CS424 Network Security: Bayesian Network Intrusion Detection (BNIDS)

Krister Johansen and Stephen Lee

May 3, 2003

1 Introduction

Although law enforcement sometimes employ informants or video/audio surveillance, often it uses simple observations to catch criminals. For example, if a guard works at a number of different banks, each of which has been robbed during or shortly after his employment, then clearly investigators should be astute enough to suspect the guard of some involvement in the crime. Although helpful, investigators should not have to track and document each action of each bank employee. Such a course would swamp the investigators with an excessive amount of detail. Investigators need to focus their efforts and not get lost in the noise.

Like the banking example, computer security has grown to become one of the foremost concerns for IT professionals. Today's networks are large complex systems based upon technology that was not necessarily designed with security in mind. Administrators must confront increasingly sophisticated attacks and intrusions, and in particular, detecting intrusions, whether in progress or post-mortem, has become a necessity to protect entire networks because of the social and legal fallout that can result from their improper and unauthorized use.

Many network intrusion detection systems (NIDS) exist, but they can be easily overwhelmed or bypassed. Additionally, current NIDS rely on specific cues or deviation detection from a valid path in finite state machine that models the system behavior of machines on the network. Instead of taking this very specific approach towards detecting intruders, we propose a system that collects seemingly innocuous data, that may be byproducts of the specific, incisive attacks that afflict networks.

We suggest a Bayesian system which would provide a solid mathematical foundation for simplifying a seemingly difficult and monstrous problem that today's NIDS fail to solve. The Bayesian NIDS (BNIDS) should have the capability to differentiate between attacks and the normal network activity by comparing metrics of each network traffic sample. Finally, such a NIDS should prove to be easily extendable and run in real-time while simple to maintain.

John McHugh chronicles a brief history of the development of NIDS. He observes that many different models(McHugh, 2001) exist and have various degrees of effectiveness and accuracy. Indeed, network intrusion detection is a complication and constantly evolving field. However, unlike other possible solutions, we believe that BNIDS, like other self-organizing statistical models, have the ability to learn and improve as they are constantly exposed to network attacks. Unlike other systems(Paxson, 1999), Our model does not employ a specific language to build filters or rules.

Due to the limited time available and scope of this class, we will restrict our work to determining whether observed traffic is an attack or is legitimate. We will not attempt to take action based upon the decision we reach. This has additional implications about how we deal with false positives, and our rate of identification of different traffic patterns. While it may be interesting to observe these characteristics for analysis of correctness and performance, performing actions to halt an intrusion is outside of the scope which we intend to address.

The gargantuan, arduous task of synchronously stepping through an attack can potentially overwhelm both automatons and humans, given the task of protecting the network. We believe that BNIDS can redefine the problem of network intrusion detection. Network administrators can use the plain sight, readily available measurables from network traffic to build statistical signatures of complex attacks.

2 Related Work

Professor Yarowsky's Information Retrieval and Web Agents class first introduced us to the application of Bayes Theorem to machine learning to solve a problem difficult for computers but simple and intuitive for human beings. Specifically, we used this probabilistic tool to differentiate between word senses, like "army tank" vs. "water tank." While human beings can naturally understand the context of tank, an untrained computer has a low probability of guessing the right sense.

Bayesian models have been used in many different ways to automatically classify seemingly incongruous data. Bayesian analysis has been very noticeably used in spam filtering (Rennie, 2000). Bayesian spam filters can be easily customized to allow different users to classify their mail differently. After all, one person's spam may be another's ham or non-spam. A Bayesian model could be customized to network administrator's tastes, who might prefer a higher or lower threshold for unsolicited traffic. Training a Bayesian model takes away a lot of the guesswork involved with directly correlating network traffic to an attack or normal noise.

Rennie's ifile email filtering software has a simple way to save the trained data and compute probabilities. In short, it keeps a list of keywords and their occurrence frequencies for each email class. In order to adapt the methodology of ifile to network traffic analysis, we need to formulate a number of measures or vector dimensions, like the characteristics proposed by Jung, et. al. While we could start with Jung's characteristics, we need more heuristics in order to fully detail the flash event, denial of service, and normal classes of network traffic. Before we can decide on the additional heuristics to use, we will need to examine the data. If we use synthetic data, we have the option of either uses Jung's characteristics to affirm his method, or possibly use a wider variety to show how his method fails.

