Machine Learning for Network Intrusion Detection

Luke Hsiao
Stanford University
lwhsiao@stanford.edu

Stephen Ibanez
Stanford University
sibanez@stanford.edu

ABSTRACT

Computer networks have become an increasingly valuable target of malicious attacks due to the increased amount of valuable user data they contain. In response, network intrusion detection systems (NIDSs) have been developed to detect suspicious network activity. We present a study of unsupervised machine learning-based approaches for NIDS and show that a non-stationary model can achieve over 35× higher quality than a simple stationary model for a NIDS which acts as a sniffer in a network. We reproduce the results of packet header-based anomaly detection for detecting potential attacks in network traffic and are able to detect 62 of 201 attacks with a total of 86 false positives (an average of under 9 per day) on the 1999 DARPA dataset. Our implementation is open source, available at https://github.com/lukehsiao/ml-ids.

1 INTRODUCTION

As the world becomes more and more connected through the Internet, computer networks have become an increasingly valuable target of malicious attacks. Large corporate networks are of particular value to attackers due to the sensitive information that they carry such as private user information or internal business data. One of the tools used to address network attacks are NIDS, which seek to monitor network traffic and detect suspicious activity or other policy violations. A NIDS typically utilizes either (1) signature detection methods such as SNORT [7], or BlindBox [9], or (2) machine-learning based techniques such as clustering [8], neural networks [3, 12], or statistical modeling [6, 11]. Learning-based approaches are of particular interest because they have the potential to detect novel attacks unlike signature based systems which would need to be updated.

One challenge involved with building a NIDS is that they often would like to examine encrypted application-level packet data in an attempt to detect more attacks. This often means that the NIDS must intercept secure connections by positioning as a man-in-the-middle, or in the case of BlindBox, employ new protocols to inspect the encrypted payload without sacrificing the end user's privacy. There are a number drawbacks to these techniques which could be avoided if the NIDS did not need to examine encrypted packet data. As a result, we explore machine learning techniques for building an NIDS that only use unencrypted packet header fields.

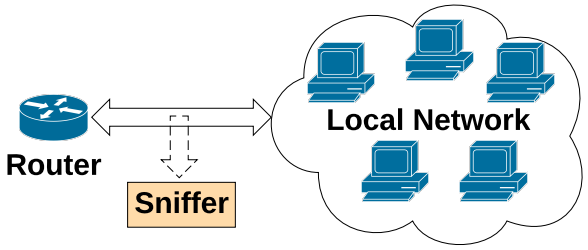

Figure 1: Our dataset consists of packets collected by a sniffer between a network and an Internet router.

2 METHOD

A common approach to using machine learning for NIDS is to frame the problem as an unsupervised anomaly detection task, where we desire to train a model to recognize normal, attack-free traffic and consequently recognize anomalous, potentially malicious traffic. We implement two methods for anomaly detection: (1) a stationary model using a mixture of Gaussians, which does not incorporate time as a parameter, and (2) a non-stationary model based on the Packet Header Anomaly Detection (PHAD) paper [5]. We use 33 fields found in packet headers as features, as opposed to other systems which perform anomaly detection by using the bytes of the payload [11]. In addition, our non-stationary model prioritizes the time since an event last occurred rather than the frequency of the occurrence.

We then train and test our models on the widely used DARPA 1999 IDS dataset[1] [4], which is still considered a useful dataset for evaluating this task despite its age [10] and consists of 5 weeks of `tcpdump` data collected by a sniffer positioned between a local network and an Internet router as shown in Figure 1. The test data in the DARPA dataset contains 201 instances of 56 types of attacks.

2.1 Stationary Model: Mixture of Gaussians

We first establish a baseline for the quality of a stationary model that does not consider time as a parameter and instead only looks at a packet's features in isolation. Specifically, we model the data as a mixture of Gaussians. During training, we train a Gaussian mixture model using `sklearn`[2]. We empirically selected to model the data with 16 components using

[1] https://ll.mit.edu/ideval/data/1999data.html
[2] http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture

full covariance matrices to keep training time manageable on consumer hardware. We found no significant differences in final quality when varying the number of Gaussians. Model parameters are initialized by first running the kmeans clustering algorithm on the data.

