An Improved Method for Anomaly-Based Network Scan Detection

Ashton Webster(✉), Margaret Gratian, Ryan Eckenrod, Daven Patel, and Michel Cukier

University of Maryland, College Park, USA
{awebste2,mgratian,eckenrod,dpatel19,mcukier}@umd.edu

Abstract. Network scans, a form of network attacker reconnaissance, often preface dangerous attacks. While many anomaly-based network scan detection methods are available, they are rarely implemented in real networks due to high false positive rates and a lack of justification for the chosen attribute sets and machine learning algorithms. In this paper, we propose a new method of scan detection by selecting and testing combinations of attribute sets, machine learning algorithms, and lower bounded data to find a Local Optimal Model.

Keywords: Machine learning · Network intrusion detection · Anomaly-based detection · Network security · Scanning

1 Introduction

Each year, new and devastating cyber attacks amplify the need for robust cybersecurity practices. Preventing novel cyber attacks requires the invention of intrusion detection systems (IDSs) that can identify previously unseen attacks. To this end, many researchers have attempted to produce anomaly-based IDSs using machine learning techniques [1, 2, 3, 4]. However, anomaly-based IDSs are not yet able to detect malicious network traffic consistently enough to warrant implementation in real networks [2, 5]. It remains a challenge for the security community to produce anomaly-based IDSs that are suitable for adoption in the real world [5, 6].

One promising field of study has been anomaly-based network scan detection. This line of research aims to detect network scans that often precede cyber attacks so that potential attackers can be identified and blocked. Specifically, many researchers have focused on using network flow data as an anomaly-based scan detection medium [7, 8, 9]. To improve upon previous research in this field, we present a method for identifying an effective network flow-based machine learning model for scan detection on a given network. Network administrators utilizing this method on their own networks can use the scan detection models produced to create personalized anomaly-based scan detection systems. In addition, we present an application of this method on the University of Maryland network.

The remainder of this paper is organized as follows. Section 2 details the background of this paper and related work. Section 3 lists our contributions and defines terms specific to this paper. Section 4 details our method and an application of this

B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 385–400, 2015.
DOI: 10.1007/978-3-319-28865-9_21

method on the University of Maryland network. Section 5 presents and discusses the results obtained and the possible limitations for our method. Finally, Section 6 presents the conclusion.

2 Background

2.1 Scanning

Network scanning, a form of network reconnaissance, often prefaces a cyber attack [7, 10]. Through various scanning techniques, an attacker will attempt to gain information about network configurations, server implementations, and potential vulnerabilities before launching more invasive exploits. Thus, scan detection is vital to the security of a network [1].

Scans can be classified into two broad categories: vertical and horizontal [10]. Vertical scans are directed at a specific host and include an in-depth examination of ports and protocols being used by the host. Horizontal scans sweep over several hosts within the targeted network and seek general information about configurations, operating system versions, and more. Vertical and horizontal scans can also be made "stealthy" by increasing the time between each successive port contact to avoid detection [11].

Rule-based thresholding is the most common method of scan detection [1, 7]. IP addresses are declared as scanners after their connection attempt count exceeds a predetermined limit. This method has a low detection rate and an "unacceptable" false alarm rate [1, 6]. Other rule-based processes are burdensome, time consuming, and prone to human error; scan detection is often skipped or overlooked for this reason [9, 12].

2.2 Anomaly Detection Systems in Scan Detection and Machine Learning

At the most fundamental level, anomaly detection involves examining data for unusual patterns [3]. This method of detection aims to classify data as either normal or abnormal based on a given definition of normalcy. In the context of anomaly-based scan detection, we use the terms 'benign' to describe normal network users and 'malicious' to describe network scanners. For the purpose of this paper, we define a user as a unique source IP address producing traffic on a network.

In an anomaly-based scan detection system, a normal network user profile is created and anomalies are treated as network scans. The system's classification success is dependent upon the number of true positives, true negatives, false positives, and false negatives it produces [13]. Based on this convention, these terms are defined as follows:

True positives: The number of correctly labeled malicious users.
False positives: The number of benign users incorrectly labeled as malicious.
False negatives: The number of malicious users incorrectly labeled as benign.
True negatives: The number of correctly labeled benign users.

In applying anomaly-based scan detection, an important goal to strive for is the reduction of false negatives and false positives [14]. This is because false positives

result in a waste of resources, while false negatives result in undetected malicious activity [2, 5, 7].

Machine learning classifies data instances using a set of predictive heuristics [12]. This technique uses learning algorithms and input data to create a predictive mathematical model for classifying further data. Machine learning has been proven to be a useful tool for anomaly detection, with successful applications in fields such as keystroke dynamics, malicious system trace detection, and user behavior at the command line [4, 15, 16]. We will refer to machine learning models created for scan detection as 'scan detection models.'

