Detecting Novel Network Intrusions Using Bayes Estimators [*]

Daniel Barbará[†]***, Ningning Wu***[†]***, and***
Sushil Jajodia [†]

1 Introduction

From the first appearance of network attacks, the internet worm, to the most recent one in which the servers of several famous e-business companies were paralyzed for several hours, causing huge financial losses, network-based attacks have been increasing in frequency and severity. As a powerful weapon to protect networks, intrusion detection has been gaining a lot of attention.

Traditionally, intrusion detection techniques are classified into two broad categories: misuse detection and anomaly detection. Misuse detection aims to detect well-known attacks as well as slight variations of them, by characterizing the rules that govern these attacks. Due to its nature, misuse detection has low false alarms but it is unable to detect any attacks that lie beyond its knowledge. Anomaly detection is designed to capture any deviations from the established profiles of users and systems normal behavior pattern. Although in principle, anomaly detection has the ability to detect new attacks, in practice this is far from easy. Anomaly detection has the potential to generate too many false alarms, and it is very time consuming and labor expensive to sift true intrusions from the false alarms.

As new network attacks emerge, the need for intrusion detection systems to detect novel attacks becomes pressing. As we stated before, this is one of the hardest tasks to accomplish, since no knowledge about the novel attacks is available. However, if we view the problem from another angle, we can find a solution. Attacks do something that is different from normal activities: if we have comprehensive knowledge about normal activities and their normal deviations, then all activities

[*]This work has been funded by AFRL Rome Labs under the contract F 30602-00-2-0512.

[†]All the authors are at George Mason University, Center for Secure Information Systems Fairfax, VA 22303

that are not normal should be suspicious. So the problem becomes one of how to derive the knowledge about unknown attacks from the existing information of normal activities as well as known attacks.

In this paper, we propose a method based on a technique called pseudo-Bayes estimators to enhance an anomaly detection system's ability to detect new attacks while reducing the false alarm rate as much as possible. Our work is based on an anomaly detection system called Audit Data Analysis and Mining (ADAM)[3] that we developed at the Center for Secure Information Systems of George Mason University. ADAM applies mining association rules techniques to look for the abnormal events in network traffic data, then it uses a classification algorithm to classify the abnormal events into normal instances and abnormal instances. The abnormal instances can be further categorized into attack names if ADAM has gained knowledge about the attacks. With the help of the classifier, the number of false alarms is greatly reduced because the abnormal associations that belong to normal instances will be filtered out. However, the normal instances and attacks that the classifier is able to recognize are limited to those that appear in the training data (due to the innate limitation of all supervised classification methods).

To overcome this limitation, we apply the pseudo-Bayes estimators method as a means to estimate the prior and posterior probabilities of new attacks. Then we construct a Naive Bayes classifier to classify the instances into normal instances, known attacks and new attacks. One advantage of pseudo-Bayes estimators is that no knowledge about new attacks is needed since the estimated prior and posterior probabilities of new attacks are derived from the information of normal instances and known attacks.

The rest of the paper is organized as follows. Section 2 gives an overview of ADAM system. Section 3 presents the pseudo-Bayes estimator technique. Section 4 shows the results of evaluating pseudo-Bayes technique with DARPA data. Section 5 reviews the related work and Section 6 offers the summary and conclusions of the research.

2 Overview of ADAM

ADAM is an anomaly detection system. It is composed of three modules: a preprocessing engine, a mining engine and a classification engine. The preprocessing engine sniffs TCP/IP traffic data, and extracts information from the header of each connection according to a predefined schema. The mining engine applies mining association rules to the connection records. It works on two modes: training mode and detecting mode. In training mode, the mining engine builds a profile of the users and systems normal behaviors, and generates labeled association rules, which will be used to train the classification engine. In detecting mode, the mining engine mines unexpected association rules that are different from the profile. The classification engine will classify the unexpected association rules into normal and abnormal events. Some abnormal events can be further classified as attack names. Although mining of association rules has previously been used to detect intrusions in audit trail data [13, 12], the ADAM is unique in two ways:

