Extending Signature-based Intrusion Detection Systems With Bayesian Abductive Reasoning

Ashwinkumar Ganesan
Dept. of Computer Science & Electrical Engineering, UMBC, Maryland, USA
gashwin1@umbc.edu

Pooja Parameshwarappa
Dept. Of Information Systems, UMBC, Maryland, USA
poojap1@umbc.edu

Akshay Peshave
Dept. of Computer Science & Electrical Engineering, UMBC, Maryland, USA
peshave1@umbc.edu

Zhiyuan Chen
Dept. Of Information Systems, UMBC, Maryland, USA
zhchen@umbc.edu

Tim Oates
Dept. of Computer Science & Electrical Engineering, UMBC, Maryland, USA
oates@cs.umbc.edu

ABSTRACT
Evolving cybersecurity threats are a persistent challenge for system administrators and security experts as new malwares are continually released. Attackers may look for vulnerabilities in commercial products or execute sophisticated reconnaissance campaigns to understand a target’s network and gather information on security products like firewalls and intrusion detection / prevention systems (network or host based). Many new attacks tend to be modifications of existing ones. In such a scenario, rule-based systems fail to detect the attack, even though there are minor differences in conditions / attributes between rules to identify the new and existing attack. To detect these differences the IDS must be able to isolate the subset of conditions that are true and predict the likely conditions (different from the original) that must be observed. In this paper, we propose a probabilistic abductive reasoning approach that augments an existing rule-based IDS (snort [29]) to detect these evolved attacks by (a) Predicting rule conditions that are likely to occur (based on existing rules) and (b) able to generate new snort rules when provided with seed rule (i.e. a starting rule) to reduce the burden on experts to constantly update them. We demonstrate the effectiveness of the approach by generating new rules from the snort 2012 rules set and testing it on the MACCDC 2012 dataset [6].

CCS CONCEPTS
• Probabilistic representations → Bayesian Networks;

KEYWORDS
Abductive Reasoning; Bayesian Networks; Intrusion Detection System; Cybersecurity

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DYNAMICS 18, December 2018, San Juan, Porto Rico, USA
© 2018 Association for Computing Machinery.

1 INTRODUCTION
The estimated loss to companies and organizations affected by cyber-crimes is increasing [22], with targets being attacked through social media platforms such as Twitter and Facebook. Cybersecurity threats are constantly evolving as adversaries design new ways to defeat existing systems. These threats are of two main types: ones that use components of known threats and integrate them to create a ‘new’ attack and zero-day1 attacks where the attacker discovers a new vulnerability in the product / system that can be exploited before it can be patched up. Although detecting zero-day attacks is the ideal expectation, in reality identifying attacks that are slight modifications of existing attacks can be difficult too. Thus, Intrusion Detection Systems (IDS) must be regularly updated with the latest attacks even though attack patterns differ in only small ways. Consider an example of the WannaCry ransomware attack2. This malware targeted machines that operated on an older version of Microsoft Windows using a known exploit called EternalBlue3. An analysis of WannaCry revealed it to be similar to previous attacks [15]. The same is true with another well known ransomware ExPetr4 and a modified version Bad Rabbit5.

This phenomenon is clearly visible when we look at snort rules that contain signature patterns for various cybersecurity threats. Table 1 shows an example set of snort rules that are similar with their corresponding CVE IDs. Snort rule MS06-040 [10] tries to alert administrators to a buffer overflow attack on the Microsoft server service while MS08-067 [11] checks for an overflow attack triggered by a specific RPC request. Both rules target the same service but have minor variations to accommodate the different methods used to trigger the buffer overflow attack.

Intrusion Detection Systems (IDS) are of three types [3]:

1) Signature based systems where the attack patterns [29] are defined. These systems cannot detect zero-day attacks.