While a mixture of Gaussians is typically used to label data based on which Gaussian is the most probable, we instead look at how unlikely a particular packet's features are given the model trained on attack free data. During detection, each packet is assigned 16 probabilities $P = [p_1, \ldots, p_{16}]$. We then score a packet as $(1 - \max P)$. Packets with a low probability of coming from any of the Gaussians have the highest scores.

2.2 Non-stationary Model: PHAD

Next, we implement the PHAD learning model which is based on computing an anomaly score for each field of a packet header. During training, we learn an estimate of the rate of anomalies for each field. If a particular packet field is observed n times and consists of r distinct values, then we approximate the probability that the next observation is anomalous by r/n. Note n is not necessarily the same as the total number of packets seen, since the packet fields that are present depend on the type of the packet. The PHAD model assumes that anomalous events occur in bursts since they are often caused by a change of network state (e.g. a change in software or configuration). During training, only the first anomalous event in a burst is added to r.

During detection, PHAD uses time as parameter to account for the dynamic behavior of real traffic and to help discount the weight of anomalous packets after the first packet in a burst. Specifically, they define a factor t for each field as the time since the previous anomaly in that particular field. Thus, if a packet occurred t seconds ago, PHAD approximates the probability that it will occur in the next second as $1/t$. PHAD looks at a stream of packets sequentially, and produces a score for each field that is inversely proportional to the probability of an anomaly in that field:

$$\text{score}_{\text{field}} = \frac{t * n}{r} \tag{1}$$

Rather than assuming that each field is independent (and thus taking the product of each of the field's probabilities as a packet score), PHAD approximates fields as occurring sequentially, and scores a packet as the probability of observing consecutive anomalies. Packets scores are the sum of the scores of its anomalous fields.

$$\text{score}_{\text{packet}} = \sum_{i \in \text{anomalous fields}} \frac{t_i * n_i}{r_i} \tag{2}$$

2.2.1 Implementing PHAD-C32 A core implementation challenge of the PHAD model proposed above is detecting whether or not a packet header is anomalous. In particular, packet header fields can range from 1 to 4 bytes, allowing up to 2^{32} values. In the original paper, the authors ultimately select a cluster-based approach for detecting anomalies. Rather than store a set of potentially millions of entries for each field, the original PHAD implementation instead stores a list of clusters (represented as a number interval) up to a limit defined as C, where PHAD-C32 uses $C = 32$. When the number of clusters exceeds C, the clusters whose ranges are closest together are combined into a single cluster. This implementation is summarized in Algorithm 1. Clusters are created during training, where each field has a set of C clusters and are used to estimate r, which is incremented each time a cluster is added, and used to determine whether a field value is anomalous or not based on whether it's value falls within any of the clusters learned from the training data. In order to best recreate the original results, we implement this same clustering structure using Python.

Algorithm 1 PHAD Clustering algorithm

```
1:  C ← 32                                          ▷ in the case of PHAD-C32
2:  clusters ← ∅
3:  procedure AddValue(a)                           ▷ adding a to the clusters
4:      for cluster ∈ clusters do
5:          if a ∈ cluster then
6:              return                              ▷ a is non-anomalous
7:      clusters ← clusters ∪ [a, a]                ▷ add new cluster
8:      if len(clusters) > C then
9:          merge two nearest clusters
10:     return
```

2.2.2 Post-processing In order to correctly identify an attack, the model must indicate the destination IP address, the date, and the time of the attack to within one minute of accuracy. After providing a list of anomalies that the model classifies as attacks, we perform a post processing step that removes duplicate detections of the same attack. We keep only the highest scoring detection for each attack.

3 EXPERIMENTS

As our core experiment, we evaluate the quality of our NIDS implementation based on two primary metrics: (1) *recall*, the percentage of attacks which we are able to identify in the dataset and (2) *precision*, the percentage of packets we classify as attacks that are true instances of attacks. These two metrics provide an assessment of how effective our NIDS is at detecting attacks, and how practical such a NIDS system would be based on its precision. For this application, a very low rate of false positives is essential, due to the limited amount of time that analysts would have to respond to these in a real-world situation. We use F1 score (the harmonic mean of precision and recall) as a unifying metric for the precision and recall of a system.

Table 1: Comparing results of our stationary and non-stationary models with a leading system in the DARPA off-line evaluation [4].

Approach	Precision	Recall	F1
Stationary	2/72	2/201	0.0147
Non-Stationary	62/148	62/201	0.3600
Original PHAD	72/172	72/201	0.3861
DARPA Expert 1[†]	85/185	85/201	0.4404

[†] This system was an offline system.

