$P(\mathbf{X}) = P(X_1, X_2, \ldots, X_n). \tag{15}$$

The algorithm approximates the joint probability distribution P($\mathbf{X}$) with a product of first order distributions of tree dependence.

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i/X_{i-1}). \tag{16}$$

In estimation, it is important to identify the goodness of approximation. The algorithm uses the Kullback-Leibler divergence measure to identify how much the product of the first-order distributions diverge from the n-dimensional joint distribution:

$$D(P, P_a) = \sum_{x} P(\mathbf{X}) log \frac{P(\mathbf{X})}{P_a(\mathbf{X})}. \tag{17}$$

such that

$$D(P, P_a) \geq 0. \tag{18}$$

with equality if both the distributions are the same. Equation 17 can further be solved to the following form:

$$D(P, P_a) = -\sum_{i=1}^{n} I(X_i, X_{j(i)}) + \sum_{i=1}^{n} H(X_i) - H(\mathbf{X}). \tag{19}$$

where $I(X_i, X_{j(i)})$ is the mutual information between two random variables. Hence, the goodness of the approximation depends on maximizing the overall mutual information between the constituent random variables in the Bayesian network. The algorithm employs the maximum-likelihood estimator to construct joint frequency distributions from the samples as follows:

$$f(X_i = u, X_j = v) = \frac{n(X_i = u, X_j = v)}{\sum_{u,v}(X_i = u, X_j = v)}. \tag{20}$$

$$f(X_i = u) = \sum_{v} f(X_i = u, X_j = v).$$

$f_{uv}(i, j)$ is used as a maximum likelihood estimator for the the probability $P(X_i = u, X_j = v)$. Applying the optimization procedure (Equation 19) on this estimator and selecting the structure on which the tree sum of the mutual information is maximized gives the most optimum Bayesian network structure learnt from data.

We experimented with two Bayesian network structures: 1) The Bayesian network structure learnt from data using the algorithm described above; and 2) A Bayesian network structure based on expert opinion. Both these Bayesian networks are presented in the following Section.

5.4.3.2 Bayesian Parameter Learning and Belief Propagation

Learning Bayesian network parameters is trivial. Conditional probability tables are populated with the empirical conditional frequencies from data [116] [113] i.e with frequency ratios [111]. This is termed as the counting method [117]. Moreover, in case of missing data, expectation maximization, gradient decent can be used for approximation [117].

Once, the Bayesian network is learnt and probability tables are set, it is used for Bayesian inference i.e. computing belief value of a node when the value of other variables is known. In performing inference, the DAG represented by the joint probability distribution (Eq. 13) is used to calculate the marginal distributions that are conditioned over the observed events. Hence, with the occurrence of bot lifecycle events, the node CPTs are updated as the belief is propagated throughout the Bayesian network. We employ the *junction tree* method [111] [118] for Bayesian inference and belief propagation because of its faster execution [111] [117]. Here we provide a brief overview of the *junction tree* algorithm, interested readers are referred to [111] for more details.

The junction tree inference algorithm proceeds as follows: 1) Moralize the Bayesian network; 2) Triangulate the graph; 3) Identify the cliques in the resulting graph which inturn form the nodes of the desired junction tree; 4) Propagate the λ and π values through the junction tree to perform inference. We explain each of these steps in detail below.

The first step in converting the Bayesian network DAG into a junction tree is

to moralize the graph [119]. This is done by inserting undirected edges between the parents of each child node in the DAG. Hence, each node and its parents now form a complete subgraph. This is followed by replacing all the directed edges with undirected edges in the original graph. The resulting graph in then triangulated [120]. A triangulated graph is an undirected graph such that any simple cycle of four nodes in the graph has atleast one cord where a chord is described as an edge that does not appear in a path. We next identify cliques, were clique is a complete subgraph, in the triangulated graph. [111] proposes an algorithm to represent the joint distribution of the DAG in terms of functions on the cliques and hence generate a list of maximal cliques. [120] describes a technique to connect the cliques using *maximum cardinality ordering* to form a junction tree. The junction trees adhere to the *running intersection property* which implies that if a node is in two cliques, it is a part of all cliques between the two cliques in the junction tree. The property ensures consistency of message passing between cliques. In the junction tree, a clique is a basic unit of local computation. Hence smaller the size of the cliques, faster the overall inference computation. Thus far, we have explained one scheme for constructing a junction tree from our Bayesian network DAG. We now explain the process of performing belief propagation in the junction tree using a message passing algorithm.

Belief propagation in junction trees is achieved by computing the marginal for each clique conditioned over the observed events, followed by computing marginal for each node (i.e. random variable) within the clique. Here we explain one such message passing algorithm, the Pearl Method [120] to propagate the λ and π values within the junction tree for belief propagation.

Let **e** be the set of observed events. For a specific variable **X**, these events can be divided into two sets: e_X^- for all events that constitute variables that are descendants of **X**, and e_X^+ for all other event variables. The following two values are used for the belief computation of variable **X**:

$$\lambda(X) = P(e_X^-/X). \tag{21}$$

$$\pi(X) = P(X/e_X^+). \tag{22}$$

Since **X** can have multiple values, λ and π will be vector valued, with a value for each state of variable **X**. These λ and π values are passed between the nodes in the junction tree in an orderly fashion, hence called the message passing algorithm. Conditioned on the observed events, the posterior probability of the variable **X** can be inferred as follows:

$$P(X/e) = \alpha.\lambda(X).\pi(X), \tag{23}$$

where $\alpha = \frac{1}{P(e)}$. Equation 23 is computed to infer the new belief of the random variable **X** on the occurrence of a set of events **e**.

We now describe briefly how the λ and π values are computed. Readers are referred to [120] for more details.

$\lambda(X)$ is computed on all variables that are descendants of variable **X**. Let these be variables: $Y_1, Y_2, \ldots, Y_n$. For cases where variable **X** is observed, λ acts as an indicator variable. However, in cases where **X** is not observed, the λ message is computed as follows:

$$\lambda(X) = \sum_{y_i} P(e_Y^-/y_i).P(y_i/X), \tag{24}$$

where $P(e_Y^-/y_i)$ is the marginal probability $\lambda(y_i)$. Hence, to compute the value of $\lambda(X)$, we need to have the λ values for all its descendant nodes. Thus, λ values are propagated upwards through the junction tree. π values, on the other hand are propagated downwards from parent node variable(s) $\mathbf{Y}$ to child node variable $\mathbf{X}$ as follow:

$$\pi(X) = \sum_{y_i} P(X/y_i).\pi(y_i). \tag{25}$$

Hence to compute the π value for a variable, we need to compute the π values for its parent nodes, and the conditional CPT values for the variable $\mathbf{X}$.

5.4.4 Architecture of the Bottleneck System

We now present an operational and architectural view of the Bottleneck system. Figure 27 shows that we subdivide the system into three components: Event generation, updates to Bayesian network, and bot classification. It is pertinent to note that our system can run both in offline (batch) or online modes as it has inherent support for churning decisions after windowing the evidence. We next explain the different components in detail.

5.4.4.1 Lifecycle Event Generation

We divide a bot detector into two components: 1) an Event Generation Engine (EGE) that flags events part of the bot lifecycle (infection, propagation, C&C communication, attack, etc.); and 2) a decision engine which incorporates the causality in the EGE events to generate a confidence in bot detection. The EGE is generally a Network IDS (NIDS) (e.g. Bro [19] or Snort [20]), above which the decision engine

will be implemented. The EGE presents the events to the post-processing decision engine in the form of a case file for each host indicating whether or not each of the specified events has been detected in the last evaluation window.

In this design, we have intentionally decoupled the decision engine (our novelty) from the lifecycle event detection. This decoupling serves two purposes. First, its helps in adapting to the changing threat landscape of botnets. While the high-level bot lifecycle will remain the same in the foreseeable future, the actual manifestation of these lifecycle events will continue to evolve. This decoupling allows the Bottleneck architecture to improve its accuracy in proportion to improvement in event detection mechanisms, without having to alter its decision engine. Secondly, and perhaps more importantly, this decoupling allows us to compare different decision engines using the same set of events. Alternately, we can also replace or aggregate multiple engines to improve the accuracy.

5.4.4.2 Bayesian Network Classifier using the Bot Lifecycle

The events generated by the EGE are input to an expert opinion based Bayesian network classifier shown in Figure 27.

Bayesian networks represent a joint probability distribution function which can be factored depending upon the causal relationships in the network structure. The conditional probability distribution reflects the causality in the Bayesian network and entails which node (random variable) is conditioned on other nodes (variables). For learning Bayesian parameters, conditional probability tables at the nodes are populated with the empirical conditional frequencies from data [116]. Once, the

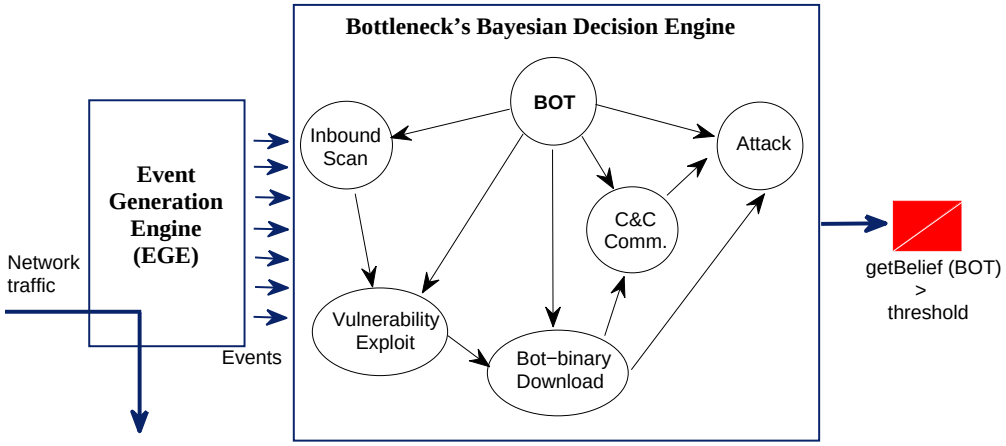


Figure 27: Bottleneck's architecture, also showing the Bayesian network (based on bot lifecycle) used to infer bot infections.

Bayesian network is learnt and probability tables are set, it is used for Bayesian inference i.e. computing belief value of a node when the value of other variables is known. We employed the *junction tree* method [118] for Bayesian inference which uses a message passing algorithm for belief propagation. Bayesian networks are described in more detail in Section 5.4.3.

Bayesian networks have very clear semantics which make them perfectly suited for classification [120]. The nodes in a classification network can be of two categories: hypothesis and evidence nodes. We use the BOT node as the hypothesis node and make it the parent of all the lifecycle nodes that are evidence nodes for bot classification. These evidence nodes represent the bot lifecycle events we obtain from study of previous work and a large corpus of bot behavior (Figure 25).

The initial conditional probabilities of this bayesian network are learnt from a data

trace. At every detection interval, the presence of lifecycle events, conveyed by the case file from the EGE, updates the conditional probability that propagates through the network to update the belief of the classification node.

5.4.4.3 Bot Classification and Action

The updated belief of the BOT classification node can then be compared with a threshold to classify a bot detection. We can easily extend this notion to compare with several different thresholds to trigger correspondingly different reactions.

5.5 Bottleneck Performance Evaluation

In this section, we validate the advantages discussed earlier, by evaluating Bottleneck as a post-processing decision engine for two existing bot detectors–namely BotHunter [35] and BotFlex [42]. Accuracy is evaluated on two independently-collected datasets: Nayatel 2.3.1 and SysNet Lab 2.3.2 trace.

5.5.1 Existing Bot Detectors

BotHunter [35] and BotFlex [42] are the only bot detection tools freely-available online. We use these tools for evaluations throughout this paper.

BotHunter is a decision engine built over the Snort IDS. It modifies the Snort ruleset to detect events possibly indicative of a bot infection, and implements a rule-based correlation layer to evaluate whether a given host is infected. BotFlex similarly implements a rule-based decision engine over an event generation engine (EGE) that uses the Bro IDS. When evaluating Bottleneck, we use as input the events gener-

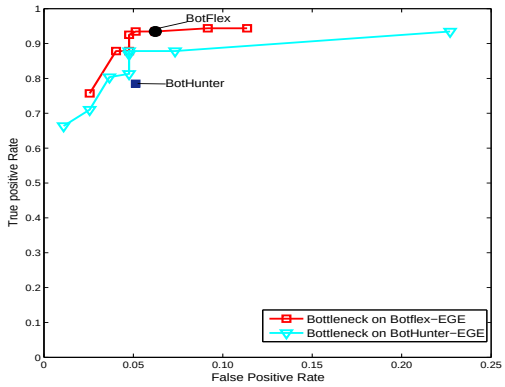


Figure 28: Accuracy evaluation of BotHunter, BotFlex and Bottleneck on Nayatel ISP dataset. Bottleneck is trained and tested on the trace using 10 fold cross validation.

ated by BotFlex and BotHunter's EGEs. We therefore evaluated Bottleneck twice, once using events generated by BotHunter-EGE and once using events generated by BotFlex-EGE.

We downloaded BotHunter and BotFlex and their various blacklists on the day of the data collection, thus allowing a judicious comparison. To ensure transparency and allow future comparison, we will anonymize and release the Snort/Bro logs on our project website [121].

5.5.2 Accuracy Evaluation

We first evaluate the overall accuracy of Bottleneck, BotHunter and BotFlex over the Nayatel dataset. All accuracy results presented henceforth were obtained using 10-fold cross validation. We generate case files for the entire evaluation trace, in effect running the system in an offline (batch) mode that is common in many commercial malware detection appliances. Figure 28 provides ROC-based accuracy evaluation for this scenario under different detection thresholds. Before we present the evaluation

Table 12: Investigating false positive and false negatives of Bottleneck, Botflex and BotHunter on the Nayatel dataset

(a) **Bottleneck and BotHunter over BotHunter-EGE**

Bottleneck	FP: 14	4 IPs: 3 or more lifecycle events observed 7 IPs: Attack observed coupled with at least one other event 3 IPs: CnC Communication observed
	FN: 13	4 IPs: No lifecycle events observed 8 IPs: Only one event observed 1 IP: Exploit,CnC,Attack observed
BotHunter	FP: 14	9 IPs: Caused by E8 5 IPs: Bot events generated so declaration conditions are met
	FN: 23	4 IPs: No lifecycle events observed 7 IPs: Event detections insufficient to trigger bot declaration conditions. 12 IPs: Neither of the two conditions satisfied due to windowing

(b) **Bottleneck and BotFlex over BotFlex-EGE**

Bottleneck	FP: 17	16 IPs: 3 or more events triggered by EGE 1 IP: Exploit and egg download observed, given high weight by Bayesian inference module
	FN: 7	Inbound scan with at most two other events (Inbound scan always lowers the overall belief)
BotFlex	FP: 17	3 or more events triggered
	FN: 7	6 IPs: Event detections insufficient to trigger bot declaration conditions 1 IP: Inbound scan and exploit with CnC communication

results, it is pertinent to reiterate that Bottleneck's design allows us to threshold the belief in the classification (BOT) node (Figure 27). This thresholding is a unique feature as it allows the user to adapt the detection engine in accordance with organizational needs. On the other hand, accuracy results of rule-based systems (BotFlex and BotHunter) are points in the ROC space.