2.3 Previous Work

Previous studies have analyzed the effectiveness of network flow-based scan detection models [7, 8, 17]. Network flow is a protocol for recording network traffic between two IP addresses and has proven to be a useful source of data for machine learning [8, 17, 18, 19]. When using network flow records, a minimum is often set on the number of network flow records a user must produce in order to be classified as a normal user or network scanner. This practice of setting a minimum number of records will be referred to as setting a 'lower bound.' This lower bound is similar to one used by the Threshold Random Walk scan detection method [17], which determines a minimal number of connection attempts a source IP address must make to distinct destination IP addresses to be accurately classified. However, network flow-based scan detection studies often choose a lower bound without giving a strong justification for the choice [7]. Moreover, other studies attempt to classify IP addresses that have only produced one record, without verifying in detail whether one record provides enough data to make an accurate classification [19].

Another important aspect of creating scan detection models is the calculation of network flow attributes. Among the attributes calculated, researchers often attempt to identify a subset of the attributes that enables high classification performance when used to create scan detection models [8, 9]. This process (called attribute set selection) reduces storage costs and computing complexity, and it eliminates extraneous attributes that reduce the accuracy of machine learning classification [14, 20, 21]. Although other researchers often identify and utilize a reduced attribute set using an attribute selection algorithm [8, 9], there may exist other attribute sets which will better classify the users. Since the "quality of the… [attribute set] is crucial to the performance of a [machine learning] algorithm," it is essential to test and evaluate multiple attribute sets [14]. Similarly, many studies classify users using only one or very few machine learning algorithms, and thus can only make claims about a specific group of algorithms and their classification success [7, 19].

Despite extensive research, the strategy of using anomaly-based scan detection in concert with machine learning has been "rarely employed in operational 'real world' settings" [2]. One of the primary challenges with machine learning applications is developing a realistic and accurately labeled network dataset for training; this issue stems from the loose definition of the term 'scan' and the high variability of networks [2]. Other studies use publicly available datasets that are over fifteen years old or are known to inaccurately model real networks [7, 12], such as the DARPA - 98/99, the KDD-99, and the Kyoto 2006 datasets [22, 23, 24]. Furthermore, some papers seek to

make claims about scan detection on datasets that have a disproportionately high number of malicious sources [5, 8]. This leads to training machine learning models on datasets that are not large enough or representative enough of the desired network to make successful classifications and prevents the work from being generalized to most other networks [2, 12, 22, 23].

3 Contributions

Motivated by previous work and the challenges associated with anomaly-based scan detection using network flow records, we present a general method for creating improved machine learning scan detection models. In order to precisely define our task, we introduce the following terms:

Aggregate Metric Value (AMV): A value calculated based on multiple classification metrics for a given scan detection model. For the purpose of this paper, we use the average of accuracy, precision, and sensitivity.
Local Optimal Model (LOM): The scan detection model with the highest performance based upon a given AMV among all of the models generated.

We frame scan detection model creation as an optimization problem with three variables: (1) a network flow attribute set, (2) a machine learning classifier, and (3) a lower bound. Our method creates multiple scan detection models based upon combinations of these variables. The AMV of each model is then calculated to find a specific network's LOM. This framework for scan detection model creation seeks to resolve challenges inherent in using machine learning for scan detection, including network-specific models and arbitrary selection of attribute sets, machine learning algorithms, and lower bounds.

By proposing a solution to this problem, we contribute the following:

- We provide a customizable method of identifying the combination of attribute set, machine learning classifier, and lower bound that creates a Local Optimal Model for a specific network.
- We compare and evaluate the implications of applying lower bounds on the number of records necessary for classification.
- We demonstrate an application of this method on the University of Maryland network.

Our method utilizes supervised machine learning and is outlined by the work flow diagram in Figure 1. To implement machine learning for our experiment, we selected the Weka machine learning library [25].

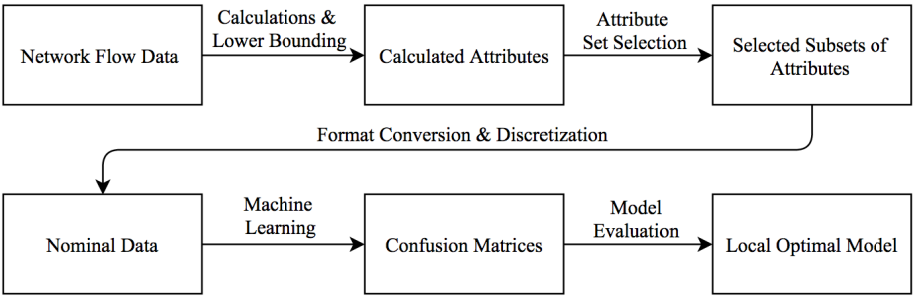

Fig. 1. This work flow diagram of our method provides an overview of the process of collecting network flow data, deriving attributes for each network user for several lower bounds, selecting attribute sets, creating scan detection models using machine learning, and finally evaluating the resulting models to find the LOM.

