Aiding Intrusion Analysis using Machine Learning

Loai Zomlot, Sathya Chandran Sundaramurthy, Doina Caragea, and Xinming Ou
Kansas State University
Manhattan, KS, USA
{lzomlot, sathya, dcaragea, xou}@ksu.edu

Abstract—**Intrusion analysis, *i.e.*, the process of combing through IDS alerts and audit logs to identify real successful and attempted attacks, remains a difficult problem in practical network security defense. The major contributing cause to this problem is the high false-positive rate in the sensors used by IDS systems to detect malicious activities. The goal of our work is to examine whether a machine-learned classifier can help a human analyst filter out non-interesting scenarios reported by an IDS alert correlator, so that analysts' time can be saved. This research is conducted in the open-source SnIPS intrusion analysis framework. Throughout observing the output of SnIPS running on our departmental network, we found that an analyst would need to perform repetitive tasks in pruning out the false positives in the correlation graphs produced by it. We hypothesized that such repetitive tasks can yield (limited) labeled data that can enable the use of a machine learning-based approach to prune SnIPS' output based on the human analysts' feedback, much similar to spam filters that can learn from users' past judgment to prune emails. Our goal is to classify the correlation graphs produced from SnIPS into "interesting" and "non-interesting", where "interesting" means that a human analyst would want to conduct further analysis on the events. We spent significant amount of time manually labeling SnIPS' output correlations based on this criterion, and built prediction models using both supervised and semi-supervised learning approaches. Our experiments revealed a number of interesting observations that give insights into the pitfalls and challenges of applying machine learning in intrusion analysis. The experimentation results also indicate that semi-supervised learning is a promising approach towards practical machine learning-based tools that can aid human analysts, when a *limited* amount of labeled data is available.**

I. Introduction

Intrusion analysis is the process of examining real-time events such as IDS alerts and audit logs to identify and confirm successful attacks into computer systems. The IDS sensors that we have to rely on for this purpose often suffer from high false-positive rate. For example, we run the well-known open-source IDS system Snort on our departmental network containing just a couple hundred machines and Snort produces tens of thousands of alerts every day, most of which happen to be false alarms. The reason for this is well-known: to prevent false negatives, *i.e.* detection misses from overly specific attack signatures, the signatures that are loaded in the IDS are often as general as possible, so that an activity with even a remote possibility of indicating an attack will trigger an alert. It then becomes the responsibility of a human analyst monitoring the IDS system to distinguish the true alarms from the enormous number of false ones. How to deal with the overwhelming prevalence of false positives is the primary challenge in making IDS sensors useful, given that the amount of attack-relevant data is minuscule compared to the titanic volume of data produced from an enterprise network. The dilemma created by this base-rate fallacy, first pointed out by Axelsson [2], has made it virtually impossible to accurately detect intrusion by a single sensor. Due to the lack of effective techniques to handle the false-positive problem, it is common among practitioners to altogether disable IDS signatures that tend to trigger large number of false positives. In our own campus network, the security analysts did not use the standard Snort signatures at all, but rather resorted to secret attack signatures that are highly specific to their experience and environment, and have small false-positive rates. However, as we were told by the security analysts, the secret signatures can only help capture some "low-hanging fruits" and many attacks are likely missed due to the disabled more generic signatures.

Alert correlation, *i.e.*, the reconstruction of high-level incidents from low-level events, has been used to remediate the false-positive problem. By looking at multiple observation points and correlating alerts, one can potentially reduce the false-positive rate and increase the confidence in intrusion analysis. Indeed, Axelsson's reasoning implies that it is impossible to achieve useful intrusion detection based on a single event such as a network packet, since the features existing in the event are too limited to provide the differentiating power needed to extract the extremely weak attack signal from the background traffic. Thus it is a reasonable assumption that to make IDS sensors useful, it is necessary to correlate events from multiple sensors as a way to increase the features available to make decisions, and there has been a long line of work in creating IDS alert correlation graphs [8], [10], [11], [19], [20], [21], [22], [26], [30], [31], [34], [35], [36]. In this work we use SnIPS [22], an open-source intrusion analysis framework which provides an alert correlation module. SnIPS also provides a prioritizing module based on an extended version of Dempster-Shafer (DS) theory [37], which ranks the correlations by calculating a *belief* value for each hypothesis that occurs in the correlation. Correlations with higher-belief hypotheses are ranked higher and presented to human analysts first.

When a correlation graph is presented to a security analyst, the analyst can "browse" the graph to explore its structure as well as the details of the supporting evidence. If the correlation proves significantly interesting, the analyst will conduct further forensic analysis on data that is outside SnIPS, to confirm or rule out the scenario. This process is manual and even with the help of the belief values, the human analyst still needs to look at the evidential details of the correlation to determine if it is worth further investigation, as the belief values do not always exactly match the priority determined by the human. Given this, the question is whether anything can be done to further automate the prioritization process, so that the human analysts' time can be saved. Throughout our experimentation with SnIPS on our departmental network, we found that the user needs to do many repetitive tasks in the analysis of the tool's output to determine whether further investigation is warranted. *We hypothesize that the rationale behind the repetitive tasks can be inferred through a machine learning approach, so that the prioritization process can be further automated.* We adopt machine learning as a candidate technique to further help prioritizing intrusion analysis since it seems that a human, after examining the correlation output, can make a decision on whether to further investigate the incident or not. Thus, there is basis to believe that the SnIPS' output can yield a set of predictive features indicating whether a correlation is interesting or not. Furthermore, the fact that a human analyst would have to look at a correlation graph and determine whether to escalate investigation implies that there would be labeled samples (albeit small in amount) available when the tool is put to use in operation. This would be similar to the approach taken in spam filtering, where machine learning has proved to be quite successful [14], [15].