It can be observed that BotHunter and Botflex provide good accuracies ($> 75\%$ TP and $5 - 6.5\%$ FP) on the ISP dataset. This is mainly because the bot traffic in the dataset triggered a large number of bot lifecycle events, which facilitated the decision engines of BotHunter and BotFlex.

In comparison to BotHunter, Bottleneck provides a significant increase in true

positives (from 78.5% to 87.9%),while reducing the false positives slightly from 5.1% to 4.8%. Moreover in comparison to BotFlex, Bottleneck matches BotFlex's TPR, while inducing a slight decrease in FPR (from 6.2% to 5.1%).

Based on the above results, we observe that Bottleneck provides comparable or better accuracy than BotHunter and BotFlex. We now provide a breakdown of the false positives and false negatives to better understand why Bottleneck offers accuracy improvements as compared to BotHunter and BotFlex, despite relying on the same underlying EGE.

5.5.2.1 Detailed Analysis of FPs and FNs

Table 12 presents a detailed analysis of the false positives and false negatives flagged by Bottleneck, BotFlex and BotHunter.

Table 12 (a) compares BotHunter and Bottleneck, with both receiving events from the BotHunter-EGE. Bottleneck, with its learning-based logic, is able to detect 4 of the 7 hosts that BotHunter misses because the lifecycle events that trigger BotHunter's conditions do not appear in the dataset. Hence these infections do not strictly adhere to the rulebase and are consequently missed by BotHunter. Bottleneck, on the other hand, is able to detect these infections as it observed and learned similar incomplete event combinations on infected hosts in the training data. Hence the Bayesian framework is able to extrapolate missing data, a feature that rule-based decision engines lack.

Regarding the six hosts detected by BotHunter but missed by Bottleneck, five do not generate any lifecycle event, and are understandably missed. These hosts do

however generate one BotHunter-specific event (E8[rb][17]) for outbound connections matching a blacklist and are thus included in the true positive count for BotHunter. In other words, a more robust blacklist would also allow Bottleneck to flag these bots. Interestingly, 9 out of 14 false positives by BotHunter are also caused by the same E8 event, so we hypothesize that this blacklist is designed to be inclusive and broad.

Table 12(b) compares BotFlex with Bottleneck, with both receiving events from the BotFlex-EGE. We observe that 15 of the 17 FPs by both Bottleneck and BotFlex are common, and are caused by bot lifecycle events being detected by BotFlex-EGE on benign hosts. Similarly, 6 out of the 7 FNs are also common to both and are caused by insufficient events being observed by the EGE. We conclude, therefore, that many of Bottleneck's FPs in our evaluation are a consequence of the liberal event generation policy employed by BotFlex.

5.5.2.2 Discussion

The results presented in this section have established that, while operating on the same event generation evidence as existing bot detectors, the Bayesian Bottleneck decision engine:

- Enables existing bot detectors to achieve comparable or better accuracies than their built-in rule-based decision engines;
- Allows the accuracy of the system to be tuned with respect to organizational needs; and

[17]Outbound to malware site: ¡Host¿ has connected to a known malware control site.

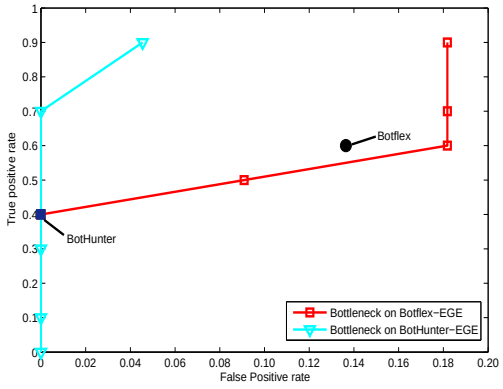


Figure 29: Bottleneck, BotHunter and BotFlex evaluated on the Sysnet trace. Bottleneck is first trained on the ISP dataset and then tested on the Sysnet trace.

- Can provide good accuracy even when the entire bot lifecycle is not present in the training dataset.

Now we present some additional benefits of employing a stochastic Bayesian decision engine for bot detection.

5.5.3 Accuracy under changing bot behavior

We argued earlier that Bayesian inference, being stochastic in nature, should be able to maintain high accuracy with changing bot behavior. To substantiate this argument, we train the Bayesian network on the Nayatel dataset, but subsequently evaluate accuracy on the SysNet dataset. Since the bot traffic in the SysNet dataset was triggered manually (by running the binary), in a controlled environment, this dataset is intrinsically different from the ISP dataset and constitutes partial bot lifecycle events. This is a realistic scenario since botnets are continuously evolving to mask their propagation activities to evade detection [1].

Figure 29 provides an accuracy comparison of BotHunter and BotFlex with Bot-

tleneck, under the present experimental setup. It can be seen in Figure 29 that as the bot behavior changes, rule-based decision engines are unable to maintain their accuracy, resulting in missed detections. It can be observed that Bottleneck provides consistently high accuracy despite changing bot behavior. It detects 9 out of the 10 infected hosts; the only host that is missed has no lifecycle events detected by either BotFlex- or BotHunter-EGE.

Moreover, when taking input from the BotHunter-EGE, Bottleneck generates only *one* false positive, which occurs because outbound scanning (which has high correlation with bot infection in the training data) is detected on the host. It can be observed that Bottleneck generates significantly higher FPs on BotFlex-EGE. A closer examination of the false positives reveals that BotFlex detects events such as inbound exploit, CnC communication and outbound scanning on these hosts, and hence Bottleneck's false positives are in this case inherited from the EGE itself. Bottleneck is able to achieve an overall TP of 90% with only one additional false positive as compared to BotFlex.

In comparison to BotHunter's 4 true positives, Bottleneck is able to detect 7 hosts with *zero* false positives, and 9 hosts with only one false positive. We hence conclude that Bottleneck, being stochastic in nature, is able to adapt to changing bot behavior. This allows Bottleneck to offer higher accuracy than rigid rule-based decision engines.

All real-time detection engines employ some windowing mechanism for detection – evidence is accumulated over a specific time window, after which a decision is made, and previous evidence is completely or partially discarded. The question that arises here is how much memory to retain between windows – too much could mean falsely

identifying no-longer malicious hosts, while too little could result in missed detections. In this section we evaluate Bottleneck for real time deployment and provide a comparison with the windowing mechanism employed by Bothunter. BotFlex does not provide any windowing and hence is not evaluated in this section.

To understand the effect of windowing on a detector's accuracy, we analyzed BotHunter's 23 missed detections, shown in Table 12. On 12 (approximately 50%) of these hosts, different lifecycle events were indeed flagged by the BotHunter EGE. However, being spread over different observation windows, these events failed to trigger any of the decision engine's conditions.[18]

Bayesian networks present a built-in mechanism to handle the windowing issue by fading the belief after every detection window. Not only is this method tunable (by changing the fading degree), it also provides a mathematically rigorous way to handle the question of how much previous information we should retain. Bottleneck implements a windowed Bayesian network with fading degree representing the extent of memory retained between windows. The fading degree varies between [0,1], with higher values signifying less memory.

5.5.4 Real-Time Evaluation of Bottleneck

Figure 30 evaluates Bottleneck on 5-minute evaluation windows with changing fading degree. It can be observed that the current evaluation does not result in any significant accuracy degradation compared to running the system in batch-mode. We

[18]BotHunter's three conditions are (1): evidence of (local host infection AND outward bot coordination or attack) (2): at least two distinct signs of outward bot coordination or attack (3): evidence that a host attempts communication with a confirmed malware site.

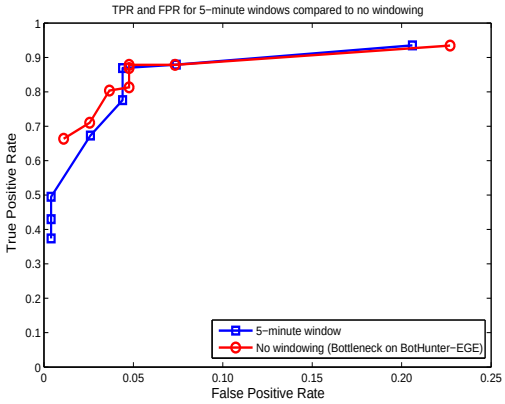


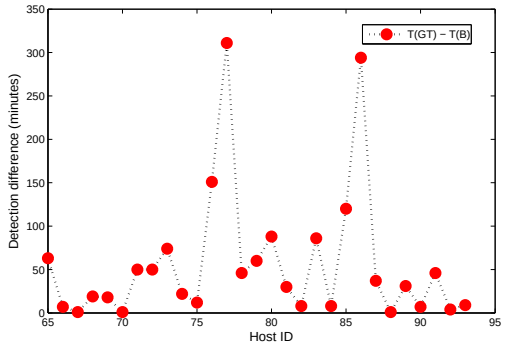
Figure 30: TPR and FPR for 5-minute windows compared to no windowing

thus conclude that Bottleneck is capable of highly accurate real-time bot detection. It can be observed that Bottleneck consistently provides high accuracy with 86.9% TP and 4.4% FP at the best operating point. Hence, Bottleneck provides 3.1% higher overall accuracy as compared to BotHunter.

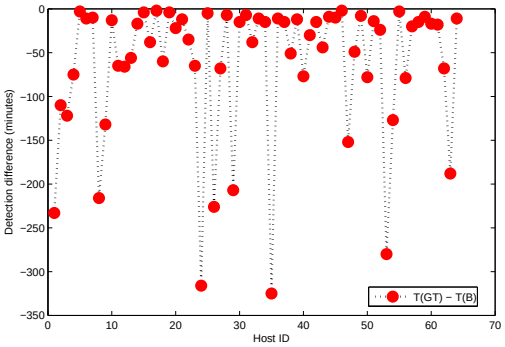
5.5.5 Detection Delay

Next we show the detection delay incurred by Bottleneck. We compute this delay with reference to the time the groundtruth flags a bot based on the observed C&C communication. We measure the delay for Bottleneck on a 5-minute window with a threshold of 0.3, Bottleneck's best operating point as seen in Figure 30.

Let T_{GT} be the time the groundtruth declares an IP as a bot and let T_B be the time Bottleneck detects it. Hence, T_{GT}-T_B is the delay incurred by Bottleneck with reference to the groundtruth. We segregate the detection delay incurred by Bottleneck into two subsets: the first subset constitutes 29 bot IPs where $T_B < T_{GT}$, i.e. Bottleneck's detection time leads the groundtruth (Figure 31(a)); the second subset constitutes 64 bot IPs where $T_B > T_{GT}$, i.e. Bottleneck's detection lags behind the



(a) Bottleneck lead time with respect to groundtruth



(b) Bottleneck lag time with respect to groundtruth

Figure 31: Bot detection delay incurred by Bottleneck.

groundtruth (Figure 31(b)).

We thus observe that when leading, Bottleneck leads the groundtruth detection by an average lead time of 57 minutes and a standard deviation of 77.5 minutes (Figure 31(a)). On the other hand, when lagging, the average lag time is 63.2 minutes with a standard deviation of 80.8 minutes (Figure 31(b)). However, overall Bottleneck lags with an average of 29.2 minutes and a standard deviation of 97.2 minutes; this delay is quite reasonable considering that we also get accuracy comparable to the batch mode operation.

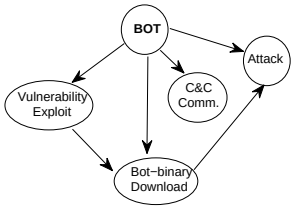


Figure 32: Learnt Bayesian Network.

Table 13: Accuracy Evaluation of the Bayesian network Learnt from Data

Threshold	TPR	FPR	FNR	TNR
0.5	95.9 %	0 %	4.1 %	100 %
0.6	95.9 %	0 %	4.1 %	100 %
0.7	95.9 %	0 %	4.1 %	100 %
0.8	95.9 %	0 %	4.1 %	100 %
0.9	95.9 %	0 %	4.1 %	100 %
0.95	95.9 %	0 %	4.1 %	100 %

5.5.6 Learning Bayesian Network from Data

Bayesian networks offer the ability to learn the Bayesian network structure from data.

Learning a Bayesian belief network includes: a) learning Bayesian network structure; and b) learning Bayesian parameters. Structure learning in Bayesian networks entails the learning of dependence and independence relationships between the domain variables. These are represented by uni-directional arrows leading from the cause to the effect variable. Parameter learning, on the other hand, is learning the conditional probability tables (CPTs) at each node. These parameters represent the conditional probability values for each state of the variable given each value of the parent node. For nodes with no parents, only prior-probabilities are specified. Once the network is learnt and parameters are set, it is then used for inference on the remaining 75% of the trace.

Figure 32 presents the Bayesian network learnt from data using the *Chow and Liu's* algorithm. Similar to earlier evaluations (Section 5.5), we trained the learnt Bayesian networks on 25% of the trace, by volume, for parameter learning. It was then evaluated for accuracy on the remaining 75% of the bot and non-bot traffic trace. Table 13 provides ROC-based evaluation of the learnt Bayesian network. The network provides an overall accuracy of 99.78% with 0 false positives. However, there are comparatively more false negatives as compared to the expert opinion based Bayesian network. Our analysis shows that missing causal relationships between nodes in the learnt model, decreased the impact of the parameter updates when new events occurred in the trace and consequently resulted in the missed detections.

5.6 Discussion of Blind Spots and Future work

In this section, we enumerate some limitations of the proposed Bayesian network based bot detection solution and propose countermeasures to mitigate these limitations.

- Like other probabilistic models, Bayesian networks are susceptible to slow poisoning and mimicry attacks. These attacks can effect both the parameter learning in Bayesian networks and the Bayesian classification leading to deliberate false positives. We believe that the poisoning of parameter learning can be prevented by either training online over sanitized and verified trace and updating the learned CPT values only after an expert vets that change. Furthermore, we can employ a larger window over which we completely reset the Bayesian

networks probabilities to their originally learnt priors. This will mitigate any slow poisoning attack.

- The accuracy provided by Bottleneck is limited by the mechanisms for event generation. We need to look into how different mechanisms can be added to the output of an EGE. Even more important is to define how these events will be combined. One option is to use simple majority, AND, and OR rules. A more interesting option to explore is to see if we can incorporate the different strengths and weakness of each mechanism into the way the Bayesian network updates its bot belief.

