Real-time Classification of IDS Alerts with Data Mining Techniques

Risto Vaarandi

Reprinted from *Proceedings of the 2009 IEEE MILCOM Conference*.
(ISBN: 978-1-4244-5239-2)

REAL-TIME CLASSIFICATION OF IDS ALERTS WITH DATA MINING TECHNIQUES

Risto Vaarandi
Cooperative Cyber Defence Centre of Excellence
Tallinn, Estonia
risto dot vaarandi at mil dot ee

ABSTRACT

During the last decade, intrusion detection systems (IDSs) have become a widely used measure for security management. However, these systems often generate many false positives and irrelevant alerts. In this paper, we propose a data mining based real-time method for distinguishing important network IDS alerts from frequently occurring false positives and events of low importance. Unlike conventional data mining based approaches, our method is fully automated and able to adjust to environment changes without a human intervention.

INTRODUCTION

During the last decade, intrusion detection systems (IDSs) have become a widely used measure for security and situation management. However, these systems are known to generate many alerts, with a considerable amount of them being either false positives or of low importance. This problem is especially common for network IDSs that monitor network segments for unwanted or malicious traffic. It is not unusual to receive thousands of alerts from a single network IDS sensor per day, with more than 90% of the alerts being irrelevant [1, 2, 3]. This often prevents the security analyst from spotting and handling events that represent a real threat.

Today, many network IDS vendors are using signature-based approach for identifying unwanted and malicious traffic – the IDS sensor is equipped with human-written signatures that describe bad network packets (e.g., a signature could contain a regular expression for matching the packet payload). Although it might seem tempting to eliminate irrelevant IDS alerts by tuning the vendor signatures for the local environment, this approach is often infeasible. First, signature development is a complex task that requires a lot of expertise and involves extensive testing in a sophisticated lab environment. However, most end users have neither such expertise nor testing resources. Second, several vendors don't reveal the signature content to the end user, in order to protect the confidentiality of proprietary attack detection techniques. Third, most vendors are frequently releasing signature updates which may conflict with customer changes. Last but not least, in many cases the alert is irrelevant not because of a faulty signature, but rather because of the surrounding context. For example, a signature for detecting SNMP probes from external networks might also trigger IDS alerts on SNMP GET queries from ISP network management servers, although these are a routine monitoring activity.

For the reasons above, false positives and other irrelevant IDS alerts can't be avoided in most environments. In order to distinguish important IDS alerts from irrelevant events, IDS alert log analysis techniques are often used. Many approaches have been suggested for this purpose like machine learning [4], time series modeling [3, 5], the use of control charts [6], etc. During the last decade, data mining based methods have also been proposed in a number of research papers [1, 2, 7, 8, 9, 10, 11]. With these methods, IDS alert logs from the recent past are mined for knowledge that is used for the creation of event filtering and correlation rules for future IDS alerts. However, existing methods are inherently semi-automated – they assume that a human expert interprets detected knowledge and creates event filtering and correlation rules by hand.

In this paper, we extend our previous research [11, 12] and propose a novel unsupervised data mining based approach for IDS alert classification. With this approach, knowledge is mined from IDS logs and processed in an automated way, in order to build an alert classifier. The classifier is then used in real-time for distinguishing important IDS alerts from frequently occurring false positives and events of low importance. The remainder of this paper is organized as follows: first, an overview of related work is provided; then, the algorithm for building the classifier is presented; the third section describes the classifier implementation and performance; the last section discusses open issues and future work.

RELATED WORK

Data mining methods were first used for knowledge discovery from telecommunication event logs more than a decade ago [13]. In the context of IDS alert log mining, a number of approaches have been suggested. Clifton and Gengo [7] have investigated the detection of frequent alert sequences, in order to use this knowledge for creating IDS alert filters. Long et al. [8] have suggested a supervised clustering algorithm for distinguishing Snort IDS true alerts from false positives. Julisch and Dacier [1] have proposed a conceptual clustering technique for IDS alert logs, so that clusters correspond to alert descriptions, and a 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% [1]. This work was later extended by Julisch who reported the reduction of alerts by 87% [2]. Al-Mamory et al. [9, 10] have proposed clustering algorithms for finding generalized alarms which help the human analyst to write filters. During the experiments, the number of alarms decreased by 93% [9] and 74% [10].