While it is possible that this prioritization process could also be automated through other means such as a rule-based system, we think it is more cost effective if the machine can learn the rules automatically. In the long run, the machine-learned models could provide insights into how to build a system to do the same job without the need for learning.

Our Contributions

1) We apply machine learning to automate the process of intrusion analysis, as opposed to most existing methods that make use of machine learning in the process of creating alarms, such as in anomaly-based detection.
2) Our method minimizes the time and effort of training the model in the deployment stage by using the human analysts' effort in investigating the correlations' validity to produce labeled data. Furthermore, using semi-supervised learning enables the model to be trained starting from as low as 10% of the required dataset size for supervised learning.

The rest of the paper is organized as follows: section II covers the background about SnIPS. Section III presents our research hypotheses followed by section IV which lays out our approach in applying machine learning in intrusion analysis using SnIPS. Section V presents our experimental results and discusses them. Section VI covers the related work and we conclude by section VII.

II. Alert Correlator

We use the open-source SnIPS IDS alert correlator [1] in our research. It works on top of IDS sensors and audit logs to further analyze the reported events to identify possible incident scenarios. SnIPS maps a triggered IDS alert to a hypothesis such as "compromised machine." It also maps the trustworthiness of the hypothesis to a discrete certainty tag such as "possible", "likely", or "certain." After the alerts are mapped to hypotheses, the hypotheses are reasoned based on an internal reasoning model. As a result, an alert correlation graph is built showing possible links among the hypotheses and alerts. SnIPS has a prioritization module [37] that can further refine the results of correlation by assigning each hypothesis in the correlation graph a belief value based on an extended version of Dempster-Shafer theory. SnIPS ranks the correlation graphs by the belief values of the "conclusion nodes" within the graphs. The goal is to further help security analysts by showing the graphs with higher belief values first.

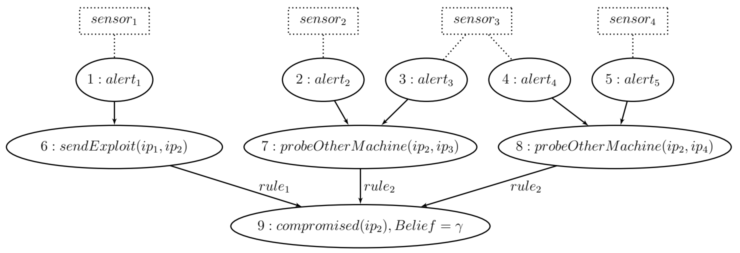
$sensor_1$
$sensor_2$
$sensor_3$
$sensor_4$
$1 : alert_1$
$2 : alert_2$
$3 : alert_3$
$4 : alert_4$
$5 : alert_5$
$6 : sendExploit(ip_1, ip_2)$
$7 : probeOtherMachine(ip_2, ip_3)$
$8 : probeOtherMachine(ip_2, ip_4)$
$rule_1$
$rule_2$
$rule_2$
$9 : compromised(ip_2), Belief = \gamma$

Fig. 1. Automatically generated correlation graph from SnIPS

Figure 1 shows a sample single-sink alert correlation graph automatically generated by SnIPS. The correlation graph is a logical inference graph. SnIPS uses predicates such as "*compromised*", "*sendExploit*", and "*probeOtherMachine*" to describe various attack hypotheses. Five groups of alerts $alert_1 - alert_5$, are triggered by four different sensors. A sensor could be one IDS signature (*e.g.*, a Snort rule), or a group of IDS signatures that capture similar patterns. The sensor nodes (the ones in dotted squares) are not part of the graph and are added here for clarity. In this example, $alert_1$ is mapped to the fact that host ip_1 sent an exploit to ip_2; both $alert_2$ and $alert_3$ are mapped to the fact that ip_2 did malicious probing to ip_3, and so on. The rationale for this correlation graph is that after ip_1 sent an exploit to ip_2 (node 6), ip_2 could be compromised (node 9). Once the attacker compromised ip_2, he could send malicious probing from there (nodes 7 & 8). Thus, these alerts are all potentially correlated in the same underlying attack sequence. For representational simplicity, time information is not shown in the example graph (but is part of the reasoning process). In this example, $alert_2 - alert_5$ happened after $alert_1$. The arrow of the arcs indicate that all of $alert_1 - alert_5$ support the hypothesis that ip_2 was compromised. The belief value (γ) of the sink (node 9) in the graph is calculated using the prioritization module.