3.1 Experimental Settings

We use the same 1999 DARPA Intrusion Detection Dataset as the original paper. These files are organized by day for each week of the data. Evaluation labels were provided via a hand-formatted text file, which required us to create additional parsing scripts in order to programmatically evaluate our results. We train our model using the attack-free data in week 3, which consists of 8 days of data (Monday through Friday with extra data for Monday, Tuesday, and Wednesday). We then test our implementation on weeks 4 and 5 of the test data, which spans 9 days (Monday through Friday of both weeks, except for Tuesday of week 4 which is not provided).

The original paper displays scores a shifted log scale as

$$0.1 \log_{10}(\text{score}_{\text{packet}}) - 0.6 \tag{3}$$

We find this equation underspecified (see Section 3.2) and instead display scores by using a log scale and scaling the results to fall in the interval $[0, 1]$. Packets can then be classified as an attack if their score exceeds a configurable threshold.

3.2 Experimental Results

The results of our experiment are summarized in Table 1. As expected, we find that a simple mixture of Gaussian model is ineffective at finding anomalous packets. When examined in isolation, attack packets are very similar to normal traffic. For example, a single packet to a common port number may in fact be part of a port scanning attack, but is undetectable without considering the timing of the packet with respect to others in the stream. By implementing a non-stationary model, we see over 35× improvement in F1. Our non-stationary model achieves comparable quality to the original PHAD implementation. In addition, it performs well when compared with Expert 1, a top off-line system (which performs detection on past traffic logs only, unlike an online systems) of the original DARPA evaluation [4]. We do notice some differences in our non-stationary model compared to the original PHAD paper, which we discuss below.

Training the PHAD Model. The key objective of training the non-stationary model proposed by PHAD is to learn an estimate of the probability of seeing an anomaly in each packet field. In Table 2, we show the results of our training compared to the results published for PHAD. We find that in our implementation, the total number of packet fields seen differs from the original PHAD results, regardless of whether or not we include the 3 days of extra training data. We suspect that the differences in r, the estimated number of anomalies seen for each field in the training data, may be due to differences in the implementation of the clustering algorithm. The order in which the values are added to the cluster can affect the final results, as clusters are merged.

Table 2: Our PHAD model vs. original after training.

	Our Implementation		PHAD-C32	
Field	**r**	**n**	**r**	**n**
Ethernet Dest Hi	9	15899443	9	12814738
Ethernet Dest Lo	12	15899443	12	12814738
Ethernet Size	508	15899443	508	12814738
Ethernet Src Hi	6	15899443	6	12814738
Ethernet Src Lo	9	15899443	9	12814738
Ethernet Type	4	15899443	4	12814738
ICMP Chksum	281	8221	1[†]	7169
ICMP Code	3	8221	3	7169
ICMP Type	3	8221	3	7169
IPv4 Chksum	1642	15785479	1[†]	12715589
IPv4 Dest	295	15785479	287	12715589
IPv4 ID	4117	15785479	4117	12715589
IPv4 Header Len	1	15785479	1	12715589
IPv4 Length	527	15785479	527	12715589
IPv4 Offset	1	15785479	2	12715589
IPv4 Protocol	3	15785479	3	12715589
IPv4 Src	301	15785479	293	12715589
IPv4 TOS	4	15785479	4	12715589
IPv4 TTL	10	15785479	10	12715589
TCP AckNo	4950	13160618	4235	10617293
TCP Chksum	1053	13160618	1[†]	10617293
TCP Header Len	2	13160618	2	10617293
TCP Dest Port	3537	13160618	3545	10617293
TCP Flags	9	13160618	9	10617293
TCP Options	2	788295	2	611126
TCP SeqNo	6310	13160618	5455	10617293
TCP Src Port	3538	13160618	3546	10617293
TCP UrgPtr	2	13160618	2	10617293
TCP Window	1064	13160618	1016	10617293
UDP Chksum	3213	2616640	2	2091127
UDP Dest Port	6048	2616640	6050	2091127
UDP Length	128	2616640	128	2091127
UDP Src Port	6050	2616640	6052	2091127

[†] The original sets checksums to 0xFFFF before training.