- Bottleneck, in its current state only provides bot detection and no defense capability. This can be incorporated by augmenting the current Bayesian network with utility nodes results in an influence diagram. The utility node is connected to the nodes on which we desire to implement policy decisions. A network administrator can simply add a utility table specifying their policy and corresponding actions can take place based on not only the belief of the BOT node, but other evidence nodes and their state.

- Bayesian inference algorithms are generally complex and require a decent processing to work. We need to evaluate this performance aspect of our algorithm to understand its feasibility for online operation.

5.7 Chapter Summary

In this chapter we proposed a machine learning-based specific-purpose ADS for the detection of bots. Bottleneck formulates its decision logic on a Bayesian network based on the high-level and abstract bot lifecycle. We show that the Bayesian network based decision logic provides comparable accuracy to rule-based bot detection tools, while providing superior performance when tested with changing traffic characteristics.

In the next chapter we model general-purpose anomaly detection systems and highlight that these systems are inherently susceptible to evasion margin estimation attacks. We propose to redesign anomaly detection systems and present some novel ADS design philosophies, to make them robust against evasion attacks.

CHAPTER 6: RETHINKING THE ADS DESIGN PHILOSOPHY

While the original models for intrusion detection system were proposed more than two decades ago [17], intrusion detection or more specifically anomaly detection, still remains an active area of research as the attacks continue to adapt and evade anomaly detection solutions [21]. In this chapter, we argue that there is a dire need to formally revisit the weak basis and assumptions on which anomaly detection systems have been designed thus far. We advocate a radically different open standards ADS design philosophy. in which the security of the system, instead of relying on the ADS detection methodology, is dependent on random network parameters.

After extensive evaluations of the current general-purpose anomaly detection systems we observed that ADS limitations in terms of low accuracies and susceptibility to evasion attacks, are deep rooted in the inherent design of these systems. Hence, we believe, there is a need to redesign the anomaly detection systems to enable them to offer acceptable accuracies to enable the commercial deployment of these systems. In the following sections, we highlight the high accuracies offered by statistical ADSs and develop a thorough understanding of an attacker's ability to design network attacks to evade state-of-the-art ADSs. We offer a few research directions to pursue a paradigm shift in the design philosophy of future general-purpose ADSs so that they are able to exhibit a high degree of robustness under any attack scenario.

6.1 Motivation

The last decade has witnessed a considerable shift in the malware trends and economics. Malware are now being designed specifically for espionage [122], to sabotage [103], and to cause large scale chaos and destruction [104]. As the detection methodologies have refined over the years, so have the attackers with stealthy malware propagation [15]. Thus, in addition to the exponentially increasing volume and impact of these new malicious code threats, the stealthiness, sophistication and impact of malware attacks are soaring at an alarming pace.

6.2 Challenges

Numerous general-purpose Anomaly Detection Systems (ADSs) have been proposed in the last few years to combat these rapidly evolving attacks. Some of these systems are now experiencing commercial deployments [123]– [23]. Over the last few years, the main research focus has been on either improving constituent components of the anomaly detection systems e.g. automating maintenance and calibration [124], optimizing training since training data is scarce [125] or providing detection solutions that build on top of existing anomaly detection systems [126]. These efforts have have not been able to make ADSs any more secure since adding more security layers does not necessarily mean more security. Hence, even today, the anomaly detection systems suffer from the same inherent limitations of low accuracies and susceptibility to attacks as they were when they were originally conceived. Hence, there is a need to revisit the ADS design philosophy and address the root cause of the limitations

inherent in the ADS design.

6.3 Statistical Anomaly Detection

It can be observed from the preliminary evaluations in Chapter 2 (Figure 3 and 7) that statistical ADSs, constituting both classes of the ADS domain (programmed as well as self learning) as shown in Figure 2, provide the best accuracy on both of the attack traffic datasets; LBNL (TRW [41]) as well as Endpoint (Maximum Entropy [47], TRW-CB [43], Kalman Filter [46]). Volumetric IDSs [32] [48] could not provide acceptable accuracy dividends on the LBNL dataset owing to the low rate of the constituent attacks. In contrast to volumetric IDSs, rule based ADS classifiers [44], [40] provided high detection rates albeit an unacceptable false alarm cost. This is deep rooted in the fact that that normal traffic behavior of a network changes with time.

Moreover statistical IDSs, due to their inherent stochastic nature, are considered more rigorous against malware attacks [18] as opposed to rule based systems and simple thresholded IDSs that need to be constantly updated with changes in network characteristics and also due to their complete inherent reliance on human factor for detection of anomalies. However, the statistical IDSs rely heavily on the underlying statistical model and parameters that mirror the network traffic characteristics. Due to their reliance on the network traffic, an intelligent and a resourceful attacker can bypass such an ADS by skillfully crafting the probes that it sends into the network so that it stays just below the radar and evades an ADS's detection. In the next section,

we stochastically model such an evasion process and show that it is rooted into the ADS detection methodology used by the ADSs today. We also provide experimental results in the form of ROC curves for the three most prominent and benchmarked statistical ADSs in Figure 3 and 7 [47], [41], [43].

6.4 Stochastically Estimating the Evasion Margin

Many research efforts in the recent past have been focused towards evading anomaly detection systems [127]– [28]. However, as the attacks became increasingly sophisticated, the ADSs grew weaker in comparison [128]. We argue that the prevalent ADS design methodology is fundamentally flawed [129] because currently the anomaly detection algorithms rely on the premise that the underlying detection principle is not known to the attacker. This assumption, however, does not hold in real world where some knowledge about the ADS principle can be obtained through several methods like social engineering, fingerprinting, trial and error, etc. [130]. In fact, in many scenarios an ADS can be evaded without knowing the exact design principles; several types of attacks (e.g., polymorphic blending attacks [130], [28], mimicry attacks [131], portscans [7], etc.) have been proposed in existing literature.

Throughout this section, we assume the role of an attacker. As a proof-of-concept, we will present stochastic methods to estimate the evasion margin for two existing, prominent and diverse statistical ADSs [47], [41], [43]. We will use real-world attack traffic datasets to provide preliminary experimental results to demonstrate the evasiveness of our methods.

6.4.1 Breaking the Maximum Entropy ADS

Maximum Entropy [47] is a general-purpose anomaly detection system that uses information theoretic methods to identify anomalies. In the following subsections, we briefly describe the detection principle of this ADS and then present stochastic methods for the estimation of basic ADS configuration parameters.

6.4.1.1 Detection Principle

The Maximum Entropy detector employs information theoretic Kullback-Leibler divergence measure for anomaly detection. The KL measure computes how much the baseline distribution $p(\omega)$ (obtained from benign traffic profiles) varies from the real-time distribution $q(\omega)$ (which might contain benign as well as malicious traffic traces). The traffic is divided into 2348 packet classes based on the destination ports and the protocol. The detector uses maximum entropy estimation to develop the baseline distribution for the traffic classes. Since the attacker does not know the network topology and/or the services running within the network, it tends to communicate with hosts that do not exist or hosts that do not have the requested service available. Thus the malicious packets from the attacker would considerably alter the real-time traffic characteristics from the baseline distribution. The Maximum entropy detector analyzes W real-time windows for attack detection in each traffic class. If the divergence between the baseline and the real-time distributions for a particular packet class exceeds the threshold τ_{KL} in h of these W windows, an anomaly is flagged by the detector.

6.4.1.2 Stochastic Methods for ADS Parameter Estimation

Aim: Our main aim, as an attacker, is to estimate the evasion margin for the ADS. This evasion margin will be the bound on the number of scan packets (t) that an attacker can send to a target entity (host/network) without detection.

The following basic configuration parameters need to be estimated by the attacker to achieve the above listed aim: (a) Baseline Distribution, $p(\omega)$; (b) Real-time Distribution, $q(\omega)$; (c) KL threshold, τ_{KL}. Let us assume that during a time window δ, the following packet sequence is observed at the end-host or the network gateway: $\{x_1, x_2, x_3, \ldots, \hat{x_i}, \hat{x_{i+1}}, \ldots, \hat{x_{i+t}}, \ldots, x_n\}$. The window constitutes t scan packets, represented by $\hat{x_i}$. Moreover let the real-time benign packet class distribution in time-window δ be represented by ${q_B}^{(\delta)} \equiv q_B$. The KL divergence measure for a particular packet class ω in time window δ is given by:

$$D_{p||q}(\omega) = p(\omega)\log_2\left({p(\omega)}/{q(\omega)}\right). \tag{26}$$

Let us consider that the attacker sends t scan packets to the target entity. This would alter the runtime distribution in the δ time window. This new distribution of the attacked packet class ω constitutes benign and anomalous scan packets (${q_{B,M}}^{(\delta)} \equiv q_{B,M}$):

$$q_{B,M}[\omega] = \frac{f_{B,M}[\omega]}{n} = \frac{\sum 1(x_i \in \omega)}{n}, \tag{27}$$

where n are the total number of packets observed during time window δ and $\sum 1(X)$ is an indicator function that takes the value 1 if X is true and 0 otherwise. Resultantly, the scan packets will alter the frequency of the attacked packet class:

$$f_{B,M}[\omega] = f_B[\omega] + t.$$

$$\Rightarrow q_{B,M}[\omega] = \frac{f_B[\omega] + t}{n} = \frac{f_B[\omega]}{n} + \frac{t}{n} = q_B[\omega] + \frac{t}{n}. \tag{28}$$

Real-time distribution of the non-attack classes $\widetilde{\omega}$ which can be represented as:

$$q_B[\widetilde{\omega}] \approx \frac{f_B[\widetilde{\omega}]}{\sum_{i=1}^{2348} f_B[x_i] + t} \approx \frac{f_B[\widetilde{\omega}]}{n}.$$

We can now compute the number of scans that the attacker can send to the target using Eq. 26:

$$D_{p||q}(\omega) = p(\omega) \log \left({}^{p(\omega)}/_{q_{B,M}(\omega)} \right) < \tau_{KL}.$$

However, $p(\omega) < q_{B,M}(\omega)$, since the extra scan packets would result in increased probability of the attack packet class. Thus

$$D_{p||q}(\omega) = p(\omega) \log \left(\frac{p(\omega)}{q_{B,M}(\omega)} \right) > -\tau_{KL} \Rightarrow \left\{ 2^{{}^{\tau_{KL}}/_{p(\omega)}} p(\omega) - q_B[\omega] \right\} \times n > t. \tag{29}$$

Eq. 29 gives the upper bound on the number of scan packets that the attacker can send into the network while just remaining below the detection threshold. Thus, for the attacker to estimate the number of evasion scans, it would have to estimate the baseline distribution $p(\omega)$, the runtime distribution $q(\omega)$ and the threshold τ_{KL}. The following sub-sections describe in detail the statistical estimation of these parameters.

6.4.1.2.1 Estimating the Baseline Distribution $p(\omega)$

Baseline distribution is learnt during the training phase on benign traffic profiles, before ADS deployment. Here we list a few scenarios for the estimation of such a benign distribution.

Real time Observation: As proposed in [130], a realistic attack scenario can be that the attacker compromises a host X in network A which communicates with the target host in another network or the target network itself. Once the attacker has control over host X, it can observe the normal traffic from host X to the target entity. From this observed normal traffic, the attacker can estimate the baseline probability mass function $\widehat{p}(\omega)$ for the attacked packet class ω (i.e. the class the attack wishes to exploit), since the attacker knows the type of services running on the remote target host.

Brute force: One of the most common modus operandi used extensively in crypt-analysis is the brute force estimation. Despite its computational complexity, it has been shown that with the current high-performance COTS (multithreaded and multi-core) hardware, it is not difficult for a craft attacker to acquire and exploit hardware parallelism to carry out a bruteforce analysis [130], [132]. Let ω be the attacked packet class with λ instances. For the brute force technique to estimate the baseline distribution, all possible combination of the connection instances would have to be experimented. This would undoubtedly require multiple IP addresses to launch the attack simultaneously. The attacker can set the initial number of connections for all instances to unity: $p(\omega) = \{\omega_1 = 1, \omega_2 = 1, \omega_3 = 1, ...\omega_\lambda = 1\}$. Then, by trial and

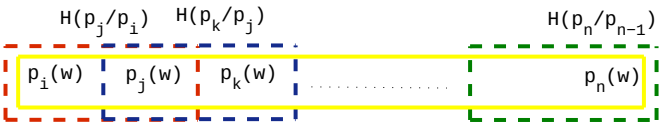


Figure 33: Conditional entropy calculation for threshold estimation.

error, the attacker can vary the connections per instance and observe the resultant behavior of the target network/host.

6.4.1.2.2 Estimating the Detection Threshold τ_{KL}

In a recent paper [133], the Co-PI proposed a Markovian stochastic model of temporal dependence in an ADS' anomaly scores. We used conditional entropy analyses for determining the order of the Markov chain that should be used for threshold estimation. While the motivation for the original work was to improve the accuracy and automation of an ADS, the threshold prediction algorithm can be adapted to estimate the KL threshold for the present attack scenario. Let us assume that the attacker has access to the real time distribution $p(\omega)$ as described above. The attacker observes $p(\omega)$ in n non-overlapping time windows. Using the first order Markov chain (where the probability of choosing the next state is only dependent on the current state), he/she can compute the conditional entropy of two random variables one in each consecutive time window (which characterizes the information remaining in one random variable when the other is already known); for the estimation of the threshold bound. Unlike [133], our main aim is to find the random variables (i.e. benign distributions) at two time indices i and j $\forall$ $i \neq j$ such that random variable at i gives minimum

information about random variable at j. Since in large order Markov chains (i.e. the number of previous states that the current state depends upon), random variables in previous time windows tend to provide some information (if not none) about the present random variable, our aim is to find the minimum information overlap. Therefore, we restrict the evaluation of conditional entropy to first order Markov chains only.

$$H\left(p_j|p_i\right) = -\sum_{\omega} p(p_i, p_j) \log\left(p\left(p_j|p_i\right)\right).$$

The conditional entropy $H(p_j|p_i)$, of two random variables p_i and p_j correspond to the information in p_j not given by p_i. Thus, computing the maximum conditional entropy between baseline distributions in two consecutive time bins, as we slide from bin 1 to bin n, can provide us the minimum information overlap in normal benign data. This is shown in Figure 33 and can be stochastically modeled as:

$$H_{\max} = \max_{i,j \in \{1,2 \ldots n\}} H\left(p_j|p_i\right).$$

This minimum information overlap can be used to identify the acceptable divergence bounds for normal traffic for which the detector does not raise an alarm. We evaluate the KL measure on benign distributions in non-overlapping windows to maximize the corresponding conditional entropy. These two benign distributions, pertaining to the attacked packet class ω, can be used for the KL divergence measure computation of Eq.26. This KL divergence, for a particular packet class ω, gives the maximum variation of the real-time distribution from the baseline that can go undetected through the Maximum Entropy ADS. Thus, greater are the number of benign

windows n that we analyze, greater the expected divergence [133].