III. Research Hypotheses

Through our empirical study of the output of the deployed SnIPS on our departmental network, we found that many correlations are not interesting since they are supported by very noisy IDS signatures and the correlation structure does not provide significant boost on the belief in attacks. When the correlation does seem to be interesting, often times one would need to consult additional information that is currently not captured by SnIPS. It seems that to use a tool like SnIPS the first task for the security analyst is to determine whether the output correlation is worthy of further investigation. While it would be ideal if the calculated belief value can be used to determine whether further investigation is warranted, our practical experience shows that this is not always the case. Thus, some manual work is still needed to filter out the more interesting output from SnIPS for further analysis.

In this work we would like to examine *the possibility to automate the manual process of determining whether an IDS alert correlation graph is interesting and worthy of further investigation through a machine-learning approach*. This hypothesis is justified for the following reasons.

1) The fact that a human analyst can make a determination on the usefulness of an IDS alert correlation graph indicates that the features within the graph are predictive to the classification of "interesting" and "non-interesting", where "interesting" means that the graph is worthy of further investigation.
2) The fact that eventually a human analyst will need to look at the alert correlation graph means that, in the end, it will be up to the human analyst to make the final decision on the usefulness of the graph. This process creates (limited amount of) labeled data to train a machine-learned classifier. Thus, a machine learning approach could be feasible for operational use, where a human analyst looks at the output and provides feedback on the classifier's precision in the form of fresh labeled data, on a continuous basis.

Since the labeled data provided in the use of the alert correlation tool will be very limited, we also hypothesize that *semi-supervised learning can help to address the problem of limited labeled data*. In the following sections we elaborate on the experiments through which we examine these hypotheses.

IV. Methodology

Our intention is to use features existing in SnIPS' output to build a prediction model to classify whether the output correlation is "interesting" or "non-interesting." Each SnIPS' output is a correlation graph, with one or more sink nodes. The sink nodes in a correlation graph represent the conclusion that the correlation draws. The classification task is to determine whether the correlation is worthy of investigation based on the features included in the correlation graph, including the graph structure, the details of the supporting evidence, and the belief value of the graph (more on this later).

A. Classes

We want to classify the correlation graphs from SnIPS into the following two classes:

1) *Interesting:* This class means that the correlation is found to be highly suspicious and worthy of more in-depth investigation. This class covers a wider area than "true positive," since the conclusion drawn by the correlation may turn out to be a false positive after further investigation.
2) *Non-interesting:* This class means that the correlation is found to be non-interesting and not worthy of further investigation. Ideally under this situation we want the conclusion drawn by the correlation to be a false positive as well. However, in reality there is always a chance that an attack is true but not sufficient evidence is captured in the correlation so it is mistakenly dismissed as "non-interesting." How to improve the true positive of intrusion analysis is an orthogonal problem of this research. The "non-interesting" class simply means that there is not sufficient evidence to warrant further investigation of the correlation.

B. Dataset Construction

We use data from our departmental network for both training and testing. This is consistent with the envisioned use of the prediction model, where each organization will train its own prediction model based on security analysts' feedback in using the correlation tool. It is known that using multiple datasets could produce stronger evidence on a prediction model's effectiveness. However, the particular problem we try to address in this research requires access to production networks. While we have been fortunate to have supportive local system administrators and were able to conduct this experiment on one production network (our departmental network), practical constraints have made obtaining access to multiple production networks infeasible for us. Indeed it is rare for academic researchers to be able to get access to production network data at all. We believe even though our experimentation was conducted on a single network, the results we obtained still provide useful insights into the effectiveness of this approach in practice, and how to apply machine learning for intrusion analysis in general.

1) Network Setup and Labeling Tool: The departmental network consists of 35 Linux and 11 Windows servers. There are over 150 workstations including Sun, Mac Pro, and PC running Windows and Linux. The departmental network consists of three VLANs: main, printer, and thin clients. The main VLAN contains all the servers and user machines and is also the entry point to the departmental network. It is a giga-bit switched network with a giga-bit uplink to the campus network. We attached an optic fiber cable to the Cisco switch of the main VLAN to mirror all traffic on it. This includes ingress and egress traffic for the departmental network, as well as internal traffic for the main VLAN. We then run Snort IDS system on the captured traffic, which produces tens of thousands of alerts per day. Both Snort and SnIPS run on a dedicated Ubuntu server running a Linux kernel version 2.6.32 with $16GB$ of RAM on an eight-core Intel Xeon processor of CPU speed 3.16GHz.

To facilitate the labeling process, we implemented a web-based interface that allows a security analyst to interact with SnIPS' output and make a determination on whether it is interesting or non-interesting. The feedback is recorded in a back-end database along with the features extracted from the correlation graph as well as the supporting evidence.

2) Labeling Process: The labeling process was the most time-consuming part of the research, due to the need of obtaining sufficient amount of labeled data for both training and testing. However, we would like to point out that when the tool is mature enough to be deployed and used in operation, the labeled data will be generated "naturally" by the users (security analysts) when they analyze the correlation graphs and make determination on whether to escalate investigation. As the tool is used over time there will be continuously generated fresh labeled data available to train the prediction model. Thus although it is tremendous amount of work for the researchers, who had to be trained as operation analysts and analyze large amounts of data for research experiments, in a deployed setting such effort would just be part of the routine work of the security analysts and there would be virtually no additional cost for labeling.