Top 20 Anomalies. The top 20 most anomalous alarms found by our system are shown in Table 3. We capture 15 of the same alarms in our top 20 as the original paper, with minor differences in rankings. Using Equation 3, their top 20 anomalies range from a score of 0.7482 to 0.5817. We find that this equation is insufficiently justified by the original authors. First, these scores can be affected by implementation details, such as that the value of t is initialized to when calculating scores for each field. Second, the equation does not guarantee that values will fall from 0 to 1 as they claim, since low scoring packets end up with negative values. The other 5 attacks present in their top 20, but are not in ours have a checksum as their most anomalous field. This difference is

Table 3: Top 20 scoring alarms on the test data along with their ranking in the original paper.

Rank	Score	Date	Time	Dest. IP	Det.	Attack	Most Anomalous Field (% of score)	Orig. Rank
1	1.000000	04/05/1999	08:39:50	172.16.112.50	TP	pod	IPv4_offset (99%)	8
2	0.994587	04/06/1999	08:59:16	172.16.112.194	FP		TCP_dataOffset (98%)	1
3	0.990693	04/01/1999	08:26:16	172.16.114.50	TP	teardrop	IPv4_offset (99%)	2
4	0.977340	04/05/1999	08:00:02	0.0.0.0	FP	(arppoison)	IPv4_ihl (100%)	-
5	0.972821	04/08/1999	08:01:20	172.16.113.50	FP		IPv4_offset (99%)	7
6	0.970337	04/05/1999	11:45:27	172.16.112.100	TP	dosnuke	TCP_urgPtr (99%)	11
7	0.962013	04/01/1999	11:00:01	172.16.112.100	TP	dosnuke	TCP_urgPtr (99%)	3
8	0.955729	03/31/1999	08:00:09	0.0.0.0	FP	(arppoison)	IPv4_ihl (100%)	4
9	0.951241	04/08/1999	23:10:00	0.0.0.0	FP	(arppoison)	Ethernet_srcHi (59%)	14
10	0.950843	04/06/1999	11:57:55	206.48.44.50	FP		IPv4_tos (99%)	15
11	0.947575	04/05/1999	11:17:50	0.0.0.0	FP	(arppoison)	Ethernet_srcHi (59%)	16
12	0.946970	04/08/1999	08:01:20	172.16.113.50	FP		TCP_dataOffset (91%)	-
13	0.943546	04/06/1999	08:32:12	172.16.114.50	TP	teardrop	IPv4_offset (99%)	-
14	0.939460	04/09/1999	08:01:26	172.16.113.50	FP		IPv4_offset (99%)	10
15	0.939037	04/08/1999	23:11:04	172.16.112.10	TP	mscan	Ethernet_dstHi (57%)	18
16	0.935665	04/08/1999	08:01:21	172.16.113.50	FP		TCP_urgPtr (99%)	19
17	0.935650	03/31/1999	11:35:13	0.0.0.0	FP	(arppoison)	Ethernet_srcHi (59%)	5
18	0.935370	04/05/1999	11:18:45	172.16.118.20	FP		Ethernet_dstHi (57%)	20
19	0.931332	04/02/1999	09:05:07	0.0.0.0	FP	(arppoison)	Ethernet_srcHi (59%)	-
20	0.931110	04/09/1999	08:01:26	172.16.113.50	FP		TCP_dataOffset (78%)	-

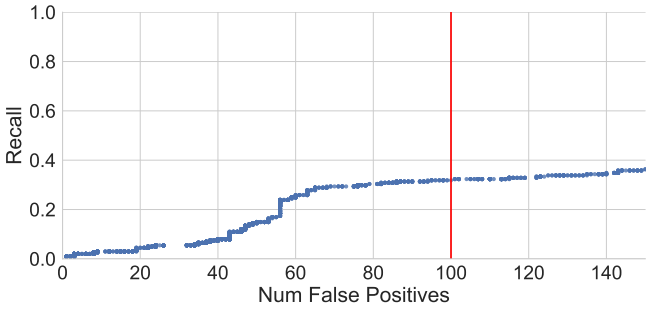

Figure 2: Recall vs. total false positives when varying classification threshold values.

likely caused by the fact that the original authors essentially ignore the checksum values during training by setting the field to 0xFFFF, as shown in Table 2.

Threshold Tuning. In Figure 2 we show the affect of varying the classification threshold on recall (the percentage of attacks detected) and the total number of false alarms. The DARPA goal of having less than 100 false alarms total is shown in red. The original authors present their results by limiting their false positives per day, which would be similar to tuning a separate threshold for each particular day. We opted for a single threshold to better mimic what a real system examining real-time traffic might do, where we cannot select a threshold after-the-fact to meet a specific rate of false positives, but instead want to flag anomalies in real-time.