4 Method

4.1 Network Flow Data Collection

Network flow data must first be collected over a network. Two sets of network flow data should be collected so that scan detection models can be trained on one set and tested on the other. Having independent training and testing datasets avoids the potential issue of overfitting the training data [26]. An additional step that can be taken is to inject network scans into the data. Injecting scans will ensure that the data contains some scans, but can also corrupt the dataset as discussed in Section 5.5.

For our experiment, we chose to create a scan detection model for the University of Maryland network. We collected the network's Cisco NetFlow [18] data over the month of November and nine days of December, 2013. We trained scan detection models using the month of November data, which we will call Dataset 1, and tested the models on the nine days of December data, which we will call Dataset 2. As the datasets were collected, they were inserted into two separate databases.

To challenge our models to detect stealthy network scans that might occur on a network, we chose to inject a sample of stealthy scans into each dataset. We set up a network of 32 virtual machines (VMs) to develop stealth scanning profiles. On the network, the open source scanning tool Nmap [27] was used to test different scans against the Snort IDS [28]. In order for the scans to be representative of stealth scans, Snort was configured with the strictest port scanning thresholds available. Stealth scanning profiles for horizontal and vertical scanning were derived from extensive testing against Snort and research into stealth scanning behavior. It was found that the slightest modifications from the default Nmap settings proved sufficient to evade Snort. The scans were developed with the additional goal of discovering enough host information to launch an exploit or attack while sending a small number of port probes over a one month period.

Once we were confident that our scan profiles were representative of stealth scanning, we configured a network to perform these scans. Five VMs were set up to scan two Class C subnets of the University of Maryland network multiple times.

The subnets were scanned over the same time period as the network's flow records were collected. Thus, the scanning records were inserted in real time into the Dataset 1 and Dataset 2 databases alongside the network's inherent flow records.

Once the scans were finished, each complete scan was relabeled with a unique source IP address within its network flow data so that every scan seemed to originate from a new IP address. This was done because more than five scanners were desired for the dataset. This was also done to replicate the difficulty of detecting scanners who perform only a few scans. In turn, the scan detection models must become more robust to accurately classify these users. The list of malicious IP addresses produced in this step was stored separately for labeling.

4.2 Network Flow Data Labeling

Once the network flow data has been collected, every user must be labeled as malicious or benign in order to perform supervised machine learning. For simplicity, our method identifies a network user as any unique source IP address that accesses a network. The advantage of injecting scans into the dataset is that the injected IP addresses can be labeled as malicious because they are the output of the Nmap scans. Labeling the remainder of the network users confidently is much more difficult. It is a time consuming and imperfect process as described in Section 5.5. To label any inherent malicious users within the data, a set of heuristics that identifies a user as a network scanner based upon the user's network flow data must be defined and applied. These heuristics can be based upon accepted definitions of scans or based on the specific network's configuration. For example, if access to port 22 is closed on the network and analyzing network flow records reveals a user attempted to access the port on several hosts, the user could potentially be labeled a scanner. Users that do not fit these heuristics should be labeled as benign users.

In our dataset, the labeling process occurred as part of the network flow attribute calculations. Users were automatically labeled as malicious if they were among the injected scans. For non-injected users, we defined a strict set of heuristics to identify any network scanners on the university network. A user was labeled as malicious if the IP address displayed horizontal or vertical scan detection behavior on the network. Additionally, any user who attempted to access a single closed port on the network was labeled as a scanner. All other users were labeled as benign.

4.3 Attribute Calculation

Statistics about a user's network flow data must be calculated to classify the user as malicious or benign using machine learning. Choosing which network flow attributes to calculate is the first step in this process. Any attribute that is believed to differ between benign network users and malicious users can be chosen. Other researchers have identified potentially useful attributes [7, 8, 19]. Once the attributes have been selected, they need to be calculated for each user that accessed the network. The user's label must be added to calculated attributes for use by the machine learning algorithms. An additional attribute that can be calculated in this step is the number of network flow records that the user produced. This attribute can be used to quickly implement lower bounding in the next step.

To perform these operations with our data, we first compiled a list of 34 network flow attributes to calculate based on the attributes used by Gates et al. [8] and Williams et al. [19]. A full listing can be can be found in Appendix A. We then created a script that individually queried the Dataset 1 and Dataset 2 databases for each user's network flow records. The 34 attributes, the user's label, and the number of records produced by the user were then calculated for each IP address in each set. This process generated two CSV files: a training dataset of Dataset 1's users and corresponding calculations, and a testing dataset that contained Dataset 2's users and calculations.

