

# A Novel Data Mining-Based Method for Alert Reduction and Analysis

Fu Xiao

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, P. R. China

Email: [fuxiao1225@hotmail.com](mailto:fuxiao1225@hotmail.com)

Shi Jin

School of State Secrecy and Department of Information Management, Nanjing University, Nanjing, P. R. China

Email: [zgnjack@hotmail.com](mailto:zgnjack@hotmail.com)

Xie Li

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, P. R. China

Email: [xieli@nju.edu.cn](mailto:xieli@nju.edu.cn)

**Abstract**—Current system managers often have to process huge amounts of alerts per day, which may be produced by all kinds of security products, network management tools or system logs. This has made it extremely difficult for managers to analyze and react to threats and attacks. So an effective technique which can automatically filter and analyze alerts has become urgent need. This paper presents a novel method for handling IDS alerts more efficiently. It introduces a new data mining technique, outlier detection, into this field, and designs a special outlier detection algorithm for identifying true alerts and reducing false positives (i.e. alerts that are triggered incorrectly by benign events). This algorithm uses frequent attribute values mined from historical alerts as the features of false positives, and then filters false alerts by the score calculated based on these features. We also proposed a three-phrase framework, which not only can filter newcomer alerts in real time, but also can learn from these alerts and automatically adjust the filtering mechanism to new situations. Moreover our method can help managers analyze the root causes of false positives. And our method needs no domain knowledge and little human assistance, so it is more practical than current ways. We have built a prototype implementation of our method. Through the experiments on DARPA 2000, we have proved that our model can effectively reduce false positives. And on real-world dataset, our model has even higher reduction rate. By comparing with other alert reduction methods, we believe that our model has better performance.

**Index Terms**—Intrusion detection, false positives, outlier detection, alert reduction

## I. INTRODUCTION

Current system managers often have to process huge amounts of alerts per day, which may be produced by all kinds of security products, network management tools or system logs. Even awfully, single product can also produce a lot of alerts. For example, people have observed that IDS can easily trigger thousands of alerts per day, up to 99% of which are false positives (i.e. alerts

that are triggered incorrectly by benign events)[1]. This flood of mostly false alerts has made it very difficult for managers to analyze security state. Moreover the reactions to dangerous attacks are often delayed, because the alerts for them are hidden in huge amounts of trivial ones and are often neglected. So how to reduce false positives in intrusion detection is an important problem needing researchers pay more attentions to.

Now one popular solution to this problem is using certain algorithms (e.g. machine learning or statistical algorithms) to identify true alerts and filter false ones from raw data. Some related methods have been proposed but most of them have 3 limitations. Firstly, they usually need a lot of labeled training data or domain knowledge to build their alert reduction model. However these data are often difficult to obtain. Secondly, most of them are off-line model which will delay the reaction to attacks. Thirdly, most of them can not adapt to new situations. In this paper we proposed a novel method, which hasn't above limitations. It is based on a new data mining technique, outlier detection. This technique has been successfully applied in many fields (e.g. fraud detection, weather prediction), but has not been used to reduce false positives. Moreover, our method can help managers find the root causes of false positives. As we know, most of current alert reduction systems can't do this work.

In order to filter IDS alerts better, we have designed a special outlier detection algorithm for this field, i.e. an improved frequent pattern-based outlier detection algorithm. It assigns each alert an FP score, which indicates the possibility that the alert is a false positive. The score is calculated based on how many frequent attribute values the alert contains. Usually the more frequent patterns an alert has, the higher its score is, and the more likely it is a false positive. In order to filter alerts in real time, we also design a three-phrase framework. In the learning phrase, we build the feature set of false positives and calculate the threshold of true alerts based

on this set. Then in the online filtering phase, we compare the FP score of each new come alerts with this threshold so as to determine whether it is false positive or not. The feature set is automatically updated so as to keep its accuracy. Finally in the phase for discovering root causes, the feature set is used to help manager analyze the root causes of false positives.

We have validated our method by experiments on both DARPA 2000 dataset and real-world data. The results on DARPA 2000 show that when 86% of alerts have been filtered by our model, 100% of true alerts still remain. On real-world dataset our model has even higher reduction rate. And after comparing with two representations of current reduction methods, we believe that our system has better performance.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces our outlier detection-based alert reduction model in detail. Section 4 presents our experiments and compares our method with others. Section 5 concludes the paper and introduces the future work.

## II. RELATED WORK

There is still not so much work on identifying and reducing IDS alerts. Current methods can be divided into three categories, which are described as follows:

**Method based on Classification.** The main idea of this method is building an alert classifier that tells true from false positives. Tadeusz Pietraszek has presented one such system in [2]. It firstly generates training examples based on the analyst's feedback. Then these data are used by machine learning techniques to initially build and subsequently update the classifier. Finally the classifier is used to process new alerts. This method can classify alerts automatically. But its main shortcoming is producing labeled training data is labor intensive and error prone.

**Method based on Root Cause Analysis.** This method firstly discovers and understands the root causes of IDS alerts. Then according to these causes, the alerts triggered by attacks can be distinguished from those triggered by benign events. In [1, 3 and 4], Klaus Julisch has proposed such a model based on conceptual clustering. It generalizes alerts according to the Generalization Hierarchies built on each alert attribute. And the final generalized alerts will be presented to users to help root causes analysis. The main disadvantage of this method is Generalization Hierarchies are not easy to build. It depends on the experience of domain experts and collecting enough background knowledge. Moreover this kind of methods only supports offline process.

**Methods based on the Assumption that "Frequent Alert Sequences Are Likely Resulted from Normal Behaviors".** Now many alert reduction methods belong to this category. For example, in [5] frequent episode mining are applied on IDS alerts. When the frequent sequences of alerts that have the same destination are found, they will be presented to users in order to determine whether they are false positives or not. [6] and [7] is similar to [5], but [6] uses continuous and

discontinuous patterns to model false positives, and [7] using frequent itemsets and association rules to model them. Paper [8] filters IDS alerts by monitoring alert flow. It models regularities in alert flows with classical time series methods. After removing the periodic components, slowly changed trend and random noise from time series, the remaining is regarded as true alerts. All methods mentioned above are based on modeling false positives by frequent or periodic alert sequences. It is effective in the environment where normal behaviors seldom change. However it often mistakes the new normal behaviors or infrequent ones for true alerts.