2https://en.wikipedia.org/wiki/Zero-day_computing
3https://en.wikipedia.org/wiki/WannaCry_ransomware_track
4https://en.wikipedia.org/wiki/EternalBlue
5https://securelist.com/schroedingers-petya/78870/
6https://securelist.com/bad-rabbit-ransomware/82851/

While signature-based systems require constant rule updates, a
\textit{ip-address} \textit{target}

The relevant datasets that are openly available (KDD1999 [8], MAC-CDC 2012 [6]) have a higher degree of malicious traffic as compared to a live stream where a disproportionately large portion of the traffic is benign. As described above, rule-based systems are unable to counter threats that deviate from pre-defined signatures. We trained from pre-existing snort rules. It performs two tasks, namely, detecting if anomalous (not necessarily malicious) execution happens. The False Alarm Rate (FAR) can be a challenge with anomaly detection mechanisms.

The pre-existing snort rules are used to learn the correlation between antecedents / attributes in a rule. Once, the correlations are learned, the model abduces antecedents that are likely occur given a seed (initialization) rule. The antecedents from the seed are used to generate new rules so as to expand the coverage of attacks detected by existing rules.

This paper is divided into the following sections. Section 2 provides an overview of various machine learning, rule based and hybrid methods for intrusion detection. Also, we discuss abductive reasoning methods. Section 3 describes the Bayesian model and pipeline used to generate snort rules. In section 4, we discuss experiments conducted with snort rules dataset and with the MACCDC 2012 dataset. Section 5 describes future directions for our work.

\section{BACKGROUND & RELATED WORK}

\subsection{Machine Learning for Cyberscience}

Over the years, a number of techniques have employed machine learning for intrusion detection. Amor et al. [1] train a Naive Bayes network to classify attacks and show that it has competitive performance. They compare it against a C4.5 decision tree. Valdes et al. [33] construct a Bayesian network (eBayes TCP) for the same purpose. The use of artificial neural networks was explored by Cannady et al. [4] who trained a neural network to perform multi-category misuse classification. A similar approach was taken by Mukkamala et al. [23] who compared against a support vector machine. Luo et al. [19] learned fuzzy association rules to construct generalized patterns to improve intrusion detection. Genetic algorithms (GA) have been utilized as well [40]. In inductive learning, rules are induced directly from the training data. One such common inductive learning process is Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [5].
Figure 1: Pipeline to build the Bayesian Abductive Reasoning model. It consists of multiple stages (a) Preprocessing stage where the rule is encoded by categorizing the attributes and then converting them to a one-hot encoded representation (b) Building the Bayesian model (c) Rule generation and attribute prediction. A (in red) is a sample snort alert. B (in green) is the generated snort rule. X represents the missing antecedent in the snort rule A.

Lee et al. [18] construct a two stage process where different algorithms (like frequent episodes) extract features that RIPPER uses to generate rules. Among more recent methods, Niyaz et al. [16] use a self-taught learning algorithm that combines a sparse autoencoder (unsupervised pre-training) and a multi-layer perceptron (supervised fine tuning) to classify normal and malicious traffic in the KDD99 dataset. Fiore et al. [13] use a Discriminative Restricted Boltzmann machine to train a semi-supervised classifier that can detect anomalies in network traffic. The neural network is trained on "normal" traffic and learns a criteria for normality. Any deviation from normal behavior is flagged as an anomaly. Wang et al. [38] similarly try to identify traffic but use a stacked auto-encoder instead while Erfani et al. [12] implement a deep belief network (DBN) and fine-tune the model to detect anomalies by using a linear SVM. Ma et al. [20] combine spectral clustering with a deep neural network to detect attacks. The network is trained in two stages. First, the training data is divided into subset using spectral clustering. The test data is then assigned the pseudo cluster labels depending on the distance of the test datapoint from each cluster. Then, the neural network is trained on the combined set of pseudo labels. Yu et al. [41] improve on the performance of these network models with a stacked dilated convolutional auto-encoder. Wang et al. [37] utilize a convolutional neural network to classify malware. They subsequently use a convolutional LSTM architecture to learn spatio-temporal features [36]. Buczak et al. [3] and Xin et al. [39] provide an overview of machine learning and deep learning methods used in cybersecurity.