4.4 Lower Bounding

A lower bound refers to the minimum number of network flow records a user must produce to be classified. Our study introduces the concept of varying lower bounds during model creation to discover how lower bounds affect a model's AMV. Different lower bounds should be chosen in search of the LOM as they may impact the AMV, as demonstrated by our findings. To test the application of lower bounds, network users that did not produce certain numbers of network flow records should be removed from the calculations dataset. Each lower bounded set of calculations should be saved separately for evaluation.

For our experiment, we chose to test the lower bounds of 2, 4, 6, 8, 10, 30, and 50 network flows, as well as no lower bound (a bound of 1 record). To perform the lower bounding, each calculation file was copied and all IP addresses that did not produce the minimum number of records removed from the file. The resulting files and the number of benign and malicious IP addresses left in each is detailed in Table 1.

Table 1. Number of Malicious and Benign Users

	Dataset 1		Dataset 2	
Lower Bound	**Benign**	**Malicious**	**Benign**	**Malicious**
1	149,600	91	103,663	57
2	115,475	86	88,042	57
4	85,874	85	73,328	57
6	71,836	83	65,446	56
8	63,384	83	59,821	56
10	57,500	78	55,918	56
30	33,913	59	39,396	50
50	25,536	56	32,943	40

4.5 Attribute Discretization

Before attribute selection or model creation can occur, the attribute calculations must be converted into a machine learning format and discretized. The conversion is a simple formatting change into a syntax on which machine learning algorithms can operate. Then the calculations must be discretized so that each attribute's range of calculated values for the users is no longer continuous but in nominal, categorized sets of values useful for machine learning.

In order for our data to be utilized by the Weka library, our calculations in CSV format were converted to ARFF (Attribute-Relation File Format) by means of a simple translation script. The ARFF files were then discretized using a modified version of Fayyad and Irani's Minimum Description Length (MDL) discretize function implemented in the Weka library that split an attribute's range of values at least once ("binary discretization") [30]. Without this modification, some attribute ranges would not have been split at all, resulting in only one discretized category for those attributes. Having a singular category for an attribute renders the attribute useless for classification since all users will have the same value for the attribute.

4.6 Attribute Set Selection

Following discretization, network flow attribute sets must be identified in order to build scan detection models. An attribute set refers to a subset of all the network flow attributes that were calculated for each IP address. Prior research has shown that choosing to classify instances based upon a strongly predictive subset of attributes instead of using all attributes can increase classification performance [30, 31]. Attribute sets are thus the second variable component of scan detection model creation that must be explored in search of the LOM.

Attribute sets can be selected in a number of ways. Every combination of the network flow attributes could be selected as an attribute set. However, testing all of the sets may prove infeasible if many attributes were calculated because the number of tests necessary grows exponentially with the number of attributes. Attribute sets can alternatively be selected manually or through the use of attribute selection algorithms. These algorithms are designed to identify which attributes are the most useful for distinguishing items of one labeled class from items of another class for a given dataset.

Given the 34 network flow attributes we identified and calculated for each user, selecting every possible attribute set combination for testing was deemed infeasible because there are 17 billion ways of combining the attributes into subsets. We therefore turned to the Weka machine learning library for attribute selection algorithms. Making no assumptions about which network flow attributes would best differentiate normal network users from network scanners, we solely relied upon these algorithms to identify useful attribute sets to test.

There are two types of attribute selection algorithms in the Weka library: subset evaluators and attribute evaluators. Subset evaluators attempt to identify the subset of all attributes that best differentiates between classes, while attribute evaluators simply rank all attributes by their perceived usefulness for differentiating between classes [32]. For our experiment, we selected every subset evaluator algorithm Weka provided that returned non-empty attribute sets, and we selected every attribute evaluator algorithm that ranked at least five attributes with nonzero scores. Thus, the CFS and Consistency subset evaluators were chosen along with the Chi-Squared, Gain Ratio, Info Gain, and Symmetrical Uncertainty attribute evaluators.

These algorithms were run on each of Dataset 1's lower bounded files. Since each attribute evaluator returned only rankings of attributes instead of a subset of them, constructing subsets from these rankings required choosing some number of the highest ranked attributes from the ranking list. For each of the attribute evaluator

algorithms we chose, we decided to create six subsets based upon the rankings returned. The first subset contained the best 5 attributes, the second contained the best 10 attributes, the third contained the best 15 attributes, and so on up to the sixth subset, which contained the best 30 attributes. We also selected a control attribute set where all 34 attributes were selected. Along with the two subsets created by the subset evaluators, this amounted to 27 selected attribute sets per lower bounded file. In some cases, duplicate subsets were produced across algorithms and files with different lower bounds, so the final number of unique subsets created for testing was 122.