The ideal research method would prefer that labeling be done by a real security analyst. However, it is difficult to recruit such professionals for research purposes. Research prototypes are not as easy to use as mature off-the-shelf products and security analysts are typically overwhelmed by a varietyof tasks and have little time to help researchers. Thus we, the researchers, must take on the task of a security analyst to further examine SnIPS' output to understand the events and label them. This is a less-than-ideal situation since usually the people who label the data need to be different from researchers to prevent the appearance of conflicts of interest. However, for this experiment it is hard to recruit people with sufficient knowledge and skills to perform the labeling, as would be possible in other problem domains. To prevent bias, we try to have multiple persons to label the data whenever possible.

As a result of the labeling process we have a dataset that comprises of 1615 data points. We found that this size is adequate to conduct our subsequent experiments, as shown later.

a) Labeling Guidelines: To ensure consistency, we followed these guidelines throughout the labeling process. For each SnIPS' correlation graph, we did the following to determine whether it is "interesting" or "non-interesting".

1) Check the graph structure. This will help understand and validate the scenario that supports the hypothesis (sink node), *e.g.*, node 9 in figure 1.
2) Check the type of the machines involved in the graph - whether they are internal machines (clients or servers) or external machines. For external machines, we use the "whois" service and IP reputation websites, *e.g.*, "Trend Micro site safety center" (http://global.sitesafety.trendmicro.com), to get a sense on whether they are benign or potentially malicious.
3) Check the open ports on the machines involved in the graph. This will help to discover any malicious services on these machines and to verify some Snort alerts.
4) Check Snort alerts' payloads. This is helpful to gain insights into the reason behind each alert, but it only provides a limited view, as Snort only stores the triggering part of the packet for the alert.
5) Check other features like time stamps, graph size, and Snort-signature categories in the correlation.

b) Observations on the Labeling Process: Our experience on manually labeling the data seems to indicate that the information that matters in determining the existence of attack or not tends to be highly specific to the nature of the IDS alerts involved and the contextual information within the network. For a machine-learning system to make high-quality classifications, such factors need to be reflected in the feature selection. However, the challenge is how to encode such fine-grained diverse information in a uniform format so that a machine-learning algorithm can consume. Our current feature selection and construction as explained in section IV-C is rather coarse-grained. This might become a limiting factor on the effectiveness of the machine-learning approach, which was later verified by our experiments. It also seems that some of the knowledge used in pruning false positives could be implemented in a systematic way through tuning the alert triggering conditions. This may significantly reduce the number of false alerts entering the system and may make a higher-level machine-learning system more effective.

3) Class Distribution: The class distribution (interesting vs. non-interesting) is skewed in this type of data, with non-interesting scenarios significantly out-numbering interesting ones. This is consistent with past estimate on IDS false-positive alerts, where it has been estimated that up to 99% of alerts reported by IDSs are not related to security issues [2]. This will make the classification process harder and prone to low accuracy [27]. In our dataset, we found the ratio of positive "interesting" to negative "non-interesting" class to be roughly $1:4$ [1]. This unbalanced data presents a challenge for machine learning, as the resulting classifier could be biased towards the majority class. To ensure that both classes are learned well, we experimented balancing the training data (while maintaining the original distribution in the test data). We have performed experiments with two balancing approaches to eliminate the imbalance bias on the classification process (section V).

C. Feature Selection

The most important task in building a machine-learned classifier is to select the features that are likely to be predictive to the classes of interest. Based on our empirical experience, we divide those features into two categories. The first category consists of information about input to SnIPS. Since currently SnIPS mostly takes Snort alerts as input evidence, we call this category Snort-related features. The second category consists of information about SnIPS' reasoning which manifests as structures of the correlation graphs. We call them SnIPS-related features. The feature set is described below.

1) Snort-related Features:

- `Snort-signature set size.` In each SnIPS' graph, there is a set of IDS alerts triggered by a set of Snort signatures. This feature concerns the size of this set. For example, in figure 1 $sensor_1 - sensor_4$ represent Snort signatures, and the size of the signature set is 4.
- `Snort-signature class groups.` In SnIPS each Snort signature is given a weight. This weight is a measure of the trustworthiness level in the alerts triggered by that signature. Snort signatures are categorized into 24 classes. The *class weight* is the maximum weight among all the weights associated with the Snort signatures in that class.

[1]A curious reader may wonder why this ratio is much higher than Axelsson's estimate [2]. This is because we are classifying based on whether a correlation is worthy of further investigation, as opposed to whether a single event represents a true attack. The latter would have a much lower ratio.

This feature is represented as a k-size vector of pairs, where each pair consists of the number of appearance of a class and its corresponding weight, as follows:$\{\{\#\text{ of appearances of class\_type}_1\text{ , weight}_1\}$ $, \cdots, \{\#\text{ of appearances of class\_type}_n\text{ , weight}_n\}\}$. If the class does not appear in the graph, we use zeros. For example, the graph in figure 1 has $sensor_1, sensor_2$ belong to $class_1$. It also has $sensor_3$ belongs to $class_2$. Thus, this feature will be $\{\{2, 0.33\}$, $\{1, 0.66\}$, $\{0, 0.0\}, \cdots, \{0, 0.0\}\}$. This means that the first pair corresponds to $class_1$, which appears twice and has a weight of 0.33, and so on.