6.4.1.2.3 Estimating the Real-time Distribution $q(\omega)$

Attacker needs to estimate the real-time distribution in time window δ. The estimate $\widehat{q_B}^{\delta} \equiv \widehat{q_B}$ can be easily obtained from Eq. 29 as:

$$\widehat{q_B}[\omega] < 2^{\frac{\tau_K L}{p(\omega)}} p(\omega). \tag{30}$$

Equipped with these estimates of the baseline distribution and the KL threshold, we show that an attacker can launch a successful evasion attack against the ADS.

6.4.2 Sequential Hypothesis Testing based TRW ADSs

TRW-based ADSs are statistical pre-programmed portscan detection classifiers [41], [43]. In this section, we provide a brief description of the basic detection principle employed by these detectors and then explain the evasion margin estimation by the attacker.

6.4.2.1 TRW Detection Principle

Both the sequential hypothesis testing based classifiers employ likelihood ratio testing for anomaly detection. However, the original TRW classifier [41] detects remote scanners (i.e., port scanners located outside the local network perimeter) while Credit-based TRW (TRW-CB) [43] detects local scanners (i.e., outgoing port scans in a network). Both these algorithms have been shown to be quite accurate and commercial ADSs also deploy these algorithms for portscan detection. The basic TRW detection

algorithm is as follows.

Let r be a host being observed by the ADS. For a given host, let Y_i be an indicator random variable with the following possible outcomes:

$$Y_i = \begin{cases} 0 & \text{if connection attemp is a success} \\ 1 & \text{if connection attemp is a failure} \end{cases}$$

Thus, Y_i is a bernoulli random variable indicating the outcome of a connection attempt towards/from a host. The sequential hypothesis testing technique considers two hypotheses: H_0 is the hypothesis that the host under observation is benign; and H_1 is the hypothesis that host is a scanner. Moreover, following a priori probabilities:

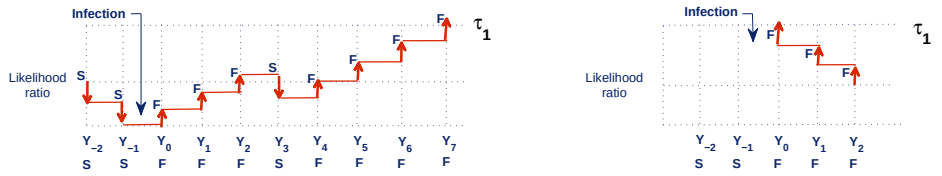
$$\begin{array}{ll} \Pr[Y_i = 0/H_0] = \theta_0 & \Pr[Y_i = 1/H_0] = 1 - \theta_0 \\ \Pr[Y_i = 0/H_1] = \theta_1 & \Pr[Y_i = 1/H_1] = 1 - \theta_1 \end{array} \quad \forall \quad \theta_0 > \theta_1 \ . \tag{31}$$

Since the scanners tend to connect to host that do not exist or hosts that do not have the requested service available [41], hence the probability of a connection to be a success is much higher for benign hosts than for scanners. Both TRW and TRW-CB leverage the above observation to detect portscans using the likelihood ratio test:

$$\Lambda(Y) = \prod_{i=1}^{n} \frac{\Pr[Y_i/H_1]}{\Pr[Y_i/H_0]}. \tag{32}$$

Thus each connection attempt is analyzed for likelihood calculation. Algorithmic details are provided in the following subsections.

TRW [41] employs the forward likelihood ratio test to identify if remote hosts are scanners. Each entity (host or network) separately computes $\Lambda(Y)$ for the incoming connection attempts from remote hosts and when it exceeds the threshold τ_1, the



(a) TRW Sequential Likelihood ratio test (b) Reverse sequential hypothesis testing in TRW-CB

Figure 34: TRW and TRW-CB SHT.

remote host is termed as a scanner. Figure 34 (a) illustrates how the likelihood ratio proceeds forward as the remote host scans the local entity for vulnerability detection.

TRW-CB [43] detects local hosts as scanners employing reverse sequential hypothesis testing (HT) to the connection attempts originating from the local entity (host or network). TRW-CB augments the HT based anomaly detection mechanism with credit based rate limiting [32]. It computes the likelihood ratio test in Eq. 32 in reverse chronological order. Thus, the HT proceeds with the most recent observation first (i.e. $\Lambda\left(Y_2, Y_1, Y_0, Y_{-1}, Y_{-2}\right)$) as shown in Figure 34 (b). Moreover, the credit-based rate limiting algorithm limits the number of first contact connections by issuing a fixed number of credits C to each host pertaining to the number of open connections that a host can afford.

6.4.2.2 Stochastic Methods for TRW Parameter Estimation

While TRW based detectors provide high accuracy dividends at both the endpoint and gateways [25], they suffer from the same inherent ADS limitation i.e. once the detection methodology is known, it is trivial to bypass the classifier. To achieve this aim, the following parameters need to be estimated: (a) a priori probabilities

(Eq. 31); and (b) threshold τ_1. The classifiers use two threshold values, the upper threshold used to identify scanners and the lower threshold to establish the host as benign. However in practical ADS deployments, the hosts are continually monitored unless found infected. Thus, we considered the upper threshold value only.

Estimating a priori probabilities Since TRW-based ADSs are pre-programmed classifiers, the a priori probabilities and the threshold are built into the system; suitable values suggested in [41] are: $\theta_0 = 0.7$ and $\theta_1 = 0.1$. As mentioned before, we are assuming attackers have multiple IP addresses at their disposal for launching an attack. This is a realistic assumption since the attacker can have multiple compromised systems that it can use to launch an attack against a target entity (host or network).

Let us consider an attacker that sends n_i random scan packets to/from a particular entity to identifying the services running on the target hosts and/or servers. Let us assume that $n_i(s)$ of these connections were successes and $n_i(f)$ were failures. This connection information can be used to easily estimate the a priori probabilities as follows:

Probability of connection success = $\theta_1 = {}^{n_i(s)}/_{n_i}$;

Probability of connection failure = 1 - $\theta_1 = {}^{n_i(f)}/_{n_i}$.

According to Eq. 31, the successful connection probability for benign hosts is greater than for scanners. With these estimated values for θ_1, θ_0 can be set to a value greater than θ_1. Likelihood ratio test is computed on two terms: ${}^{\theta_1}/_{\theta_0}$ and ${}^{1-\theta_1}/_{1-\theta_0}$. Since scanners tend to experience more failed connection attempts, the second term would have a greater impact on the likelihood ratio of Eq. 32. Thus greater the value of θ_0 as compared to θ_1, higher the impact of connection failure on

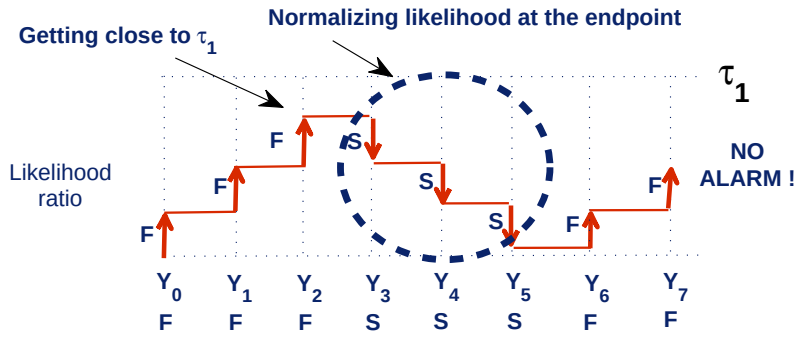


Figure 35: Attack scenario for evading TRW detection.

the subsequent likelihood ratio computation.

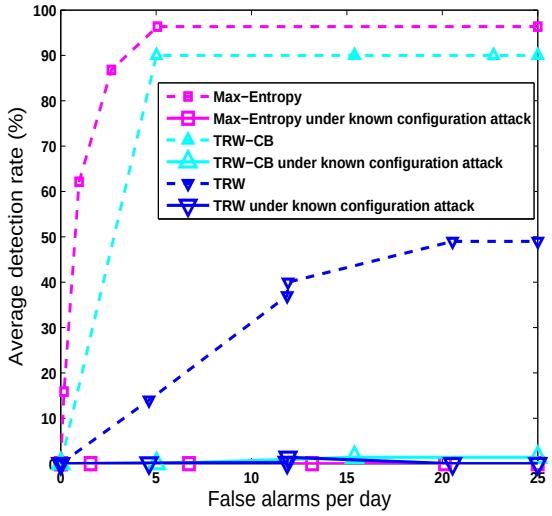
Estimating TRW's Threshold Threshold value can be accurately estimated by experimental evaluation. Using the above a priori probabilities, the attacker can scan a target entity and observe the corresponding likelihood ratio values generated with each new scan packet. The attacker can scan computers such that it knows whether a scan will be successful or not. A local scanner can easily achieve this by setting up remote computers on the Internet; in fact, multiple virtual hosts on one remote computer can also provide the same results. For a remote scanner, the attacker can keep track of previous successful scans. These prior successful scans can be used to bring the likelihood ratio down whenever required. When the attacker exceeds the threshold, any new connections from the attacker would be deferred by the ADS. The likelihood ratio test value when this happens can be set as a bound for the upper threshold τ_1.

6.4.3 Dataset Formation

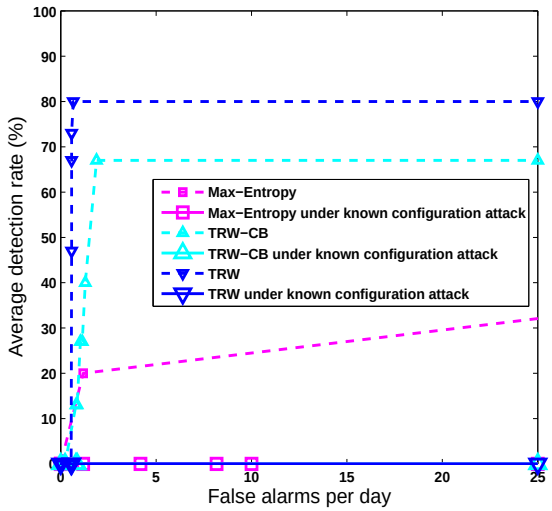
Two datasets, with complementary strengths, were used for the ROC evaluation given in Chapter 2. We use benign and attack traces from the same datasets in the following evaluation. However, here we explain the formation of the datasets by assuming the role of an attacker.

Using the stochastic estimates of Section 6.4.1, the number of scan packets t, were computed using Eq. 29. For both the datasets, t scan packets for each attacked class, pertaining to the specific malware, were inserted in each time window δ. For example, for Blaster worm, attack sessions were inserted for class pertaining to TCP port 135. For malware that spans more than one attack class, t attack sessions for each attack class were computed and inserted in each time window.

In the TRW-based detectors we interleave the scanning sessions with the benign sessions such that the estimated threshold τ_1 is not exceeded. Using the parameter estimates as presented in Section 6.4.2, scan packets were interleaved with benign connection traces, as shown in Figure 35. With the TRW ADSs, the attacker has the ability to itself compute the likelihood ratio that would be used at the target entity for the evaluation of the maliciousness of the attacker. Thus, scan packets were inserted in a window unless the likelihood ratio was ε (error threshold) close to the threshold τ_1. When this limit was reached, successful benign connections were inserted to normalize the likelihood.



(a) Endpoint dataset



(b) LBNL dataset

Figure 36: ROC analysis of TRW-based ADS classifiers under configuration estimation attack.

6.4.4 Preliminary Experimental Results for the Proposed Attack

Figure 36 provides an ROC analysis for the Maximum Entropy, TRW and Credit-based TRW classifiers on the original dataset and on the newly formed dataset with parameter estimation. Maximum Entropy detector provided the best accuracy div-

idends on the endpoint dataset as was shown in Figure 3 in Chapter 2. However, the detector could not maintain its accuracy on the gateway LBNL dataset. On the new datasets formed with ADS parameter estimation, the detector was completely paralyzed and all attacks, without exception, were able to bypass the ADS. Thus, an intelligent attacker can reliably compute the scans that it can send into the network without detection. Similarly, the TRW based detectors have also been unable to detect intrusions by the attacker (local and remote), as shown in Figure 36. On the Endpoint dataset some detections were observed. These were due to the estimation errors in the parameter computations. However these detections are negligible compared to the aggressiveness of the attacker. Thus the ROCs show that an intelligent attacker can easily paralyze and bypass even the most accurate of the ADS.

In light of these preliminary evaluations, we reiterate the fact that the current anomaly detection systems need to be majorly redesigned owing to the disability of the ADS detectors to maintain accuracy in the presence of a resourceful and an intelligent attacker.

6.4.5 Discussion

Hence, the last section has reiterated our claim that while some ADSs are able to provide high accuracy dividends at specific points of network deployment, these can be bypassed by intelligent stochastic analysis and estimations. The experimental evaluations of Figure 3, 7 and 36 authenticate our claims. Consequently, we argue that there is a need to revisit the weak basis on which anomaly detection systems are still being designed. In the next sections, we aim to present some preliminary

evaluations of a few proposals for ADS redesign that we are currently working on.

6.5 Cryptographically-Inspired Anomaly Detection System Design

In the previous sections, we have highlighted that, while a few current IDSs provide acceptable accuracy dividends at specific deployment points, these systems are inherently trivial to bypass. The fundamental weakness of these IDSs is the fallacious design premise that the attacker does not have the knowledge of the underlying system used for detection. In this section, we provide arguments in support of a cryptographically-inspired rethinking of the ADS detection philosophy in accordance with Kerckhoff's principle of cryptography.

Cryptographic Systems: Cryptographic cipher systems employ randomness for the protection of communication data [134]– [135]. Principally randomness is introduced by two factors: permutation and substitution. Thus data is transformed and replaced using secret keys until the ciphertext is so randomized that deciphering the original plaintext is infeasible. We argue that this notion of randomness can also be used for securing traffic information in anomaly detection systems by introducing uncertainty to the baseline distribution using secret key transformations. The remaining of this section describes some cryptographically inspired designs for ID Systems.

