Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering

Risto Vaarandi and Kārlis Podiņš

Reprinted from *Proceedings of the 2010 IEEE Conference on Network and Service Management* (ISBN: 978-1-4244-8908-4)

Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering

Risto Vaarandi and Kārlis Podiņš
Cooperative Cyber Defence Centre of Excellence
Tallinn, Estonia
firstname.lastname@ccdcoe.org

Abstract—**Network IDS is a well-known security measure for network monitoring and protection. Unfortunately, IDSs are known to generate large amounts of alerts, with many of them being either false positives or of low importance. This makes it hard for the human to spot alerts which need more attention. In order to tackle this issue, this paper proposes an IDS alert classification method which is based on data mining techniques.**

Keywords-alert classification; intrusion detection; data mining

I. Introduction

Network IDS (intrusion detection system) is a well-known security measure for network monitoring and protection. Network IDS monitors network segments for malicious or other unwanted traffic and produces alerts when such traffic is observed. Many network IDSs use the signature based approach for detecting unwanted traffic – IDS sensors are equipped with human written rules (signatures) which describe bad network packets. Despite IDS solutions have been used for more than a decade, an important problem is still not fully addressed – IDS can produce a large number of alerts which are overwhelming to the human. For example, a single IDS sensor can generate tens of thousands of alerts in a day [1, 2]. Furthermore, often vast majority of the alerts are false positives or of low importance [2–7].

For these reasons, IDS alert processing techniques have been extensively researched [1–15]. Among proposed methods, data mining based approaches have been frequently suggested during the past decade [2–10]. With these approaches, IDS alert logs from the recent past are mined for previously unknown regularities and irregularities, and the detected knowledge is then used by the human for developing alert correlation rules (such as filters for false positives). Unfortunately, this inherently semi-automated procedure is expensive, since the domain expert has to interpret the knowledge and write correlation rules by hand. Moreover, since most IT environments are constantly changing, this procedure has to be carried out regularly which further increases its cost [2, 5].

In this paper, we extend our previous work on IDS alert classification [2], and present a novel unsupervised real time alert classification method which is based on frequent itemset mining and data clustering techniques. Unlike other data mining based approaches, our method fully automates the process of knowledge interpretation and construction of alert filtering rules. Furthermore, due to its unsupervised and automated nature, the method does not need human-labeled training data and is able to adjust to environment changes without a human intervention.

This work was supported by the SEB financial group. This paper is a product of the authors; it does not represent the opinions or official policies of the CCD CoE or NATO and is designed to provide an independent position.

The remainder of this paper is organized as follows – section II presents an overview of related work, section III provides an in-depth discussion of our alert classification method, section IV describes experiments for evaluating the performance of the method, and section V concludes the paper.

II. Related Work

During the last decade, IDS alert processing techniques have received considerable amount of attention in the research community. Pietraszek has proposed machine learning methods for IDS alert classification, in order to reduce the amount of false positives [13]. Viinikka et al. have suggested the use of time series modeling for modeling regularities in large alert volumes [1, 12]. Other proposed methods include the application of EWMA control charts for monitoring IDS alerts produced by verbose signatures [11], the application of the chronicles formalism for temporal alert correlation [14], and graph-based correlation methods [15]. For IDS alert log mining, a number of approaches have been proposed. Treinen and Thurimella have investigated the application of association rule mining for the detection of rules for novel attack types [6]. Clifton and Gengo have used a similar approach for building IDS alert filters [8]. Long, Schwartz and Stoecklin have suggested a supervised clustering algorithm for distinguishing Snort IDS true alerts from false positives [7]. Julisch and Dacier have proposed a conceptual clustering technique for IDS alert logs [3–5]. With this approach, detected clusters correspond to alert descriptions, and the human expert can use them for developing filtering and correlation rules for future IDS alerts. During their experiments, Julisch and Dacier found that these hand written rules reduced the number of alerts by an average of 75% [4] and 87% [5]. Al-Mamory, Zhang and Abbas have suggested clustering algorithms for finding generalized alarms which help the human analyst to write filters [9, 10]. During the experiments, the number of alarms decreased by 93% [9] and 74% [10].