- `Host categories.` We grouped the monitored network IP addresses into three categories: client, server, and external IPs. This feature is a vector consisting of the number of appearances of the IPs in each categories in the *Snort alerts* supporting the correlation. This feature is: $\{\{\#\text{ of appearances of client}\}$, $\{\#\text{ of appearances of server}\}$, $\{\#\text{ of appearances of external}\}\}$. For example, in figure 1 suppose that ip_2 belongs to the server category, ip_3 and ip_4 belong to the client category, and ip_1 belongs to the external category. Then, if we count these IPs from the $alert_1 - alert_5$ (not shown on the graph) the feature will be {4, 5, 1}.

2) SnIPS-Related Features:

- `Belief value of the correlation graph.` The maximum belief value of the sink nodes in a correlation graph is the belief value of the whole graph.
- `Correlation-graph size.` This feature is the number of nodes in the graph. For example, the correlation graph in figure 1 has 9 nodes. Note that the sensor nodes (the ones in dotted squares) are not part of the graph.
- `SnIPS' inference rules set.` This feature is about the participating SnIPS' inference rules that created the correlation graph. This feature is represented as a vector of SnIPS' inference rules appearing in the graph. Currently, SnIPS has three internal rules. Thus, this feature is: $\{\{\#\text{ of appearances of rule}_1\}$ $, \{\#\text{ of appearances of rule}_2\}$ $, \{\#\text{ of appearances of rule}_3\}\}$. If a rule does not appear in the graph, we assign zero for it. For example, our sample graph in figure 1 has this vector $\{1, 2, 0\}$ - $rule_1$ is used to connect nodes (6 to 9), $rule_2$ is used to connect nodes (7 to 9) and nodes (8 to 9).

D. Learning Approaches

The nature of our problem implies that supervised or semi-supervised learning approaches are possible candidates. Supervised approaches often yield better results if enough labeled training data are available. Since the labeling process for our problem domain is time-consuming, it is oftentimes hard to have a large number of labeled samples. This gives rise to the possibility of applying semi-supervised learning techniques to address the problem of scarcity of labeled data. By using semi-supervised learning, the system can start with a small amount of labeled data, train a classifier and then make this classifier labels more data iteratively until the process stabilizes. There are two well known approaches in semi-supervised learning, Co-training [4], and Expectation Maximization (EM) [12] and its variants such as self-training (a.k.a., self-teaching or bootstrapping) [33]. In section V, we illustrate the results using both supervised and semi-supervised approaches.

For the various classification methods, Support Vector Machine (SVM) algorithm [9] has been used widely in the application of machine learning, including in cybersecurity. SVM is a binary classifier in its original formulation. In the linearly separable case, it works by maximizing the separating boundary between the two classes (a.k.a., margin), and selects a number of critical boundary instances, called support vectors, from each class. Then, it builds a linear discriminant function (a.k.a., a hyperplane) that separates the two classes. When the data is not linearly separable, the algorithms implicitly maps the data to a higher dimensional space (through the means of a kernel), where data becomes linearly separable, and it builds a linear discriminant function in that space.

V. Experimentation and Discussion

For the experimentation section we used our dataset that comprises of 1615 data points. We conducted experiments with supervised and semi-supervised learning using SMO (Sequential Minimal Optimization) classifier, which is an implementation of the SVM algorithm from Weka [16]. We used SMO with logistic models option from Weka to produce probability estimates for the output. All the experiments were implemented in Java.

a) Validation Method: We use the n-fold cross-validation [32] method to evaluate the results of our experiments. In n-fold cross-validation, the original sample is randomly partitioned into *n* subsamples: *n-1* subsamples are used for training and the remaining one is used for testing. The procedure iterates *n* times to cover each possible splits as testing exactly once. Empirically, 10 folds have been shown to give the most reliable results [32], and this is what we used in our experiments.

b) Performance Metrics: We use accuracy, area under the ROC curve (ROC AUC), precision, recall, and F-measure to measure the performance. The measures' range is from 0 to 1, and the closer the measure to 1, the better it is. We use the ROC AUC as it has been shown to be a better classification performance measure when compared to overall accuracy [6]. Precision, recall, and F-measure are used to better understand the behavior of the classifier for the two classes (interesting vs. non-interesting). Precision measures the fraction of the classified instances that are relevant to the right class. Recall measures the fraction of correctly classified instances out of the total number of instances in that class. F-measure is the harmonic mean that combines recall and precision measures; thus, it shows the trade-off between precision and recall.