Strategy: The underlying principle of statistical anomaly detection is that the normal data instances occur in high probability spaces of the stochastic model, while intrusive instances do not [18]. Thus in non-parametric statistical anomaly detection, the baseline distributions are obtained from the benign traffic profiles. The real-time

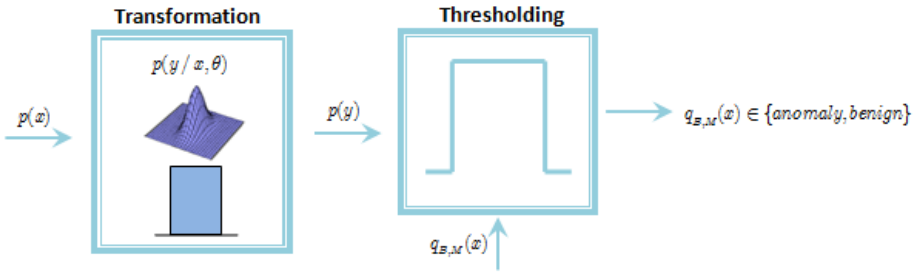


Figure 37: Partitioning the ADS detection process.

distributions, during actual ADS deployment, are compared with the baseline distributions for the detection of intrusions using statistical analysis. Since the baseline and the run-time distributions can be obtained by the attacker, as was explained in Section 6.4, the information about these distributions can be manipulated by the attacker to its advantage. Thus, we propose that the baseline distribution should be randomized using a secret parameter before statistical evaluation. Thus the cryptographically inspired transformation as well as the added randomness would add robustness to the system thereby preventing the attacker to develop an estimate for these configuration parameters. We propose that the complete ADS thresholding be partitioned into two phases: transformation and thresholding. This is shown in Figure 37. Thus, in the first phase, the baseline distribution is transformed using transformation matrix which is described in detail in the next section. The ADS detection principle is then applied to the transformed baseline and the real-time distributions for the detection of maliciousness in the traffic. The main aim is to transform the baseline distribution based on a secret key which consequently has a large configuration space, thus making it impossible for the attacker to guess/estimate these parameters. In the following sections, we propose a few channel coding based models of the ADS configuration

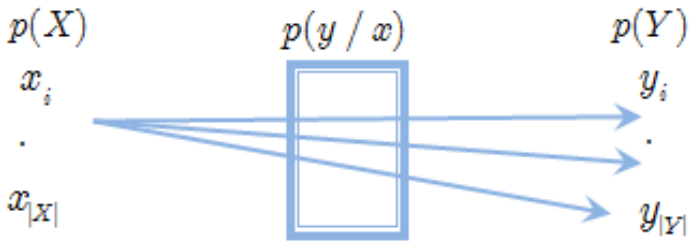


Figure 38: Discrete memoryless Symmetric channel.

parameters that can be used to introduce this randomness.

6.5.1 Modeling ADS detection as a Information Channel Coding problem

Discrete memoryless symmetric channels [136] are characterized by an input alphabet X, an output alphabet Y, and the conditional probability mass function $p\left(y/x\right),\ \forall x \in X,\, y \in Y$. Moreover, since the channel is memoryless and symmetric, the output of the channel depends only on the current input i.e. $p(y) = f(p(x))$ and the alphabet size is the same i.e. $|X| = |Y|$. Such a channel is shown in Figure 38. The channel can be represented in the form of a matrix called channel transition matrix represented as:

$$p(y/x) = \begin{bmatrix} p\left(y_0/x_0\right) & p\left(y_1/x_0\right) & . & . & . & p\left(y_{|Y|-1}/x_0\right) \\ p\left(y_0/x_1\right) & p\left(y_1/x_1\right) & . & . & . & p\left(y_{|Y|-1}/x_1\right) \\ . & & & & & \\ . & & & & & \\ p\left(y_0/x_{|X|-1}\right) & p\left(y_1/x_{X|-1}\right) & . & . & . & p\left(y_{|Y|-1}/x_{|X|-1}\right) \end{bmatrix}. \tag{33}$$

Such a model is used in information channel capacity domain to maximize the channel capacity based on the input distribution $p(x)$ for a given channel $p(y/x)$.

We can use information channel coding as a baseline to model our information randomization process. The channel transition matrix can be used to perform linear transformations of the baseline distribution in each time window with the same alphabet size ω at the output (i.e. $|X| = |Y| = \omega$). The matrix can be parameterized with a random secret key θ, that can be the source of randomness in the ADS detection process. The transition matrix can hence be represented as $p(y; \theta/x)$. Moreover, each secret key value has a large configuration space. Thus, this two tier randomness: secret key for configuration space selection and the specific configuration parameters from within that space would render the detection process impractical for the attacker to evade. The remaining of this section describes a method for the randomization of the ADS detection process through the parameterized transition matrix $p(y; \theta/x)$.

6.5.1.1 Minimizing the mutual information

Our main aim is to transform the input baseline distribution $p(x)$ before ADS detection is performed on the real-time distribution $q_{B,M}(x)$. Thus the attacker not knowing the secret key and the ultimately the parameter configuration would not be able to determine the evasion margin. Thus, in order to achieve this objective, we need to minimize the mutual information between the input baseline distribution $p(x)$ and the transformed distribution $p(y)$. Mutual information is the measure of the information overlap between two random variables (or distributions). If one of the random variable is known (e.g. X and consequently $p(x)$) then the other one can be predicted if the information overlap between the two random variables is considerable. Mutual information between two random variables X and Y, in terms of the transition

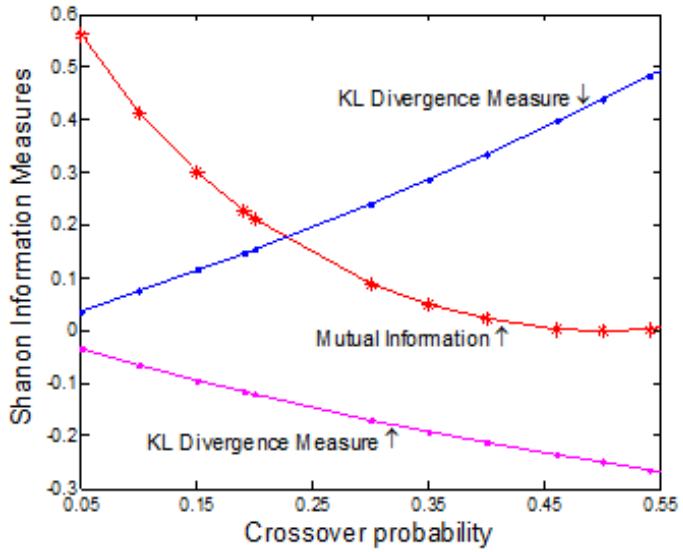


Figure 39: Dependence of information measures on crossover probability.

matrix, is given by:

$$I(X;Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2\left(\frac{p(y;\theta/x)}{p(y)}\right) = \sum_{x,y\in\omega} p(x,y)\log_2\left(\frac{p(y;\theta/x)}{\sum\limits_{x'\in\omega} p(x')p(y;\theta/x')}\right). \tag{34}$$

Our cryptographical-inspired ADS selects a transition matrix that minimizes the information overlap between the input baseline distribution $p(x)$ and the output $p(y)$, i.e.

$$\min_{p(y;\theta/x)} I(X;Y) = \min_{p(y;\theta/x)} \sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2\left(\frac{p(y;\theta/x)}{p(y)}\right) = \sum_{x,y\in\omega} p(x,y)\log_2\left(\frac{p(y;\theta/x)}{\sum\limits_{x'\in\omega} p(x')p(y;\theta/x')}\right). \tag{35}$$

The probability that the input x_i will be mapped to the output y_j where $i \neq j$ is called the crossover probability. This is shown in Figure 38. As the cross over probability increases, the corresponding mutual information between the input baseline distribution and the output decreases. However, as we minimize the mutual information measure, we consequently increase the divergence between the input and

output baseline and the real-time distribution. This is shown in Figure 39. It can be observed that the mutual information decreases as the crossover probability increases and ultimately becomes zero at approximate probability value of 0.5. Thus our legitimate interval for the crossover probability is limited to $\{0.0.5\}$. Figure 39 also gives the divergence measure for two classes in $p(x)$. It can be seen that as the crossover probability increases, so does the divergence between the input and the output distributions. Thus the amount of divergence should be parameterized (by Δ), based on the secret key θ for randomization. Thus, the mutual information is minimized based on the transition matrix $p(y;\theta/x)$, provided the divergence between the input baseline distribution $p(x)$ and the output $p(y)$ does not exceed beyond Δ, which is a function of the secret key θ. Thus, this would result in randomizing the baseline distribution used for anomaly detection by the ADS.

6.5.2 Modeling ADS detection as an Information Source Coding problem

Here we reiterate the fact that the basic purpose of transforming the baseline distribution is to conceal and obscure so as to make it impossible for the attacker to estimate the underlying ADS parameters. This is achieved by parameterizing the transformation with the secret key which randomizes the resultant process. Another approach to model the $p(x)$ transformation is modeling it as a source coding problem. Hence, we propose to randomize the input random variable (X) based on a source information measure by transforming these outcomes using a uniform transition matrix $(U(1,\omega))$ confining the probability variations to an interval $\{0,\phi\}$ with maximum

divergence variance of Δ which is dependent on the value of the secret key θ. Thus for the secret key defined divergence variation of Δ, many possible probability limits exist, any of which can be selected by the ADS for transition matrix computation $p(y;\theta/x)$. Subsequently, the transformation module in Figure 39 can be further split into two constituent phases as shown in Figure 10. These are described in detail in the following subsections:

6.5.2.1 Information Content based instance selection

The baseline distribution mostly constitutes instances that contain approximately the same amount of information, except a few that have a large information variance. Today the malware targets one of these two traffic class categories based on his/her intend to identify either the well known applications running on well known ports pertaining to traffic classes which occur with high probability i.e. low information content unlike the remaining less probable classes. Thus, a plausible transformation scheme can be based on the information content of these classes to consequently prevent the attacker from developing an accurate estimate. In order to achieve this objective, we propose to use the information content ($I(x_i)$) or the expected value of the information content (i.e. entropy $H(x_i)$) of the instances of the baseline distribution to define the uniform transition matrix $p(y;\theta/x)$. These information measures present the amount of information content associated with each outcome in the image of the random variable. The self information and the entropy for the i^{th} outcome in

the baseline distribution $p(x)$ is given by:

$$I(x_i) = \log_2\left(\frac{1}{p(x_i)}\right), \quad H(x_i) = E\left\{\log_2\left(\frac{1}{p(x_i)}\right)\right\} = p(x_i)\log_2\left(\frac{1}{p(x_i)}\right). \quad (36)$$

The instances of higher probability are more likely to occur in the real-time distribution and thus have lower information content. The less likely instances however have higher information content. Based on this similarity between the instances of the baseline distribution, the transition matrix $p(y;\theta/x)$ can be defined as a uniform stochastic distribution as follows:

$$p(y;\theta/x) = \begin{cases} f(p(x_{i,j}),\phi) & I(x_i) \approx I(x_j)\,\forall i,j \in \omega \\ 0 & \text{otherwise,} \end{cases} \quad (37)$$

where ϕ defines the maximum uniform probability transformation $\left(1/\phi\right)$ for the outcomes with similar information content $I(x_i)$ (or entropy $H(x_i)$) such that the maximum divergence between the input baseline and the output does not exceed Δ (defined as a function of the secret key); $f(p(x_{i,j}),\phi)$ can be an equiareal geometric transformation function (e.g. scaling, shearing etc.); and ω are the total traffic classes constituting the baseline distribution. The secret key determines the maximum variation of the output distribution from the input, which consequently determines the acceptable bound on the maximum probability variation for the similar ($I(x_i) \approx I(x_j)$) instances of the baseline distribution.

6.5.2.2 Transformation & Normalization

Once the transition matrix is defined, based on the similarity of instances constituting the baseline distribution, instances of $p(x)$ are transformed by $p(y;\theta/x)$ and

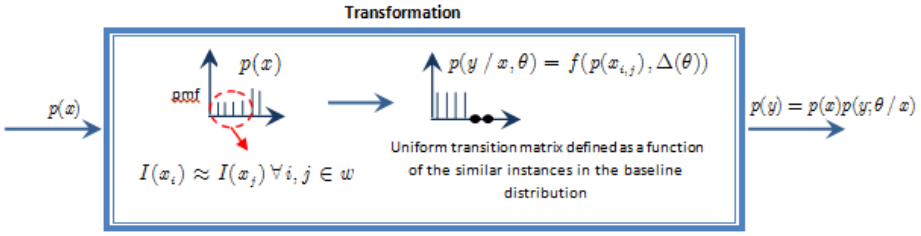


Figure 40: Partitioning the ADS Transformation process.

then normalized to form a valid $p(y)$. Normalization would result in a decrease in the probability of occurrence of the remaining instances not selected for transformation. Thus, the transformed baseline distribution $p(y)$ can be built with respect to either the high information or the low information instances. However, the degree of transformation is bounded by the parameter Δ which is defined as a function of the secret key θ. Thus greater is Δ, greater is the allowed deviation of transformed $p(y)$ (i.e. greater can be the value of ϕ) from the baseline distribution $p(x)$ and vice versa.

In the next section we describe how the baseline and run-time distributions and parameters can be used to evaluate the accuracy of the ADS in each time window.

6.5.3 Anomaly Detection

Let the maximum acceptable divergence between the input baseline distribution and the output is Δ. This divergence is represented in Figure 41 by the red circle on the left. The circumference of the circle marks the maximum divergence limit for the transformation process. Thus, the information overlap between the input $p(x)$ and the output $p(y)$ is minimized based on this value of the divergence Δ. Thus for accurate detection, the real-time distribution $q_{B,M}(x)$ should be compared with both the original baseline distribution $p(x)$ and the transformed $p(y)$.

A robust attacker would intend to cause maximum damage to the network in terms of intrusions while just staying within the evasion margin. Since the attacker can only estimate the baseline distribution $p(x)$ but cannot estimate $p(y)$, the real-time distribution comprising attack traffic ($q_{B,M}(x)$) would tend to lie on the boundary of the red circle on the left; an attacker would always try to maximize the attack impact without detection. However, the boundary of $p(x)$, can be partitioned into two spaces for attack detection: non-overlapping boundary space; and overlapping boundary space between $p(x)$ and $p(y)$. Then in order to differentiate intrusions from legitimate benign activity within a time window, the ADS would need to threshold on the following two conditions:

- Anomaly detection on the non-overlapping boundary region between $p(x)$ & $p(y)$:

$$D(p(x)||q_{B,M}(x)) \approx \Delta => D(p(y)||q_{B,M}(x)) \approx 2\Delta.$$

- Anomaly detection on the overlapping boundary region between $p(x)$ & $p(y)$:

$$D(p(x)||q_{B,M}(x)) \approx \Delta => D(p(y)||q_{B,M}(x)) << \Delta(\approx 0).$$

Similar thresholding conditions can also be defined for benign real-time distribution as follow:

- Benign traffic window:

$$D(p(x)||q_B(x)) << \Delta => D(p(y)||q_B(x)) \in \{(< 2\Delta \cap > \Delta) \cup \approx \Delta\}.$$

Thus, an attacker who has complete knowledge of the working of the system will

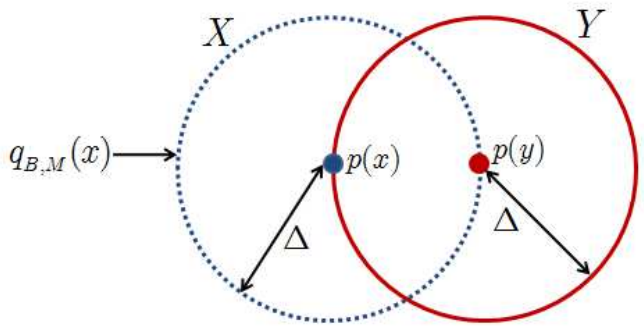


Figure 41: Transformation of the baseline distribution.

not be able to evade the ADS due to the inherent randomness in the proposed design methodology.