4.7 Machine Learning Model Creation

After attribute selection, machine learning algorithms must be selected so that scan detection models can be created and tested. Different machine learning algorithms may classify a dataset differently and are thus the third variable that should be tested in search of the LOM. Machine learning algorithm selection can be based upon the unique benefits of certain algorithms, an algorithm's classification performance in other settings, the distribution of malicious and benign users within the data, or some other prior knowledge. Since justifying the selection of a machine learning algorithm can still be challenging, we propose selecting many algorithms to test in order to compare their classification results.

Once the machine learning algorithms are selected, scan detection models are created by training each algorithm on the training datasets according to attribute sets. Each algorithm should train on each training dataset generated by applying a lower bound, using every attribute set selected for testing. This results in one unique scan detection model for every combination of the three variables. Afterwards, the models are tested on the corresponding lower bound testing dataset. The output of this stage is a classification confusion matrix for each of the scan detection models.

For our experiment, we selected the following five machine learning algorithms implemented in the Weka Machine Learning Library: Random Forest, AODE, PRISM, SMO, and Decision Table. We sampled algorithms from different categories of machine learning algorithms, including Tree Based (Random Forest), Rule Based (Decision Table), and Bayes (AODE), among others. We trained each of these machine learning algorithms on each of the 8 lower bounds files with each of the 122 attribute sets, resulting in 4,880 unique scan detection models. We then tested every scan detection data model on the corresponding lower bounded data from Dataset 2 with the same attribute sets to generate a set of confusion matrices for comparison of the models.

4.8 Model Evaluation

Once scan detection models are created and confusion matrices are generated from testing, classification metrics can be derived from the matrices to determine which model best classified the data. Models can be evaluated according to a single metric such as accuracy. However, the base rate fallacy is a serious problem for scan detection, as the vast majority of the network users are usually benign [5]. This means that if a model classified every user as benign, it will still have a high classification accu-

racy. Therefore, it is advisable to combine and weigh multiple classification metrics into a single score: what we call an Aggregate Metric Value (AMV). We term the model with the highest AMV to be the Local Optimal Model (LOM). The general method to produce a simple weighted AMV is as follows:

$$\sum_{i=0}^{n} w_i * m_i \tag{1}$$

where w_i is the assigned weight for the metric m_i and n is the total number of metrics.

Once an AMV is selected, it should be calculated for every scan detection model generated from the previous step based upon its testing confusion matrix. Models can then be sorted by descending AMV to find the LOM. If the AMV of this model is deemed sufficient, it can be deployed as an anomaly-based scan detection system on the live network. For our data, we used the following AMV, based on conventional definitions of precision (p), sensitivity (s), and accuracy (a):

$$AMV = \tfrac{1}{3} * p + \tfrac{1}{3} * s + \tfrac{1}{3} * a \tag{2}$$

where

$$p = \frac{TP}{TP+FP} \tag{3}$$

$$s = \frac{TP}{TP+FN} \tag{4}$$

$$a = \frac{TP+TN}{TP+FP+TN+FN} \tag{5}$$

We calculated this AMV for each of the 4,880 scan detection models generated in the previous step and sorted by descending AMV to find the LOM for our data.

5 Results and Discussion

The following section presents the results of implementing our method on the University of Maryland network. With these results, we will illustrate how each variable of lower bound, attribute set, and machine learning classifier impacts the AMV performance of a scan detection model. While we analyze which values of the variables performed well on our network dataset, we recognize that these specific values may not extend to other networks.

Table 2 displays the classification results of the LOMs created based on different AMVs. From this table, we see that the selection of our AMV for model evaluation returns nearly three times as many correctly identified scans than evaluating solely by accuracy. While evaluating models by our AMV identified the same LOM as evaluating by precision, this is simply a coincidence based on our particular selection of AMV as the average of sensitivity, precision, and accuracy.

Table 2. Comparison of AMVs

Metric	Lower Bound	True Positives	False Positives	False Negatives	True Negatives
Accuracy	1	6	4	51	103659
Precision	4	17	4	40	73324
Sensitivity	1	57	103612	0	51
Our AMV	4	17	4	40	73324

5.1 The Role of Lower Bound on Metric Performance

One of the unique aspects of our method is treating lower bound as another scan detection model input variable. Table 3 compares the models with the same machine learning classifier and attribute set as our LOM, but with different lower bounds. The table illustrates a noticeable drop in performance if a lower bound other than 4 is chosen. If no lower bound is used, (designated by the lower bound row 1) only two instances are correctly classified as malicious. These results suggest that evaluating multiple lower bounds can produce models with higher AMV performance.

The tradeoff of using lower bounds is that the model ignores users who only produce a few network flow records. Essentially, this is equivalent to requiring network scans to consist of at least a minimum number of flows. It is possible that the unlabeled users for one model could be labeled as malicious in another model with a different lower bound. However, neither model is "mislabeling" the data, as they are attempting to detect scans based on fundamentally different definitions. Based on these facts, it is important to note that the LOM returned by our method will use the definition of a scan based on the lower bound with the best performance for the given AMV.