In our recent paper [2], we have proposed an IDS alert classification algorithm which distinguishes important alerts from redundant ones. Our algorithm first employs frequent itemset mining for detecting patterns that describe frequently occurring redundant alerts. The algorithm then extracts signature IDs from detected patterns and finds *frequent endpoint sets* which describe strong associations between alert attribute values for each ID. Detected sets will be used for restricting the matches produced by too generic patterns. Although this heuristic was found to be efficient [2], frequent endpoint sets don't capture all strong associations between alert attribute values. For instance, the heuristic makes an assumption that all associations for a given signature ID are of the same type (e.g., between values of transport protocol and destination port attributes), while this is not always the case. In the following section, we will describe a new classification method which addresses this issue.

III. IDS Alert Classification

In this section, we propose an IDS alert classification method which relies on frequent itemset mining and data clustering algorithms. Our method first applies a frequent itemset mining algorithm to past IDS alert logs, in order to discover patterns that describe redundant alerts. After that, data clustering techniques are used for finding fine-grained subpatterns for each detected pattern. Finally, the detected knowledge is interpreted and employed for real time classification of IDS alerts, in order to distinguish important alerts from irrelevant ones. The mining and interpretation steps are periodically repeated for keeping the classification knowledge up-to-date, thus the method is able to adjust to environment changes without a human intervention.

A. *Properties of IDS Alert Log Data*

Before providing a detailed description of our classification method, we will discuss special properties of IDS alert log data which the method relies on. We have observed these properties when analyzing the logs of three Snort IDS sensors of a large financial institution. Two sensors are deployed in the external network perimeter and one sensor in the intranet; all sensors are equipped with more than 15,000 signatures. The logs for three sensors covered a time frame of one year, and contained 55,484,833 alerts, 2,444,414 alerts, and 57,303,494 alerts, respectively.

First, we discovered that only a few signatures trigger most of the alerts – for three sensors described above, 10 most prolific signatures triggered 97.54%, 81.81%, and 95.53% of all alerts, respectively. During our past research, we have observed the same property [2], and experiments by other researchers have yielded similar results [1, 12]. Second, we discovered that prolific signatures have a tendency to trigger alerts over longer periods of time. For example, when we inspected 10 most prolific signatures for the three aforementioned IDS sensors, we found that for the first sensor, 9 signatures produced alerts for more than 170 days and 6 for more than 360 days. For the second sensor, 8 signatures produced alerts for more than 258 days and 7 for more than 344 days. For the third sensor, all 10 signatures produced alerts for more than 164 days and 6 for more than 351 days.

Third, when we inspected the signatures that trigger alerts frequently over long periods of time, we discovered that the number of such signatures is relatively small, yet they produce most of the alerts, with vast majority of them being either false positives or alerts of low importance. During one experiment, we divided the IDS logs into 365 slices, with each slice covering one day. We found that for the three IDS sensors described above, only 25, 17, and 20 signatures triggered alerts for at least 300 days, respectively. However, these signatures produced 90.94%, 79.68%, and 70.14% of all alerts, respectively, with the majority of them being irrelevant. Similar property has been observed by other researchers [1]. Table I lists data for top 5 signatures for one Internet and one intranet IDS sensor. The signatures from Table I trigger alerts which represent either well-known threats of low importance or false positives in the local environment. One of the well-known threats is the MS Slammer Sapphire worm which is detected by signatures 1:2003 and 1:2050. Other threats of low importance are very frequently occurring *ping* scans which are detected by signatures 1:483 and 1:469. As for false positive alerts, they are generated for routine network management traffic (signatures 1:2006779, 1:1419 and 1:480), for legitimate workstation traffic (signature 1:466), and for normal SSL and SSH traffic (signatures 1:8428 and 1:2001980). In the rest of this paper, we call such frequently occurring redundant alerts *routine alerts*.

Given the properties of IDS alert log data, it is obvious that IDS alert logs contain strong patterns in many environments, and these patterns often correspond to routine alerts. Therefore, the mining of frequent patterns from IDS alert logs yields valuable knowledge for alert classification. Since the output from different IDS sensors varies even within the same organization and is highly dependent on the sensor location, it often makes sense to mine patterns from each sensor log separately. For the sake of simplicity, we assume in the remainder of the paper that the IDS alert log is produced by a single sensor.