A. Supervised Learning

1) SVM Kernels Experiments: We conducted multiple experiments with multiple SMO kernels, to find the best kernel. Choosing the right kernel function highly depends on the nature of the dataset and in practice the best mapping function is often determined experimentally. This is done by applying various kernel functions and selecting the best kernel and parameters that have the highest generalization performance on a validation dataset. We conducted experiments with normalized polynomial, polynomial, Gaussian Radial Basis Function (RBF) kernel, and Pearson VII Universal Kernel (PUK). PUK [29] is a universal kernel that can be calibrated to work as any of the standard SMO kernels, by appropriately adjusting its two parameters σ and ω. We used different parameter values provided in Üstün *et al.*'s [29] work to get the effect of the standard kernels. We also used Weka's default values for the parameters. The experiments show that using the default values give the best performance metrics (table I). Thus, we used these parameters to conduct the remaining experiments.

a) Cost Parameter Value: To avoid over fitting, SMO has a regularization parameter C that controls the tradeoff between allowing training errors and enforcing rigid margins (a.k.a., bias-variance problem). Thus, this parameter can be seen as a penalty for training errors. Increasing the value of C increases the cost of misclassifying points and forces the creation of a classifier that fits the training dataset well, but may not generalize well on new data. Decreasing the value of C too much may result in a very simple classifier that does a poor classification job, in general. There is no widely accepted standard for selecting the best value for C. One method recommends trying exponentially growing sequences, *e.g.*, $C \in \{2^{-5}, 2^{-3}, \cdots, 2^{13}, 2^{15}\}$. Other method uses $C \in \{0.5, 1, 2, 3, 5, 7, 10, 15, 20\}$ [25]. We tried both and found that the best value for our experiments is around 10.

TABLE I. RESULTS FROM RUNNING SMO'S PUK KERNEL ON UNBALANCED DATASET; WITH σ=1,ω=1,AND C=10

Accuracy	Class*	ROC AUC	Precision	Recall	F-measure
0.927	1	0.956	0.824	0.797	0.810
	0	0.956	0.951	0.958	0.955
	Weighted Avg.	*0.956*	*0.926*	*0.927*	*0.926*

* Class: interesting (positive) is "1" and non-interesting (negative) is "0".

2) Balancing the Data : As mentioned before, the distribution of classes (interesting vs. non-interesting) is skewed. This could result in models skewed toward the negative class (non-interesting), which would affect the prediction ability for the positive case. As table I shows, the performance metrics for the interesting class are usually worse than for the non-interesting class. This is because the classifier does not have enough data to learn how to predict the interesting class.

There are several approaches to address unbalanced data in classification. Over-sampling balances the class populations through over-weighting the minority class instances, for example in our case we had weight "4" for each instance in the positive (interesting) class. Under-sampling balances the class populations through eliminating the majority instances [18].We conducted experiments with over-sampling and under-sampling using SMO PUK default kernel with parameter ($C = 10$). The overall results in the over-sampling are better than under-sampling; however, there is no indication that either over-sampling or under-sampling significantly improves the classifier's performance. This may be due to a number of factors such as the overall small number of labeled data, and the simplicity in over-sampling by duplicating data samples.

B. Semi-Supervised Learning

The idea behind the semi-supervised learning is that the system can start with very small amount of labeled data and grow by retraining itself until it consumes all the unlabeled data. Thus, to validate our hypothesis (section III) we ran experiments with two of the widely used semi-supervised approaches. Co-training [4] and a popular variant of Expectation Maximization (EM) [12].

Table II shows the comparison between the two approaches, where 10% of the training data is used as labeled and the rest of the data is used as unlabeled. For comparison, we also show an upper-bound (the best results that SMO classifier can achieve with a fully labeled dataset, or in other words a supervised learning) and a lower-bound (results for running SMO supervised classification only with the 10% of labeled data from our dataset, as used also in co-training).The results show that co-training seems to be the most promising approach.

The co-training idea is to split the feature set into two subsets (views). Each view should be sufficient to build a classifier (predictive of the class label), and the two views should be conditionally independent given the class. The algorithm starts from a small labeled sample and builds two classifiers from the two views. Then, it consumes some unlabeled instances from the unlabeled data pool, and each classifier labels them. Newly labeled data points are added to the training pool. Instances where the two classifiers disagree are ignored (thus, effectively removing possible noise in the labeling process). Next, the algorithm builds two new classifiers from the bigger labeled data pool, and iterates until the unlabeled data pool is consumed or a maximum number of iterations is reached [4]. We use SVM classifiers to learn from the two views in the co-training approach.

TABLE II. RESULTS FOR SEMI-SUPERVISED WITH SMO'S DEFAULT PUK KERNEL ON UNBALANCED DATA, WITH σ=1,ω=1,AND C=10;$^{+}$ EM: EXPECTATION MAXIMIZATION

Approach	Accuracy	ROC AUC	Precision	Recall	F-measure
Lower bound	0.875	0.900	0.880	0.875	0.877
Co-training	0.900	0.948	0.907	0.900	0.903
EM^{+}	0.893	0.895	0.889	0.893	0.890
Upper bound	0.927	0.956	0.926	0.927	0.926