Table 3. Comparison of Lower Bounds by Descending AMV

Lower Bound	True Positives	False Positives	False Negatives	True Negatives	Our AMV
4*	17	4	40	73324	0.7024
6	15	9	41	65437	0.6307
10	10	8	46	55910	0.5777
8	12	13	44	59808	0.5644
2	4	4	53	88038	0.5232
30	5	10	45	39386	0.4773
1	2	4	55	10365	0.4560
50	4	11	36	32932	0.4551

*Lower bound selected by method

5.2 The Role of Attribute Sets on Metric Performance

Table 4 shows the impact of the attribute set on performance, controlling for the lower bound and classifier. The impact of attribute set is more subtle than machine learning algorithm or lower bound selection, resulting in only minor variations in the false positive rate and the number of correctly classified instances. This reflects the tendency of the method to generate multiple viable attribute sets.

Table 4. Comparison of Top 5 Attribute Sets by Descending AMV

Rank	True Positives	False Positives	False Negatives	True Negatives	Our AMV
1*	17	4	40	73324	0.7024
2	18	6	39	73322	0.6884
3	18	6	39	73322	0.6884
4	14	4	43	73324	0.6743
5	16	6	41	73322	0.6701

*Attribute set selected by method

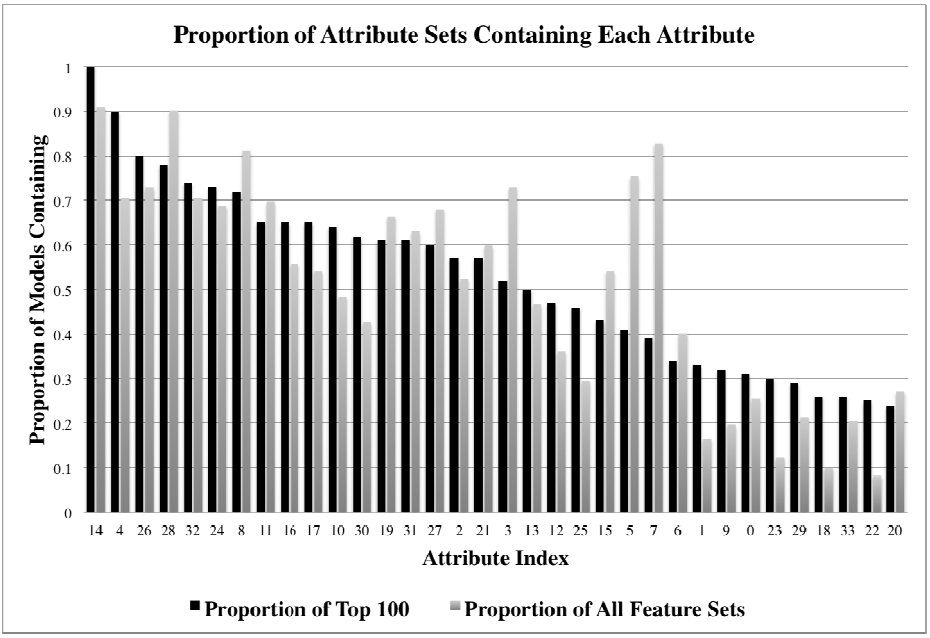

Fig. 2. Proportion of top 100 models and all models containing each attribute

Figure 2 displays two calculations: (a) the proportion of the attribute sets from the top 100 models containing a given attribute and (b) the proportion of all attribute sets containing a given attribute. Every one of the top 100 models' attribute sets contained attribute 14, "rt_std_pttrn" (see Appendix A for details). One interesting finding is that although more than 70% of all the attribute sets generated by the attribute selection algorithms contained attributes 5 and 7, the attributes were only present in the top 100 models about 40% of the time. This suggests that attribute set selection algorithms may choose attributes that appear less frequently in the attribute sets of models with the highest AMVs. This reinforces the importance of trying various combinations of attribute sets for comparison based on results.

5.3 The Role of Machine Learning Classifiers on Metric Performance

By controlling for lower bound and attribute set, Table 5 shows that machine learning algorithm selection has a significant impact on AMV performance. Random Forest outperforms the other algorithms by a considerable margin in terms of false positive rate, with only 4 false positives for the 17 correctly identified scans.

Table 5. Comparison of Machine Learning Algorithm by Descending AMV

Classifier	True Positives	False Negatives	False Positives	True Positives	Our AMV
Random Forest*	17	40	4	73324	0.7024
AODE	14	43	38	73290	0.5046
Prism	24	30	425	72883	0.4972
SMO	1	56	18	73310	0.3564
Decision Table	0	57	67	73261	0.3328

*Machine learning algorithm selected by method

For classifiers, Random Forest was the most successful at achieving high AMV values. In fact, the top 26 models by AMV were all achieved using Random Forest, with AODE first appearing at position 27. The Prism machine learning algorithm was not used to generate any of the top 100 results, largely due to its propensity to label large portions of the data as malicious.