TABLE I. SIGNATURES THAT TRIGGER REDUNDANT ALERTS FREQUENTLY

Sensor location	Signature ID	Signature description	# of days	# of alerts
Internet	1:483	ICMP Ping Cyberkit 2.2 Windows	361	46,967,725
Internet	1:469	ICMP Ping Nmap	362	750,748
Internet	1:2050	SQL version overflow attempt	365	465,087
Internet	1:2003	SQL Worm propagation attempt	365	465,086
Internet	1:2006779	Nagios HTTP Monitoring Connection	365	288,520
Intranet	1:466	ICMP L3retreiver Ping	365	19,318,057
Intranet	1:1419	SNMP trap udp	365	14,922,302
Intranet	1:2001980	SSH Client Banner Detected on Unusual Port	359	1,909,080
Intranet	1:480	ICMP ping speedera	352	1,332,600
Intranet	1:8428	SSLv2 openssl get shared ciphers overflow attempt	354	729,523

B. *Frequent Itemset Mining for IDS Alert Logs*

We model the IDS alert log L as a finite sequence of alerts $L = (A^1, \ldots, A^m)$, and assume that each alert has attributes *no, time, ID, proto, srcIP, srcPort, destIP,* and *destPort*. If α is an attribute and A is an alert, A_α denotes the value of attribute α for alert A. If A is the k-th alert from L (i.e., $A = A^k$), we view it as a tuple $A = (A_{no}, A_{time}, A_{ID}, A_{proto}, A_{srcIP}, A_{srcPort}, A_{destIP}, A_{destPort})$, where A_{no} equals to k $(1 \leq A_{no} \leq m)$, A_{time} is the occurrence time of A, A_{ID} is the ID of the signature that produced A, A_{proto} is the network protocol for the traffic which triggered A, while A_{srcIP}, $A_{srcPort}$, A_{destIP}, and $A_{destPort}$ are the source IP address, source port, destination IP address, and destination port for the traffic which triggered A (if the network protocol does not involve ports, $A_{srcPort}$ and $A_{destPort}$ equal to the constant '-').

Let $I = \{i_1,\ldots,i_n\}$ be a set of items. If $X \subseteq I$, X is called an *itemset*. A *transaction* is a tuple *(tid, X)*, where *tid* is a transaction identifier and X is an itemset. A *transaction database D* is a set of transactions, and the *support* of an itemset X is the number of transactions that contain X: $supp(X) = |\{tid \mid (tid, Y) \in D, X \subseteq Y\}|$. If s is a support threshold and $supp(X) \geq s$, X is called a *frequent itemset*. If itemset X does not have any proper supersets with the same support, X is called a *closed itemset*. If D is a transaction database and s is a support threshold, $F_{closed}(D, s)$ denotes the set of all closed frequent itemsets for D and s. Note that the support threshold is often specified as a percentage $p\%$ which means that $s = |D|*p/100$.

During our past research, we have developed a tool called LogHound which implements an efficient frequent itemset mining algorithm for finding patterns from log files [16]. In order to apply LogHound to IDS alert log L, we have configured it to view each alert A from L as a transaction (A_{no}, X), where $X = \{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$. If L is viewed as a transaction database $D(L)$ in such a way, frequent closed itemsets from $D(L)$ correspond to IDS alert patterns. For example, the itemset *{(1:2003,1), (UDP,2), (1434,6)}* indicates that the MS Slammer Sapphire worm (detected by the signature 1:2003) frequently hits the destination port *1434/udp*. Note that this alert pattern can be also written as a string *1:2003 UDP * * * 1434*. In the remainder of this paper, we will often use this string notation for itemsets, and we will also use the terms pattern and itemset interchangeably.

If $A = (A_{no}, A_{time}, A_{ID}, A_{proto}, A_{srcIP}, A_{srcPort}, A_{destIP}, A_{destPort})$ is an alert and P is a pattern, we say that A *matches* P if $P \subseteq \{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$. If $\mathcal{A}$ is the set of all valid alerts that the IDS sensor can produce, the alert classifier is a function $f: \mathcal{A} \rightarrow \{0, 1\}$, where the function value 1 denotes "interesting" and 0 "routine".