Our work differs from the above in that we use an unsupervised data mining technique that needs no domain knowledge and labeled training data. In addition, it needs little human assistance. So our model can overcome the shortages that classification or root cause analysis based methods have. In addition, our method can help managers find the root causes of false positives. Now only [1] can do this. Although our method is also based on frequent patterns, it is quite different from the third category of methods, because they are build on different assumptions. Our method assumes that alerts containing many frequent attribute values are likely resulted from normal behaviors. However the third category uses frequent alert sequences to identify these alerts. In other words, in our model, the frequent pattern is made up of alert attributes. But in the third category of methods, frequent pattern is constituted by alerts. Moreover current alert reduction methods such as [8] and [1, 3 and 4] can obtain true alerts only after removing all false positives or clustering all alerts. But our method can directly identify the outliers (true alerts) in larger amounts of data, so the cost wasted for processing false positives will be reduced and the reaction to attacks will be more rapid. In addition, our model can constantly update the feature set so that it is adaptive to new normal behaviors.

Alert correlation [9], which is a little similar to above work, is another kind of technique for analyzing intrusion alerts. However it mainly focuses on the reconstruction of attack scenarios, while our focus is identifying and reducing false positives.

## III. OUTLIER DETECTION-BASED ALERT PROCESS

Outlier detection is a new data mining technique which has absorbed many attentions recently. It is able to identify abnormal data (called *outlier*) in large dataset. Now in the field of network security, people have successfully used outlier detection to implement IDS [10, 11]. However it has not been applied in alert reduction. In fact, compared with false positives which are the majority of IDS alerts, true alerts (i.e. alerts related to attacks) are just so-called "outliers". So this technique can also be used to filter IDS alerts. Moreover, for its high efficiency, low cost and the unsupervised character, methods based on this technique can achieve good performance.

In this paper, we use a frequent pattern-based outlier detection technique (FP-Outlier) to filter false positives. We choose this algorithm because it can process the uneven IDS alerts (i.e. alerts are often the mixture of

several alert clusters, and the densities of these clusters are not equal). Moreover this algorithm can meet all requirements that Abdulrahman et al. proposed for alert reduction technique in [6] (i.e. high scalability, low cost, independent reduction process, and the capability of dealing with noisy data and any type of attributes contained in the alerts). In order to make this algorithm more fit for processing IDS alerts, we have improved it in two ways. In this section, we firstly introduce our improved algorithm, and then present an alert reduction and analysis framework based on it.

#### A. Improved FP-Outlier Algorithm

Frequent pattern-based outlier detection (i.e. FP-Outlier) is presented by Zengyou He et al. in [12]. This method is built on the following truth: given a set of supermarket transactions, where each transaction is a set of literals (called *items*). *Frequent itemsets* (also called *frequent patterns*) are those combinations of items that have transaction support above predefined minimum support (*support* means percentage of transactions containing these itemsets). If a transaction contains many frequent itemsets, it means that this transaction is unlikely to be an outlier because it possesses the “common features”. There are mainly 3 steps in this algorithm: Firstly, all frequent itemsets are found in the dataset by certain mining algorithm. Secondly the outlier score of each transaction is calculated based on these frequent patterns. Finally, transactions are sorted ascendingly according to their outlier scores, and the top p% of which are selected as candidate outliers. In the second step, the outlier score is measured by Frequent Pattern Outlier Factor (FPOF). Its definition is given as follows:

**Definition 1 (Frequent Pattern Outlier Factor)** Let  $D = \{t_1, \dots, t_n\}$  be a set of  $n$  transactions.  $Support(X)$  denotes the ratio between the number of transactions that contain itemset  $X$  and the number of transactions in  $D$ .  $minisupport$  is a user defined threshold. It defines the minimal support of frequent itemset. And  $FPS(D, minisupport)$  is a set of frequent patterns mined from  $D$  with  $minisupport$ . Given a transaction  $t$ , its Frequent Pattern Outlier Factor (FPOF) is calculated as follows:

$$FPOF(t) = \frac{\sum_{X \subseteq t, X \in FPS(D, minisupport)} support(X)}{|FPS(D, minisupport)|} \quad (1)$$

The key idea of FP-Outlier is using frequent itemsets as features to identify “normal” data. So the more frequent and repeated the normal behaviors are, the better this algorithm can work. After analyzing IDS alerts, we found that they just had above characters. For example, a Snort deployed in the network of our laboratory can produce more than 30,000 alerts a week, and 95% of them are 4 types, viz., “SNMP public access udp”, “SNMP request udp”, “(http\_inspect) BARE BYTE UNICODE ENCODING” and “NETBIOS SMB IPC\$ unicode share access”. Through carefully analyzing, we believe that they are all triggered by configuration problems. It means all of them are false positives. Thus it is obvious that if we mine frequent itemsets from IDS

alerts (each alert is regarded as a transaction, and each alert attribute is regarded as an item), the frequent patterns we get will mostly come from false positives. So these patterns can be regarded as features of false positives and used to filter them.

In order to make the FP-Outlier algorithm more fit for IDS alerts, we have improved it in two aspects.

Our first improvement is assigning each alert attribute a weight. That is because we have observed that when determining whether or not an alert is false positive, the attributes of this alert usually have different impact to the final decision. For example, there are two alerts. The first one contains frequent pattern “AlertType = SNMP public access udp”. And the second one contains frequent pattern “Destination Port = 80”. In general the first one is more likely to be a false positive than the second, because port 80 is common to both attacks and normal events. In order to show this difference, we should assign a higher weight to the “AlertType” attribute than to the “Destination Port” attribute.