EM-type learning is an iterative statistical technique for maximum likelihood estimation for small labeled dataset. Given a model, and data with missing class values, EM will locally maximize the likelihood of the parameters and predicts with estimates for the missing class values [12]. The self-training variant is similar with co-training, but uses only one view. At each iteration, instances from the unlabeled pool that are classified with high confidence by the current classifier are added to the training data pool. As opposed to co-training, the use of one view makes it impossible to detect labeling conflicts. Self-training is also used with SVM in our experiments. The results show that this approach can also produce good results but not as good as co-training in our application. We think this

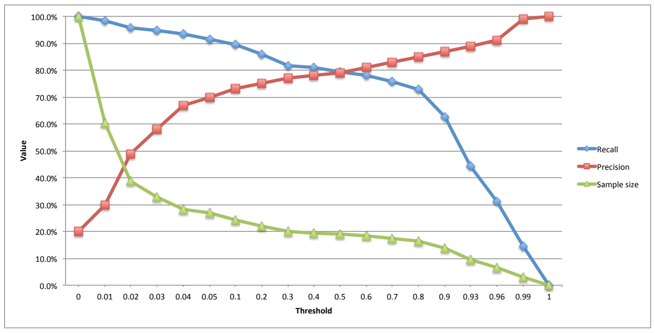

Fig. 2. Effort analysis

is because we have conflicts between some instances (a.k.a., noise), which self-training is not able to detect. The two views used in co-training help in this respect as the two independent "experts" can point to conflicting instances.

C. The Benefit of Machine Learning Approach

We conducted another experiment to see if the machine learning approach is an improvement over SnIPS' prioritization module or not. To do so, we thresholded the belief value of the SnIPS' correlations and calculated the performance metrics. Table III shows the advantage of the machine learning approach that uses *multiple* features including the belief value, over the approach that relies *solely* on the belief value in SnIPS (denoted "Baseline" in the table).

TABLE III. COMPARISON BETWEEN USING JUST THE SNIPS' BELIEF VALUE AS A CLASSIFIER (BASELINE) AND MACHINE LEARNING USING SVM (SUPERVISED) AND CO-TRAINING (SEMI-SUPERVISED) LEARNING; WITH σ=1,ω=1,AND C=10

Approach	Accuracy	ROC AUC	Precision	Recall	F-measure
Baseline	0.803	0.679	0.755	0.803	0.753
Semi-supervised	0.900	0.948	0.907	0.900	0.903
Supervised	0.927	0.956	0.926	0.927	0.926

Furthermore, we conducted another analysis using precision, recall, and dataset sample size to show the benefit of using machine learning in reducing the workload of the security analyst. Figure 2 shows the changes when the threshold (x axis) is increasing. When one starts with small amount of alerts correlations, the precision is high meaning more of the effort is devoted to useful tasks. When the work increases, the cumulative precision decreases as well.

This is a good indication that the classification helps in presenting the interesting (potentially malicious) correlation to the security analyst.

VI. Related Work

IDS alert correlation has been extensively studied in the past ten years [8], [10], [11], [19], [20], [21], [26], [30], [31], [34], [35], [36]. However, just because a correlation exists does not automatically mean the associated alerts are high confidence. The correlation itself are often "false correlations". From our conversation with system administrators, it is highly desirable that alert correlation tools prioritize their output based on the likelihood of true attacks and other correlation related attributes. Our work provides a possible approach to address this need.

Beaver *et al.* [3], [28] propose an interesting approach by using "in-situ" learning and semi-supervised learning to build a model for intrusion detection. Attack traffic is introduced to the network where the detection tool is deployed and this labeled data was used for training the classifier. Their model, as opposed to anomaly detection, learns both the known attacks and the normal traffic. It remains to be seen how effective the approach will be when the system is deployed in production networks. Our application of machine learning has a different objective. Instead of using machine learning to make a decision on whether an event is malicious or not, we use it to prioritize alert-correlation graphs from an up-stream analysis tool, with the goal of saving analysts' time. We also conducted our experiment in a live production network.

Pietraszek [23] has used machine learning to classify IDS alerts into true and false positives. In a later work, Pietraszek and Tanner [24] propose a more complete system, where noisy alerts are eliminated before feeding them into the system. This line of work differs from ours in two ways. First it operates at the IDS alerts level, and it uses RIPPER rule learner. The rule learner approach has an explicit classification logic, which allows a human expert to inspect the classifier and verify its correctness. Instead of a rule learner, we use the SVM algorithm to classify interesting and non-interesting correlation graphs. In our framework, the classifier will adapt according to the new incoming labels and will present the non intuitive conflicting results to the expert for further study. The RIPPER rule learner is applied before the correlation stage, while our approach uses correlation graph related features to build the classifier. Second, in our work the noisy alerts are not removed. This is desirable since even the most noisy alerts can have some link to a true attack.

Bolzoni *et al.* [5] present a system that automatically classifies attacks (*e.g.*, buffer overflow, SQL Injection) detected by an anomaly-based network intrusion detection system. This is done by comparing the extracted byte sequences from an alert payload to previously collected data, *e.g.*, Snort alert classification. Our work's goal is to classify intrusion analysis correlation graphs into "interesting" and "non-interesting", where "interesting" means that a human analyst would want to conduct further analysis on the events.