The alert classifier could be defined in a trivial way – an alert is considered routine if some frequent closed pattern matches it, otherwise it is considered interesting. While building such a classifier is straightforward, it has several drawbacks. First, unusual and intensive short-term malicious network activity might trigger many uncommon alerts. If the number of such alerts is sufficiently large, patterns that reflect them would be detected as frequent, even if the alerts appear only in a small time frame of the log. Therefore, the classifier would mistakenly consider similar future alerts as routine. Second, IDS log mining often involves the over-generalization problem – some detected patterns could be too generic for alert classification [2, 4]. For example, the pattern * *TCP * * 10.2.1.17 25* would mistakenly label uncommon alerts from non-verbose signatures as "routine".

In order to address the first issue and handle the bursts of uncommon alerts, we split the IDS alert log into smaller slices, where each slice covers a time frame of equal length. Formally, if L is an IDS alert log, t is time, and d is a time interval, the *log slice* $L_{t,d}$ is defined as follows: $L_{t,d} = \{A \mid A \in L, t \leq A_{time} < t + d\}$. We then mine patterns from each slice, and a pattern will be used for classification only if it is discovered in many slices. The set of patterns selected for alert classification is denoted by F_C. Since only a few slices contain patterns for uncommon alerts, such patterns will be excluded from F_C (see Fig. 2 for details). Please see our previous paper [2] for a more thorough discussion of this technique.

C. *Data Clustering for Pattern Refinement*

As mentioned in the previous subsection, some frequent patterns are too generic for IDS alert classification. For example, suppose that only two Nagios network management servers 10.6.1.1 and 10.23.5.7 can issue legitimate monitoring queries to the web server 10.1.1.1, and that corresponding frequent patterns *1:2006779 TCP 10.6.1.1 * 10.1.1.1 80* and *1:2006779 TCP 10.23.5.7 * 10.1.1.1 80* are detected for all log slices. In that case, however, an overly generic pattern *1:2006779 TCP * * 10.1.1.1 80* will also be found frequent for all slices, since its support can't be smaller than the supports of two previous patterns.

For handling the over-generalization problem, we first exclude patterns without the signature ID information (e.g., the pattern * *TCP * * 10.1.1.1 80*) from further analysis and remove them from the set F_C. We will then analyze each remaining pattern in F_C, in order to find if the majority of alerts from L that match the pattern can be described with a moderate number of more specific subpatterns. If such subpatterns exist, they are used for improving the quality of alert classification – apart from matching a pattern P from F_C, an alert must also match one of the subpatterns of P, in order to be classified as "routine".

For the subpattern discovery, we are using the SLCT data clustering tool that has been developed previously by us for efficient analysis of large log files [17]. SLCT favors specific patterns over generic ones and thus suits well for the given task. SLCT divides log file entries into clusters, where entries from one cluster are similar to each other and are described with a certain pattern. Entries that don't fit to any of the detected clusters are arranged to a special *cluster of outliers*. When processing the log file, SLCT splits each entry into tokens. If the entry is a sequence of n tokens $w_1\, w_2 \ldots w_n$, SLCT considers the entry as a set of words: $\{(w_1,1),\ldots,(w_n,n)\}$ (i.e., each word is a token-position pair). In order to cluster the log file, SLCT first finds frequent words that occur at least c times in the data set, where c is the user given *clustering* *threshold*. SLCT will then make a second pass over the data set, in order to create cluster candidates – for each log file entry, all frequent words are extracted from it and the set of frequent words becomes a cluster candidate (i.e., if the entry contains m frequent words $(v_1,p_1),\dots,(v_m,p_m)$, the candidate is $\{(v_1,p_1),\dots,(v_m,p_m)\}$). For each candidate, SLCT maintains an occurrence counter, and after the data pass the candidates that have occurred at least c times will be selected as clusters.

We have configured SLCT to view each alert A as a set of six words $\{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$. It is easy to see that with this approach, each detected cluster has the same structure as an alert pattern, and thus we can define *alert A matches cluster C* in the following way: $C \subseteq \{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$. Note that if alerts that match a pattern P are clustered, then $P \subseteq C$ for each detected cluster C, since elements of P become words that are part of every alert. In other words, every alert that matches C must also match P, and therefore C acts as a more specific subpattern for P that improves the quality of alert classification.