Our idea has been additionally explored by some previous research (Burroughs, Wilson, and Cybenko, 2002). Burroughs, et. al. suggests a distributed system to collect data on an attacker's activities. This system would then be capable of assessing the general state of the entire network to be protected. While Burroughs, et. al. proposes a distributed system, we suggest a system that does not necessarily have to be distributed. Instead, network traffic could used post-mortem to train a model that could warn network administrators of a future attacks of similar characteristics.

After discussing in class methods to differentiate between flash events and denial of service attacks (Jung, Krishnamurthy, and Rabinovich, 2002), we noted how network traffic can be represented as an a traffic vector based on traffic patterns, client characteristics, and file reference characteristics. Using a similar representation system, network traffic data could be fed into a probabilistic model that could detect an attack and distinguish the traffic influx from either a flash event or normal traffic. Moreover, deciding which characteristics to measure would have a significant effect on the performance of BNIDS. There are substantial differences between observing that a person who is wearing a red shirt, a person holding a butter knife, and a person brandishing a switch blade.

BNIDS and Protocol scrubbers(Malan et al., 2000; Paxson, Handley, and Kreibich, 2001) come from a conceptually similar root. Scrubbers attempt to generate some well-formed norm of network traffic so that a NIDS may have a consistent and unambiguous view of the network. Similarly, BNIDS attempts to take network traffic, reduce it to fundamental characteristics, and use statistical methods to determine whether the observed traffic fits the accepted norm. While these are quite different behaviors, they are conceptually related in that both attempt to reduce ambiguity in traffic flows. For a scrubber, the goal is to provide a consistent view of traffic. For BNIDS, the goal is to determine whether ambiguous traffic is harmful or benign. We also believe that use of a fully functional BNIDS may obsolete the need for a protocol scrubber. Although there are subtle variances in traffic and packet form, a well trained Bayesian model would be able to discern which of these characteristics are indicative of harmful behavior.

Although our architecture is different from the methods employed in Bro(Paxson, 1999), we believe that the design goals outlined by Paxton are relevant and important for any NIDS that intends to function in a hostile production environment. However, for the purposes of this project, it was unrealistic to attempt to achieve that level of performance. So, for our purposes, we have set out to provide a proof-of-concept BNIDS that may well validate the applicability of Bayesian techniques to intrusion detection.

As a training set for BNIDS, we used data compiled by MIT Lincoln Laboratory (MIT-LL), which labeled sets of network traffic. The MIT-LL also sub-classed the attack into specific types of attack. Although a significant problem on its own, we decided that determining specific attacks like Ping of Death or buffer overflows is beyond the scope of the work we present. Secondly, we originally considered differentiating between attack, flash event, and normal traffic, but we could not correlate the completely different data sets. Specifically, we originally intended to use the World Cup data from the Internet Trace Archive as a flash event data set, but since the data only represented the web objects that were requested over the World Cup period, we could not use similar measures between our Lincoln Labs attack data and the World Cup data. Thus, we limit our results to detecting attacks from non-attacks.

We considered mapping the web objects requested in the MIT Lincoln Labs data to the objects requested in the World Cup data, but we decided that such a modification could incorrectly bias the overall results. Instead, we narrowed our focus to the labeled Lincoln Labs. Bayesian modeling requires accurate statistical data that properly maps to the deployed network topology. If we trained the model using data from MIT-LL, then the rest of our analysis is dependent upon using other data from the same MIT-LL topologies. Since the statistical significance, and likewise, the actual significance of network data is entirely context dependent, we are required to evaluate our models within a consistent context so that we can obtain accurate and meaningful results. If other data sources are used, our statistical model becomes polluted and may generate false positives,

or fail to properly identify attacks. For this reason, we have used only data from MIT-LL's 1999 DARPA traces. These traces simulated attacks on a network maintained by MIT-LL for just this purpose. The datasets were labeled, and contained traces of attack traffic, normal traffic, and mixed traffic. This seemed ideal for our purposes, since it gave us traffic variety, but maintained the relative homogeneity that would provide ideal results for statistical modeling.