In our model, the frequent pattern mined from IDS alerts is composed of alert attributes, so after assigning alert attributes different weights, frequent pattern will have a weight too. Its weight is defined as follows:

**Definition 2 (Weight of a Frequent Pattern)** Let  $A = \{A_1, \dots, A_n\}$  be a set of alert attributes name, each with an associated domain of value. The weight of  $A_i$  has been given as  $w_i$  ( $1 \leq i \leq n$ ). A frequent pattern  $X$  which contains  $A$  is a tuple over  $A$ . Then the weight of  $X$  is defined as:

$Weight(X) = \max\{w_i \mid w_i \text{ is the weight of } A_i, A_i \in A \text{ and } A_i \text{ is contained by } X, 1 \leq i \leq n\}$ .

In other words, the weight of a frequent pattern is equal to the largest weight of all alert attributes contained by it.

We have proved by experiments that assigning alert attributes different weights can really improve performance (re. section IV.A). And currently in our prototype, the weight of alert attributes is set by a semi-automated method, which includes following steps: Firstly, we sampled  $n$  alerts randomly from all IDS alerts (in our experiment,  $n = 30$ ), and manually classified them into true alerts and false positives. Secondly, for each alert attributes  $i$ , all frequent values of it were found, and the total number (recorded as  $S_i$ ) of frequent values that appeared in both false positives and true ones was calculated. After that, the weight of attribute  $i$  (i.e.  $weight_i$ ) was calculated according to formula 2:

$$weight_i = 1 - \frac{S_i}{\text{the total number of frequent values of attribute } i} \quad (2)$$

Finally, above steps were repeated for several times, and the weight of each attribute was set to the mean of the results we got in each round. The weight value can also be designed by an automated method, such as the technique frequently used in information retrieval. For example the formula proposed by A. Leuski in [13] can be used to calculate the weight value automatically. But the computational cost of this formula is quite high. So we are still looking for better automated method now.

Our second improvement to FP-Outlier is using dynamic feature set to describe normal behaviors (i.e. false positives). With the arriving of new alerts, new type of normal behavior also emerges. So in order to keep the feature set up to date, the frequent pattern describing new behaviors should be constantly added to features set. This improvement will cause the value of  $\|FPS(D, minisupport)\|$  in formula 1 also change constantly. Then the outlier score of alerts in different period will become incomparable. For example, with the growth of alert set, the number of frequent patterns will also increase. Then even for the same alert, the outlier score calculated in the early time will be bigger than the one calculated later. It is obviously inappropriate. In order to solve this problem, we removed  $\|FPS(D, minisupport)\|$  from our calculating of outlier score. This change will not violate FP-Outlier's basic idea (i.e. determine the outlier degree of an object by how many frequent pattern it contains). So it is the needless cost for us in deed.

According to above improvements, we propose our algorithm for identifying false positives. It can be described as following.

Firstly, we assign each alert a False Positive Score (FP score) which represents the possibility that this alert is a false positive. This score is measured by Common Feature Factor (CFF). Its definition is:

**Definition 3 (Common Feature Factor)** Let  $D = \{a_1, \dots, a_n\}$  be a set of alerts, and each attribute of alert in  $D$  has a predefined weight. Given  $minisupport$ , the minimal support of frequent itemset,  $FPS(D, minisupport)$  is a set of frequent itemsets mined from  $D$  with  $minisupport$ .  $Support(X)$  denotes the ratio between the number of alerts that contain itemset  $X$  and the number of alerts in  $D$ . And  $weight(X)$  is the weight of  $X$ . Then for each alert  $a$ , its Common Feature Factor (CFF) can be calculated as follows:

$$CFF(a) = \sum_{X \subseteq a, X \in FPS(D, minisupport)} support(X) \times weight(X) \cdot (3)$$

Then base on this formula, we present an algorithm CalculateFPScore to calculate each alert's FP score. Following is the pseudo code of this algorithm.

```

Algorithm CalculateFPScore
Input: FS // the feature set of false positives
        alert // the alert to be processed
Output: FPScore // indicates the possibility that
        an alert is a false positive

01 begin
02 FPScore = 0
03 for each frequent itemset X in FS do begin
04   if alert contains X then
05     FPScore = FPScore + weight(X)*support(X)
06   end if
07 end
08 return FPScore
09 end
    
```

Figure 1. Algorithm for Calculate Each Alert's FP Score

*B. Framework for Alert Reduction and Analysis*

This section gives a detailed introduction on how to use the improved algorithm to filter false positives. The original FP-Outlier algorithms are only designed for off-line process. The main shortcoming of off-line process is that attack-related alerts can't be presented to users immediately and the reactions to them are often delayed. So we design a real-time framework to solve this problem. This framework consists of 3 phrases: the learning phrase and the online filtering phrase (shown in figure 2) are used to filter alerts in real time. And the third phrase, phrase for discovering root causes, is used to help managers analyze causes of false positives.

**Learning Phrase.** The main work in this phrase is building the feature set of false positives and calculating the threshold of true alerts. Feature set is composed of frequent itemsets mined from IDS alerts. These itemsets, which are the combinations of attribute values, are regarded as the features of false positives and used to identify them. The threshold of true alert is used to tell true from false positives. Alerts whose FP scores bigger than this threshold are treated as false positives, otherwise true. In this paper, the threshold is equal to the largest FP score of candidate true alerts.

There are two difficulties in this phrase: Firstly, new type of false positives continually emerges, so it is difficult to determine how many data are enough for building the feature set. Secondly, not all frequent patterns mined from IDS alerts can be used as features. That's because some attacks (called *frequent attack* in the rest of paper) can also trigger many similar IDS alerts (e.g. some probing or DoS attacks). Frequent patterns mined from these alerts are obviously not features of false positives.