When using SLCT for clustering alerts that match frequent patterns from F_C, the patterns themselves and the amount of matching alerts from L could vary a lot. Therefore, it is often impossible to find a common clustering threshold suitable for all patterns. In order to address this problem, SLCT is applied to each set of alerts several times with different thresholds, and after each clustering round the results are evaluated. After all rounds, the threshold which yields the best results is selected for clustering. We denote by $L(P)$ the set of alerts from L which match the pattern P from F_C, while by C_P we denote the best set of clusters for P. In order to evaluate the clustering results for $L(P)$, we define $G_{L(P),c}$ for a given $L(P)$ and threshold c as follows: $G_{L(P),c} = \min_{C' \in CA_{L(P),c}} |C'|$, where $CA_{L(P),c}$ is the set of all clusters for $L(P)$ and threshold c. Intuitively, larger values of $G_{L(P),c}$ indicate that detected clusters are more specific, since all clusters contain at least $G_{L(P),c}$ elements. We also denote the cluster of outliers for $L(P)$ and c by $O_{L(P),c}$, and find clusters for P with the *FindBestClustering* procedure from Fig. 1 (during our experiments, typical values for M and pct have ranged between 20...30 and 1...5, respectively). Informally, the procedure seeks for the threshold which produces a moderate number of clusters that cover the majority of alerts and are as specific as possible. If no such threshold is found, C_P is set to $\{ P \}$ which means that C_P will not restrict the matches produced by the pattern P (because for every alert that matches P there is always a matching cluster in C_P).

In order to address the over-generalization problem, the set C_P is detected for each pattern P from F_C. Unfortunately, although only a few signatures trigger routine alerts which create frequent patterns in logs, the set F_C could nevertheless become quite large if lower support thresholds are used during frequent itemset mining. In that case, there are often only a few fairly generic patterns $X_1,\dots,X_k$ present in F_C with many specific patterns $Y_1,\dots,Y_m$, so that each Y_j is a superset of some cluster C for a certain X_i: $\exists X_i \in F_C\ \exists C \in C_{X_i}, C \subseteq Y_j$. However, since $X_i \subseteq C$, then every alert A that matches Y_j (i.e., $Y_j \subseteq \{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$) must also match both X_i and C, and is thus classified as "routine" regardless of Y_j. Therefore, the pattern Y_j is redundant and can be excluded from F_C.

```
Procedure FindBestClustering

Input:
$P$ – pattern for which clusters are found
$L(P)$ – set of IDS alerts which is clustered
$seq = (c_1, ..., c_r)$ – sequence of clustering thresholds
$pct$ – max percentage of alerts in the cluster of outliers
$M$ – max number of clusters

Output: set of clusters $C_P$

$G := 0$
$c := -1$
for each $i := c_1...c_r$ do
  if ($|O_{L(P),i}| / |L(P)| \leq pct/100$ AND $|CA_{L(P),i}| \leq M$ AND $G_{L(P),i} \geq G$)
  then
    $G := G_{L(P),i}$
    $c := i$
  fi
done
if ($c = -1$) then $C_P := \{ P \}$ else $C_P := CA_{L(P),c}$ fi
return $C_P$
```

Figure 1. The procedure for finding the best set of clusters for a pattern.

```
Procedure BuildClassifier

Input:
$L$ – IDS alert log
$t_s$ – beginning of the time window for mining
$t_e$ – end of the time window for mining
$s$ – support threshold
$N$ – number of slices
$K$ – pattern relevance threshold ($1 \leq K \leq N$)
$seq$ – sequence of clustering thresholds
$pct$ – max percentage of alerts in the cluster of outliers
$M$ – max number of clusters

Output: definition of the alert classifier $f: \mathcal{A} \rightarrow \{0, 1\}$

$d := (t_e - t_s) / N$
for each $i = 1...N$ do $F_i := F_{closed}(D(L_{t_s+(i-1)*d,d}), s)$ done
$F := F_1 \cup ... \cup F_N$
for each $P$ in $F$ do $k_P := |\{F_j \mid 1 \leq j \leq N, P \in F_j\}|$ done
$F_C := \{P \mid P \in F, k_P \geq K\}$
$F_{withoutID} := \{P \mid P \in F_C, \neg\exists id, (id,1) \in P\}$
$F_C := F_C \setminus F_{withoutID}$
$OrderPatterns(F_C)$
$n := |F_C|$
$F_{redundant} := \emptyset$
for each $j = 1...n$, $P_j$ in $F_C$ do
  if ($\neg\exists i, i < j, C \subseteq P_j, C \in C_{P_i}$) then
    $L(P_j) := \{A \mid A \in L_{t_s,t_e-t_s}, A \text{ matches } P_j\}$
    $C_{P_j} := FindBestClustering(P_j, L(P_j), seq, pct, M)$
    $C_{notWellDistr} := \{C \mid C \in C_{P_j}, IsWellDistributed(C) = FALSE\}$
    $C_{P_j} := C_{P_j} \setminus C_{notWellDistr}$
  else
    $F_{redundant} := F_{redundant} \cup \{P_j\}$
  fi
done
$F_C := F_C \setminus F_{redundant}$
return the following definition for the alert classifier:
  $f(A) =$ $0$, if $\exists P \in F_C\ \exists C \in C_P$, $A$ matches $P$ AND $A$ matches $C$;
           $1$, otherwise
```