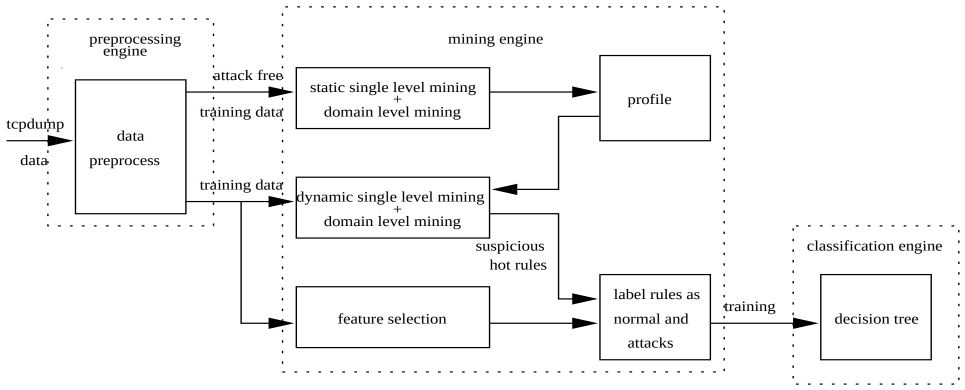

Figure 1. *The training phase of ADAM*

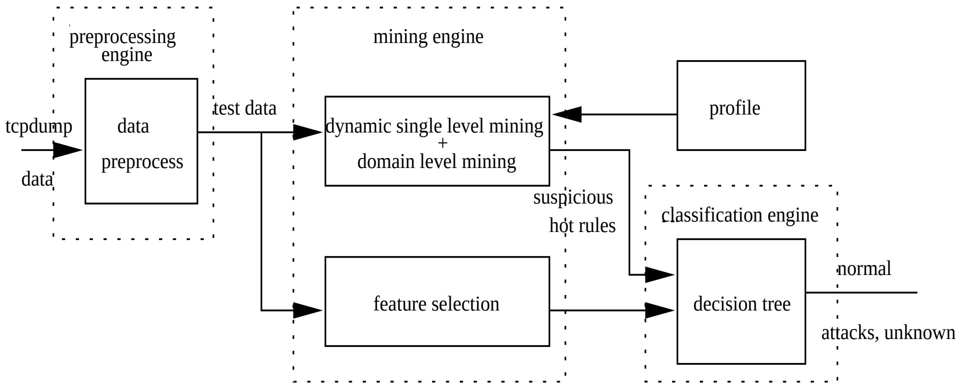

Figure 2. *The detecting phase of ADAM*

- It is real-time: it uses an incremental mining (on-line mining) which does not look at a batch of TCP/IP connections, but rather employs a sliding window of time to find the suspicious rules within that window.

- It is an anomaly detection system (instead of a misuse detection system like previous systems that use data mining for intrusion detection, such as those described in [13, 12], which aim to characterize the rules that govern misuse of the system), since it aims to detect anomalous behavior relative to a profile. For this, the technique builds, a-priori, a profile of "normal" rules, obtained by mining past periods of time in which there were no attacks. Any rule discovered during the on-line mining that also belongs to this profile is ignored, assuming it corresponds to a normal behavior. In this sense, our technique looks for unexpected rules. This helps in reducing the number of false positives flagged by the technique.

Figures 1 and 2 show the basic architecture of ADAM. ADAM performs its task in two phases. In the training phase, depicted in Figure 1, we use a data stream for which we know where the attacks are located (we also know the types of attacks). The attack-free parts of the stream are fed into a module that combines a static algorithm for mining association rules and an algorithm for mining domain level association rules [1] (both techniques will be discussed later in the paper). The output of this module is a profile of rules that we call "normal," i.e., that depict the behaviors during periods where there are no attacks. The profile along with the training data set are also fed into a module that uses a combination of a dynamic algorithm for association rules (discussed later) plus the domain level algorithm, whose output consists of rules that characterize attacks to the system. These rules, along with a set of features extracted from the data stream by a features selection module are used as the training set for a classifier. This whole phase takes place once (off-line), before we use the system to detect intrusions.

The actual detection of intrusions is implemented as depicted in Figure 2. Here the dynamic algorithm is used to produce rules that are considered as suspicious which, along with the features extracted by the features selection module are fed to the (already trained) classifier. The classifier labels the events as attacks (including its presumed type), normal events, or unknown. When the classifier labels connections as normal events, it is filtering them out of the attack set, avoiding passing them to the security officer. The last class, i.e., unknown, is reserved for events whose exact nature cannot be pinpointed by the classifier (they cannot be classified as known attacks). Conservatively, we consider those as attacks and include them in the set of alarms passed to the security officer. In the rest of the section, we further clarify the concepts and techniques involved in our system.

2.1 Preprocessing

Preprocessing module looks at TCP/IP traffic, and generates a record for each connection from the header information of its packets based on the following schema:

$$R(T_s, Src.IP, Src.Port, Dst.IP, Dst.Port, FLAG).$$

In this schema, T_s represents the beginning time of a connection, $Src.IP$ and $Src.Port$ refer to source IP and port number respectively, while $Dst.IP$ and $Dst.Port$ represent the destination IP and port number. The attribute $FLAG$ describes the status of a TCP/IP connection. The relation R contains the dataset that is subject of the association mining. Notice that in this context, the association rules we can come up with are more restrictive than in the general market-basket data case. For instance, there can be no rules with two different $Src.IP$ values in the antecedent. (No connection comes from two sources.) Nevertheless, the number of potential rules is large: connections may come from a large base of source IP addresses and ports.

[1] this is called multi-level mining in data mining terminology; to avoid confusion with the multi-level security concept, we use the term domain level instead in this paper

2.2 Mining engine

The basic idea behind the mining engine is the concept of association rules from the connection records. The task of mining association rules, first presented in [2] consists in deriving a set of rules in the form of $X \longrightarrow Y$ where X and Y are sets of attribute-values, with $X \bigcap Y = \emptyset$ and $\|Y\| = 1$. The set X is called the antecedent of the rule while the item Y is called consequent. For example, in a market-basket data of supermarket transactions, one may find that customers who buy *milk* also buy *honey* in the same transaction, generating the rule $milk \longrightarrow honey$. There are two parameters associated with a rule: *support* and *confidence*. The rule $X \longrightarrow Y$ has *support* s in the transaction set T if $s\%$ of transactions in T contain $X \cup Y$. The rule $X \longrightarrow Y$ has *confidence* c if $c\%$ of transactions in T that contain X also contain Y. The most difficult and dominating part of an association rules discovery algorithm is to find the itemsets $X \bigcup Y$, that have strong support. (Once an itemset is deemed to have strong support, it is an easy task to decide which item in the itemset can be the consequent by using the confidence threshold.) For this reason, we actually aim to find high-support itemsets in our technique, rather than their rules. We use the terms rules and itemsets interchangeable throughout the paper.