In order to address the first problem, we design a block-by-block mining method. At first, the alerts sequence is divided into several blocks. Each block contains n alerts which have neighboring timestamps. Then we mine frequent patterns in every block, and the patterns found are put into feature set. If the next two or three blocks all can not contribute new features, the feature set will be regarded as stable and we can stop collecting new alerts. Because some itemsets (called *global frequent itemset* in the rest of paper) are only frequent in a long run, we have to combine all blocks and mine them to find this kind of pattern in the end. It should be noticed that the supporting instances of frequent patterns are all more than one predefined minimal value.

As to the second problem, we designed an algorithm to automatically filter this kind of pattern. We have observed that alerts triggered by frequent attack usually occur many times in a short period, and they often have the same destination or source address. But false positives usually don't have these characters. So this is helpful to identify them. In our filtering algorithm, we designed two statistics: the first one is the number of certain type of alerts from the same source IP in the last T seconds, and the second one is the number of certain type of alerts to the same destination IP in the last T seconds. We calculate the two statistics for each alert type in the

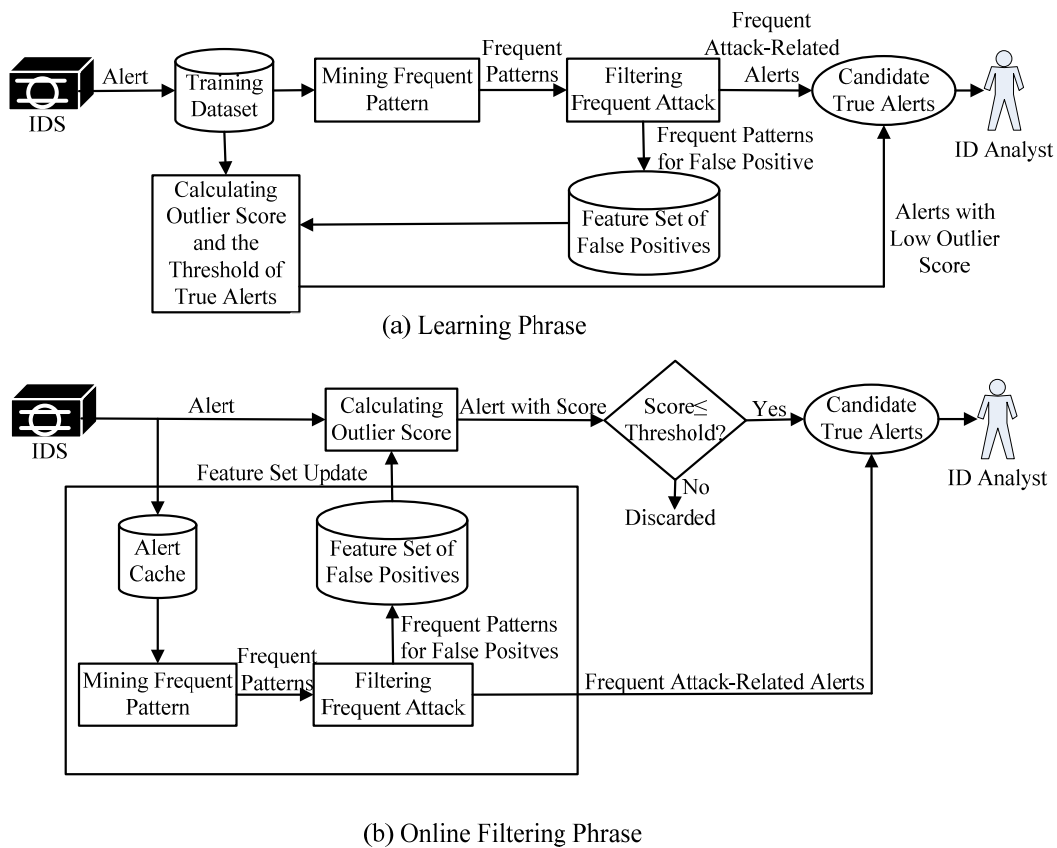


Figure 2. Real-time Framework for Alert Reduction

frequent itemsets. If any statistic is large than predefined threshold (e.g. in DARPA 2000, we define this threshold is 15 alerts per minute), this type of alerts will be regarded as triggered by frequent attack.

Detailed process in learning phrase is presented as follows (shown in figure 2):

*Step1, Constructing Feature Set.* Collect n alerts in the order of arrival as one block and mine them for frequent patterns. Add new patterns into the feature set, and then collect next n alerts. Repeat above process until the next k blocks (we choose  $k = 3$ ) all contributes no new patterns. Combine all blocks into one set and mine the whole dataset for global frequent itemsets. Put these patterns into feature set. In our system, we use Apriori algorithm, which is a famous data mining algorithm, to mine frequent itemset. The detail of this algorithm can be found in paper [14].

*Step2, Filtering Frequent Attacks.* Check all alert types that appear in feature set, and determine whether or not they are frequent attacks. For those frequent attacks, the FP score of related alerts are set to 0, and related frequent patterns are removed from the feature set.

*Step3, Calculating the Threshold of True Alerts.* Based on the feature set obtained in above steps, use CalculateFPScore algorithm to calculate the FP score of each remaining alert. Alerts are sorted ascendingly according to their score and top p% of them is put into the set of candidate true alerts. Set the threshold of true alerts to the largest score in the set of candidate true

alerts. And after all alerts in this set are recommended to users, the set of candidate true alerts is reset to empty.

**Online Filtering Phrase.** The main work in this phrase is calculating the FP score for each newcomer alert, and then comparing this score with the threshold of true alerts, so as to determine it is true or false. Besides this, the feature set is updated continually in order to keep its accuracy. Detailed process in this phrase is presented as follows (shown in figure 2):

*Step1, Filtering Each Newcome Alert.* When a new alert come, it is firstly put into the alert cache, then its FP score is calculated based on the feature set by CalculateFPScore algorithm. If the score is not bigger than the threshold, this alert is regarded as true and is recommended to users, otherwise it is discarded.