But training effective machine learning methods is a challenge. Data-driven models require the network traffic on which they are trained to represent the likely distribution of the traffic when they are employed. This forces security analysts to re-train the model on a regular (sometimes daily) basis [3]. Also, there is a lack of good quality labeled data that contains normal and malicious traffic, even though the volume of data available is high. As new attacks are discovered, annotating the data is a continuous and expensive process. Vu et al. [35] try to solve this problem by generating synthetic network traffic using a Auxiliary Generative Adversarial Network. The additional data can be utilized to better train a classifier. In this paper though, we look at methods to enhance existing rule-based systems, benefiting from the rules already created by security analysts.

Although patterns are manually crafted in signature-based IDS, their performance can be improved by automatically generating rules. Gomez et al. [14] use a pareto-based multi-objective evolutionary algorithm to evolve snort rules. Vollmer et al. [34] try to reduce the effort of creating rules when an intrusion is detected by automating the rule creation process. In our research, a Bayesian network is trained on snort rules rather than an evolutionary genetic algorithm.

2.2 Abductive Reasoning

As described in the introduction, abductive reasoning is the process of hypothesizing a cause given observed effects. Broadly, abductive reasoning methods can be classified into two types: logic-based [32] and probabilistic [17] methods. Kate et al. [17] build a probabilistic abductive reasoning algorithm with a markov logic network. Raghavan et al. [28] designed a Bayesian abductive logic program framework to perform tasks such as plan recognition where the set of observable facts are inadequate to reason deductively. Logic Tensor Networks [31] proposed by Serafini et al. create a single framework to represent first-order predicate logic so as to deductively reason over a knowledge base.

3 PROBLEM DEFINITION

3.1 Preliminaries

Consider a set of $n$ rules $R = \{R^1, ..., R^n\}$. Let $A = \{A_1, ..., A_l\}$ represents the complete set of $l$ antecedents. Each rule $R^i$ has a set
of $m$ antecedents given by $R^i = \{a^i_1 \ldots a^i_m\}$ where $1 < m < l$ ($a^i_1$ is a specific value while $A^i_1$ is the category / feature / variable). The rules can have varying numbers of antecedents. In the following sections, antecedents and attributes are used interchangeably to define the attribute in a snort rule. Snort rules are defined as follows:

$$R^i := \{a^i_1 \ldots a^i_m | a^i_1 \in A^i_1 \ldots a^i_m \in A^i_m\}$$

is equivalent to:

$$a^i_1 \land \ldots \land a^i_m \Rightarrow R^i$$

In snort, each rule is considered to be a conjunction of attributes (defined in the alert) / antecedents. Also, snort rules do not use consequent ($R^i$) variables as antecedents (i.e. on LHS of the rule). Each component of the rule is associated with an antecedent that is assumed to be a categorical variable with a finite set of possible values. For example, the feature network protocol has a finite set of values: tcp, udp and other protocols. As the number of antecedents in a rule may vary, the variable also has an UNK token for rules where the antecedent is not present. Each rule represents a particular attack / threat. When $R^i$ is provided as a seed (initialization) rule to the model, it is represented as $O^i$.

<table>
<thead>
<tr>
<th>Antecedent Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>tcp</td>
</tr>
<tr>
<td>source IP</td>
<td>$$EXTERNAL_NET</td>
</tr>
<tr>
<td>source port</td>
<td>any</td>
</tr>
<tr>
<td>target IP</td>
<td>$$HOME_NET</td>
</tr>
<tr>
<td>target port</td>
<td>$[139,445], [135,139,445,593,1024:]$</td>
</tr>
<tr>
<td>flow</td>
<td>established, to_server</td>
</tr>
<tr>
<td>dce_iface</td>
<td>12345678-1234-abcdef-ef00-0123456789ab</td>
</tr>
<tr>
<td>metadata</td>
<td>policy balanced-ips, drop, policy security-ips, drop, service netbios-ssn</td>
</tr>
<tr>
<td>dce_opnum</td>
<td>0, 1</td>
</tr>
<tr>
<td>byte_test</td>
<td>$4,&gt;,256,8,relative,dce, 4,&gt;,512,8,relative,dce$</td>
</tr>
</tbody>
</table>

Table 2: A sample set of antecedents that are extracted from two rules. The rules have all same antecedents except the target port number where one rule checks more ports, byte_test and dce_opnum.