3.3 Feature Ablation Study

In this study, we remove a single feature of the 33 used by our non-stationary model and evaluate the final F1 achieved using the remaining 32 features. The results are shown in Figure 3, where the baseline setting of all 33 features enabled is shown in red. Most fields, such as `IPv4_ihl` (the IP header

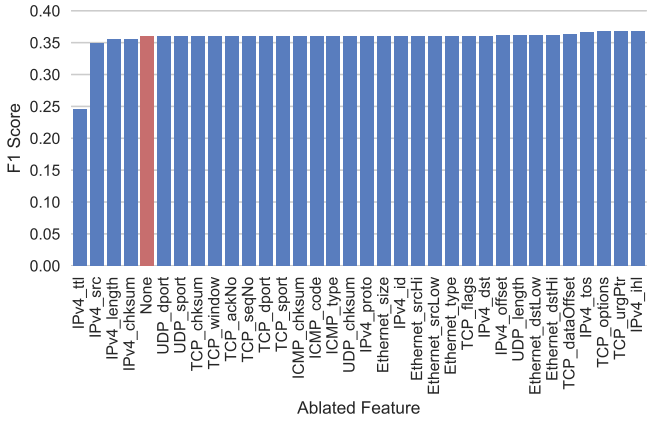

Figure 3: For each feature on the x-axis, we remove that feature alone and report the resulting F1.

length), contain signals that are either of little impact or are redundant, and thus do not affect the F1 significantly when removed. However, removing the `IPv4_ttl` feature (the time-to-live value) significantly hurts the F1.

Running our non-stationary model with *only* the `IPv4_ttl`, we achieve an F1 score of 0.38 with a lower recall of 0.26 but a much higher precision of 0.68. During training, we only see 10 distinct values for the TTL field: 2, 32, 60, 62-64, 127, 128, 254, and 255. However, we see that attack packets have a different TTL values than the attack-free training data such as 43 and 253, despite the specific value of the TTL field being unrelated to the attack itself. We suspect that this is likely an unintentional artifact of the simulated data, which is well known to be a difficult task [2], and represents one of the shortcomings of the 1999 DARPA dataset. As future work, we may be able to address these simulation artifacts by mixing in real-world traffic into the training data.

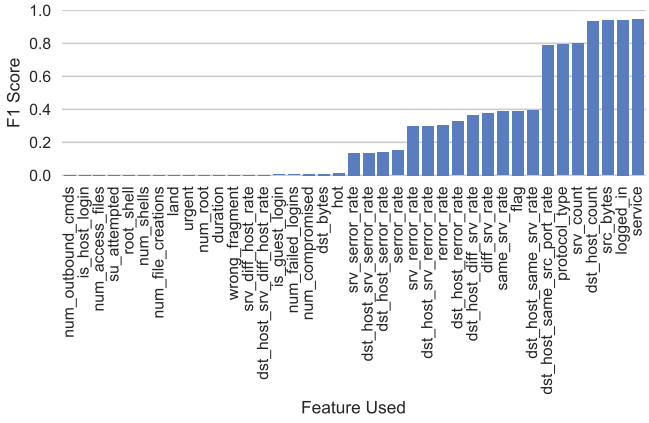

Figure 4: For each feature on the x-axis, we use that feature alone and report the resulting F1.

3.4 Evaluating Additional Features

Thus far, the best quality we have been able to achieve on the 1999 DARPA dataset is an F1 score of 0.36. While promising, this is lower than our desired level of quality. This suggests that only using unencrypted packet header fields as features may not enough to achieve high F1. In this section, we describe some preliminary results in our search for additional features to use. We are looking for features that can be exploited by an online NIDS that only examines unencrypted packet header fields. We find some promising candidate features that can derived by extending the NIDS to store a small amount of state per connection.

To facilitate our search for additional features, we examine the dataset provided in the 1999 Knowledge Discovery and Data Mining Tools Competition, which we refer to as the KDD dataset [1]. The KDD dataset was derived from a set of log files and `tcpdump` traces which are very similar to the ones provided in the 1999 DARPA dataset. These log files and `tcpdump` traces have been processed into connection entries and domain specific knowledge was used to construct a set of 39 different features per connection.