In this section we proposed to redesign the ADS by randomizing the normal baseline distribution to make it practically infeasible for the attacker to correctly estimate it. In the following section we propose to randomize the ADS feature space to achieve the same objective of making parameter estimation infeasible for the attacker.

6.6 Moving Target-based Randomized Feature Space

In this section we propose to redesign the current anomaly detection systems by enabling these systems to perform detection by mutating between multiple feature dimensions instead of relying solely on a specific feature space. We call these multi-dimensional features the Moving target-based feature space (MTFS).

In current anomaly detection systems, features used for detection do not vary throughout the operational life of the ADS. Moreover, the statistical feature distributions of the underlying static feature space are thresholded for anomaly detection. Hence as shown in Section 6.4, the anomaly detection system can be easily evaded by

disseminating stealthy malware that stays below the detection threshold. This is not necessarily a daunting task due to numerous off-the-shelf tools available to perform network traffic forensics. The current anomaly detectors employ fixed features for detection purposes. This makes evading these AD systems possible. Hence, in order to cater for this inherent limitation in anomaly detection systems, we propose a moving target (MT) defense based anomaly detection. The MTFS based detector mutates the underlying detection feature(s) across time. Thus at different time instances different features are employed for detection of network attacks. This makes it difficult for the adversary to predict the detection feature(s) being monitored at a given time for anomaly detection because the features being analyzed for detection constantly vary. It makes the job of the adversary harder because now it would also have to estimate the features being analyzed by the ADS for detection at current time window before evasion procedure could be framed. Moreover, since the MTFS feature being analyzed by the ADS changes across time, it consequently makes the task of ascertaining what type of traffic will pass through the ADS undetected, even more challenging. The inherent randomness introduced by such a time varying feature space mutation results in a detection system that is robust to exploitation by network traffic analysis. We believe that the proposed approach can make it economically impractical (since time and effort has a cost against the gain, in the event of a successful intrusion) for the adversary to launch an evasion attack.

Moving Target based feature space mutation can be applied to any type of anomaly detection system e.g. rule-based ADS or volumetric ADS or threshold-based ADS etc. Hence, for example an MTFS rule based system would tend to have a mutating

feature space and hence a mutating rule base. Similarly, in threshold-based systems the features space employed for detection is thresholded for detecting anomalies. Hence, an MTFS threshold-based ADS would tend to mutate the features thresholded in different time windows and consequently the threshold space. Similarly it can be applied to statistical ADSs by varying the feature space being analyzed by the ADS for attack detection in different time windows. Hence the moving target based detection is a general scheme that can be tailored towards any type of anomaly detection system.

In this work we aim to define optimality and performance in terms of unpredictability, accuracy and detection delay. Hence in employing an MTFS based anomaly detection system, we aim to achieve a detector which is resilient to evasion by network traffic analysis and in turn yields optimal performance. Mutating between features for ADS detection randomizes the detection model and thus increases the deterrence of the system against evasion attacks without compromise on performance. In the following subsection we perform some preliminary analysis of feature space mutation. We provide experimental results on both LBNL and endpoint traffic traces. We also present results that provide sound basis for mutating feature space across time, which leads to detecting attack sessions that were previously undetected by non-MTFS based Maximum Entropy ADS, used as a proof of concept, due to evasion estimation. Thus, the MTFS-based ADS would be capable of detecting a wide range of attacks . This is based on the premise that attack traffic perturbs network traffic semantics and hence some feature distribution(s). In the following subsections, we provide detailed discussions with extensive experimental results to support our claims.

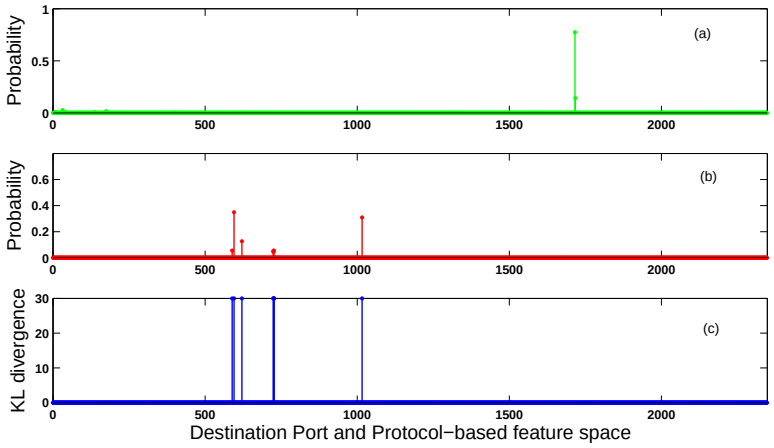


Figure 42: Kullback Leibler divergence based attack detection on destination port & protocol features.

6.6.1 Vertical Correlation - Multiple Feature Perturbations Observed for a Single Attack Class

In this section we provide experimental results to verify that different features have the capability to detect the same class of attacks. This will in turn assist us in identifying the valid set of features that can be used for detecting a particular class/type of attack. We provide results on LBNL traces (i.e. inbound scan attacks).

Motivation: An attack tends to perturb network traffic semantics. Hence, if multiple feature distributions get affected by an attack, any of those perturbed feature distributions can be employed for detection. Thus mutating between these features can guarantee the detection of the attack in different time windows.

Experimental Evaluation: We perform some preliminary evaluation of the moving target based ADS using the Kullback Leibler divergence measure of the Maximum entropy ADS. Maximum Entropy ADS by default uses Destination port and protocol based 2348 feature classes for attack detection. We used the Maximum Entropy detec-

tion principle on an MTFS (mutation based feature space) constituting the following features:

- Destination Port & Protocol (default);
- Packet Lengths; and
- Destination Ports.

Figure 42, Figure 44, and Figure 45 present maximum entropy estimation on the above mentioned features, respectively, of the MTFS space on the LBNL traces. The figures also provide the probability density function (PDF) for the benign and attack window distributions. It can be easily analyzed that attack sessions introduce significant perturbations in terms of spreading out of the attack window PDF that deviates significantly from the benign distribution, which is inadvertently skewed. Moreover, another significant insight is that multiple features get perturbed within the same attack window for the same type of attack. Hence, in turn, any of these features could be used for attack detection.

The plots reiterate the underlying principle employed in statistical analysis that in benign distribution a few feature classes constitute a major share of the traffic. However, the major share shifts during the attack period to other (attacked) feature classes. Hence, the run-time distribution diverges from the benign distribution and thus the attack gets detected. Probability density (PDF) plots illustrate this shifting of the trend from one set of prominent features to others.

Figure 42 (a) and (b) present the benign and attack window distributions for the 2348 destination port & protocol-based classes respectively. Figure 42 (c) gives the

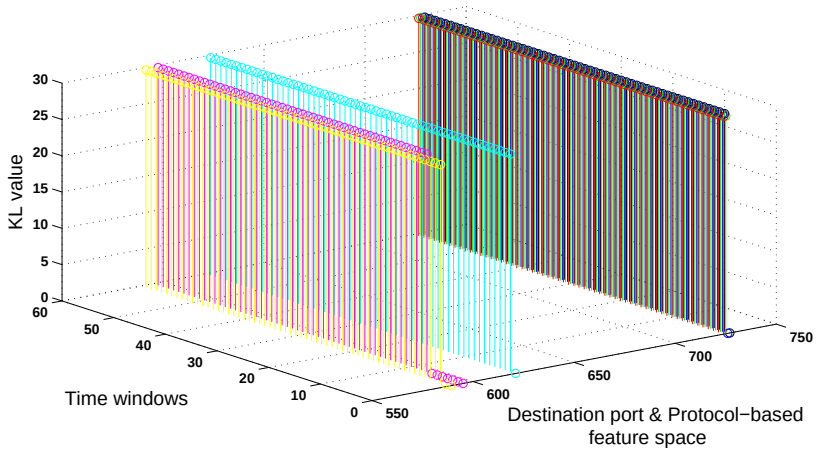


Figure 43: KL divergence exceeding the threshold in multiple time windows.

Kullback Leibler divergence for the 2348 feature classes over the 60, 1sec evaluation windows used by Maximum Entropy detector as explained in Section 6.4.1.

The LBNL traces have scan-based attack packets, these scan packets are targeted towards multiple hosts (i.e. multiple destination ports) within the network. This in turn spreads out the PDF of the attack window which ultimately leads to a greater divergence between the benign PDF and the attack window PDF and hence the detection of the attack. This can be clearly seen in Figure 42 (c).

Most of the benign traffic is web-based and hence is dominated by packets with destination port 80. However, in the attack window, multiple packets scanning on different destination ports tend to vary the port semantics by shifting the emphasis from web to scan ports.

Figure 43 shows the feature classes from the perspective of Maximum Entropy ADS. For the ADS to raise an alarm, the KL divergence is to exceed the threshold for atleast 30 1sec windows in a total of 60 evaluated windows. It can be seen that the classes that diverge significantly in Figure 42 (c), diverge in multiple windows for

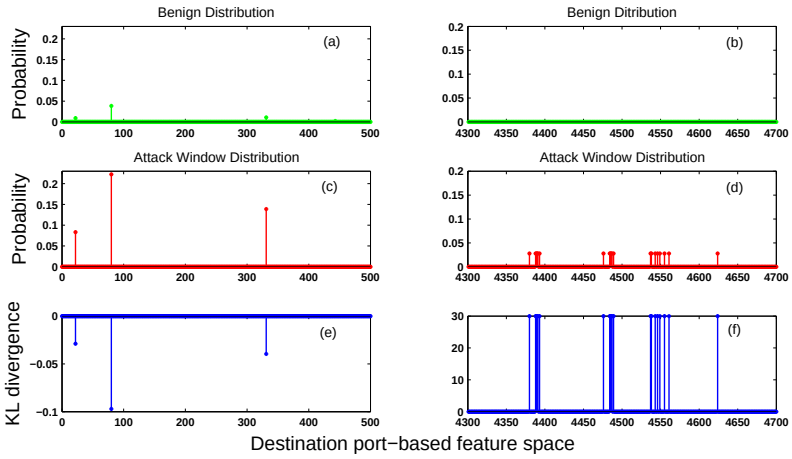


Figure 44: Kullback Leibler divergence based attack detection on destination port feature.

Maximum Entropy ADS to raise an alarm as shown in Figure 43.

Figure 44 provides the PDF and the KL divergence plots for destination port-based 587 feature classes. The plots (a),(c),(e) and (b),(d),(f) provide distribution and divergence values, respectively, for different regions of the feature space emphasizing two important insights: (i) For ports observed in benign traffic, an increased frequency of occurrence in attack window tends to generate negative KL divergence values; (ii) for ports not observed in the benign trace, an occurrence in the attack window exponentially increases the KL divergence value. Hence in cases where a destination port based feature class that is unobserved in benign trace is seen in attack window, the Maximum Entropy ADS will tend to detect it. However the same does not happen with an increase in the frequency of the already observed classes. Figure 10 provides similar plots for packet length based feature classes.

It can be observed that all three features detect the LBNL attack. This is the inherent strength of statistical detection. Attack traffic perturbs the semantic balance

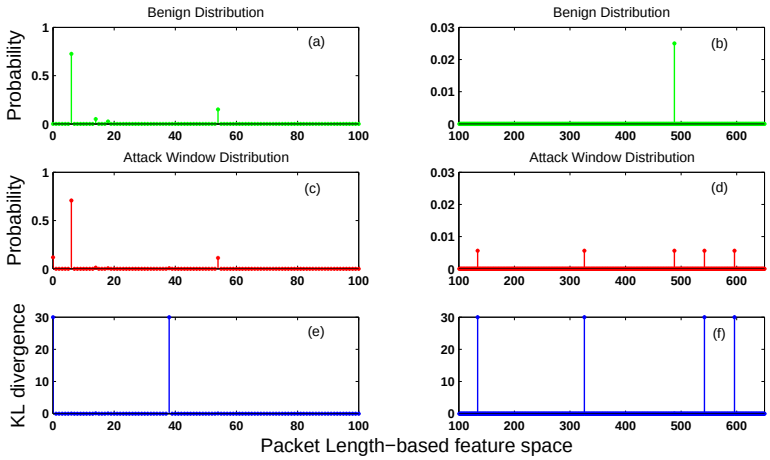


Figure 45: Kullback Leibler divergence based attack detection on 'Packet Length' feature.

between features, learnt in the training dataset. Hence attacks are detected based on variations introduced by the attack packets in the underlying model. Moreover, these attacks perturb multiple features and these can be employed across time for a randomized detection model.

Hence, anomalies in network traffic translate into perturbations in network traffic semantics. As already mentioned, these variations are detected by identifying statistical variations in traffic features. Hence if some malware is introduced in the network, some feature(s) inadvertently get disturbed. However, if the feature perturbed is not the feature being analyzed or does not have direct or indirect effect on the analyzed feature; then it is possible that the malware can pass undetected. However, this is an inherent limitation of the statistical feature based detection and not of the moving target based scheme.

Challenges: It is important to identify the valid set of mutating features such that the performance of the MTFS-based ADS is not compromised. There are multiple

scenarios to consider in this regard. For example, for an attack that spans a single time window, the features employed for detection should be diverse enough to detect the attack class. If however, an attack spans multiple time windows, mutating feature space should select features that ensure the detection of the anomaly.

6.6.2 Horizontal Correlation - Effect of Features on Detecting Different Types of Attacks

In this section we argue that a feature can be perturbed by the occurrence of different types of attacks. Thus, if there are N different attacks that an ADS is configured to detect, different attacks might perturb the same feature. Thus we believe that the features that get perturbed by multiple attacks overlap.

Motivation: If we can identify the common feature classes that the different types of attacks, the ADS is configured to detect; we can in turn formulate mutating feature classes based on a combination of all these features. Mutating between disjoint sets of these horizontally correlated feature classes would result in an ADS that can achieve optimal performance and is robust to evasion attacks.

Since malicious insiders pose a greater security threat than outsiders [137], we perform MTFS analysis on the endpoint dataset [80]. In this dataset most of the attacks originate from within the network. i.e. from malicious entities inside the network. For this section we provide results on network traces with different types of outbound attacks.