There has been a long line of work on applying machine learning in anomaly-based intrusion detection [7], [13]. It has been pointed out that significant challenges exist in applying machine learning in this area [27]. Our application of machine learning has a different goal than those past works. Our machine-learned model will help a human analyst to prioritize output from an intrusion analysis system, which relies upon (multiple) IDS systems. Our method is *not* to build an intrusion detector through machine learning. Our application of machine learning is justified due to the nature of the problem.

VII. Conclusion and Future work

In this paper, we presented our results from experimenting with machine learning techniques in intrusion analysis. Our goal is to use machine learning to reduce the workload on the security analyst by automating the pruning of non-interesting alert correlations. We conducted experiments with both supervised and semi-supervised learning approaches using SVM classifiers. Our results indicate that the manual process of determining whether a correlation graph is worthy of further investigation can be automated with acceptable performance through co-training using SVM classifiers. The main lesson learned from this work is that proper labeling is an important step in this research and the insights in labeling helped in feature selection and construction. It is also clear from the work that encoding all relevant features to build a classifier is a challenging task and one must make trade-offs in deciding how fine-grained the features are.

We left the full integration of this system with SnIPS as future work. This will help to further test our hypothesis by collecting more labeled data. To further help the labeling task, we have started the process of setting up a time machine[17] and integrate it with SnIPS' web interface. This will help the security analyst to have full view of the attack scenario.

References

[1] Argus Lab. Snort Intrusion Analysis using Proof Strengthening (SnIPS). http://www.arguslab.org/projects/snips.

[2] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS)*, 1999.

[3] J. Beaver, C. Symons, and R. Gillen. A learning system for discriminating variants of malicious network traffic. In *8th Cyber Security and Information Intelligence Research Workshop*, January 2013.

[4] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th annual conference on Computational Learning Theory (COLT)*, 1998.

[5] D. Bolzoni, S. Etalle, and P. H. Hartel. Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2009.

[6] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30, 1997.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41, 2009.

[8] S. Cheung, U. Lindqvist, and M. W. Fong. Modeling multistep cyber attacks for scenario recognition. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[9] C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, 1995.

[10] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, 2002.

[11] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, pages 85–103. Springer, 2001.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *journal of the royal statistical society, series b*, 39, 1977.

[13] D. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2), 1987.

[14] J. Goodman, D. Heckerman, and R. Rounthwaite. Stopping spam. *Scientific American*, 292(4), 2005.

[15] P. Graham. *Hackers and Painters: Big Ideas from the Computer Age*. O'Reilly, 2004.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1), 2009.

[17] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[18] S. Li, Z. Wang, G. Zhou, and S. Y. M. Lee. Semi-supervised learning for imbalanced sentiment classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence(IJCAI)*, 2011.

[19] P. Ning, Y. Cui, D. Reeves, and D. Xu. Tools and techniques for analyzing intrusion alerts. *ACM Transactions on Information and System Security*, 7(2), 2004.

[20] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer & Communications Security (CCS)*, 2002.

[21] S. Noel, E. Robertson, and S. Jajodia. Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances. In *20th Annual Computer Security Applications Conference (ACSAC)*, 2004.

[22] X. Ou, S. R. Rajagopalan, and S. Sakthivelmurugan. An empirical approach to modeling uncertainty in intrusion analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[23] T. Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection*, 2004.

[24] T. Pietraszek and A. Tanner. Data mining and machine learning—towards reducing false positives in intrusion detection. 2005.

[25] G. Rätsch, S. Sonnenburg, and B. Schölkopf. Rase: recognition of alternatively spliced exons in c.elegans. *Bioinformatics*, 21(1), 2005.

[26] R. Smith, N. Japkowicz, M. Dondo, and P. Mason. Using unsupervised learning for network alert correlation. *Advances in Artificial Intelligence*, 2008.

[27] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy (S&P)*, 2010.

[28] C. T. Symons and J. M. Beaver. Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training. In *the 5TH ACM workshop on Artificial Intelligence and Security (AISec)*, 2012.

[29] B. Üstün, W. Melssen, and L. Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81, 2006.

[30] F. Valeur. *Real-Time Intrusion Detection Alert Correlation*. PhD thesis, University of California, Santa Barbara, May 2006.

[31] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3), 2004.

[32] I. H. Witten, E. Frank, and M. A. Hall. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann Publishers Inc., 3rd edition.

[33] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, 1995.

[34] D. Yu and D. Frincke. A novel framework for alert correlation and understanding. In *International Conference on Applied Cryptography and Network Security (ACNS)*, 2004.

[35] Y. Zhai, P. Ning, and J. Xu. Integrating IDS alert correlation and OS-level dependency tracking. *Intelligence and Security Informatics*, 2006.

[36] J. Zhou, M. Heckman, B. Reynolds, A. Carlson, and M. Bishop. Modeling network intrusion detection alerts for correlation. *ACM Transactions on Information and System Security (TISSEC)*, 10(1):4, 2007.

[37] L. Zomlot, S. C. Sundaramurthy, K. Luo, X. Ou, and S. Rajagopalan. Prioritizing intrusion analysis using DempsterShafer theory. In *the 4TH ACM workshop on Artificial Intelligence and Security (AISec)*, 2011.