BUILDING THE ALERT CLASSIFIER

In this section, we propose a frequent itemset mining based method for discovering knowledge from historical IDS alert logs and creating alert classifiers from this knowledge in an automated manner. Our method is motivated by the main drawback of previously proposed data mining approaches – a human expert has to analyze detected knowledge and manually create event filtering and correlation rules. Unfortunately, this is an expensive procedure that requires a lot of skill and that has to be performed regularly (e.g., on a monthly basis [2]), because the environment is changing constantly. For instance, signature database updates from vendors may introduce new alert messages to IDS alert logs.

Our method is also motivated by the following observations. First, most alerts are triggered by only a few signatures. For example, when we inspected the logs of three IDS sensors of a large financial institution for the January-March 2009 time frame, we found that ten most prolific signatures produced 85.6%, 86.2% and 96.2% of the alerts, respectively (each sensor had over 16,000 signatures deployed which triggered more than 300,000 alerts per month). Other researchers have observed the same phenomenon – e.g., in [3] it is reported that 68% of the alerts were produced by five signatures, while according to [5] seven signatures produced 77% of the alerts. Second, if a signature has triggered many alerts over longer periods of time, it is also likely to do so in the future. During our experiments, we identified twenty most prolific signatures for all days in the January-March 2009 time frame (90 days), using the logs of previously mentioned three IDS sensors. When we examined these 90 lists for each sensor, we found that 12, 10 and 11 signatures appeared in all lists, respectively. Taking into account these observations, we argue that in many environments a relatively small number of very frequent alert patterns can be found in IDS alert logs. Furthermore, these patterns describe the majority of alerts that will appear in the future. We also argue that since the patterns represent alerts that occur very frequently over longer periods of time, these alerts are well-known to security analysts who review IDS logs regularly. Due to their frequency and persistent nature, we call them *routine alerts* in the remainder of this paper.

When we investigated routine alerts from previously mentioned three IDS sensors, we found that they are either false positives or events of low importance. One of the commonly occurring false positive alerts is triggered by network monitoring traffic coming from trusted sources (ISP network management servers). A prominent alert of low importance is related to MS Slammer Sapphire worm which infected many computers around the world in 2003. Despite vast majority of these computers were cleaned and patched, there are still infected nodes around that are constantly scanning the Internet for victims. Although this malicious network traffic triggers many alerts, they represent a very frequent and well-known attack that doesn't pose a threat to properly maintained systems.

We believe that the automated real-time identification of such routine alerts is important in many environments. First, it helps to save human effort that is spent for editing alert filters. Therefore, security analysts will have more time for reviewing alerts which don't match routine alert patterns and thus deserve closer investigation. Second, since most IDS alerts are routine events, there will be much less alerts to investigate than in the original IDS log. In order to detect routine alert patterns, we employ frequent itemset mining which is a prominent data mining technique for finding regularities in various data sets [14], including event logs [12, 13]. From

detected knowledge, an alert classifier is created for real-time processing of future alerts. Since network IDS sensors might be of various types and deployed in a wide range of environments (e.g., public networks, intranets or DMZs), they might produce very different outputs, therefore it often makes sense to apply our method for individual IDS sensors separately. For the sake of simplicity, we assume during the following discussion that the IDS alert log is produced by a single sensor.