We find that if we use all 39 features and build a simple one-nearest-neighbor classifier then we achieve an F1 score of 0.95. This result indicates that these features make it relatively easy to distinguish between normal and attack connections on this dataset. The obvious questions we would like to ask are: which features are most useful? And, can we use any of the useful features in an online NIDS that only examines unencrypted header fields?

In order to answer these questions, we perform an ablation study where we use each feature individually and run our one-nearest-neighbor classifier; ties are broken by a majority vote. Figure 4 shows the results of this experiment. We see that there are 7 features, which, when used alone, result in an F1 score of about 0.80 or higher:

- `service`: network service running on the destination host
- `logged_in`: if the connection successfully logged in
- `src_bytes`: the number of data bytes transferred from the source to the destination
- `dst_host_count`: the number of connections to the same host as the current connection during the last two seconds
- `srv_count`: the number of connections to the same service as the current connection during the last two seconds
- `protocol_type`: the type of transport layer protocol being used for the connection
- `dst_host_same_src_port_rate`: the percentage of connections to the same host with the same port number on the source host during the past two seconds

The `service`, `protocol_type`, and `srv_count` features work well due to the unique characteristics of the KDD dataset and consequently do not generalize well to other datasets such as the DARPA dataset. The `logged_in` feature would be unavailable to our NIDS because it would require examining packet payloads or server log files. However, the remaining 3 features (`src_bytes`, `dst_host_count`, and `dst_host_same_src_port_rate`) can in fact be computed by our NIDS by keeping a small amount of state per connection. As future work, we would like to build a NIDS that computes these three features and attempt to use them to help identify attacks in network traffic.

4 CONCLUSION

When building an NIDS, it is very important to consider the timing of packets. This importance is reflected in the significantly superior quality of the non-stationary model over the stationary model. In addition to examining time, using raw unencrypted header fields is not enough. In order to achieve higher levels of quality we must use additional features, some of which can be derived by extending the NIDS to store a small amount of state per connection. Building an NIDS is a difficult task due to challenges in collecting data to use for training and testing. Labeling real world data requires a significant amount of time and expertise, and simulation data often does not model the real world accurately. For example, in both of the datasets that we examined we found artifacts that made classification easier than it likely would be in the real world. Hence, it is often a good idea to perform an ablative analysis experiment to see which features are most useful and sanity check that these results make sense by cross referencing with other datasets. It is also difficult to create standard datasets because network traffic is always evolving, and in the real world, it is difficult to capture real network traffic that is truly attack-free, making it challenging to distinguish between normal and anomalous traffic.

5 CONTRIBUTIONS

We have worked together closely on the project. Stephen led the implementation of our evaluation scripts, as well as processing both our training and testing data. In addition, he led experiments with the KDD dataset and the additional features from Section 3.4. Luke has led the PHAD-C32 implementation and ablation studies with this non-stationary model, in addition to leading the implementation and experiments with our mixture of Gaussians baseline. Both of us have tested and contributed to each part of the project by coding, testing, and writing.

We would also like to thank Professors Andrew Ng and Dan Boneh, along with their teaching staff, for their support, advice, and feedback throughout this project.

References

[1] K. Cup. Dataset. *available at the following website http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*, 72, 1999.

[2] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (ToN)*, 9(4):392–403, 2001.

[3] K. Labib and R. Vemuri. Nsom: A real-time network-based intrusion detection system using self-organizing maps. *Networks and Security*, pages 1–6, 2002.

[4] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 darpa off-line intrusion detection evaluation. *Computer networks*, 34(4):579–595, 2000.

[5] M. V. Mahoney and P. K. Chan. Phad: Packet header anomaly detection for identifying hostile network traffic. Technical report, 2001.

[6] M. V. Mahoney and P. K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385. ACM, 2002.

[7] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[8] K. Sequeira and M. Zaki. Admit: anomaly-based data mining for intrusions. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 386–395. ACM, 2002.

[9] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 213–226. ACM, 2015.

[10] C. Thomas, V. Sharma, and N. Balakrishnan. Usefulness of darpa dataset for intrusion detection system evaluation. The International Society for Optical Engineering, 2008.

[11] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *RAID*, volume 4, pages 203–222. Springer, 2004.

[12] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles. Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proc. IEEE Workshop on Information Assurance and Security*, pages 85–90, 2001.