*Step2, Updating the Feature Set.* As soon as the account of alerts in cache is equal to n, a frequent itemset mining is executed on these alerts. If the result contain some frequent patterns that the feature set hasn't, filter frequent attacks from these new patterns (similar to step2 in the learning phrase) and the remaining are added into the feature set. As to the patterns that the feature set already has, we only need to update the support, i.e., their supports in feature set are set to the mean of new support and the old one. All frequent attack-related alerts are put into the set of candidate true alerts and then recommended to users. Finally, store all alerts in cache into alert database, and then clear up the cache.

We should mention that with the coming of new alerts, the threshold calculated based on the early feature set

will become inaccurate gradually. Moreover, although the feature set is updated continually, some global patterns are still possibly missed. So it is necessary to mine the whole alert databases so as to adjust the threshold and the support of frequent patterns. This process can be done at comparatively long intervals. And in order to avoid affecting the real-time filtering, it had better be executed in background offline.

**Phrase for Discovering Root Causes.** The main work in this phrase is using the feature set obtained before to help managers find the root causes of false positives. After careful observation, we found that the frequent patterns in the feature set not only can be used to identify false positives, but also can disclose the general rules of normal behaviors that trigger false positives. And these rules are very useful for understanding the root causes of these false positives. For example, table I shows part of feature set obtained after mining 4000 true IDS alerts (come from the Snort deployed on a proxy of our laboratory). From it, we can know that there are totally 887 alerts whose type is “SNMP public access udp” in the 4000 alerts. Moreover the 887 alerts all aims at the 161 port of site 202.119.36.253. As we known, site 202.119.36.253 is a printer in fact. It is a HP Laser printer shared by the whole laboratory through intranet. Considering this, we can conclude that these alerts are triggered by normal print tasks and are likely to be false positives. Then after careful analysis, we found the root causes of these alerts are that the client of HP Printer needs to use SNMP to identify the extended state of printer. So our guess is right.

TABLE I.  
PART OF FEATURE SET FROM TRUE ALERTS

Frequent Itemset	Support Instance
...	...
AlertType=SNMP public access udp	887
AlertType=SNMP public access udp Destination IP=202.119.36.253	887
AlertType=SNMP public access udp Destination Port=161	887
AlertType=SNMP public access udp Destination IP=202.119.36.253 Destination Port=161	887
...	...

Now our model directly presents the feature set to users, and some mechanism such as querying and sorting interfaces are also provided so as to help analysis. The root causes of false positives still can not be automatically found currently. However based on the feature set and before-mentioned mechanisms, the analyzing task will become easier. And in the future, we will try to develop an automatic way to find root causes.

#### IV. EXPERIMENTS

We have built a prototype implementation of our alert reduction method using the Weka framework [15]. The prototype has been validated on DARPA 2000 dataset.

And in order to assess the performance of our method in a real setting, we also applied it to real network data from our laboratory. We summarize the results obtained in this section. Before presenting the results, we will introduce the measures used in our experiments firstly. They are reduction rate, completeness and soundness. Their definitions are given as follows:

**Definition 4 (Reduction Rate)** Let  $N$  be the total number of alerts and  $N_f$  be the number of alerts filtered by the systems. The reduction rate  $R_r$ , which assesses how many alerts can be filtered by the system, is defined as the ratio between  $N_f$  and  $N$ , i.e.,

$$R_r = \frac{N_f}{N}. \tag{4}$$

**Definition 5 (Completeness)** Let  $td$  be the number of true alerts correctly detected by the system, and  $tm$  be the number of true alerts missed by the system. The completeness  $R_c$ , which evaluates how well the system can detect true alerts, is defined as the ratio between  $td$  and the total number of true alerts, i.e.,

$$R_c = \frac{td}{td + tm}. \tag{5}$$

**Definition 6 (Soundness)** Let  $td$  be the number of true alerts correctly detected by the system,  $N$  be the total number of alerts and  $N_f$  be the number of alerts filtered by the systems. Then  $N - N_f$  means the number of candidate true alerts. The soundness  $R_s$ , which measures how correctly the alerts are recommended, is defined as the ratio between  $td$  and the total number of candidate true alerts recommended by the system, i.e.,

$$R_s = \frac{td}{N - N_f}. \tag{6}$$

##### A. Results Obtained from DARPA 2000 Data Set

DARPA 2000 is a famous synthetic data set for intrusion detection-related tests. It includes two scenarios: LLDOS 1.0 and LLDOS 2.0.2. Because the purpose of our experiment on DARPA 2000 is just verifying the feasibility and studying how to set the parameters, we did not use too many data so as to simplify our implementation. The dataset we used is phrase 2 to phrase 4 from DMZ in LLDOS 1.0. On this dataset, the IDS (we use RealSecure) can produce 886 alerts, which include 51 true alerts and 835 false positives. There are 45 attributes in each alert. And in our alert reduction method, we mainly use 5 of them for analysis. They are “alert type”, “source IP”, “source port”, “destination IP” and “destination port”.

Our experiment on DARPA 2000 is composed by two parts: Firstly we studied the effect of main parameters (including  $p$ , minisupport and weights of alert attributes) used in our model so as to choose appropriate values for them. And then we test how well our real-time framework work.

##### Experiments on Different Parameters.

*Effect of p.* Parameter  $p$  denotes the proportion of candidate true alerts in the whole dataset (re. Section III.B). It is mainly used in the learning phrase and determines the threshold of true alerts. In order to study how  $p$  influences the performance of our method, we change the value of  $p$  and obtain two groups of data. In this experiment, the weights of alert attributes are set to  $\{1, 0.6, 0.4, 0.8, 0.2\}$  (they are respectively the weights of “alert type”, “source IP”, “source port”, “destination IP”, and “destination port”), and the minisupport is set to 5%. Figure 3 shows the result:

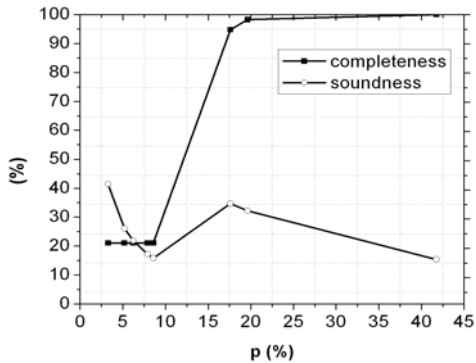


Figure 3. Note how the caption is centered in the column.