Our procedure can be described as follows. We first create a database of association rules (itemsets) that have strong support (above a certain threshold) for datasets for which we are absolutely sure there were no attacks. This constitutes our *training data* for the system. The rules in the database can be cataloged according to the time of the day and day of the week, to further refine the specificity of these rules to variations of workload during the different time periods. Next, we use an incremental, on-line algorithm to detect rules that are currently receiving strong support. In order to do this, we use a sliding window of predetermined size δ, and an algorithm that outputs rules that have received strong support during this window. We compare any rule that starts receiving support with rules in the database for an analogous time and day of the week situation. If the rule is present in the database, we do not pay attention to it (i.e., we do not devote storage resources to keep track of its support). On the other hand, if the rule is not in the database, we keep a counter that will track the support that the rule receives. If the rule's support exceeds a threshold, that rule is reported as suspicious. For a set of suspicious rules, we provide two services. First the ability to drill down and find the raw data in the audit trail that gives rise to these rules. Secondly, we characterize the set of rules with a vector of parameters (based on the raw audit trail data that gave rise to the rules) and use this vector as an input to a classifier. The classifier sorts each suspicious activity as a normal event, known attack class or an unknown type.

There are three components in the mining engine:

Single level mining

Single level mining has two working modes: static mining and dynamic mining. Static mining, also called off-line mining, is used to generate the profile by finding the "ordinary" itemsets that occur during non-intrusion times. Dynamic mining uses a sliding window method that implements incremental, on-line association rule mining. It only needs to look at each connection once, which makes it appropriate for on-line detection.

Domain level mining

Sometimes an attack will take place as a coordinated effort from several hosts or/and to multiple hosts. In that case, it is likely that itemsets of the form Src.IP, Dst.IP will not have enough support to be flagged as suspicious. However, if we aggregate the support of all the itemsets of that form where all the source IPs or destination IPs belong to the same subnet, the support may be enough to recognize this as a suspicious event.

Indeed, we have realized that, in practice such situations arise and cannot be captured by just mining associations on the schema level. So we complemented our technique by using domain level mining, which roughly speaking, can be thought of clustering itemsets that have some commonality and aggregating the support that the individual (flat) itemsets exhibit. The mining of multi-level association rules were first proposed by J. Han, et al. [9]. They use a top-down progressive deepening method for mining multiple-level association rules. Although we borrow the concept, we instead use a bottom-up method to produce the domain level rules.

Given the schema $R(T_s, Src.IP, Src.Port, Dst.IP, Dst.Port, FLAG)$, we can produce higher abstractions of the IP related attributes. For instance, the first byte in the host ID portion of the IP address usually identifies the subnet to which the host belongs, while the first two bytes of the IP address identify the net ID. In the domain level mining, we define 4 layered subnets from the lowest level to highest level: Sub_1 which is identified by the first 3 bytes of the IP address, Sub_2 by the first 2 bytes of the IP address, Sub_3 by the first 1 byte of the IP address, Sub_4 is the highest level subnet that contains all possible IPs. Clearly Sub_1 is the first level abstraction on IP address, Sub_2 is the first level abstraction on Sub_1, and so on. The itemsets of interest of domain level are similar to those of single level mining, except that the every IP will be replaced with every possible item of the set $\{IP, sub_1, sub_2, sub_3, sub_4\}$.

Feature selection

Feature selection by nature is a multi-window mining process. Both *Dynamic Mining* and *Domain Level Mining* algorithms use single size time window, and it is very hard to choose an optimum window size that can capture all kinds of rules that appear at different frequencies. For instance, if the time window is too big, the algorithms may miss some rules that are hot only in a short period of time. On the other hand, if the time window is too small, they may miss the rules that span a long time period in a slow manner. Our motivation to do feature selection is to overcome the limitations of single window size mining algorithms. Two time windows are used here. One is 3 seconds wide, and it is used to capture the rules that are only hot in a very short period of time and can easily be missed in a wide window. The other is 24 hours long, and it is used to capture the rules that appear at very low frequency but last for long time. Both time windows are applied on *Dynamic Mining* and *Domain Level Mining* algorithms. Some features like average connection rate per second, contiguity index of a source IP to a set of destination IP, etc., are extracted from the mining results and will be used for the further analysis.

2.3 Classification Engine

The abnormal rules generated by mining association rules algorithm are intended to guide the further detection work. To filter out as many false positives as possible, we complement the technique by adding a classification step. Using a set of parameters for each rule in the training data, we build a classifier (usually a modified decision tree, however, we have test the system with a variety of classifiers).

By studying the attacks of training data, we choose a number of attributes, or parameters to characterize each itemset, which reflect the properties of single level mining, domain level mining and feature selection. Then, a classifier is trained by the itemsets derived from training data. The classifier will be used to classify the itemsets derived from the test data.