Experimental Evaluation: Different types of attacks perturb different features in network traffic. We believe if we can identify the overlapping features that detect

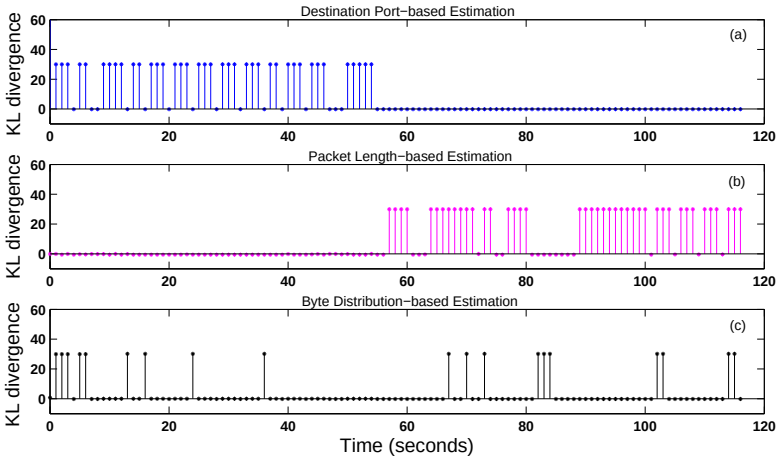


Figure 46: Different types of attacks detected by different feature classes.

different types of attacks, we can have a much smaller feature space that can detect diverse attacks and hence can provide optimal performance.

Figure 46 presents the KL divergence for two different types of attacks detected by three different feature classes. The attack window shown, has traffic from a scanning worm (Dloader-NY), followed by an SQL injection attack on a web server on port 80. Dloader has an average scan rate of approximately 46 scans per second (sps). This is a very high scan rate as compared to other malware [24]. However, the SQL injection attack packets are bundled with large payloads targeted for a webserver on destination port 80.

Figure 46 (a) shows the KL divergence using destination port based feature classes. This feature is able to detect the scanning attack owing to its high scan rate. This is because attack packets (with destination port 135 and 139) alter the probability distribution of destination port-based feature classes. Thus the attack is detected due to significant divergence of the port distribution in the attack window from the benign distribution. However, since the only feature employed for detection is destination

port, it is unable to detect the SQL injection attack on port 80 for one significant reason: port 80 has high probability of occurrence in the benign distribution and hence such payload based attacks do not significantly perturb the attack window distribution, for port 80.

Figure 46 (b), on the other hand, shows the KL divergence when packet length based feature classes are employed for detection. It can be clearly seen that it is unable to detect the scanning malware despite its extremely high scan rate. However, the SQL injection attack is detected owing to large payloads.

Figure 46 (c) shows the KL-divergence on the packet byte distribution for benign and malicious time window. The byte distribution is calculated using IP-wise n-gram analysis (where n=1). The distribution was computed for each IP in the network. It can be seen that the byte distribution-based feature class is able to detect both the attacks. This is because under normal circumstances an IP tends to have a byte sequence that is different than the skewed sequence observed during scanning (very small or no payload at all). Similarly, and payload based attacks like buffer over flow and SQL injection attacks are bundles with comparatively large payloads. Thus, this perturbs the normal sequence of bytes due to complex SQL queries or buffer overflow code in the payload. We call this feature that can detect multiple classes of attacks, the overlapping feature.

Hence, the classes of anomalies that an ADS can detect strongly relies on the features employ for detection. We thus propose that a mutating feature space should have a minimal, pre-defined number of constituent features for optimal accuracy. Mutating between such feature spaces would in turn result in an ADS that is robust

against evasion estimation attacks and at the same time provides optimal performance.

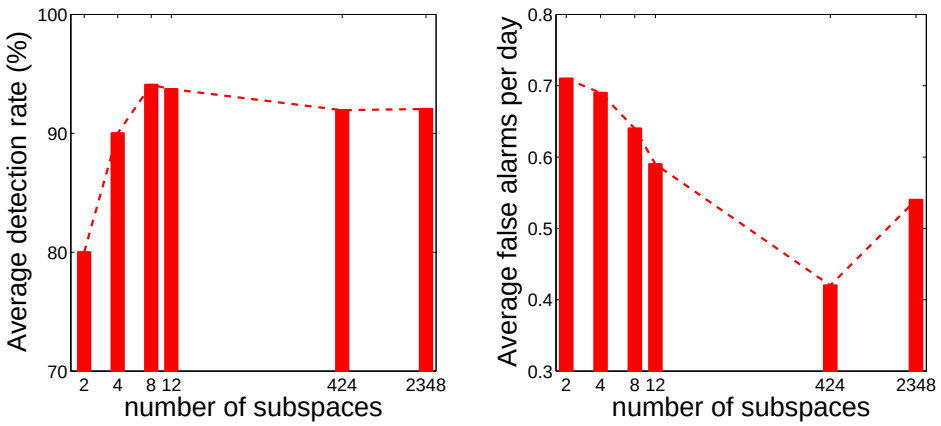
Challenge: The main challenge here is to find efficient feature(s) that can detect most of the attack types. This will enable an ADS to use features that are more sensitive to different attack types and in turn shorten the size of the feature space employed for detection as well. Therefore, it can be seen as a set of efficient features capable of dealing with different attack types.

Following are some of the open research problems with regard to the evaluations presented thus far: 1. How many features to be employed for detection within a detection window? 2. What features to use for detection within a detection window? and 3. When to mutate from one set of features to another? We elaborate on these research questions in the following subsections.

6.6.3 ADS Features Space

We argue that simultaneously employing all the features for detection, while increases complexity, does not necessarily yield optimal performance. We can judiciously select the feature classes maximizing the use of overlapping features where ever possible.

Motivation: Firstly, mutation based anomaly detection i.e. mutating feature space across time results in a more rigorous detection technique than using a high dimensional feature space across all time windows. Having a high dimensional feature space does improve performance [87], provided the features are correlated judiciously for attack detection. However, the ADS is still susceptible to evasion margin estima-



(a) Detection rate (b) False alarm rate

Figure 47: Accuracy of the Maximum Entropy detector with varying number of subspaces; results are generated for the Endpoint dataset and are averaged over all endpoints and attacks.

tion and hence can be bypassed by an intelligent and a resourceful attacker. This is because the underlying feature space does not change with time and hence can be leveraged to analyze network traffic semantics and craft attacks accordingly. The moving target based ADS design would, however, make it infeasible for an adversary to craft evasion estimation attacks due to inherent randomness of the detection process introduced by the mutating feature space. However, it is important to identify the right number of features to be employed for detection in different time windows, by an MTFS-based ADS, for optimal performance.

Experimental Evaluation: Using a high-dimensional feature space does not necessarily improve accuracy. To support our claim we reuse some earlier results from Chapter 4 (Figure 13 shown again in Figure 47. Figure 47 provides the accuracy gain on the endpoint dataset achieved by the Maximum Entropy ADS as the feature subspaces are varied from 2 to 2348 using the detector's slicing technique progressively.

Figure 47(a) shows that the detection rate does not increase progressively as we increase the number of analyzed static feature classes from 2 to 2348. The same trend is observed in the false alarms as well.

Moreover, using a high dimensional feature space translates into higher memory consumption as well. Thus the resultant ADS is more complex and hence needs more resource for its operation. Maximum Entropy has the highest runtime memory utilization (approx 67 KB) as compared to other ADSs evaluated in Chapter 2 [25].

Thus, while using all the features simultaneously does not improve robustness, it also does not ensure high accuracy. Hence, selecting a few feature class(es) judiciously for detection in a time window would suffice to introduce enough randomness in the process of ADS detection so as to render the parameter estimation attacks impractical.

Challenges: It is imperative to identify the right number of features that can be employed for detection. The number of features implies those features that can provide acceptable performance and have low complexity.

6.6.4 Traffic-aware Mutation

Random (not traffic aware) mutation strategy is applicable in the cases where we don't have any knowledge about the attacker behavior. This strategy makes the evasion harder for the attacker without making any assumption about the attacker's behavior. However, traffic aware mutation can be employed in the case where behavior of the attacker can be learned from traffic. For the purpose of traffic aware mutation, game theoretic approach can be used. In this case, the game model can assume two players i.e., attacker j and defender i. We assume that we do not have any knowledge

about the attacker's history of actions.

The goal of the attacker is to stay undetected; however, the goal of the defender is to detect the attack. Consider that the attacker wants to either launch a scan, buffer overflow or other similar attacks while staying undetected. In order to stay undetected, the attacker would have possible actions of either changing the header fields or introduce payload depending upon the defense mechanism being employed. Similarly, in order to combat the attacker, the defense mechanism has actions of selecting a feature or set of features from a given feature space which was derived after horizontal and vertical correlation discussed in the previous sections.

For every selected action, a payoff is given to the player i.e., defender or attacker. The attacker receives a payoff in terms of effectiveness of attack while staying undetected. However, the defender receives the payoff in terms of effectiveness of defense mechanism i.e., attack being detected or not. Since the goal of the attacker is to scan the network without being detected, the payoff can be defined in terms of relative number of nodes that has been scanned without being detected. This means the higher the number of nodes being scanned without detection, the higher the payoff for the attacker. On the contrary, the defender's payoff can be defined in terms of misdetection i.e., inverse of the number of nodes being scanned without detection.

Assuming that the attacker is rational, the strategy of the attacker can be calculated using the Nash equilibrium. It reveals a pure strategy, which consists of the sequence of the same action from the given action set since the attacks were not detected, or a mixed strategy which is a probabilistic distribution over the corresponding pure strategies. The regular quantal response equilibrium (QRE) can be

used to generalize the Nash equilibrium by introducing an error parameter to the payoff function since payoff functions can be erroneous. The error parameter is also called rational parameter. If the player is completely irrational (attacker), which is not covered by Nash equilibrium, the rational parameter will converge to 0. However, in case of complete rational behavior it will converge to infinity. Therefore, a corresponding defend policy (which features to select) can be employed in order to combat with the attacker behavior thus resulting to traffic aware or behavior aware mutation.

6.7 Chapter Summary

In this chapter, we stochastically modeled statistical anomaly detection systems and showed that these systems are inherently susceptible to evasion attacks. We have presented some novel ADS design philosophies to make future ADSs robust against such attacks. This is part of our ongoing research. Next chapter concludes this thesis.

CHAPTER 7: CONCLUSION

In this thesis we focused on: i) Proposing techniques to improve existing general-purpose ADSs; ii) Design and develop a specific-purpose ADS post-processor for bot detection based on the bot lifecycle; iii) Stochastically model current general-purpose anomaly detection systems; and iv) Outline alternative designs for general-purpose ADSs to make them robust against parameter estimation attacks.

To improve the accuracy of current general-purpose ADSs, we proposed a pre-processing and a post-processing method. We first proposed a feature space slicing based pre-processor to improve the performance of existing general-purpose ADSs. The inherent design of current ADSs, does not allow them to exploit parallelism available in modern day hardware. We hence propose a framework for parallelizing current day ADSs which, as a by product, also improves the performance of these systems. We illustrate that accuracy of existing statistical ADSs *can* be improved if more computational resources are available for their deployment. The key idea is to segregate benign and malicious instances into separate subspaces to mitigate averaging out and noise artifacts. We tested our feature space slicing framework on a diverse set of network and host-based ADSs. Each ADS was evaluated on its own defined set of feature(s) employed by the ADS for detection of traffic anomalies. Our experimental evaluation shows that, if feature quantification and number of subspaces

is determined judiciously, a general-purpose ADS operating simultaneously on the reduced subspaces can achieve dramatic improvements in detection and false alarm rates.

We also proposed a multi classifier based ADS post-processing method that judiciously aggregates the outputs of N parallel connected classifiers. We modeled and evaluated conventional voting based combining schemes, ENCORE and bayesian network based median rule and sum rule. We proposed a standard deviation normalized Entropy of Accuracy (SDnEA) weighted scheme. We showed experimentally that the proposed schemes provide consistently better accuracy (detection rates and false alarm) than existing combining schemes by also considering the accuracy and the variance in accuracies of the general-purpose ADSs.

Keeping up with the evolving threat landscape, we also designed and evaluated a novel specific-purpose ADS post-processor for the detection of bots. Our proposed bot detection post-processor basis its decision logic on a Bayesian network based on the high-level and abstract bot lifecycle. We showed that the bot lifecycle when used as a signature for bot detection yields low false positives. We further showed that the Bayesian network using the formal notion of fading can provide an accuracy of 87.9% with only 4.8% false positives. When evaluated on changing bot behavior, Bottleneck provided an improvement of detection rate from 40% to 90% at the cost of 1 false positive. Furthermore, Bottleneck provided 90% accuracy, with one additional false positives, over a 5 minute window as compared to batch mode execution. Hence, Bottleneck merits consideration for real-time, online deployment.

Finally we showed that current general-purpose ADSs are vulnerable to parameter

estimation attacks. This is deep rooted in the design of these systems. Hence there is a need to re-design the anomaly detection systems to make it infeasible for the attacker to devise evasion attacks. We proposed two cryptographically-inspired designs and another moving target-based ADS design. In the crypto-inspired ADS design we propose to randomize the baseline distribution, while in the moving target based design we propose to randomize the feature space of an ADS. These ADS designs introduce randomness in the detection process which in turn renders evasion attacks computationally infeasible.

REFERENCES

[1] J. Stewart, "Inside the storm: Protocols and encryption of the storm botnet," in *Black Hat Technical Security Conference, USA*, 2008.

[2] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Technical report, Tech. Rep., 2000.

[3] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 201–206.

[4] "Symantec internet security threat reports i–xi," Jan 2002–Jan 2008.

[5] M. Corp., "Mcafee virtual criminology report: North american study into organized crime and the internet," 2005.

[6] "Computer economics: 2001 economic impact of malicious code attacks." [Online]. Available: http://www.computereconomics.com/cei/press/pr92101.html

[7] D. Moore, C. Shannon *et al.*, "Code-red: a case study on the spread and victims of an internet worm," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 273–284.

[8] "Cyber secure institute." [Online]. Available: http://cybersecureinstitute.org/blog/?p=15

[9] S. Shin and G. Gu, "Conficker and beyond: a large-scale empirical study," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 151–160.

[10] F. Cohen, "Computer viruses: theory and experiments," *Computers & security*, vol. 6, no. 1, pp. 22–35, 1987.

[11] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *Security & Privacy, IEEE*, vol. 1, no. 4, pp. 33–39, 2003.

[12] E. Rescorla, "Security holes... who cares," in *Proceedings of the 12th USENIX Security Symposium*, 2003, pp. 75–90.

[13] Z. Chen and C. Ji, "A self-learning worm using importance scanning," in *Proceedings of the 2005 ACM workshop on Rapid malcode*. ACM, 2005, pp. 22–29.

[14] J. Ma, G. M. Voelker, and S. Savage, "Self-stopping worms," in *Proceedings of the 2005 ACM workshop on Rapid malcode*. ACM, 2005, pp. 12–21.

[15] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms." in *USENIX Security Symposium*, vol. 2, 2004, pp. 16–85.

[16] "Advanced threat report - 2012," http://www.fireeye.com/info-center/.