6 Conclusions

By treating the creation of scan detection models as an optimization of an AMV using the best combination of lower bound, attribute set, and machine learning algorithm, a flexible framework for identifying LOMs is created. We were able to evaluate our model on the University of Maryland network and successfully identify the LOM. Our results demonstrate that different lower bounds, attribute sets, and machine learning algorithms are necessary to evaluate because they impact the AMV of a scan detection model. We improve upon an arbitrary selection of these variables when creating models by using a model's performance to justify the variables' values. This will provide a more practical method of creating network specific scan detection models in operational settings.

While our method successfully identified the LOM for the University of Maryland network, the method should be easily extendable to other networks. Network administrators should start by selecting their own network flow attributes to calculate. Then, they can create models using their own selection of lower bounds, attribute sets, and machine learning algorithms. Finally, the models should be compared using a customized AMV to produce a network specific LOM.

Despite our method's benefits, it is limited by its reliance upon supervised machine learning. Performing supervised learning requires every source IP address in the network flow data to be labeled as malicious or benign prior to testing. This labeling is time consuming, and it requires a network administrator to have thorough knowledge of a network's configuration and network scans to label every IP address in the network flow data confidently. Even if a network administrator labels every IP address according to some strict set of heuristics, there is no ground truth regarding which

users are truly scanners. We attempted to counteract this problem by injecting scans into the network flow data that could be labeled with ground truth. However, injecting anomalous data into a network dataset can make the dataset no longer representative of a real world network [25, 34].

The alternative approach of semi-supervised machine learning would require administrators to only label a few IP addresses in the network that are known to be malicious or benign such as injected network scans or websites commonly visited over the network. An avenue for future research is evaluating if applying semi-supervised learning to such a method can produce scan detection models with classification success similar to that of models produced using supervised learning.

Appendix A: Calculated Network Flow Attributes

Network Flow Attributes Calculated for Each Source IP Address		
Index	**Attribute**	**Description**
0	rt_w/o_ACK	Ratio of flows that do not have the ACK bit set to all flows
1	rt_under_3	Ratio of flows with fewer than 3 flows to all flows
2	max_ips_1sub	Maximum number of IP addresses contacted in any one /24 subnet
3	max_high	Maximum number of high destination ports contacted on any one host
4	max_low	Maximum number of low destination ports contacted on any one host
5	max_cnsc_high	Maximum number of consecutive high destination ports contacted on any one host
6	max_cnsc_low	Maximum number of consecutive low destination ports contacted on any one host
7	num_uniq_dsts	Number of unique destination IP addresses contacted
8	num_uniq_srcp	Number of unique source ports
9	avg_srcp/dest	Average number of source ports per destination IP address
10	rt_std_flags	Ratio of flows with “standard” flag combinations (SYN and ACK set, along with either the FIN or RST bit set) to all flows
11	rt_over_60	Ratio of the number of flows with the average bytes/packet > 60 to all flows
12	med_pack/dst	Median value of packets per destination IP address
13	rt_std_pttrn	Ratio of flows with “standard” combination (standard flag combination and at least three packets and at least 60 bytes/packet on average) to all flows
14	rt_bksctr_pttrn	Ratio of flows with backscatter combination (RST, RST-ACK, or SYN-ACK for the flag combination and the average number of bytes/packet is <= 60 and the number of packets per flow is <= 2) to all flows
15	rt_dst	Ratio of unique destination IP addresses to the number of flows
16	rt_srcp	Ratio of unique source ports to the number of flows
17	rt_bksctr_flags	Ratio of flows with backscatter flag combinations (R/RA/SA) to all flows
18	min_pack	Minimum number of packets of any one flow

19	max_pack	Maximum number of packets of any one flow
20	mean_pack	Mean packets per flow
21	std_dev_pack	Standard deviation of packets per flow
22	min_dur	Minimum duration of any one flow
23	max_dur	Maximum duration of any one flow
24	mean_dur	Mean duration per flow
25	std_dev_dur	Standard deviation of duration per flow
26	min_bytes	Minimum number of bytes of any one flow
27	max_bytes	Maximum number of bytes of any one flow
28	mean_bytes	Mean bytes per flow
29	std_dev_bytes	Standard deviation of bytes per flow
30	min_bpp	Minimum number of bytes per packet of any one flow
31	max_bpp	Maximum number of bytes per packet of any one flow
32	mean_bpp	Mean bytes per packet per flow
33	std_dev_bpp	Standard deviation of bytes per packet per flow

Acknowledgements. We would like to thank Benjamin Klimkowski and Bertrand Sobesto for their assistance on this project. This material is based upon work supported by the National Science Foundation under Grant No. 1223634.