We model each alert A as a tuple $A = (A_{time}, A_{ID}, A_{proto}, A_{srcIP}, A_{srcPort}, A_{destIP}, A_{destPort})$, where the *time* attribute reflects the occurrence time of the alert, the *ID* attribute describes the ID of the signature that produced the alert, and the *proto* attribute identifies the network protocol for the traffic that triggered the alert. The *srcIP*, *srcPort*, *destIP*, and *destPort* attributes describe the source IP address, source port, destination IP address, and destination port of the traffic. If the protocol does not involve ports (e.g., ICMP), we use the constant "-" for the *srcPort* and *destPort* attribute values. Figure 1 presents an example how we model Snort IDS alerts.

```
Mar 31 00:12:23 2009 mysensor [auth.alert]
snort[8903]: [1:1852:4] WEB-MISC robots.txt
access [Classification: access to a potentially
vulnerable web application] [Priority: 2]:
{TCP} 10.219.73.73:36167 -> 192.168.10.2:80

Mar 31 00:12:39 2009 mysensor [auth.alert]
snort[8903]: [1:2003:12] SQL Worm propagation
attempt [Classification: Misc Attack]
[Priority: 2]:
{UDP} 10.183.11.200:1298 -> 192.168.10.245:1434

Mar 31 00:13:09 2009 mysensor [auth.alert]
snort[8903]: [1:483:6] ICMP PING CyberKit 2.2
Windows [Classification: Misc activity]
[Priority: 3]:
{ICMP} 10.16.16.146 -> 192.168.10.60


(1238458343, 1:1852, TCP, 10.219.73.73, 36167,
                          192.168.10.2, 80)
(1238458359, 1:2003, UDP, 10.183.11.200, 1298,
                          192.168.10.245, 1434)
(1238458389, 1:483, ICMP, 10.16.16.146, -,
                          192.168.10.60, -)
```

Figure 1. Sample Snort IDS log messages with corresponding alert tuples

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". IDS alert log L is the set of alerts that the IDS sensor has logged in the past. Given the time t and the time interval d, the *log slice* $L_{t,d}$ is defined as follows: $L_{t,d} = \{A \mid A \in L, t \leq A_{time} < t + d\}$ (i.e., $L_{t,d}$ contains all logged alerts from time t to $t+d-1$).

Let $I = \{i_1,...,i_n\}$ 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 *cover* of an itemset X is the set of identifiers of transactions that contain X: $cover(X) = \{tid \mid (tid, Y) \in D, X \subseteq Y\}$. The *support* of an itemset X is defined as the number of elements in its cover: $supp(X) = |cover(X)|$. If itemset X does not have any proper supersets with the same support ($\neg\exists Y, X \subset Y, supp(X) = supp(Y)$), X is called a *closed itemset*. The *frequent itemset mining problem* is defined as follows [14] – given the transaction database D and the support threshold s, find all itemsets with the support s or higher (each such set is called a *frequent itemset*), and the supports of frequent itemsets. Instead of all frequent itemsets, we focus on mining closed frequent itemsets from IDS logs, since they are a compact and lossless representation of all frequent itemsets [15]. If D is a transaction database and s is a support threshold, $F_{closed}(D, s)$ denotes the set of closed frequent itemsets for D and s.

In order to apply frequent itemset mining formalism to IDS alert log L, we order the alerts in L in the occurrence time ascending order (alerts with identical occurrence times can be ordered by other attributes). If A is the m-th alert from L, we can view it as a transaction (m, X), where $X = \{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$. In other words, in order to distinguish identical values of different attributes, we order the relevant six attributes and store both the attribute value and number into each item. In that way, the IDS alert log L becomes a transaction database (we denote it by $D(L)$), and frequent itemset mining algorithms can be applied to this database. Detected closed frequent itemsets describe frequent alert patterns which not merely identify prolific signatures, but rather reveal strong associations between all types of attribute values. E.g., the itemset *{(PHPBufferOverflow,1), (TCP,2), (10.1.1.1,5), (80,6)}* indicates that the PHP buffer overflow attack is often attempted from various sources against the HTTP server running at the node 10.1.1.1. In the remainder of this paper, we use the terms pattern and itemset interchangeably. We also use the string notation for itemsets (patterns), where attribute values are extracted from items and written in the order of attributes; if there is

no item for an attribute, asterisk (*) is written. For instance, the above itemset is denoted by *PHPBufferOverflow TCP * * 10.1.1.1 80*.