3 Our technique: pseudo-Bayes estimators

3.1 Introduction

Pseudo-Bayes estimators is a well used technique in the area of discrete multivariate analysis [4]. It is used to provide the estimated cell values of contingency tables which may contain a large number of sampling zeros. All too often, the observed table of counts provides an unsatisfactory table of estimated values since there are many cells and few observations per cell, and some zeros are “smaller” than others, especially when we are computing rates. For instance, it may be misleading to report both 0/5 and 0/500 as being equal to zero, since as rates they carry quite different information. In order to distinguish such zero properties from one another , the observed counts need to be smoothed since the observed table of counts seems too abrupt. The basic problem here is one of simultaneously estimating a large number of parameters (the expected cell frequencies). One way to provide such estimates is to assume an underlying parametric model for the expected frequencies, where the number of parameters in the model is typically much smaller than the total number of cells. pseudo-Bayes is another approach to solve the problem which does not involve the problem of model selection. The idea behind pseudo-Bayes is as follows [4]:

Let $\mathbf{X} = (X_1, \ldots, X_t)$ have the multinomial distribution with parameters $N = \sum_{i=1}^{t} X_i$ and $\mathbf{p} = (p_1, \ldots, p_t)$. We observe a vector of values $\mathbf{x} = (x_1, \ldots, x_t)$, where x_i is the observed count in the ith category and $\sum_{i=1}^{t} x_i = N$. The vector $\mathbf{p}$ takes values in the parameter space

$$L_t = \mathbf{p} = (p_1, \ldots, p_t) : p_t \geq 0 \qquad and \qquad \sum_{i=1}^{t} p_t = 1 \tag{1}$$

and we denote the center of L_t by $\mathbf{c} = (t^{-1}, \ldots, t^{-1})$.

The kernel of the likelihood function for this multinomial distribution is

$$l(\mathbf{p} \mid \mathbf{x}) = l(p_1, \ldots, p_t \mid x_1, \ldots, x_t) = \prod_{i=1}^{t} p_i^{x_i}. \tag{2}$$

The natural conjugate family of prior distribution for this likelihood is the Dirichlet, whose densities have the form

$$f(\mathbf{p} \mid \beta) = \Gamma(\sum_{i=1}^{t} \beta_i) \prod_{i=1}^{t} \frac{p_i^{\beta^{-1}}}{\Gamma(\beta_i)} \tag{3}$$

where $\beta_i > 0$ for all i and $\Gamma(y)$ is the gamma function given by $\Gamma(y) = \int_0^\infty e^{-z} z^{y-1} dz$. When the prior distribution is Dirichlet with parameters $\beta = (\beta_1, \ldots, \beta_t)$, the posterior distribution is also Dirichlet with parameters $\beta + \mathbf{x} = (\beta_1 + x_1, \ldots, \beta_t + x_t)$.

If we let

$$D(\beta_1, \beta_2, \ldots, \beta_t) = \frac{\Gamma(\sum_{i=1}^{t} \beta_i)}{\prod_{i=1}^{t} \Gamma(\beta_i)} \tag{4}$$

then the moments of the Dirichlet distribution are given by

$$E(\prod_{i=1}^{t} p_i^{a_i} \mid \beta) = \frac{D(\beta_1, \ldots, \beta_t)}{D(\beta_1 + a_1, \ldots, \beta_t + a_t)} \tag{5}$$

If we set

$$K = \sum_{i=1}^{t} \beta_i, \qquad \lambda_i = \frac{\beta_i}{K} \tag{6}$$

we see that the prior and posterior means of p_i are given by

$$E(p_i \mid K, \lambda) = \lambda_i \qquad (\textit{prior mean}) \tag{7}$$

$$E(p_i \mid K, \lambda, \mathbf{x}) = \frac{x_i + K\lambda_i}{N + K} \qquad (\textit{posterior mean}) \tag{8}$$

We can rewrite (8) in vector notation as

$$E(\mathbf{p} \mid K, \lambda, \mathbf{x}) = \frac{N}{N+K}(\mathbf{x}/N) + \frac{K}{N+K}\lambda \tag{9}$$

The posterior mean is the Bayesian point estimate of $\mathbf{p}$. When the prior distribution is Dirichlet with parameters K and λ, this Bayesian point estimate is given by (9).

We define the risk function as the expected value of the squared distance from an estimator $\mathbf{T}$ to $\mathbf{p}$ as follows:

$$R(\mathbf{T}, \mathbf{p}) = NE\|\mathbf{T} - \mathbf{p}\|^2 = N \sum_{i=1}^{t} E(T_i - p_i)^2 \tag{10}$$

We denote the random variable version of the Bayes estimator given in (9) by

$$\hat{\mathbf{q}} = \hat{\mathbf{q}}(K, \lambda) = \frac{N}{N+K}(\mathbf{X}/N) + \frac{K}{N+K}\lambda \tag{11}$$

Since K and λ are constants, we can easily compute the risk function of $\hat{\mathbf{q}}$:

$$R(\hat{\mathbf{q}}, \mathbf{p}) = (\frac{N}{N+K})^2(1 - \|\mathbf{p}\|)^2 + (\frac{K}{N+K})^2 N\|\mathbf{p} - \lambda\|^2 \tag{12}$$