3 Methodology

We broke the problem into distinct components, each comprising a part of the BNIDS. One component has the task of extracting statistics from a stream of network traffic, whether replayed or real-time, and generating output as a vector. If BNIDS is presently populating its database, the second component must take the vectors of extracted statistics and store each vector into their respective traffic class: ATTACK, FE, or BENIGN. If our system is currently evaluating a sample of traffic, our next module should take a test vector from the first component and compute a metric for the similiarity of the test vector to each traffic class. Finally, if the metric is high enough, the last subsystem will send out a notification to the network administrator.

The first BNIDS component relies on the libpcap library to calculate some statistics from a sample of network traffic. We implemented a C program to dynamically load a set of shared objects which do the actual packet parsing. Each shared object has a function which conforms to the pcap_handler and a second function which outputs the actual statistics for each packet parser. The libpcap library provides a looping function which will call each packet parser to collect their respective metrics. Once the traffic has been parsed, each shared object's output function is called to produce the traffic vector representation used by BNIDS.

As inferred in the previous paragraph, we produced a number of parsers which extract various statistics from a set of network traffic. One metric, as described in the Jung, et. al paper(Jung, Krishnamurthy, and Rabinovich, 2002), is to calculate the topological distribution(Krishnamurthy and Wang, 2000) of clients. We used routing tables to build a trie whose nodes would keep a counter of network clients that match. In this way, we would be able to compute the number of clients that come from specific networks.

We wrote another parser, similar to the one proposed by Jung, et. al, which would count the number of web requests for the same file references on web server traffic captured by the data set.

To make our BNIDS instance more network specific, in order to identify particularly vulnerable or active network machines, we introduced some simple characteristics like packets per destination or packets per destination:port. Additionally, similar to flow calculation, we captured the number of packets sent between two hosts. We believe that BNIDS would be more successful using statistics that are specific to the network.

To gain a broader sense of the traffic, we also produced metrics which showed the number of flows, average bandwidth usage, and per protocol network activity. Moreover, to make our BNIDS specific to a network, we produced a number of characters which would do per source or destination traffic measures. Our overall hope was to capture enough data in a simple form to summarize the traffic.

Once BNIDS has collected various metrics, we pass the vector to the BNIDS classifier. The classifier iterates through the dimensions of the newly created vector and for each class, computes a similarity measure. In short, if an untested traffic vector has a high value for a dimension that the attack class also has a high value for, then the untested vector's likelihood of being an attack has increased. Conversely, an untested traffic vector's similarity to a class decreases if its dimensions weakly match the magnitudes of the class.

If the classifier has strongly matched the new vector to one of the classes, the new vector might be incorporated into the database, depending on some predetermined, manually selected thresholds.

Our architecture is conceptually similar to the CIDF(Ptacek and Newsham, 1998) model, where one component collects data, another analyzes it, a third provides for countermeasures, and a final component keeps logs. However, since our architecture is currently an offline tool, we train our data using tools that derive characteristics from a packet trace and then insert them into a database. We can then generate characteristics from other traces and run them trough the analyzer to determine their categorization (attack, benign, etc. . .). Right now, we do not perform any further action once we classify the data, however, it would not be unduly complicated to extend the model to archive new attacks and add them to the working set, or to activate countermeasures upon detection of an attack.

Some of the statistical methods we used to generate and analyze statistics are Bayes theorem and a class proximity equation. Bayes Theorem describes the probability of a class occurring based upon the presence of a characteristic. Bayes Theorem is given in figure 1. This states that the probability

$$P(A_i|A) = \frac{P(A_i)P(A|A_i)}{\sum_{j=1}^{N} P(A_j)P(A|A_j)}$$

Figure 1: Bayes Theorem

of an occurrence of A is a function of a characteristic A_i and is some fraction of the probability of finding A within the set of all characteristics. The proximity metric we used to determine whether a new vector was closer to one class or another is given in figure 2. This effectively states that the

$$proximity = \sum_{i=0}^{N} \log\left(\frac{dim_i/vecs}{dims}\right)$$

Figure 2: Proximity Metric

proximity from one class to another is the log of the sum of a chosen dimension divided by the vectors in the class over the total dimensions in the model.