If $A = (A_{time}, A_{ID}, A_{proto}, A_{srcIP}, A_{srcPort}, A_{destIP}, A_{destPort})$ is an alert and P is a pattern, we say that A *matches* P if $P \subseteq \{(A_{ID},1), (A_{proto},2), (A_{srcIP},3), (A_{srcPort},4), (A_{destIP},5), (A_{destPort},6)\}$. For example, the alert *(100, DNSprobe, UDP, 10.1.1.1, 3761, 10.1.1.2, 53)* matches the pattern *DNSprobe UDP * * * 53*, but not the pattern *DNSprobe UDP * * 10.1.1.3 53*. If P is a pattern and there exists a valid signature ID *id*, so that $(id,1) \in P$, we say that *P has the ID attribute*. For other attributes $\alpha \in \{proto, srcIP, srcPort, destIP, destPort\}$, we define *P has the α attribute* in a similar fashion.

During our previous research, we have developed a data mining tool LogHound that implements an efficient frequent itemset mining algorithm for event log data sets [12]. LogHound can be configured to store both the attribute name and value into items and is thus suitable for mining patterns from IDS alert logs according to our model. Figure 2 depicts sample Snort IDS alert patterns that have been detected with LogHound.

```
# VNC horizontal port scan from 10.22.50.53
1:2002911 TCP 10.22.50.53 * * 5900
Support: 20

# SQL Worm propagation attempt
1:2003 UDP * * * 1434
Support: 98

# bad TCP traffic to 192.168.19.20:80
* TCP * * 192.168.19.20 80
Support: 137

# Cyberkit ping from 10.16.16.1 to 192.168.10.60
1:483 ICMP 10.16.16.1 - 192.168.10.60 -
Support: 288
```

Figure 2. Sample alert patterns detected with LogHound

With traditional IDS log mining systems, human experts would use the knowledge depicted in Figure 2 for writing event filtering and correlation rules. At first glance, it might seem tempting to automate this procedure in a straightforward manner and define the alert classifier in the following way (L is the IDS alert log and s is the support threshold):

$$f(A) = \begin{cases} 0, \text{ if } \exists X \in F_{closed}(D(L), s), A \text{ matches } X; \\ 1, \text{ otherwise} \end{cases}$$

In other words, the classifier labels the alert as "interesting" if it does not match any of the closed frequent patterns, otherwise the alert is labeled as "routine". However, the above method for building the classifier has several drawbacks. First, an intensive attack might trigger many IDS alerts and create frequent patterns, although the attack itself could only last for a short time. If a similar attack occurs in the future, the classifier will mistakenly label attack alerts as "routine", although they are highly interesting. Second, some patterns from $F_{closed}(D(L), s)$ are too generic. For example, the third pattern in Figure 2 implies that all alerts regarding the target "HTTP server at the node 192.168.19.20" will be classified as "routine".

For the reasons above, we propose another algorithm for building the classifier that considers IDS alerts from a time window $t_s \ldots t_e$. The algorithm divides these alerts into n slices, with each slice covering a time frame of the same size (e.g., 1 hour). After that, closed frequent patterns are mined from each slice. For building the classifier, only these patterns are considered which were detected for k or more slices, and which have the *ID* attribute. If the time window $t_s \ldots t_e$ covers a sufficiently large time frame from the recent past (e.g., 4 weeks), and the number of slices n and threshold k are large enough (e.g., there are 672 slices of 1 hour with a threshold of 336), the algorithm will detect only these patterns that correspond to routine alerts. Figure 3 provides a detailed description of the algorithm.