References

1. Dua, S., Xian, D.: Data Mining and Machine Learning in Cybersecurity. Auerbach, Boca Raton (2011)
2. Sommer, R., Paxson, V.: Outside the closed world: on using machine learning for network intrusion detection. In: IEEE Symposium on Security and Privacy, Oakland (2010)
3. Denning, D.E.: An Intrusion-Detection model. IEEE Transactions on Software Engineering (1987)
4. Lane, T.D.: Machine Learning Techniques for Computer Security Domain of Anomaly Detection. Purdue University, Department of Electrical and Computer Engineering and the COAST Laboratory (1998)
5. Axelsson, S.: The Base-Rate Fallacy and the Difficulty of Intrusion Detection. ACM Transactions on Information and System Security (TISSEC), 2008
6. Lippmann, R.P., et al.: Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In: DARPA Information Survivability Conference and Exposition (2000)
7. Simon, G.J., et al.: Scan detection: a data mining approach. In: Proceedings of the Sixth SIAM International Conference on Data Mining, SIAM (2006)
8. Gates, C., et al.: Scan detection on very large networks using logistic regression modeling. In: Proceedings of the IEEE Symposium on Computers and Communications (2006)
9. Ertöz, L., et al.: Scan Detection - Revisited. Technical Report AHPCRC 127, University of Minnesota – Twin Cities (2004)
10. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Surveying Port Scans and Their Detection Methodologies. The Computer Journal (2011)

11. Yegneswaran, V., Barford, P., Ullrich, J.: Internet intrusions: global characteristics and prevalence. In: SIGMETRICS Performance Evaluation. ACM, New York (2003)
12. Symons, C.T., Beaver, J.M.: Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training. In: Proceedings of the 5th ACM Workshop on Artificial Intelligence and Security (2012)
13. Davis, J., Goadrich, M.: The relationship between precision-recall and ROC curves. In: Proceedings of the 23rd ACM International Conference on Machine Learning (2006)
14. Nguyen, T.T.T., Armitage, G.: A Survey of Techniques for Internet Traffic Classification using Machine Learning. Communications Surveys & Tutorials (2008). IEEE
15. Killourhy, K.S., Maxion, R.: Comparing anomaly-detection algorithms for keystroke dynamics. In: International Conference on Dependable Systems & Networks (2009)
16. Lee, W., Stolfo, S.J., Chan, P.K.: Learning patterns from unix process execution traces for intrusion detection. In: AAAI Workshop on AI Approaches to Fraud Detection and Risk Management (1997)
17. Jung, J., et al.: Fast portscan detection using sequential hypothesis testing. In: IEEE Symposium on Security and Privacy (2004)
18. Cisco: Introduction to Cisco IOS Network Flow, March 2015. www.cisco.com
19. Williams, N., Zander, S., Armitage, G.: A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. SIGCOMM Computer Communication Review **36**(5), 5–16 (2006)
20. Dash, M., Liu, H., Motoda, H.: Consistency based feature selection. In: PADKK 2000 Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications (2000)
21. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufman, Burlington (2005)
22. Tavallaee, M., et al.: A detailed analysis for the KDD CUP 99 data set. In: IEEE Symposium on Computational Intelligence for Security and Defense Applications (2009)
23. Song, J., et al.: Statistical analysis of honeypot data and building of kyoto 2006+ dataset for NIDS evaluation. In: Proceeding of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (2011)
24. McHugh, J.: Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratories. ACM Transactions on Information and System Security, November 2000
25. The University at Waikato: Weka 3: Data Mining Software in Java, March 2015. http://www.cs.waikato.ac.nz/ml/
26. Dietterich, T.: On Overfitting and Undercomputing in Machine Learning. ACM Computing Surveys (1995)
27. Nmap, March 2015. http://nmap.org/
28. Cisco. Snort, March 2015. https://www.snort.org/
29. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: International Joint Conferences on Artificial Intelligence (1993)
30. Jain, A., Zongker, D.: Feature Selection: Evaluation, Application, and Small Sample Performance. IEEE Transactions on Pattern Analysis and Machine Intelligence (1997)
31. Jain, A.K., Chandrasekaran, B.: Dimensionality and sample size considerations in pattern recognition practice. In: Handbook of Statistics (1982)
32. University of Waikato: Performing Attribute Selection. https://weka.wikispaces.com/Performing+attribute+selection
33. Mahoney, M.V., Chan, P.K.: An analysis of the 1999 DARPA/lincoln laboratory evaluation data for network anomaly detection. In: Proceedings of the 6th Intl. Symposium on Recent Advances in Intrusion Detection (2003)