From figure 3, we can see that it is difficult to achieve both high completeness and good soundness. Increasing of  $p$  leads to higher completeness, but the soundness often decreases too. In alert reduction systems, missing true alerts is usually more serious than incorrectly recommending false positives to users, so completeness is more important. In other words, the value of  $p$  should firstly ensure high completeness. Then on the base of it, the soundness and reduction rate should also be comparatively high. So according to figure 3, we think that the value between 15% and 20% is appropriate for  $p$ .

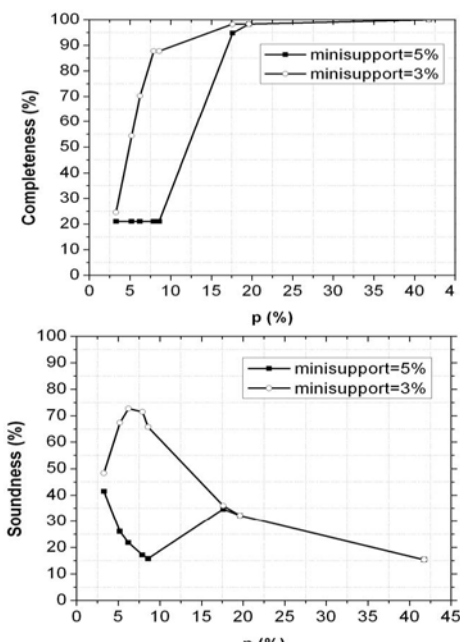


Figure 4. Performance with Different Value of minisupport

*Effect of minisupport.* Parameter minisupport defines the minimal value of support for frequent patterns mining. Small minisupport will bring more frequent patterns, but the possibility that the patterns are mined from frequent attacks also grows. In order to study what value of minisupport is appropriate, we set it respectively to 5% and 3%. When minisupport was equal to 5%, the frequent patterns mined from the dataset did not contain any frequent attacks, while equal to 3%, they contained a frequent attack ‘Sadmind Amslverify Overflow’.

The final result of this experiment is shown in figure 4 (In this experiment, the weight is still  $\{1, 0.6, 0.4, 0.8, 0.2\}$ ). From it, we can see that when minisupport is 3%, our method has better completeness and soundness. We believe that is because the feature set of false positives in this situation is more precise than that when minisupport is 5% (It should be noticed that this precision is based on removing all unsuitable features, i.e., patterns from frequent attacks).

*Effect of Alert Attribute’s Weight.* In order to verify our conclusion on the weight of alert attribute (re. Section III.A), we assigned three groups of weights for alerts and observed the corresponding results. In the first group, all alert attributes had the same weight values (i.e.1). In the second group, the weights of “alert type”, “destination IP”, “source IP”, “source port”, and “destination port” were respectively 1, 0.8, 0.6, 0.4, and 0.2. And in the third group, the weight of “alert type” was 1, the weight of “destination IP” and “source IP” were both 0.5, and the weight of “source port” and “destination port” were both 0.1. Then we set minisupport to 3%.

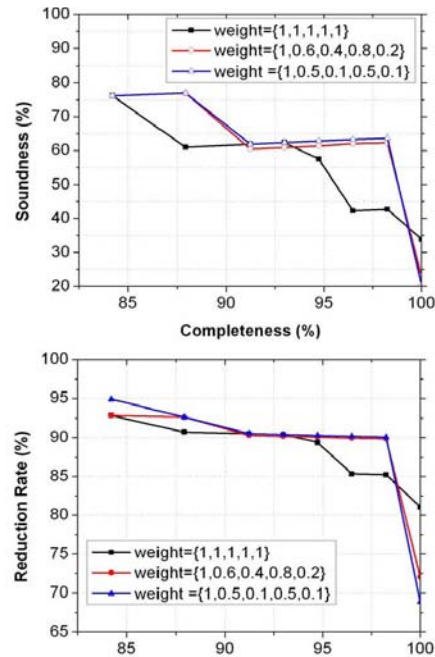


Figure 5. Performance with Different Attribute Weights

The result of this experiment (shown in figure 5) exactly proved our assumption. From it, we can see that after setting different weights for alert attributes according to their importance, both soundness and

reduction rate of our model are improved to some extent. And we are also surprised to find when the weights are respectively set to {1, 0.6, 0.4, 0.8, 0.2} and {1, 0.5, 0.1, 0.5, 0.1}, the results obtained are very similar. We think that is caused by the characters of data in DARPA 2000. It also means our model is not very sensitive to parameters. And slight inaccuracy of the parameter's value does not influence the result too much. This will make our model more practical.

**Efficiency of Our Real-time Framework.**

Based on above result, we also test the efficiency of our real-time framework on DARPA 2000. In the learning phrase, we firstly regarded every 100 alerts as one block in the order of arrival. Then we mined each block for frequent patterns. After mining 4 blocks, the feature set of false positives was stable, so we stop collecting training data. 12 frequent patterns can be found after mining these 400 alerts. Because there is no frequent attack in them, all patterns were regarded as features. Then we set weights to {1, 0.6, 0.4, 0.8, 0.2}, and calculated the FP score for these alerts. After sorting alerts ascendingly by their FP scores, we found that the maximal score in the top 15% alerts is 0.054. So the threshold of true alerts is set to 0.054.

In the online filtering phrase, the remaining 485 alerts were input to our prototype one by one. We mined every 100 alerts so that the new feature of false positives can be found and added into the feature set in time. In our experiment, new patterns can be found only when mining the first 100 alerts. Among these patterns (i.e., "Alert Type = Admind", "Alert Type = Sadmin\_Amslverify\_Overflow" and "Source IP = 202.077.162.213"), only the alerts that type of "Sadmin\_Amslverify\_Overflow" denoted a frequent attack, so the corresponding patterns (i.e., "Alert Type = Sadmin\_Amslverify\_Overflow") can not be added into the feature set.