In order to use $\hat{\mathbf{q}}(K, \lambda)$, we need to know the values of K and λ. K can be chosen in a way such that it depends on the data and the choice of λ. If λ is regarded as fixed, then we can find the value of K that minimizes the risk $R(\hat{\mathbf{q}}(K, \lambda), \mathbf{p})$ by differentiating (10) in K and solving the resulting equation. This yields

$$K = K(\mathbf{p}, \lambda) = \frac{1 - \|\mathbf{p}\|^{\mathbf{2}}}{\|\mathbf{p} - \lambda\|^{\mathbf{2}}} \tag{13}$$

This optimal value of K depends on the unknown value of $\mathbf{p}$. We may obtain an estimate of this unknown optimal value of K by replacing $\mathbf{p}$ by $\hat{\mathbf{p}} = \mathbf{X}/N$, yielding

$$\hat{K} = K(\hat{\mathbf{p}}, \lambda) = \frac{1 - \|\hat{\mathbf{p}}\|^2}{\|\hat{\mathbf{p}} - \lambda\|^2} \tag{14}$$

or, in terms of $\mathbf{x}$, the observed value of the random variable $\mathbf{X}$,

$$\hat{K} = \frac{N^2 - \sum_{i=1}^{t} x_i^2}{\sum_{i=1}^{t} x_i^2 - 2N \sum_{i=1}^{t} x_i \lambda_i + N^2 \sum_{i=1}^{t} \lambda_i^2} \tag{15}$$

A pseudo-Bayes estimator of $\mathbf{p}$ is then

$$p* = \hat{\mathbf{q}}(\hat{K}, \lambda) = (\frac{N}{N+\hat{K}})\hat{\mathbf{p}} + \frac{\hat{K}}{N+\hat{K}}\lambda \tag{16}$$

where $\hat{K}$ is given in (15). Other pseudo-Bayes estimators of $\mathbf{p}$ are possible, and they correspond to alternative ways of estimating the optimal value of K.

3.2 Building Naive Bayes classifier using Pseudo-Bayes estimators

Given a training data that is composed of both normal and known attack instances, we can construct a contingency table, in which each column refers to an attribute

$$\begin{array}{ccccccccc}
 & & X_0 & & \dots & & X_j & & \dots \\
A_0 & t_{00_0} & \dots & t_{00_{g_0}} & \dots & t_{0j_0} & \dots & t_{0j_{g_j}} & \dots \\
 & & & & & & \vdots & & \\
A_i & t_{j0_0} & \dots & t_{i0_{g_0}} & \dots & t_{ij_0} & \dots & t_{ij_{g_j}} & \dots \\
 & & & & & & \vdots & & \\
A_{I-1} & t_{(I-1)0_0} & \dots & t_{(I-1)0_{g_0}} & \dots & t_{(I-1)j_0} & \dots & t_{(I-1)j_{g_j}} & \dots \\
A_I & t_{I0_0} & \dots & t_{I0_{g_0}} & \dots & t_{Ij_0} & \dots & t_{Ij_{g_j}} & \dots
\end{array}$$

Figure 3. Contingency table of the training data

characterizing an aspect of the instances and each row refers to a class of the training data that is either normal or an attack name. The attributes can be discrete variables or categorical variables. For simplicity, we will transform discrete variables into categorical variables. Besides all the classes in training data, an extra class will be added to represent new attacks. The table is built in such way that the cell value of ith row and jth column denotes the number of instances in training data that class i and attribute j both are present. The cell values of new attacks will be initialized with zeros. By pseudo-Bayes, the contingency table will be smoothed and each cell will be given an estimated value. The estimated cell values of unknown attacks will be $K\lambda_i/(N+K)$ by (8).

Let $A_0, \dots, A_{I-1}$ denote I class types in the training data and n_i be the number of instances of the class A_i in the training data. Let $\mathbf{x} = x_0, \dots, x_{J-1}$ be an instance with J attributes whose domains are denoted as $D = (D_0, \dots, D_{J-1})$, and g_j be the number of categories in D_j. The training data can be represented by a $(I+1) * M$ table T shown in Figure 3, where $M = \sum_{j=0}^{J-1} g_j$. Here t_{ij_k} denotes the number of instances that both class A_i and x_{j_k} are present. x_{j_k} refers to the kth categorical value of X_j. A_I represents unknown attacks, and $t_{ij_k} = 0$ if $i = I$. $\sum_{i=0}^{I-1} \sum_{k=0}^{g_j} t_{ij_k} = N$, for $j = 0, \dots, J-1$. N is the number of instances in training data. If we assume all attributes are conditionally independent given a class A_i, then we can partition the table into J parts. Each part represents an attribute and will be smoothed by pseudo-Bayes individually.