4 Testing

Testing involved training the model with MIT-LL provided labeled data sets. These traces were conveniently labeled for our training pleasure. These traces were organized into weekly samples

from MIT-LL's network. The first and third weeks were attack free. However, the second, fourth, and fifth weeks contained attack data. In order to determine whether we were properly classifying data, we trained our BNIDS using the traces from the first and second weeks, and held weeks three, four, and five for evaluation purposes.

The statistics and results that we were attempting to divine included tracking what the BNIDS guessed, what the data was actually classified as, the similarity of the vector to the attack and benign classes, and the difference between these two similarities. We also ran additional traces where we withheld a certain characteristic in an attempt to determine how that affected the classification and similarity computations. The characteristics that we used in these classifications were:

bgp_cp - A BGP clustering metric which counted the number of packets per cluster.

bgp_c - A BGP clustering metric which counted the number of clients per cluster.

arp - Number of observed ARP packets per-host.

tcp_ip_ip - TCP packets per ip address, for source-destination pairs.

udp_ip_ip - TCP packets per ip address, for source-destination pairs.

http - Http objects observed in trace.

ip - Packets per IP address.

tcp_ip_port - TCP packets per IP-address per port.

udp_ip_port - UDP packets per IP-address per port.

tcp_port - TCP packets per port.

udp_port - UDP packets per port.

host_flow - Observed flows per source address.

tcpdstat - Various ancillary statistics about aggregate traffic observed.

5 Results and Analysis

Although our results were not encouraging, they did provide some insights as to the underlying proof of our concepts, and information about the utility of our characteristics. For the most part, we were unable to correctly classify much of the observed data, and certainly could not claim that we had an appreciable success at detecting network intrusions. However, we found that our model did correctly classify traffic when it bore a close similarity to both the attack and benign classes. We believe that this is an indication that our model is still valid for correctly identifying traffic that may be different in subtle, and complicated ways. This gives us hope that with more data and more accurate characteristics, BNIDS may well perform the tasks we had intended.

While the removal of a particular characteristic did not generally seem to affect the classification of most data, there were notable exceptions. The tcp_ip_port, and to a lesser degree udp_ip_port metrics turned out to be more important than we had anticipated. Although we are not precisely sure as to why they were so important, removal of these metrics from the dataset caused the BNIDS to incorrectly classify many more traces than it had been on average. This leads us to believe that these characteristics are particularly important for the classification of network data.

Just as we had characteristics that were particularly useful, we also observed characteristics that appeared to hinder the classification of our network data. The tcp_port metric, when removed from the database, caused more vectors to be correctly identified. This leads us to believe that this, and a number of other metrics like it, are too general and may be polluting the database. Part of the reason that we believe more granular metrics would be useful is from this result.

We have presented the results of our characterizations in tabular form. The first table shows how our model performed with all characteristics present, and subsequent tables name the amputated metric. The data sets we used are from weeks three, four, and five of MIT-LL's 1999 DARPA funded traces. (Available from: `http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html`) We took their Monday, Wednesday, and Friday datasets and used them to run our classifications.

6 Conclusions and Future Work

Although we were unsuccessful at ultimately differentiating attack data from normal data, we believe that this method has the potential to provide fruitful results in the future. We simply had an inadequate amount of data to properly perform the task. Due to the volume and complexity of the datasets we trained the model on 6 vectors. Each vector represented a trace of network events. Some were benign, others were attacks. We then used 9 other vectors to test the model. We believe that much of the problem with making accurate determinations is based upon the fact that we had a very limited number of vectors for training and testing. In Professor Yarowsky's text sense determination project, we trained our models on a set of 4,000 vectors and were asked to determine the sense of an additional 250-500. Because of this, we believe our dataset to be rather lacking. However, coming up with enough data from a consistent topology is a significant challenge. Another way this model could be improved is to constantly monitor the network for attacks and dynamically add them to the dataset.

We are also not certain that our vector characteristics were granular enough. While we attempted to track flows, network clustering, occurrence of odd packets and IP-addresses, http object requests, and various host and service counts, we remain unconvinced that this is the ideal set of characteristics to implement for network traffic. Further work on this topic would likely yield more impressive results.

Despite a failure to achieve our desired results, we still believe that our method is sound and that with additional data, and time to investigate the problem, meaningful and promising results can be achieved.