Input:
t_s – the beginning of the time window
t_e – the end of the time window
n – the number of slices
k – the pattern relevance threshold ($1 \le k \le n$)
s – the support threshold

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

1. $d := (t_e - t_s) / n$
2. For each i in $\{0,..., n-1\}$ $F_{i+1} := F_{closed}(D(Lt_s+i*d,d), s)$
3. $F := F_1 \cup F_2 \cup \ldots \cup F_n$
4. For each P in F $count_P := |\{F_j \mid 1 \le j \le n, P \in F_j\}|$
5. $F_C := \{P \mid P \in F, count_P \ge k, P \text{ has the ID attribute}\}$
6. Return the following definition of the alert classifier:

$$f(A) = \begin{cases} 0, \text{ if } \exists X \in F_C, A \text{ matches } X; \\ 1, \text{ otherwise} \end{cases}$$

Figure 3. An algorithm for building the alert classifier

The algorithm has several important properties. First, it does not require any human intervention.

Second, it can be used for rebuilding the classifier after certain time periods (e.g., once a day), in order to make the classifier adaptable to environment changes. Third, the algorithm is not sensitive to a short-term noise, since only those patterns are used for classifying alerts which appear frequently in many time frames over longer periods of time.

However, although the algorithm removes patterns without the *ID* attribute from further consideration, it is still susceptible to *over-generalization* with respect to other attributes (in IDS alert mining context, this problem has also been identified in [1]). For example, suppose that the IDS sensor has a signature for detecting SNMP GET requests, and the application server 10.1.1.1 normally receives SNMP GET queries only from two monitoring servers 192.168.1.1 and 192.168.1.2. If patterns *SNMPGet UDP 192.168.1.1 * 10.1.1.1 161* and *SNMPGet UDP 192.168.1.2 * 10.1.1.1 161* appear in the set F_C, then F_C could also contain *SNMPGet UDP * * 10.1.1.1 161* (because its support can't be smaller than the sum of supports of the first two patterns). Unfortunately, the latter pattern is too generic, since it classifies SNMP GET request alerts for all source nodes as "routine", although requests from other nodes than 192.168.1.1 and 192.168.1.2 are unusual.

In order to address this problem, we augment the classifier building algorithm in the following way. After the set F_C has been found (Figure 3, step 5), all distinct signature ID values are extracted from all patterns in F_C. Recent alert log slice is then scanned and for each signature ID, the algorithm checks whether the signature mostly triggers alerts for a few source and/or destination combinations only. This information is later used by the alert classifier for making too generic patterns in F_C more restrictive. The algorithm starts with checking if each signature *id* produces alerts for only a few combinations of *proto*, *srcIP*, *destIP*, and *destPort* attribute values (during our experiments, typical values for d, l, m, and p have been "two weeks", 0.1…0.25, 10…50, and 95…99, respectively):

1. For all alerts A in $L_{t,d}$ that have $A_{ID} = id$, count the occurrence times of all distinct combinations $(A_{proto}, A_{srcIP}, A_{destIP}, A_{destPort})$, and find the total number of alerts: $j = |\{A \mid A \in L_{t,d}, A_{ID} = id\}|$.
2. In order to filter out noise, remove all very infrequent combinations (combinations that occur only once or twice), and set c to the number of remaining combinations.
3. Remove infrequent combinations that have occurred much less than the average (less than $j/c*l$ times, where $l < 1$).
4. If the number of remaining combinations is smaller than m and they cover more than p% of the alerts with the signature ID *id*, the combinations are returned as the *frequent endpoint set* H_{id}.

If the above estimation procedure does not yield a result, it is repeated for the *(proto, destIP, destPort)*, *(proto, srcIP, destPort)*, and *(proto, destPort)* attribute combinations (in the given order), until one of the procedures returns a result or the last procedure terminates. Note that during the analysis, we are focusing on protocol- and destination-related attributes and have excluded the *srcPort* attribute altogether, since source attributes are usually associated with attackers and can have a wide range of possible values (especially *srcPort*). However, if the *srcIP* value belongs to a victim, we swap the source and destination attribute values and apply the estimation procedure in the normal way.