The pseudo-Bayes estimator method is shown as an algorithm in Figure 4. Before applying pseudo-Bayes estimator, a small constant is added to each cell of the table to ensure $\lambda_{ij_k} \neq 0$. The symbol x_j denotes the jth column value of a part table. If $\hat{t}_{ij_k}$, is the pseudo-Bayes estimated cell value of T_j, then the posterior probability $P(x_j \mid A_i)$ can be easily computed as shown in Equation 17

```
For a $T_j$, select $\lambda_{ij_k}$ as following:
$\lambda_{ij_k} = \frac{1}{g_0}\frac{t_{ij_k}}{S_{j_k}}, \qquad S_{j_k} = \sum_{i=0}^{I} t_{ij_k}$
compute the weighted factor $\hat{K} = (N^2 - \sum t_{ij_k}^2)/\sum_{i,j}(t_{ij_k} - N\lambda_{ij_k})^2$
compute the cell estimates $\hat{t_{ij_k}} = Np_{ij_k}^* = N(t_{ij_k} + \hat{K}\lambda_{ij_k})/(N + \hat{K})$
```

Figure 4. *Algorithm* Pseudo-Bayes estimator algorithm

$$P(x_j \mid A_i) = \frac{\hat{t}_{ij_k}}{\sum_{k=0}^{g_j} \hat{t}_{ij_k}} \tag{17}$$

Using Naive Bayes,

$$\begin{aligned}
P(C = A_i \mid X = x) &= \frac{P(X = x, C = A_i)}{P(X = x)} \\
&= \frac{P(X = x \mid C = A_i)P(C = A_i)}{P(X = x)} \\
&= \frac{P(X_0 = x_{0_{m0}} \mid C = A_i) \ldots P(X_{J-1} = x_{J-1_{m_{J-1}}} \mid C = A_i)P(C = A_i)}{P(X = x)} \\
&= \frac{(\prod_{j=0}^{J-1} P(x_{m_j})P(C = A_i))}{P(X = x)} \qquad (18)
\end{aligned}$$

Here, x_{m_j} refers to the m_jth column of the table T_j, $P(x_{m_j} \mid C_=A_i)$ is computed by (17), and $P(C = A_i) = (1/J)(\sum_{j=0}^{J-1}(\sum_{k=0}^{g_j} \hat{t}_{ij_k}/N))$.

4 Experimental Results

We applied the pseudo-Bayes estimator method on two years of DARPA Intrusion Detection Evaluation Data[5] (1998, and 1999). Each year's data contains two types of data: training data and test data. The 1998 training data consists of 7 weeks of network-based attacks in the midst of normal background data. Attacks are labeled in training data. The test data consists of 2 week network-based attacks and normal background data. Both Tcpdump and BSM data are provided for each day. The 1999 data consists of 3 weeks training data and 2 weeks test data. Besides Tcpdump and BSM data, NT audit data are added into 1999 evaluation. Currently, ADAM only works only on Tcpdump data.

To evaluate our approach, we configure the experiments in two ways:

- *Configuration1* Given a training data, we build a Naive Bayes classifier by removing one attack type at a time, then test against the removed attack, using the test data.

Attack	#instances	#classified as unknown	detection rate by unknown	#classified as other attacks	detection rate by misclassification	false alarm rate
smurf	156	156	100%	0	0%	0.1%
mscan	6	6	100%	0	0%	0.1%
ipsweep	3	3	100%	0	0	1.7%
portsweep	5	5	100%	0	0%	1.7%
pod	73	73	100%	0	0%	1.7%
mailbomb	2	2	100%	0	0%	4.3%
teardrop	12	12	100%	0	0%	1.2%
snmpgetattack	13	0	0%	0	0%	0.2%
back	8	0	0%	0	0%	0.1%
neptune	23	6	26%	0	0%	2.9%

Figure 5. *Experiment of Configuration1 on DARPA 1998 test data*

- *Configuration2* Given a set of training data and test data, we build a Naive Bayes classifier based on training data, then test against the test data. To evaluate the classifier's ability to detect new attacks, we choose the test data in such way that it consists of at least one attack that are not appeared in the training data.

Figure 5 shows the experimental results of *Configuration1* on DARPA 1998 test data.[2] To better evaluate the performance of pseudo-Bayes estimator, we pick a set of attacks that behave very differently, while for the attacks that share some similarities, we only select one candidate to represent the rest. The reason we do so is to try to avoid the influences of the existence of similar attacks in the training data when testing an attack. The first column refers to the attack that is in the test data but not in training data. The column *#instance* gives the occurrence of the attack in test data. The column *#classified as unknown* gives the number of instances that are classified as unknown. The column *detection rate by unknown* is defined as the ratio of *#classified as unknown* to *#instances.* The column *#classified as other names* gives the number of instances in which an attack is misclassified as some other attack. The column *detection rate by misclassification* is defined as the ratio of *#classified as other names* to *#instances.* The experiment shows that the pseudo-Bayes estimators technique works very well in detecting new attacks: 7 out 10 attacks can be fully detected. One attack is partially detected, i.e., some instances of the attack are detected, but some are not. 2 attacks are totally missed. As we studied the partially detected and totally missed attacks, we found that they share some similarities with normal instances so that they cannot be easily distinguished from the normal ones. The missed attacks are more similar to the normal instances than the partial detected attacks, while the full detected attacks are very different from the normal instances. [3] The experiment also hints that pseudo-Bayes method works well when a new attack is substantially different from normal instances.

Figures 6–8 show an experiment of *Configuration2.*[4] In the experiment, the

[2]Please note that each row shows the result of a different experiment, and thus it is has its own false alarm rate.

[3]Here the selection of attributes and their values are derived from the mining engine, so the degree of similarity between instances is scaled in terms of their attribute values.