Figure 2. The procedure for building the alert classifier.

If there are many such redundant patterns, substantial amount of CPU time is wasted for finding clusters for them. Also, their presence in F_C will slow down the alert classification process. In order to address this problem, we use the following *OrderPatterns* procedure for sorting the patterns from F_C:

1. patterns are sorted by the signature ID (in numeric ascending order if IDs are numbers, otherwise in alphabetic order),
2. patterns with the same signature ID are then sorted in numeric ascending order by the number of elements,
3. patterns with the same number of elements are finally sorted in alphabetic order of their string notations.

Since the *OrderPatterns* procedure defines a complete order on F_C, we can now assume that $P_1 < \ldots < P_n$, where $P_i \in F_C$, $1 \le i \le n$, $n = |F_C|$. We then find clusters for patterns from F_C in this order, and alerts for pattern P_j will only be clustered if there is no previously detected cluster C for pattern P_i ($i < j$), so that $C \subseteq P_j$. If such C exists, alerts for P_j will not be clustered but P_j is rather removed from F_C. Since the *OrderPatterns* procedure ensures that generic patterns with lesser number of elements are processed before more specific patterns for the same signature ID, this considerably increases the likelihood that many specific patterns will be found redundant and discarded. During our experiments, we have observed a significant decrease in the number of patterns (e.g., for one IDS sensor the set F_C originally contained 511-831 patterns during one month period, which were reduced to 19-23 patterns during daily rebuilds of the alert classifier).

Although the previous technique allows for much faster building of alert classifiers, some clusters might not adequately describe routine alerts. While a pattern P is included in F_C only if it is detected for many slices, clusters for P are mined from a single alert set $L(P)$. As a consequence, a short term flood of atypical alerts that match P could create a cluster which mistakenly labels uncommon alerts as "routine". For tackling this issue, we check for each cluster from C_P how the alerts it matches are distributed over time. Currently a simple procedure called *IsWellDistributed* is used for this purpose which returns *FALSE* if the cluster matches alerts for less than D days, and *TRUE* otherwise (during our experiments, D has ranged between 7…14). However, more complex methods can be easily implemented within our framework.

Fig. 2 presents the *BuildClassifier* procedure for building the alert classifier which employs all techniques described in this section. In the next section, we will discuss our experiments for evaluating the classifier performance.

IV. EXPERIMENT RESULTS

In order to evaluate the performance of the alert classification method in a production environment, we have implemented it for three IDS sensors of a large financial institution (the sensors have been described in section IIIA). The logs of three sensors are monitored in real time with Simple Event Correlator (SEC) [18], and the alert classification functionality has been implemented as a SEC Perl module.

```
Mar 16 19:20:43 2010 mysensor.mydomain snort[***]: [1:2002911:4]
ET SCAN Potential VNC Scan 5900-5920 [Classification: Attempted
Information Leak] [Priority: 2]: {TCP}
192.168.75.59:2700 -> 10.20.19.98:5900

Mar 18 20:44:04 2010 mysensor.mydomain snort[***]: [1:1156:10]
WEB-MISC apache directory disclosure attempt [Classification:
Attempted Denial of Service] [Priority: 2]: {TCP}
192.168.207.131:2579 -> 10.20.81.22:80

Mar 19 16:44:52 2010 mysensor.mydomain snort[***]: [1:2010920:2]
ET WEB_SERVER Exploit Suspected PHP Injection Attack (cmd=)
[Classification: Web Application Attack] [Priority: 1]: {TCP}
192.168.65.22:47551 -> 10.20.11.72:80

Mar 21 23:14:18 2010 mysensor.mydomain snort[***]: [1:2002:8]
WEB-PHP remote include path [Classification: Web Application
Attack] [Priority: 1]: {TCP} 192.168.78.178:54459 -> 10.20.24.7:80
```

Figure 3. Examples of interesting alerts.