Note that frequent endpoint sets don't consume much storage space, because sets are created only for signature IDs that are extracted from patterns in F_C. Fortunately, the number of such signatures is small, since only a few signatures are prolific (see the discussion in the beginning of this section). Furthermore, a set is created only if it is known to be compact, since the number of its elements can not exceed the given threshold. Therefore, frequent endpoint sets easily fit into memory and the alert classifier can access them in a fast way. Figure 4 presents an enhanced version of the alert classifier. Note that it is a generalization of the classifier described in Figure 3 – if frequent endpoint sets are not calculated, two classifiers are identical.

In order to classify an input alert A, the classifier scans the set F_C for matching patterns. If A matches P from F_C and there is no frequent endpoint set for A_{ID}, A is classified as "routine" (Figure 4, line 6). If $H_{A_{ID}}$ exists and P is specific enough, A is also classified as "routine" (Figure 4, line 8). However, if P is too generic (it does not have all the attributes that are stored in $H_{A_{ID}}$), the algorithm classifies A as "routine" only if the tuple of relevant attribute values of A is found in $H_{A_{ID}}$ (Figure 4, line 9). Otherwise, the next pattern from F_C is tried, and if the search ends without a result, A is classified as "interesting". For example, if $H_{SNMPGet}$ = *{(UDP, 192.168.1.1, 10.1.1.1, 161), (UDP, 192.168.1.2, 10.1.1.1, 161)}*, the pattern *SNMPGet UDP*

*192.168.1.1 * 10.1.1.1 161* is specific enough and classifies any matching alert as "routine". On the other hand, the pattern *SNMPGet UDP * * 10.1.1.1 161* is too generic, and the frequent endpoint set $H_{SNMPGet}$ is thus consulted. In that case, the alert *(10, SNMPGet, UDP, 192.168.1.2, 36129, 10.1.1.1, 161)* is classified as "routine", while for *(11, SNMPGet, UDP, 192.168.1.3, 36129, 10.1.1.1, 161)* the search of F_C for matching patterns will continue.

Input: A – an IDS alert
Output: 0 if A is classified as "routine",
1 if A is classified as "interesting"

Functions used by the classifier:
1) the *getattr()* function returns the set of attribute names for the frequent endpoint set H, e.g., if H contains *(proto, destPort)* tuples, *getattr(H)* returns *{proto, destPort}*;
2) the *createtuple(A, AttrSet)* function extracts the values of attributes *AttrSet* from the alarm A and creates a tuple from them, in order to search a frequent endpoint set, e.g., *createtuple((10, DNSprobe, UDP, 192.168.1.1, 1234, 10.1.1.1, 53), {proto, destPort})* returns *(UDP, 53)*

```
1: function f
2:  {
3:    for each P in F_C {
4:      if (A matches P) then {
5:        id := A_ID
6:        if (H_id does not exist) then return 0
7:        AttrSet := getattr(H_id)
8:        if ( ∀α ∈ AttrSet, P has the α attribute) then return 0
9:        if (createtuple(A, AttrSet) ∈ H_id) then return 0
10:     }
11:   }
12:   return 1
13: }
```

Figure 4. An enhanced alert classifier

IMPLEMENTATION AND PERFORMANCE

In this section, we describe our classifier implementation and experiments for estimating its performance. During the experiments, we have applied our method for three Internet and intranet IDS sensors of a large financial institution. In our setup, alerts classified as "interesting" are written to a separate log file for further review. Classifiers are rebuilt every midnight, using the sensor log data of the last two weeks that is divided into 1 hour slices. After experimenting with several pattern relevance thresholds (input parameter k in Figure 3), we finally settled for 168. In other words, once the closed frequent pattern has been detected for at least half of the 1 hour time frames during the last two weeks, it will be used for further alert classification. This allows for the classifier to adapt to new routine alert patterns with a reasonable learning time, without being sensitive to occasional bursts of similar alerts.