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B		128238.529331	130114.295894	1875.766563
w3_m	B	B		112363.074963	112466.100311	103.025348
w3_w	B	B		139277.807388	141281.091600	2003.284212
w4_f	A	A		85237.478845	84561.473877	676.004968
w4_m	A	A		61487.317583	61379.892840	107.424743
w4_w	A	B		95380.861600	95610.942065	230.080465
w5_f	A	B		272933.209546	275119.937429	2186.727883
w5_m	A	B		108556.751290	108697.248366	140.497076
w5_w	A	A		117106.966938	117054.420711	52.546227

Table 1: Evaluation with all characteristics present

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	bgp_cp	127925.365940	129795.959879	1870.593939
w3_m	B	B	bgp_cp	112067.268336	112165.981403	98.713067
w3_w	B	B	bgp_cp	138930.229156	140928.090283	1997.861127
w4_f	A	A	bgp_cp	84986.976624	84307.241167	679.735457
w4_m	A	A	bgp_cp	61256.882581	61146.293342	110.589239
w4_w	A	B	bgp_cp	95118.053761	95344.225598	226.171837
w5_f	A	B	bgp_cp	272396.402868	274574.335103	2177.932235
w5_m	A	B	bgp_cp	108248.447806	108384.346207	135.898401
w5_w	A	A	bgp_cp	116825.813356	116769.111938	56.701418

Table 2: Evaluation with bgp_cp removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	bgp_c	128237.154995	130112.766711	1875.611716
w3_m	B	B	bgp_c	112361.700139	112464.789912	103.089773
w3_w	B	B	bgp_c	139276.352538	141279.635007	2003.282469
w4_f	A	A	bgp_c	85236.077748	84560.006417	676.071331
w4_m	A	A	bgp_c	61486.230944	61378.873764	107.357180
w4_w	A	B	bgp_c	95379.892564	95609.829878	229.937314
w5_f	A	B	bgp_c	272930.995185	275117.501968	2186.506783
w5_m	A	B	bgp_c	108555.675775	108696.048532	140.372757
w5_w	A	A	bgp_c	117105.768738	117053.088844	52.679894

Table 3: Evaluation with bgp_c removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	arp	128238.396877	130114.161627	1875.764750
w3_m	B	B	arp	112362.991718	112466.014991	103.023273
w3_w	B	B	arp	139277.630441	141280.957644	2003.327203
w4_f	A	A	arp	85237.459056	84561.457867	676.001189
w4_m	A	A	arp	61487.316066	61379.892840	107.423226
w4_w	A	B	arp	95380.792744	95610.871193	230.078449
w5_f	A	B	arp	272933.059500	275119.783384	2186.723884
w5_m	A	B	arp	108556.680925	108697.176738	140.495813
w5_w	A	A	arp	117106.937524	117054.393182	52.544342

Table 4: Evaluation with arp removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	tcp_ip_ip	128161.830237	130036.690395	1874.860158
w3_m	B	B	tcp_ip_ip	112280.568608	112382.554243	101.985635
w3_w	B	B	tcp_ip_ip	139170.578365	141173.082745	2002.504380
w4_f	A	A	tcp_ip_ip	85143.941733	84466.659688	677.282045
w4_m	A	A	tcp_ip_ip	61408.348580	61300.041176	108.307404
w4_w	A	B	tcp_ip_ip	95310.672992	95540.164008	229.491016
w5_f	A	B	tcp_ip_ip	272815.042960	275000.367240	2185.324280
w5_m	A	B	tcp_ip_ip	108486.856908	108626.970809	140.113901
w5_w	A	A	tcp_ip_ip	117021.649106	116968.130755	53.518351

Table 5: Evaluation with tcp_ip_ip removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	udp_ip_ip	128238.529009	130114.294605	1875.765596
w3_m	B	B	udp_ip_ip	112363.072717	112466.095063	103.022346
w3_w	B	B	udp_ip_ip	139277.806564	141281.090723	2003.284159
w4_f	A	A	udp_ip_ip	85237.478845	84561.473493	676.005352
w4_m	A	A	udp_ip_ip	61487.317579	61379.892833	107.424746
w4_w	A	B	udp_ip_ip	95380.861595	95610.940890	230.079295
w5_f	A	B	udp_ip_ip	272933.207738	275119.935048	2186.727310
w5_m	A	B	udp_ip_ip	108556.750801	108697.248358	140.497557
w5_w	A	A	udp_ip_ip	117106.966938	117054.419042	52.547896