After 485 alerts had been processed, 70 alerts were recommended to users as true alerts. The reduction rate is 86%. Because 50 alerts among them were really true, and there are totally 51 true alerts in the dataset, the completeness of our prototype is 98%, and the soundness is 71%. However, after analyzing all recommended alerts, we found that the only true alert missed by our prototype was a duplicated one. In fact, when RealSecure detects an intruder breaks into a host by telnet, it will generate 3 types of alerts for this single attack step, i.e., "TelnetEnvAll", "TelnetXdisplay" and "TelnetTerminaltype". And we only missed the alert that type of "TelnetTerminaltype". Users can still identify this attack step by the other two alerts. So the completeness of our prototype can be regarded as 100% considering this.

**B. Results Obtained from Real Data**

Due to various limitations of synthetic dataset [16], we have repeated our experiments on real-world IDS alerts. This dataset is collected over the period of 2 weeks in the network of our laboratory. The network includes 10 hosts, which connect to the Internet through one proxy server. A network-based IDS (snort) recorded information exchanged between the Internet and the intranet. There

are totally 65528 alerts in this dataset, 95% of which are 4 types of alerts, i.e., "SNMP public access udp", "SNMP request udp", "(http\_inspect) BARE BYTE UNICODE ENCODING" and "NETBIOS SMB IPCS unicode share access".

In this experiment, we regarded 1000 alerts as one block, and used the same set of parameters as for DARPA 2000. By executing similar steps as in the previous paragraph, we obtained 5258 candidate true alerts finally, and the reduction rate is 92%. After thoroughly analyzing, we found that all alerts filtered by our prototype were false positives, so the completeness is 100%. Among the true alerts recommended by our prototype, the account of alerts triggered by real attacks was 2620, so the soundness is 50%. By comparing with the result obtained from DARPA 2000, we found that our model has higher reduction rate but lower soundness on real dataset. The likely explanation of this fact is that real dataset contains fewer intrusions and more redundancy than DARPA 2000. We believe that if we fit the parameters to the real data, the performance of our prototype can be even high.

**C. Comparison with current methods**

In order to show the performance of our method, we have also compared it with other two typical alert reduction systems. The first one is the system presented by Klaus Julisch in [1, 3 and 4] (recorded as "Klaus's method" in table II). It can identify false positives based on root cause analysis. And the second one is a system presented by Abdulrahman Alharby et al [6] (recorded as "Alharby's method" in table II). It processes alerts based on continuous and discontinuous patterns and can filter false positives in real time. The result of comparison is shown in table II.

TABLE II.  
COMPARISON BETWEEN OUR PROTOTYPE AND OTHER METHODS

	Reduction Rate	Completeness	Time Consumption	Real Time?
Our method	86%-92%	100%	Low	Yes
Klaus's method	87%	100%	High	No
Alharby's method	80%	100%	Middle	Yes

From above table, we can find that the reduction rate of our method is a little better than Klaus's method. However Klaus's method is not able to filter alerts in real time. It can distinguish true alerts and false ones only after clustering all alerts, so its time consumption is high. Alharby's method has the capability of real-time filtering, but its reduction rate is far lower than ours. Moreover, this method needs a lot of labeled data to build its model and can not filter alerts in training phrase, while our method does not have these limits. So using our method, security managers can response to attacks more quickly. In another word, the time consumption of our method is lowest. From above comparison, we believe that our system has better performance than current methods.

#### D. Discussion on our method

**About the measures we used.** It should be noticed that almost all of current alert reduction systems did not consider the soundness of their filtering results. However soundness is an important measure, because low soundness will make users feel that the alert reduction system is unworthy of belief. And it is useful especially in the environment where the amount of alerts is large. In this situation, the alerts recommended to users are usually numerous. If the soundness is low, the recommendation that contains a lot of false positives will be useless to users. Considering this, we evaluated our prototype by this measure. On DARPA 2000 dataset, the soundness of our prototype is 71%. It is well but not perfect enough. And on real dataset, its soundness is even low, so we will do more research on improving it. In fact, the soundness of our model is mainly influenced by two factors. Firstly, the more accurate the feature set is, the higher soundness we can obtain (See in Fig 4, lower minisupport leads more features and higher soundness). In addition, it can also be affected by how exact the FP score is (See in Fig 5, different weights cause different score and different soundness). So our improvement in the future will mainly focus on these two aspects.

**About the method for setting weights.** In our prototype, we calculated the value of weight by random sampling. This method has several advantages, for example, it is easy to implement and the weights it produced are very fit for current dataset. However, this method needs manual analysis of sample alerts. Fortunately, the amount of samples is little, so it is still better than the alert reduction methods that need a lot of labeled training data (e.g. method based on classification). Moreover, the calculation of weight value needs not be repeated for every dataset. In fact, the weights calculated by this method can be regarded as reference values of other dataset. And we have proved in section IV.B that even on different dataset, we can also get well result by the same parameter values.

**About how can our framework achieve “real-time”.** Our framework can process alerts in real-time, i.e. when a new alert coming, the framework can determine whether it is false positive or not almost immediately. This is implemented by following steps (re. section III.B): Firstly, based on a feature set of false positives, our framework calculates the FP score for each new alert. Secondly, the FP score is compared with the threshold of true alerts, so as to determine this alert is true or false. In order to finish above steps, a feature set, which can describe false positives accurately, and a threshold, which can distinguish true alerts with false one, have to be built in advance. These are just what the learning phrase of our framework needs to do. As far as the algorithm for calculating the FP score, before-mentioned experiments have proved that our algorithm is both accurate and with low time consumption. It is also an important pre-condition for real-time process.

**About the adaptability of our model.** In practice, the behaviors of attackers and normal users are both

frequently changed. So the alert reduction system must be adaptive to this change, i.e. it should be able to identify the false positives triggered by new normal behaviors, and avoid to mistakenly filter out true alerts caused by new attacks. Our model has done some efforts in this aspect.