To explain the objectives and system architecture, let us consider an example rules set consisting of two rules. Consider the first rule $R^1$ has a SID: 13162 (from table 3). Table 2 has the complete set of antecedents identified from the two rules. Thus, the two rules differ in only the target port numbers, byte_test and dce_opnum. Given all the antecedents and their possible values, the total number of rules possible are 8 (for each combination target port, byte_test and dce_opnum). Let $R^2$ contain the combination target port = $[139,445]$, byte_test = $4,>,256,8,relative,dce$, dce_opnum = 0 while $R^2$ has target port = $[135,139,445,593,1024:]$, and byte_test = $4,>,512,8,relative,dce$ as well as dce_opnum = 1. Let $R^1$ be the seed rule (represented as $O^1$).

3.2 Definition

In this section, we provide the exact definition of abductive reasoning. Generating a hypothesis rule can be categorized into the following tasks:

Abducing Antecedents. In this task, each hypothesis is considered to be an existing rule with a single antecedent $A^p$ being different. Thus, $A^p \neq \{A^i_j | A^i_j \in O^i\}$ for given observation $O^i$ (seed rule).

In case of the example described above, we compute the probability of the hypothesis rule having the target port = $[135,139,445,593,1024:]$ given $R^1$ with byte_test = $4,>,256,8,relative,dce$ (the combination does not exist in the rules set).

Once the probabilities of all the antecedent values that are not part of the seed rule, are computed, the next step is to select an antecedent as a replacement for its corresponding value in the seed. There are three strategies that can be applied, i.e., choosing the antecedent $A^p$ that has maximum-likelihood, selecting the top $k$ most likely antecedents, or defining a threshold likelihood $t$, above which all antecedents are selected.

Abducing Rules. In this task, each hypothesis is a rule that can have multiple antecedents from the seed rule replaced or inserted. Generating hypotheses where likely antecedents are inserted into the seed rule, has a high computational cost. This is because the possible combinations are exponential. Hence rule abduction is constrained to replacing a set of antecedents in the seed rules only.

For the previous example, this will lead to a hypothesis rule having target port = $[135,139,445,593,1024:]$ and dce_opnum = 1 if both antecedents are selected.

3.3 System Architecture

In this section, each component of the pipeline is described. Figure 1 shows the overall system architecture and how a snort rule is processed. We use Scikit-learn [26] for data preprocessing and constructing the Bayesian network.

Rule Preprocessing. In this step, the snort rules are parsed and converted to a categorical variable. Snort consists of a fixed set of position attributes, namely, alert, source IP, source port number, destination IP and destination port number. The remaining antecedents / attributes are in the form of key-value pairs. We use the keys as nodes in a graphical model and the values represent the set of possible states the variable can take. Thus, a vocabulary of possible values is built for each attribute including attributes such as content and pcre. We note that, although, content and pcre are strings that can potentially have infinite permutations, in this paper they are considered to have a finite set of possible values bounded by the rules set $R$. Thus each attribute is treated as a categorical variable. Since we construct a multivariate Bayesian model, the attribute values are then converted to one-hot encoded / binary features. In figure 1, $A$ represents the snort rule (from the rules set $R$), $B$ is the snort rule that is generated and $X$ represents the missing / likely antecedents when a seed rule is provided.