Table 6: Evaluation with udp_ip_ip removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	http	128232.146262	130107.940222	1875.793960
w3_m	B	B	http	112356.728081	112459.903533	103.175452
w3_w	B	B	http	139270.900838	141274.022533	2003.121695
w4_f	A	A	http	85228.524469	84553.053110	675.471359
w4_m	A	A	http	61479.031549	61372.060949	106.970600
w4_w	A	B	http	95373.926078	95603.949117	230.023039
w5_f	A	B	http	272925.647834	275112.529810	2186.881976
w5_m	A	B	http	108544.630833	108686.039847	141.409014
w5_w	A	A	http	117098.230730	117045.959457	52.271273

Table 7: Evaluation with http metric removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	ip	128232.372242	130107.900072	1875.527830
w3_m	B	B	ip	112357.904494	112460.860987	102.956493
w3_w	B	B	ip	139273.217722	141276.117816	2002.900094
w4_f	A	A	ip	85232.292782	84556.128246	676.164536
w4_m	A	A	ip	61483.794685	61376.241573	107.553112
w4_w	A	B	ip	95375.730149	95605.643012	229.912863
w5_f	A	B	ip	272925.416118	275111.766620	2186.350502
w5_m	A	B	ip	108551.775447	108692.328892	140.553445
w5_w	A	A	ip	117100.748574	117047.984027	52.764547

Table 8: Evaluation with ip metric removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	tcp_ip_port	25911.884743	26231.712674	319.827931
w3_m	B	B	tcp_ip_port	31786.834576	32119.288235	332.453659
w3_w	B	B	tcp_ip_port	30676.123992	31034.111632	357.987640
w4_f	A	B	tcp_ip_port	25118.221449	25378.092424	259.870975
w4_m	A	B	tcp_ip_port	26677.311386	27022.563524	345.252138
w4_w	A	B	tcp_ip_port	25839.045426	26112.840948	273.795522
w5_f	A	B	tcp_ip_port	41354.023220	41807.617356	453.594136
w5_m	A	B	tcp_ip_port	45599.688187	46206.175967	606.487780
w5_w	A	B	tcp_ip_port	35896.821731	36332.677105	435.855374

Table 9: Evaluation with tcp_ip_port removed – suggesting a useful metric

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	udp_ip_port	124563.753302	126524.814930	1961.061628
w3_m	B	B	udp_ip_port	106565.031722	106805.721115	240.689393
w3_w	B	B	udp_ip_port	134541.738821	136656.474594	2114.735773
w4_f	A	A	udp_ip_port	80655.242252	80088.717795	566.524457
w4_m	A	A	udp_ip_port	58295.675361	58264.383899	31.291462
w4_w	A	B	udp_ip_port	90768.474019	91108.216234	339.742215
w5_f	A	B	udp_ip_port	265727.354404	268081.892706	2354.538302
w5_m	A	B	udp_ip_port	103203.648488	103473.031214	269.382726
w5_w	A	B	udp_ip_port	111911.787465	111982.331847	70.544382

Table 10: Evaluation with udp_ip_port removed – also suggesting a useful metric

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	tcp_port	106405.330374	107882.421929	1477.091555
w3_m	B	A	tcp_port	86765.574280	86403.693321	361.880959
w3_w	B	B	tcp_port	113805.688057	115346.241419	1540.553362
w4_f	A	A	tcp_port	65061.094647	64020.434684	1040.659963
w4_m	A	A	tcp_port	38323.949518	37798.791660	525.157858
w4_w	A	A	tcp_port	74500.305068	74351.797702	148.507366
w5_f	A	B	tcp_port	239457.736087	241033.707629	1575.971542
w5_m	A	A	tcp_port	68706.606443	68115.868724	590.737719
w5_w	A	A	tcp_port	86787.978283	86181.690203	606.288080