TABLE II. PERFORMANCE OF IDS ALERT CLASSIFIERS

	Sensor1	**Sensor2**	**Sensor3**
Sensor location	Internet	Internet	Intranet
Max. / min. / average number of alerts per day	379,981 / 25,649 / 168,008.85	57,217 / 2,524 / 11,610.58	393,884 / 75,417 / 232,327.75
Total number of alerts	9,408,496	650,193	13,010,354
Support threshold for 1-hour / 1-day slices	10 / 100	10 / 100	50 / 500
Max. / min. / average number of patterns per day	32 / 18 / 24.89	34 / 16 / 17.07	26 / 18 / 21.12
Max. / min. / average number of alerts classified as "routine" per day	99.83% / 75.35% / 98.41%	97.05% / 43.31% / 90.72%	99.30% / 89.66% / 97.34%
Total number of alerts classified as "interesting"	126,335	69,557	314,135

If an alert is classified as "interesting", it is written to a separate log file which is regularly reviewed by security analysts. Fig. 3 depicts examples of interesting alerts (the IP addresses and other sensitive data have been obfuscated for the reasons of privacy). The *BuildClassifier* procedures (see Fig. 2) for all sensors have been implemented as UNIX *cron* jobs and are invoked once in 24 hours, in order to rebuild the classifiers on a daily basis and make them adaptable to environment changes.

When choosing the input parameters for the *BuildClassifier* procedure, we discovered that the classification framework produces the best results when two classifiers are used simultaneously for each sensor. For the first classifier, frequent patterns are mined for the last 2 weeks log data which are divided into 336 1-hour slices, while for the second classifier, frequent patterns are mined for the last 30 days log data which are divided into 30 24-hour slices. An IDS alert A is classified as "interesting" if both classifiers return 1 for A, otherwise A is classified as "routine". We have discovered that the first classifier facilitates fast learning of strong new routine alert patterns, while the second classifier helps to correctly identify routine alerts (such as various port scans) which often don't appear as patterns in 1-hour time frames [2]. For both classifiers, the pattern relevance threshold K has been set to 50% of the number of slices (i.e., to 168 and 15, respectively), the clustering threshold sequence *seq* has been set to 10%...1%, while clustering parameters *pct* and M have been set to 5 and 30 (see Fig. 2). As for the support threshold s, we finally settled for values which usually produce 15-30 classification patterns for each sensor, so that the human analyst can easily grasp the patterns and follow the classification process. Table II presents the results of our experiments for the 8 week period.

During 8 weeks, the amount of routine alerts for Sensor 2 was unexpectedly small for three days, with only 43.31% of alerts classified as "routine" in the worst case. When we investigated this issue, we discovered that this was caused by the flaw introduced to a vendor signature [19] which affected several sites and triggered 29,189 false positive alerts in our environment. Since the signature had not produced any alerts in the past, the false positives were classified as "interesting". Without these alerts taken into account, we found that the performance of our method is comparable to the performance of other recently proposed data mining approaches (their authors have reported the reduction of alerts by 74-93%, see section II for details).

In order to evaluate how well does our classification method recognize alerts that require attention from the security personnel, we define precision and recall in the following way. If *TP* is the set of alerts that are correctly classified as "interesting", *FP* is the set of alerts that are incorrectly classified as "interesting", and *FN* is the set of alerts incorrectly classified as "routine", then *precision* is defined as $|TP| / (|TP|+|FP|)$ and *recall* is defined as $|TP| / (|TP|+|FN|)$. For estimating recall and precision, we extracted 274,354 penetration test alerts (produced in December 2008 by 660 signatures) and 495,670 known irrelevant alerts (triggered between December 2008 and March 2010 by 43 signatures) from one sensor log and replayed them to the classifier of the same sensor. From 282,682 alerts classified as "interesting", 8,411 were irrelevant alerts, yielding the precision of 97.02%. Only 83 penetration test alerts from 274,354 were incorrectly classified as "routine", yielding the recall of 99.96%.