Not all attributes in a snort rule are useful though. A list of excluded attributes is created that contains attributes like sid, rev and reference. The sid is the signature ID of the rule and rev is the revision number for the snort rule. Reference contains external links to information about the attack the snort rule is capturing.
Generating Snort Rules. After building the Bayesian network, the model is able to predict the maximum likely antecedent values or the missing attributes when provided with a seed rule $O^2$. As discussed before, we can choose the antecedent either using MLE or the most likely topk values or based on a threshold set manually. Selections based on the MLE can be highly restrictive. Instead, we select all antecedent predictions that are above a threshold $t$. The antecedent is known to be true only when a snort rule generates an alert. Thus, while inferring the values of antecedents, attributes that are predicted.

Building a Bayesian Model. Once the snort rules are preprocessed, the one-hot encoded attributes are concatenated to form a feature vector to train the Bayesian network. To train the model and infer efficiently, we assume the attributes to be conditionally independent.

$$P(a_j | a_i, ... , a_n) = \arg \max_{a_j \in A_j} \prod_{i=1}^{n} P(a_j | a_i)$$  

where, $a_j$ is the specific antecedent value ($A_j$ is the categorical variable) to be predicted given the other observed antecedents.

By inferring, individual antecedents are not observable. The antecedent is known to be true only when a snort rule generates an alert. Thus, while inferring the values of antecedents, attributes that are not a part of the snort rule are assumed to be UNK. To generalize the model better for UNK (unknown) tokens a Laplace smoothing is applied.

$$P(a_j | a_i) = \frac{F(a_j, a_i) + \alpha}{F(a_j) + \alpha | T|}$$

where, $|T|$ represents all the samples in the training set.

Generating Snort Rules. After building the Bayesian network, the model is able to predict the maximum likely antecedent values or the missing attributes when provided with a seed rule $O^2$. As discussed before, we can choose the antecedent either using MLE or the most likely topk values or based on a threshold set manually. Selections based on the MLE can be highly restrictive. Instead, we select all antecedent predictions that are above a threshold $t$. They are combined with the original antecedents of the seed rule in an unordered list. They then form a graph (represented as adjacency matrix) where each value for an antecedent is linked to a value of the next antecedent in the list. We use a depth first search (DFS) method to generate each possible combinations from the predicted values and eliminate the rules that are copies of the seed. The threshold $t$ has a direct impact on how many rules are generated as it controls values that are predicted.

In the example, let us assume the target port value $[139, 1445, 445, 593, 1024:]$ and dce_opnum value 1 have likelihoods greater than the threshold $t$. The additional antecedents are added to the adjacency matrix formed from the attributes in rule $R^1$. We generate all combinations of rules from this graph and eliminate copies of $R^1$. The final rule generated is shown in table 3 (rule SIG-ID: 250001).

### 3.4 Expanding Features Using Clustering

Although attributes like content and pcre are assumed to be categorical for the purpose of training the model, they can provide insight into rules that are similar based on how close the content/pcre string is to another rule. Thus, clustering can be used to identify similar rules. One of the ways to cluster snort rules is by using hierarchical agglomerative clustering with a customized Levenshtein distance measure computed in the following manner:

$$D(r_i, r_j) = w_1 \cdot KD_{i,j} + w_2 \cdot \sum_{c \in k_i \land c \in k_j} lev_c(r_i^c, r_j^c)$$

As seen in equation 5, the metric is a weighted distance where, $r_i$ and $r_j$ represent the snort rules, $KD_{i,j}$, the key distance, is the symmetric difference between the attributes set in $r_i$ and $r_j$, $lev_c(r_i^c, r_j^c)$ is the Levenshtein distance between the value of $r_i^c$ and $r_j^c$ ($c$ is a common attribute between $r_i$ and $r_j$). Weights $w_1$ and $w_2$ are hyperparameters.