In order to identify new false positives, we designed a dynamic feature set (re. section III.A), i.e. with the arriving of new alerts, the new frequent patterns which represent new false positives are continually mined and put into the feature set, so that the accuracy of feature set can be kept. The experiment shown before has proved the effectiveness of this method. However it still has some limitations. For example, only when the amount of false positives triggered by new normal behaviors is larger than a predefined threshold, can their feature patterns be discovered. In order to identify these false positives more rapidly, we can choose low threshold. But this will possibly bring unrelated frequent patterns into feature set. So a trade-off has to be found between them.

As far as the change of attackers' behaviors, we believe that it has little effect on our model. Because our model is based on profiling false positives by frequent attribute values. And any alert (no matter it has new type or not) which does not possess the features of false positives will be regarded as true alert. This is similar to the idea of anomaly detection in the field of IDS. In other words, new or unknown attacks can be identified by our model unless they have the same features with false positives. However even in the later situation, our model can also identify most of them by the algorithms for filtering frequent attacks (re. section III.B).

#### V. CONCLUSIONS

In this paper, we present a novel alert reduction method based on a new data mining technique, i.e., outlier detection. And we also build a real-time framework to filter false IDS alerts using this method. Now we have finished the prototype implementation of our model. And through the experiments on DARPA 2000, we have proved that when 86% of alerts have been filtered by our model, 100% of true alerts still remain. And on real-world dataset, our model has even higher reduction rate. By comparing with current other alert reduction methods, we believe that our model has better performance. However, our method also has some shortages, for example, now its soundness is still not perfect. In the future, we will study how to improve the soundness of our model. In addition, we will look for low-cost automatic method for setting weights and do more experiments on real-world IDS alerts so as to know how our model works with alerts generated by multiple IDSs.

#### ACKNOWLEDGMENT

This work is supported in part by the High Technology Research Project of Jiangsu Province of China under Grant No.BG2005029, and in part by the

Natural Science Foundation of Jiangsu Province under Grant No.BK2007136.

#### REFERENCES

- [1] K. Julisch, M. Dacier, "Mining Intrusion Detection Alarms for Actionable Knowledge", *Proceedings of KDD'02*, ACM Press, New York, pp. 366-375, 2002.
- [4] K. Julisch, "Clustering Intrusion Detection Alarms to Support Root Cause Analysis", *ACM Transactions on Information and System Security*, 6(4), pp. 443-471, 2003.
- [5] C. Clifton, G. Gengo, "Developing Custom Intrusion Detection Filters Using Data Mining", *Proceedings of MILCOM 2000*, IEEE Press, New York, pp. 440-443, 2000.
- [6] A. Alharby, H. Imai, "IDS False Alarm Reduction Using Continuous and Discontinuous Patterns". *Proceedings of ACNS 2005*, Springer, Heidelberg, pp. 192-205, 2005.
- [7] S. Manganaris, M. Christensen, D. Zerkle, et al, "A Data Mining Analysis of RTID Alarms", *Computer Networks*, 34(4), pp. 571-577, 2000.
- [8] J. Viinikka, H. Debar, L. Mé, et al, "Time Series Modeling for IDS Alert Management", *Proceedings of AsiaCCS'06*, ACM Press, New York, pp. 102-113, 2006.
- [9] P. Ning, Y. Cui, D. Reeves, et al, "Tools and Techniques for Analyzing Intrusion Alerts", *ACM Transactions on Information and System Security*, 7(2), pages 273-318, 2004.
- [10] L. Ertöz, E. Eilertson, A. Lazarevic, et al, "Detection of Novel Network Attacks Using Data Mining", *Proceedings of DMSEC 2003*, IEEE Press, New York, pp. 1-10, 2003.
- [11] P. Dokas, L. Ertöz, V. Kumar, et al, "Data Mining for Network Intrusion Detection", *Proceedings of NSF Workshop on Next Generation Data Mining*, AAAI/MIT Press, Cambridge, pp. 21-30, 2002.
- [12] Z. He, X. Xu, J.Z. Huang, et al, "FP-Outlier: Frequent Pattern Based Outlier Detection", *Computer Science and Information System*, 2(1), pp. 103-118, 2005.
- [13] A. Leuski, "Evaluation Document Clustering of Interactive Information Retrieval", *Proceedings of ACM CIKM'01*, ACM Press, New York, pp. 33-40, 2001.
- [14] R. Agrawal, R Srikant, "Fast Algorithms for Mining Association Rules", *Proceedings of VLDB'94*, Morgan Kaufmann Publishers, San Francisco, pp.487-499, 1994.
- [2] T. Pietraszek, "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection", *Proceedings of RAID'04*, Springer, Heidelberg, pp. 102-124, 2004.
- [3] K. Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency", *Proceedings of ACSAC'01*, IEEE Press, New York, pp. 12-21, 2001.
- [15] I.H. Witten, E Frank, *Data Mining: Practical Machine Learning Tools with Java Implementations*. Morgan Kaufmann, San Francisco, 2000.
- [16] J. McHugh, "The 1998 Lincoln Laboratory IDS Evaluation (A Critique)", *Proceedings of RAID 2000*, Springer, Heidelberg, pp. 145-161, 2000.

**Fu Xiao**, born in Jiangsu Province, China and on 25th December, 1979. She is a Ph. D. candidate of Department of Computer Science and Technology, Nanjing University, Nanjing, China. Her main research interests include network security and machine learning. And she has published several papers in the areas of network security and machine learning.

**Shi Jin**, born in Anhui, China and on 4th July, 1976, and received the Ph.D. degree in computer science and technology from the Nanjing University, Nanjing, China, in 2008. He is an Assistant Professor in the School of State Secrecy and Department of Information Management, Nanjing University, where his research focuses on network security, privacy protection. He published over 20 papers in the areas of network security and privacy protection.

**Xie Li**, born in 1942, professor and Ph. D. Supervisor of Computer Science and Technology, Nanjing University, Nanjing, China. His current research interests include distributed computing and advanced operation system.