[4]The Figure 7 shows the result of a single experiment, and thus only the overal false alarm rate

Name	dict	pod	satan	nmap	guest	smurf
#instances	2	23	10	1	2	348

Name	teardrop	ipsweep	neptune	portsweep	back	normal
#instances	31	47	50	39	4	1294

Figure 6. *Attack distribution of DARPA 1998 training data*

Attack	#instances	#classified as unknown	detection rate by unknown	#classified as other attacks	detection rate by misclassification	false alarm rate
mailbomb	4	2	50%	2	50%	6.8%
apache2	9	1	11%	0	0%	
udpstorm	12	8	75%	4	25%	
telnet	1	1	100%	0	0%	
guessftp	1	1	100%	0	0%	
guesspop	1	1	100%	0	0%	
process	3	3	100%	0	0%	
pod	6	6	100%	0	0%	
dict	2	1	50%	0	0%	
teardrop	2	2	100%	0	0%	
neptune	130	13	10%	0	0%	
back	3	0	0%	0	0%	
smurf	62	60	99%	0	0%	
satan	6	1	16%	6	100%	
portsweep	3	3	100%	0	0%	

Figure 7. *Experiment of Configuration 2 on DARPA 1999 inside data when trained by 1998 training data*

classifier is trained with DARPA 1998 training data and tested by DARPA 1999 test data. Figure 6 shows the attack distribution of the training data. Figures 7 and 8 are experiment results of two datasets obtained from an insider sniffer and an outside sniffer respectively. The attacks in bold refer to the new attacks in test data. For inside data, 4 of 7 new attacks are fully detected, while the other 3 attacks are partially detected, For outside data, 4 of 7 new attacks are fully detected, but 2 attacks are totally missed, and 1 attack is partially detected.

Figures 9-11 show another experiment of *Configuration2*. Here the test data is kept unchanged, but the training data uses DARPA 1999. Figure 9 shows the attack distribution of 1999 training data which contains fewer attacks than 1998 training data. The names in bold refers to the new attacks in test data. Once again, the experiment shows very good performance of the pseudo-Bayes technique. For inside data, 6 of 10 new attacks are fully detected, 2 attacks are partial detected, while 2 are totally missed. For the outside data, 4 out of 8 attacks are fully detected, and 2 attacks are partially detected, and 2 are totally missed.

Comparing the two experiments showed in figures 7-8 and figures 10-11, we can see that the first experiment has lower false alarm rates than the second one because it is trained with more training classes. The misclassification rate of attack *teardrop* in the first experiment is smaller than the second one since *teardrop* is a known attack to the first one, and the misclassification rate of the attack *mailbomb* in the second experiment is smaller than the first one for the same reason. If an attack cannot be distinguished from the normal instances, like *back*, then whether the training data contains the attack nor not does not affect the results much.

is shown. So on for the other figures of Configuration2.

Attack	#instances	#classified as unknown	detection rate by unknown	#classified as other attacks	detection rate by misclassification	false alarm rate
mailbomb	8	3	37.5%	3	37.5%	5.7%
crashiis	1	0	0%	0	0%	
telnet	3	3	100%	0	0%	
guessftp	1	1	100%	0	0%	
guesspop	1	1	100%	0	0%	
process	2	2	100%	0	0%	
apache2	8	0	0%	0	0%	
pod	11	11	100%	0	0%	
teardrop	3	3	100%	0	0%	
neptune	134	19	14%	0	0%	
back	6	0	0%	0	0%	
smurf	24	24	100%	0	0%	
ipsweep	1	1	100%	0	0%	
portsweep	3	3	100%	0	0%	
satan	7	1	14%	6	86%	

Figure 8. *Experiment of Configuration 2 on DARPA 1999 outside data when trained by 1998 training data*

Name	pod	mailbomb	satan	ipsweep	neptune	portsweep	normal
#instance	11	1	3	22	13	5	1053

Figure 9. *Attack distribution of DARPA 1999 training data*

Attack	#instances	#classified as unknown	detection rate by unknown	#classified as other attacks	detection rate by misclassification	false alarm rate
apache2	9	0	0%	0	0%	7.1%
udpstorm	12	12	100%	0	0%	
telnet	1	1	100%	0	0%	
guessftp	1	1	100%	0	0%	
guesspop	1	1	100%	0	0%	
process	3	3	100%	0	0%	
dict	2	1	50%	0	0%	
teardrop	2	1	50%	1	50%	
back	3	0	0%	0	0%	
smurf	62	60	99%	0	0%	
mailbomb	4	4	100%	0	0%	
pod	6	6	100%	0	0%	
neptune	130	13	10%	0	0%	
satan	6	1	16%	1	16%	
portsweep	3	3	100%	0	0%	

Figure 10. *Experiment of Configuration 2 on DARPA 1999 inside data when trained by 1999 training data*

Attack	#instances	#classified as unknown	detection rate by unknown	#classified as other attacks	detection rate by misclassification	false alarm rate
crashiis	1	0	0%	0	0%	5.6%
telnet	3	3	100%	0	0%	
guessftp	1	1	100%	0	0%	
guesspop	1	1	100%	0	0%	
process	2	1	50%	0	0%	
apache2	8	0	0%	0	0%	
teardrop	3	2	67%	1	33%	
smurf	24	24	100%	0	0%	
back	6	0	0%	0	0%	
pod	11	11	100%	0	0%	
neptune	134	19	14%	0	0%	
ipsweep	1	1	100%	0	0%	
mailbomb	8	3	37.5%	3	37.5%	
portsweep	3	3	100%	0	0%	
satan	7	1	14%	2	30%	