[17] D. E. Denning, "An intrusion-detection model," *Software Engineering, IEEE Transactions on*, no. 2, pp. 222–232, 1987.

[18] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[19] "Bro," http://www.bro.org/, Online May 2012.

[20] M. Roesch *et al.*, "Snort-lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX conference on System administration*. Seattle, Washington, 1999, pp. 229–238.

[21] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3. IEEE, 2003, pp. 1901–1910.

[22] "Fireeye, next-generation threats." [Online]. Available: http://www.fireeye.com/threat-protection/why-dont-traditional-defenses-work.html

[23] "Damballa csp," https://www.damballa.com/solutions/damballa_csp.php Online April 2013.

[24] C. Wong, S. Bielski, A. Studer, and C. Wang, "Empirical analysis of rate limiting mechanisms," in *Recent Advances in Intrusion Detection*. Springer, 2006, pp. 22–42.

[25] A. B. Ashfaq, M. J. Robert, A. Mumtaz, M. Q. Ali, A. Sajjad, and S. A. Khayam, "A comparative evaluation of anomaly detectors under portscan attacks," in *Recent Advances in Intrusion Detection*. Springer, 2008, pp. 351–371.

[26] K. Nyalkalkar, S. Sinhay, M. Bailey, and F. Jahanian, "A comparative study of two network-based anomaly detection methods," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 176–180.

[27] C. Smith, A. Matrawy, S. Chow, and B. Abdelaziz, "Computer worms: Architectures, evasion strategies, and detection mechanisms," *Journal of Information Assurance and Security*, vol. 4, pp. 69–83, 2008.

[28] P. Fogla and W. Lee, "Evading network anomaly detection systems: formal reasoning and practical techniques," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 59–68.

[29] "Emerging cyber threats report for 2009," October 15, 2008.

[30] B. Lowe, "Symantec report: The underground economy," Dec 2008. [Online]. Available: http://www.symantec.com/connect/sites/default/files/SYMC_ISTR_and_Underground_Economy__SHARE_0.ppt

[31] Z. Li, Q. Liao, and A. Striegel, "Botnet economics: uncertainty matters," in *Managing Information Risk and the Economics of Security*. Springer, 2009, pp. 245–267.

[32] M. M. Williamson, "Throttling viruses: Restricting propagation to defeat malicious mobile code," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*. IEEE, 2002, pp. 61–68.

[33] D.-K. Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*. IEEE, 2005, pp. 118–125.

[34] "Ddos attack launchged against spamhaus," 2013. [Online]. Available: http://www.bbc.co.uk/news/technology-21954636

[35] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: detecting malware infection through ids-driven dialog correlation," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362915

[36] "Bibliography of anomaly detection systems," 2012. [Online]. Available: http://wisnet.niit.edu.pk/projects/adeval/Literature$_$survey/Bibliography.doc

[37] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.

[38] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, vol. 2. IEEE, 2000, pp. 12–26.

[39] S. A. Khayam, "Wireless channel modeling and malware detection using statistical and information-theoretic tools," Ph.D. dissertation, Michigan State University, 2006.

[40] M. Mahoney and P. K. Chan, "Phad: Packet header anomaly detection for identifying hostile network traffic," *Florida Institute of Technology technical report CS-2001-04*, pp. 1–17, 2001.

[41] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 211–225.

[42] S. Khattak, Z. Ahmed, A. A. Syed, and S. A. Khayam, "Poster: Botflex: A community-driven tool for botnet detection."

[43] S. E. Schechter, J. Jung, and A. W. Berger, "Fast detection of scanning worm infections," in *Recent Advances in Intrusion Detection*. Springer, 2004, pp. 59–81.

[44] M. V. Mahoney, "Network traffic anomaly detection based on packet bytes," in *Proceedings of the 2003 ACM symposium on Applied computing*. ACM, 2003, pp. 346–350.

[45] S. A. Khayam, H. Radha, and D. Loguinov, "Worm detection at network endpoints using information-theoretic traffic perturbations," in *Communications, 2008. ICC'08. IEEE International Conference on*. IEEE, 2008, pp. 1561–1565.

[46] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 31–31.

[47] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 32–32.

[48] "Next-generation intrusion detection expert system (nides)." [Online]. Available: http://www.csl.sri.com/projects/nides/

[49] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 2001, pp. 130–143.

[50] "Wisnet ads comparison homepage." [Online]. Available: http://wisnet.seecs.nust.edu.pk/projects/ENS/source_code.html

[51] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *Proceedings of the 12th USENIX Security Symposium*, vol. 285, 2003, p. 294.

[52] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of pca for traffic anomaly detection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 109–120, 2007.

[53] T. Joachims, "Making large scale svm learning practical," 1999.

[54] "Nayatel," http://www.nayatel.pk/index.php Online April, 2013.

[55] "Computer immune systems, datasets." [Online]. Available: http://www.cs.unm.edu/~immsec/data/synth-sm.html

[56] "Cert-polaska," http://www.cert.pl/PDF/Report_Virut_EN.pdf Online April, 2013.

[57] [Online]. Available: http://www.team-cymru.org/

[58] [Online]. Available: http://www.icir.org/

[59] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 29–38, 2006.

[60] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A first look at modern enterprise traffic," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 2–2.

[61] "Winpcap homepage." [Online]. Available: http://www.winpcap.org/

[62] "Symantec security response." [Online]. Available: http://securityresponse.symantec.com/avcenter

[63] C. Shannon and D. Moore, "The spread of the witty worm," *Security & Privacy, IEEE*, vol. 2, no. 4, pp. 46–50, 2004.

[64] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 120–128.

[65] "Information security hype," 2003.

[66] "Gartner information security hype cycle declares intrusion detection systems a market failure–money slated for intrusion detection should be invested in firewalls," June 2003. [Online]. Available: http://www.gartner.com/5_about/press_releases/pr11june2003c.jsp

[67] "Fireeye malware protect system (mps) homepage." [Online]. Available: http://www.fireeye.com/products/

[68] "Cisco anomaly guard module homepage." [Online]. Available: www.cisco.com/en/US/products/ps6235/

[69] "Arbor networks peakflow homepage." [Online]. Available: http://www.arbornetworks.com/en/peakflow-x.html

[70] "Endace ninjabox homepage." [Online]. Available: http://www.endace.com/ninjabox.html

[71] "Intel core^{TM}2 duo processor." [Online]. Available: http://www.intel.com/products/processor/core2duo/index.htm

[72] "Amd athlon^{TM}ii x2 dual-core processor." [Online]. Available: http://www.amd.com/us-en/Processors/ProductInformation/\newline0,,30_118_9485_16006,00.html

[73] "nvidia motherboard gpus." [Online]. Available: http://www.nvidia.com/page/gpu_mobo.html

[74] "Intel larrabee integrated graphics accelerator." [Online]. Available: http://software.intel.com/en-us/articles/larrabee/

[75] "Intel single-chip cloud computer (ssc) homepage." [Online]. Available: http://techresearch.intel.com/articles/Tera-Scale/1826.htm

[76] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High performance network intrusion detection using graphics processors," in *Recent Advances in Intrusion Detection*. Springer, 2008, pp. 116–134.

[77] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 1, pp. 99–107, 2005.

[78] P. Piyachon and Y. Luo, "Efficient memory utilization on network processors for deep packet inspection," in *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*. ACM, 2006, pp. 71–80.

[79] S. Yi, B.-k. Kim, J. Oh, J. Jang, G. Kesidis, and C. R. Das, "Memory-efficient content filtering hardware for high-speed intrusion detection systems," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007, pp. 264–269.

[80] "Nust dataset." [Online]. Available: http://www.wisnet.seecs.nust.edu.pk/projects/ENS/DataSets.html

[81] "Lbnl/icsi enterprise tracing project." [Online]. Available: http://www.icir.org/enterprise-tracing/download.html

[82] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[83] ——, "Mining anomalies using traffic feature distributions," in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4. ACM, 2005, pp. 217–228.

[84] D. Turner, M. Fossi, E. Johnson, T. Mark, J. Blackbird, S. Entwise, M. Low, D. McKinney, and C. Wueest, "Symantec global internet security threat report–trends for 2008," *h ttp://eval. symantec. com/mktginfo/enterprise/white papers/b-whitepaper internet security threat report*, vol. 14, pp. 04–2009, 2009.

[85] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu *et al.*, "The ghost in the browser analysis of web-based malware," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007, pp. 4–4.

[86] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[87] A. B. Ashfaq, M. Javed, S. A. Khayam, and H. Radha, "An information-theoretic combining method for multi-classifier anomaly detection systems," in *Communications (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.

[88] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, no. 6, pp. 859–874, 2011.

[89] T. M. Mitchell, "Machine learning. wcb," 1997.

[90] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[91] D. J. Ketchen and C. L. Shook, "The application of cluster analysis in strategic management research: an analysis and critique," *Strategic management journal*, vol. 17, no. 6, pp. 441–458, 1996.

[92] L. J. Heyer, S. Kruglyak, and S. Yooseph, "Exploring expression data: identification and analysis of coexpressed genes," *Genome research*, vol. 9, no. 11, pp. 1106–1115, 1999.

[93] G. Giacinto, R. Perdisci, M. Del Rio, and F. Roli, "Intrusion detection in computer networks by a modular ensemble of one-class classifiers," *Information Fusion*, vol. 9, no. 1, pp. 69–82, 2008.

[94] S. L. Scott, "A bayesian paradigm for designing intrusion detection systems," *Computational statistics & data analysis*, vol. 45, no. 1, pp. 69–83, 2004.

[95] A. Al-Ani and M. Deriche, "A new technique for combining multiple classifiers using the dempster-shafer theory of evidence," *arXiv preprint arXiv:1107.0018*, 2011.

[96] N. Hatami and R. Ebrahimpour, "Combining multiple classifiers: diversify with boosting and combining by stacking," *Int. J. Comput. Sci. Network Security*, vol. 7, no. 1, pp. 127–31, 2007.

[97] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of combining multiple classifiers and their applications to handwriting recognition," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 3, pp. 418–435, 1992.

[98] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 3, pp. 226–239, 1998.

[99] A. F. R. Rahman, H. Alam, and M. C. Fairhurst, "Multiple classifier combination for character recognition: Revisiting the majority voting system and its variations," in *Document Analysis Systems V*. Springer, 2002, pp. 167–178.

[100] M. Fairhurst and A. Rahman, "Enhancing consensus in multiple expert decision fusion," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 147, no. 1, pp. 39–46, 2000.

[101] A. Rahman and M. Fairhurst, "Exploiting second order information to design a novel multiple expert decision combination platform for pattern classification," *Electronics Letters*, vol. 33, no. 6, pp. 476–477, 1997.

[102]

[103] "Tracking ghostnet: Investigating a cyber espionage network," 2009, information Warfare Monitor.

[104] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, 2011.

[105] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*, may 2010, pp. 305 –316.

[106] F. V. Jensen and T. D. Nielsen, *Bayesian networks and decision graphs*. Springer, 2007.

[107] M. Knysz, X. Hu, and K. G. Shin, "Good guys vs. bot guise: Mimicry attacks against fast-flux detection systems," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1844–1852.

[108] M. A. R. J. Z. Fabian and M. A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 2006 ACM SIGCOMM Internet Measurement Conference (IMC)*, vol. 2006, 2006.

[109] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.

[110] N. R. Ramay, S. Khattak, A. A. Syed, and S. A. Khayam, "Poster: Bottleneck: A generalized, flexible, and extensible framework for botnet defense."

[111] R. Cowell, "Introduction to inference for bayesian networks," *NATO ASI SERIES D BEHAVIOURAL AND SOCIAL SCIENCES*, vol. 89, pp. 9–26, 1998.

[112] X. Qin and W. Lee, "Discovering novel attack strategies from infosec alerts," in *In Proceedings of the 9th European Symposium on Research in Computer Security, Sophia Antipolis*, 2004, pp. 439–456.

[113] J. Cheng and R. Greiner, "Learning bayesian belief network classifiers: Algorithms and system," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, E. Stroulia and S. Matwin, Eds. Springer Berlin Heidelberg, 2001, vol. 2056, pp. 141–151. [Online]. Available: http://dx.doi.org/10.1007/3-540-45153-6_14

[114] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *Information Theory, IEEE Transactions on*, vol. 14, no. 3, pp. 462–467, May 1968.

[115] C. Chow and T. Wagner, "Consistency of an estimate of tree-dependent probability distributions (corresp.)," *Information Theory, IEEE Transactions on*, vol. 19, no. 3, pp. 369–371, May 1973.

[116] G. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00994110

[117] A. Netica, "Programers library reference manual."

[118] D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell, "Bayesian analysis in expert systems," *Statistical science*, pp. 219–247, 1993.

[119] F. V. Jensen, *An introduction to Bayesian networks*. UCL press London, 1996, vol. 74.

[120] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[121] "Bottleneck," http://sysnet.org.pk/w/Code_and_Tools#BotFlex.

[122] FireEye, "Protecting against advanced malware and targeted apt attacks," 2012. [Online]. Available: http://www.fireeye.com/resources/pdfs/FireEye_Gartner_RB_Feb2012.pdf

[123] "Arbor network's peakflow." [Online]. Available: http://www.arbornetworks.com

[124] G. F. Cretu-Ciocarlie, A. Stavrou, M. E. Locasto, and S. J. Stolfo, "Adaptive anomaly detection via self-calibration and dynamic updating," in *Recent Advances in Intrusion Detection*. Springer, 2009, pp. 41–60.

[125] W. K. Robertson, F. Maggi, C. Kruegel, and G. Vigna, "Effective anomaly detection with scarce training data." in *NDSS*, 2010.

[126] F. Silveira and C. Diot, "Urca: Pulling out anomalies by their root causes," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[127] F. Cohen, "50 ways to defeat your intrusion detection system," 2003.

[128] G. V. Hulme, "Gartner: Intrusion detection on the way out," *Information Week, June*, vol. 13, p. 2003, 2003.

[129] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," DTIC Document, Tech. Rep., 1998.

[130] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *Proceedings of the 15th USENIX Security Symposium*, 2006, pp. 241–256.

[131] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 255–264.

[132] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[133] M. Q. Ali, H. Khan, A. Sajjad, and S. A. Khayam, "On achieving good operating points on an roc plane using stochastic anomaly score prediction," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 314–323.

[134] O. Goldreich, "Foundations of cryptography, volume i," *Basic Tools*, 2003.

[135] B. Schneier, "The data encryption standard (des)," *CryptoGram newsletter*, 2000.

[136] I. Land, "A note on symmetric discrete memoryless channels," 2005. [Online]. Available: http://kom.aau.dk/project/sipcom/SIPCom06/sites/sipcom8/courses/SIPCom8-1/SymDMCs.pdf

[137] "Malicious insider threats report, computer economics," May 2010. [Online]. Available: http://securityresponse.symantec.com/avcenter