V. Conclusion and Future Work

In this paper, we have presented a novel unsupervised IDS alert classification method which is based on data mining techniques. Our experiments suggest that the method works well in a production environment and significantly reduces the workload of the security personnel.

We have also experimented with extending the alert notion to include source and destination country information, obtained via GeoIPWhois database. This allowed for the detection of some more specific patterns (for some signature IDs we noticed that traffic is often coming from a specific country), but they did not change the classification results significantly. Nevertheless, we feel that this research idea deserves further study. As discussed in [1, 2, 11, 12], major changes in the arrival rates of routine alerts might be of interest to the security personnel, but unfortunately it is impossible to detect these changes with our method. For this reason, we also plan to research the applicability of various statistical methods for IDS alert processing, in order to detect unexpected floods of routine alerts.

Acknowledgment

The authors express their gratitude to Mr. Kaido Raiend, Dr. Paul Leis, Mr. Ants Leitmäe, and Mr. Ain Rasva from SEB Estonia for supporting this work.

References

[1] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen, "Processing intrusion detection alert aggregates with time series modeling," Information Fusion Journal, vol. 10(4), 2009, pp. 312-324.

[2] R. Vaarandi, "Real-Time Classification of IDS Alerts with Data Mining Techniques," in *Proc. of 2009 MILCOM Conference*, 7 pp.

[3] K. Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency," in *Proc. of 2001 Annual Computer Security Applications Conference*, pp. 12-21.

[4] K. Julisch and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," in *Proc. of 2002 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 366-375.

[5] K. Julisch, "Clustering Intrusion Detection Alarms to Support Root Cause Analysis," ACM Transactions on Information and System Security, vol. 6(4), 2003, pp. 443-471.

[6] J. J. Treinen and R. Thurimella, "A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures," in *Proc. of 2006 RAID Symposium*, pp. 1-18.

[7] J. Long, D. Schwartz, and S. Stoecklin, "Distinguishing False from True Alerts in Snort by Data Mining Patterns of Alerts," in *Proc. of 2006 SPIE Defense and Security Symposium*, pp. 62410B-1--62410B-10.

[8] C. Clifton and G. Gengo, "Developing Custom Intrusion Detection Filters Using Data Mining," in *Proc. of 2000 MILCOM Symposium*, pp. 440-443.

[9] S. O. Al-Mamory, H. Zhang, and A. R. Abbas, "IDS Alarms Reduction Using Data Mining," in *Proc. of 2008 IEEE World Congress on Computational Intelligence*, pp. 3564-3570.

[10] S. O. Al-Mamory and H. Zhang, "Intrusion Detection Alarms Reduction Using Root Cause Analysis and Clustering," Computer Communications, vol. 32(2), 2009, pp. 419-430.

[11] J. Viinikka and H. Debar, "Monitoring IDS Background Noise Using EWMA Control Charts and Alert Information," in *Proc. of 2004 RAID Symposium*, pp. 166-187.

[12] J. Viinikka, H. Debar, L. Mé, and R. Séguier, "Time Series Modeling for IDS Alert Management," in *Proc. of 2006 ACM Symposium on Information, Computer and Communications Security*, pp. 102-113.

[13] T. Pietraszek, "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection," in *Proc. of 2004 RAID Symposium*, pp. 102-124.

[14] B. Morin and H. Debar, "Correlation of Intrusion Symptoms: an Application of Chronicles," in *Proc. of 2003 RAID Symposium*, pp. 94-112.

[15] P. Ning, Y. Cui, and D. S. Reeves, "Analyzing Intensive Intrusion Alerts via Correlation," in *Proc. of 2002 RAID Symposium*, pp. 74-94.

[16] R. Vaarandi, "A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs," in *Proc. of 2004 IFIP International Conference on Intelligence in Communication Systems*, pp. 293-308.

[17] R. Vaarandi, "A Data Clustering Algorithm for Mining Patterns From Event Logs," in *Proc. of 2003 IEEE Workshop on IP Operations and Management*, pp. 119-126.

[18] R. Vaarandi, "Simple Event Correlator for real-time security log monitoring," Hakin9 Magazine, vol. 1/2006 (6), 2006, pp. 28-39.

[19] http://seclists.org/snort/2010/q1/481