Table 11: Evaluation with tcp_port removed – suggesting a harmful metric

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	udp_port	128238.529331	130114.295894	1875.766563
w3_m	B	B	udp_port	112363.074963	112466.100311	103.025348
w3_w	B	B	udp_port	139277.807388	141281.091600	2003.284212
w4_f	A	A	udp_port	85237.478845	84561.473877	676.004968
w4_m	A	A	udp_port	61487.317583	61379.892840	107.424743
w4_w	A	B	udp_port	95380.861600	95610.942065	230.080465
w5_f	A	B	udp_port	272933.209546	275119.937429	2186.727883
w5_m	A	B	udp_port	108556.751290	108697.248366	140.497076
w5_w	A	A	udp_port	117106.966938	117054.420711	52.546227

Table 12: Evaluation with udp_port removed

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	host_flow	128238.529331	130114.295894	1875.766563
w3_m	B	B	host_flow	112363.074963	112466.100311	103.025348
w3_w	B	B	host_flow	139277.807388	141281.091600	2003.284212
w4_f	A	A	host_flow	85237.478845	84561.473877	676.004968
w4_m	A	A	host_flow	61487.317583	61379.892840	107.424743
w4_w	A	B	host_flow	95380.861600	95610.942065	230.080465
w5_f	A	B	host_flow	272933.209546	275119.937429	2186.727883
w5_m	A	B	host_flow	108556.751290	108697.248366	140.497076
w5_w	A	A	host_flow	117106.966938	117054.420711	52.546227

Table 13: Evaluation without host_flow metric

Data Set	Truth	Guess	Removed Metric	Attack Similarity	Benign Similarity	Sim Difference
w3_f	B	B	tcpdstat	128238.529331	130114.295894	1875.766563
w3_m	B	B	tcpdstat	112363.074963	112466.100311	103.025348
w3_w	B	B	tcpdstat	139277.807388	141281.091600	2003.284212
w4_f	A	A	tcpdstat	85237.478845	84561.473877	676.004968
w4_m	A	A	tcpdstat	61487.317583	61379.892840	107.424743
w4_w	A	B	tcpdstat	95380.861600	95610.942065	230.080465
w5_f	A	B	tcpdstat	272933.209546	275119.937429	2186.727883
w5_m	A	B	tcpdstat	108556.751290	108697.248366	140.497076
w5_w	A	A	tcpdstat	117106.966938	117054.420711	52.546227

Table 14: Evaluation without tcpdstat metric

References

Burroughs, Daniel J., Linda F. Wilson, and George V. Cybenko. 2002. Analysis of Distributed Intrusion Detection Systems Using Bayesian Methods. In *IPCCC 2002*.

Burschka, Darius, Stephen Lee, and Gregory Hager. 2002. Stereo-Based Obstacle Avoidance in Indoor Environments with Active Sensor Re-Calibration. In *ICRA 2002*, pages 2066–2072, Washington, DC, USA.

Jung, Jaeyeon, Balachander Krishnamurthy, and Michael Rabinovich. 2002. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *WWW2002*, Honolulu, HW, USA.

Krishnamurthy, Balachandar and Jia Wang. 2000. On network-aware clustering of web clients. In *SIGCOMM*, pages 97–110.

Malan, G. Robert, David Watson, Farnam Jahanian, and Paul Howell. 2000. Transport and application protocol scrubbing. In *INFOCOM (3)*, pages 1381–1390.

McHugh, John. 2001. Intrusion and intrusion detection. Technical Report IJIS (2001) 1:14-35, Computer Emergency Response Team (CERT), Pittsburgh, Pennsylvania, July 27th.

Paxson, Vern. 1999. Bro: A system for detecting intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, December 14th.

Paxson, Vern, Mark Handley, and Christian Kreibich. 2001. Network intrustion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium*, Washington DC, August. USENIX.

Ptacek, Thomas H. and Timothy N. Newsham. 1998. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., January.

Rennie, Jason D. M. 2000. ifile: An Application of Machine Learnign to E-Mail Filtering. In *KDD-2000 Text Mining Workshop*, Boston, MA, USA.

Yarowsky, David. 1992. Word-Sense Disambiguation Using Statistical Models of Roget's Categories Trained on Large Corpora. In *COLING-92*, pages 454–460, Nantes, France.