### 4 EXPERIMENTAL RESULTS & ANALYSIS

#### 4.1 Dataset

We use two datasets in our experiments. The first is the community edition of snort rules from 2012. The rules are tested on snort version 2.9.21 that was released in 2012. The dataset contains 43792 rules. A wide variety of rules are available from these, five categories are selected, namely, system, web-misc, web-cgi, and netbios (selected categories have the large number of rules). Table 4 shows the different types of rules and the number of rules available for each type. As the abduced antecedents or rules generated are specific to each rule set type, the models and experiments are performed independently. The second dataset is the MACCDDC 2012 dataset [6]. This dataset consists of series of raw pcap files collected from various attack simulations.
4.2 Abducing Antecedents

As described in task 1 (sub-section 3.2), we test whether the model is able to abduce a single missing antecedent. The experiment gives us an idea about the correlations between different attributes that form the rules. The test is conducted in the form of a leave one column out (LOCO) experiment where the column left out is considered missing. Each rule set in Table 4 is randomly split into a training and test set with 90% of data used for training. We perform 10-fold cross validation to test. Also, the rules are clustered with the customized Levenstein distance (refer to subsection 3.4). After agglomerative clustering is executed, a clusterID is assigned to rules that are similar. The clusterID is used as a feature while training the network. The Bayesian model (with and without cluster features) are in green and blue respectively.

The performance of the two networks is compared against a random baseline (red) and max frequency classifier (purple) (i.e. a classifier that predicts the same label with the maximum frequency in the training dataset). Figure 2-6 shows the performance of the different classifiers.

4.3 Analysis & Discussion

As seen in Figures 2-6, the performance of the Bayesian network with and without clusterID feature is better than the max frequency classifier for most attributes. They perform better than the random baseline for all attributes. For attributes such as distance and depth in figure 2, detection_filter and dsize in figure 3, depth in figure 4, the performance of the max frequency classifier is equal or better than Bayesian models. This is because the classifier performs well when the attribute has a skewed set of values. Consider figure 3 that shows the classification performance for Netbios. The Bayesian classifiers have equal accuracy with the maximum frequency classifier for attributes dsize and detection_filter. The following table shows the frequency of individual labels for each of these attributes. As seen in the tables 5 and 6, the majority of the values in detection_filter are UNK tokens (as they are rarely present in snort rules) leading the Bayesian models to perform poorly with respect to other labels. In comparison, the model has a better performance for an attribute like flow (table 7) that has a distribution of unique labels that is less skewed.

### Table 5: Frequency of unique detection_filter attribute values in the dataset of size 540.

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>track by_dst,count 10,seconds 60</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 6: Frequency of unique dsize attribute values in the dataset of size 540. The high frequency of UNK labels leads a higher performance by the maximum frequency classifier.

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNK</td>
<td>538</td>
</tr>
<tr>
<td>&lt;56</td>
<td>1</td>
</tr>
<tr>
<td>&gt;100</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 7: Frequency of unique flow attribute values. The unique values have a distribution where the skew is limited. This leads to of a maximum frequency classifier that performs poorly in comparison.

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>established,to_server</td>
<td>271</td>
</tr>
<tr>
<td>UNK</td>
<td>119</td>
</tr>
<tr>
<td>to_server, established</td>
<td>80</td>
</tr>
<tr>
<td>established,to_client</td>
<td>26</td>
</tr>
<tr>
<td>stateless</td>
<td>24</td>
</tr>
<tr>
<td>to_client,established</td>
<td>11</td>
</tr>
<tr>
<td>established, to_server</td>
<td>4</td>
</tr>
<tr>
<td>to_server</td>
<td>4</td>
</tr>
<tr>
<td>established,to_server,no_stream</td>
<td>1</td>
</tr>
</tbody>
</table>

4.4 Qualitative Analysis of New Rules

To test the quality of the new rules generated (task 2 in sub-section 3.2), we compare the alerts observed using the seed rule and those generated when the new rules are added to the snort configuration. When snort configuration is updated with them, the seed rule is deactivated. This is measure their impact independent of the seed.

To generate the rules a single seed rule is provided to the model. To generate the rules a threshold posterior probability is defined for each attribute and used to retrieve the topk values for each antecedent. We perform our tests with a threshold of 0.01.

To compare the alerts generated by both configurations, the pcap file from the MACCDC 2012 dataset is replayed on snort with each individual setting. We analyze if any of the alerts generated while using the new rules are false alarms by associating the timestamps of these alerts for both configurations. Table 8 shows the timestamps when alerts for seed and new rules are generated. Then, we calculate the number of alerts generated.