Figure 11. *Experiment of Configuration 2 on DARPA 1999 outside data when trained by 1999 training data*

5 Related work

The statistical anomaly detector of SRI's IDES[6, 15] uses a deductive process based on statistics to determine if current activity is atypical. Audited activity is described by a vector of intrusion detection variables that correspond to the measures recorded in the profiles. Forrest et al.[7] record frequent subsequences of system call that are invoked in the execution of a program. Absence of subsequences in the current execution of the same program from the stored sequences constitutes a potential anomaly. Lee[14] using a rule learning program, generated rules that predict the current system call based on a window of previous system calls. Abnormality is suspected when the predicted system call deviates from the actual system call. Lane and Brodley[11] used a similar approach but they focused on an incremental algorithm that updates the stored sequences and used data from UNIX shell commands. Ghosh and Schwartzbard [8] proposed using a neural network to learn a profile of normality. Teng et al. [16] proposed an approach of anomaly detection by using inductively generated sequential patterns which are used to characterize users' behavior over time. A rulebase is used to store patterns of user activities and anomalies are reported when a user's activity deviates significantly from those specified in the rules.

This work reveals a method of detecting new attacks by estimating the prior and posterior probabilities of the new attacks from the knowledge of normal activities. We are not aware of any closely related work.

6 Conclusions

We have shown that the pseudo-Bayes estimator method can enhance ADAM's ability to detect new attacks. The experimental results show that the method helps in detecting new attacks whose properties are different and distinguishable from the normal instances of training data. For new attacks that do not differ much from the normal instances, the method does not work well and will misclassify them as normal instances. (Misclassification is not unique in pseudo-Bayes, and it exists in every supervised classifier when some classes are not distinguishable.) However, the ability to capture new classes is rarely present in classifiers, and that is precisely what we have achieved in this work. We will continue our research on exploring more information from training data to: 1) optimally select the parameters λ. There are 2 ways to choose λ: data independent and data dependent. In this work, we use data dependent method since we believe training data carries the information about the distribution of known and unknown attacks as well as their probabilities. As the selection of the parameter λ is very important and affect the performance of pseudo-Bayes, the selection should take into account of the information of training data. 2) guide the initialization of cell values of new attacks instead of blindly assigning them to zeros, which may characterize new attacks more efficiently and accurately.

Bibliography

[1] R. Agrawal and R. Srikant, *Fast Algorithms for Mining Assocation Rules*, in Proceedings of the 20th VLDB Conference, Santiago, Chile, May, 1994.

[2] R. Agrawal and T. Imielinski and A. Swami, *Mining Association rules between sets of items in large databases*, in Proceedings of the ACM SIGMOD Conference on Management of Data, Washington D.C., May, 1993.

[3] D. Barbará and S. Jajodia and N. Wu and B. Speegle, *The ADAM project*, http://www.isse.gmu.edu/ dbarbara/adam.html.

[4] Y.M.M. Bishop and S.E. Fienberg, *Discrete Multivariate Analysis: Theory and Practice*, The MIT Press, 1975.

[5] DARPA, DARPA 1998 Intrusion Detection Evaluation, http://ideval.ll.mit.edu/1998_index.html.

[6] D. Denning and P. Neuman, *IDES*, http://www.csl.sri.com/intrusion/intrusion-main.html#IDES.

[7] S. Forrest and S. Hofmeyr and A. Somayaji and T. Longstaff, *A sense of self for unix processes*, in Proc. of IEEE symp. Security and Privacy, 1996.

[8] A. Ghosh and A. Schwartzbard, *A study in using neural networks for anomaly and misuse detection*, in Proc.. USENIX Security Symp., 1999.

[9] J. Han and Y. Fu, *Discovery of Multiple-Level Association Rules from Large Databases*, in Proceedings of the 21st Very Large Data Bases Conference, Zurich, Swizerland, 1995.

[10] G.H. John and P. Langley, *Estimating Continuous Distributions in Bayesian Classifiers*, in Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 1995.

[11] T. Lane and C. Brodley, *Approaches to online learning and concept drift for user identification in computer security*, in Proc. Intl. Conf. Knowledge Discovery and Data Mining, 1998.

[12] W. Lee and S.J. Stolfo, *Data Mining Approaches for Intrusion Detection*, in Proceedings of the Seventh USENIX Security Symposium, San Antonio, TX, January, 1998.

[13] W. Lee and S.J. Stolfo and K.W. Mok, *Mining Audit Data to Build Intrusion Detection Models*, in Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98), New York, NY, August, 1998.

[14] W. Lee and S. Stolfo and K. Mok, *A Data Mining Framework for Building Intrusion Detection Models*, in Proceedings of the IEEE Symposium on Security and Privacy, 1999.

[15] *SRI-International*, *Next-Generation Intrusion Detection Expert System*, http://www.csl.sri.com/nides/index.html.

[16] H.S. Teng and Kaihu chen and S.C. Lu, *Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns*, in IEEE Symposium on Security and Privacy, Oakland, CA, May, 1990.