Since the original rule is a Netbios rule (SID: 13162), we look at the alerts generated for Netbios only. With the original rules, 330 alerts are generated while with the new rules, 421 alerts are generated.

### Table 8: Number of alerts generated with different configurations.

<table>
<thead>
<tr>
<th>Antecedent</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_client,established</td>
<td>2</td>
</tr>
<tr>
<td>netbios</td>
<td>1</td>
</tr>
<tr>
<td>established,to_server,no_stream</td>
<td>2</td>
</tr>
<tr>
<td>established, to_server</td>
<td>4</td>
</tr>
<tr>
<td>to_server</td>
<td>4</td>
</tr>
</tbody>
</table>

4.5 Impact Of Threshold

To test how the threshold affects the rules generated, consider the same seed rule (SID: 13162) as before. We check the rules generated from the seed for varying threshold conditions. The threshold parameter controls the size of the topk predicted values for each attribute and thus the types of rules that are generated. A low
Figure 2: Classification accuracy for each attribute on the Web-Misc ruleset.

Figure 3: Classification accuracy for each attribute on the Netbios ruleset.

threshold increases the size of the topk list and generates more combinations of rules. On the other hand, with a high threshold, the system may be unable to generate rules at all. To understand the impact of this parameter, we checked the number of rules that are generated for a range of threshold values (as shown in Figure 7).
Figure 4: Classification accuracy for each attribute on the Special Attacks ruleset.

Figure 5: Classification accuracy for each attribute on the Web-CGI ruleset.

5 CONCLUSION & FUTURE WORK

In this paper, we show that a Bayesian model trained on snort rules can be utilized to abduce antecedents and to generate a set of new snort rules that are modifications of an existing rule. By treating the missing antecedents either as incomplete or modified conditions, the model provides snort the ability to predict missing antecedents and generate alerts for rules that are likely to be triggered but whose conditions have not been met yet because a potential attacker has modified the threat vector. Also, we show that snort rules are inherently incomplete and designed for specific attacks whose
rules set, prepare the system better for attacks in the future and also provides a measure of the incompleteness of the rule set.

In the future, we will experiment with different graphical and neural network models such as Markov Logic Networks (MLN) [17] and Logic Tensor Networks (LTN) [31] as a substitute for our Bayesian approach. Today, LTNs are trained for deductive reasoning but they can be extended to perform abductive reasoning. We can expand the current reasoning approach to abduce rules that have antecedents more than the seed rule and experiment against multiple missing values in the observation as compared to a single missing attribute in this paper.

Apart from enhancing the current generation of signature based systems with additional reasoning, the bayesian model provides us with a template for more abstract reasoning. More et. al [21] demonstrate a system that can detect potential attacks by combining information from various "sensors" on the network i.e. IDS, pattern is well established. Abducing new snort rules expands the

Table 8: The first alert is generated from an existing snort rule (SID:2349) while the second alert is from a snort rule (SID: 2500016) derived using the Bayesian model.
network traffic analyzers, system logs and so on taking a more holistic view to detect a potential attack. The system can be extended when the ontology is grounded with knowledge about prior attacks. The Cybersecurity Ontology (UCO) does so by combining cybersecurity concepts from multiple known security ontologies like CVE [7] and STIX [2]. In our view, the bayesian model can be extended to reason with set of rules designed to operate on such ontologies. Cognitive Cybersecurity System (CCS) [24] is an example of such a system.

The experiments in this paper make an implicit assumption that given the increase in the number of alerts and the timestamp of when the alerts were generated, the rules are detecting potentially new attacks. Our future experiments will analyze the false alarm rates (FAR) for abduced rules.

6 ACKNOWLEDGMENTS

This research is being conducted in the UMBC Accelerated Cognitive Computing Lab (ACCL) that is supported in part by a gift from IBM Research. We thank the other members of the ACCL Lab for their input, suggestions and guidance in developing this system.

REFERENCES