After initial experiments, we found that several port scan alerts were classified as "interesting", although they occurred frequently in almost every slice. A closer study revealed that each such port scan was done from a single IP address that often never appeared again in the IDS log. Therefore, each distinct pattern (e.g., *PortScan TCP 10.10.1.1 * * 22*) appeared only in a couple of slices, and more generic patterns without source-related attributes (e.g., *PortScan TCP * * * 22*) were seldom detected. As a consequence, no patterns describing these alerts were included in F_C. In order to prevent this, we decided to augment F_C with closed frequent patterns mined from the last 60 days log data, setting the slice size to 24 hours and the relevance threshold to 30. When we used the larger slice size, port scan patterns without source attributes were found in most slices and included in F_C. As a result, port scan alerts were classified correctly.

Frequent pattern mining tasks (step 2 in Figure 3) are the most CPU and memory intensive parts of the classifier building algorithm. In order to save CPU and memory consumption and distribute it over time, the mining tasks have been implemented as UNIX *cron* jobs. At the 5th minute of every hour, frequent patterns for the previous hour are mined and saved to disk, so that detected patterns can be reused for daily rebuilds of the classifier during the next two weeks. In a similar fashion, frequent patterns for the previous day are mined at every midnight and stored to disk for further reuse. In order to allow for decreasing the support threshold (input parameter s in Figure 3) in the future, patterns are mined with a threshold somewhat lower than s, while the classifier building algorithm considers only patterns with the support s or higher.

Table 1 presents our experiment results for 31 days in March-April 2009 (each sensor had over 16,000 signatures deployed). We found that very few signatures produce alerts which have a chance to be classified as "routine", and for these signatures frequent endpoint sets are often found which make generic patterns more restrictive. During the experiments, 81-99% of alerts were classified as "routine", and the set of "interesting" alerts was 5-100 times smaller than the original alert log. These results are comparable with the performance of other

recent data mining methods which have reduced the number of alerts by 74-93% [1, 2, 9, 10].

Table 1. Classifier performance

	Sensor 1	Sensor 2	Sensor 3
max./min./average # of alerts per day	88955 / 8672 / 37569	16289 / 3475 / 8941	819061 / 105247 / 194063
support threshold for 1h/1d slices	10 / 100	10 / 100	20 / 200
max./min. # of patterns in F_C	52 / 46	102 / 40	354 / 341
max./min. # of signature IDs in patterns of F_C	16 / 15	12 / 10	16 / 16
max./min. # of freq. endpoint sets	16 / 14	5 / 3	12 / 10
max./min. amount of "routine" alerts	98.96% / 86.36%	94.27% / 84.98%	99.78% / 81.85%
average amount of "routine" alerts	93.63%	90.18%	96.44%

If T_p is the set of alerts that are correctly classified as "interesting", F_p is the set of alerts incorrectly classified as "interesting", and F_n is the set of alerts incorrectly classified as "routine", then *precision* is defined as $|T_p| / (|T_p| + |F_p|)$ and *recall* is defined as $|T_p| / (|T_p| + |F_n|)$. In order to estimate the classifier recall and precision, we extracted 274,354 penetration test alerts (triggered by 660 signatures) and 47,257 known irrelevant alerts (triggered by 42 signatures) from one sensor log, and replayed them to the classifier of the same sensor. All alerts were produced during 4 days in December 2008; IDs of 5 signatures that triggered 31,407 pen-test alerts also appeared in F_C. Nevertheless, 274,207 pen-test alerts were correctly classified as "interesting", yielding the recall of 99.94%. From 277,932 alerts classified as "interesting", only 3,725 were irrelevant alerts, yielding the precision of 98.65%.

OPEN ISSUES AND FUTURE WORK

In this paper, we have presented a novel data mining based IDS alert classification method. Although our preliminary results are promising, one issue remains open – major changes in the arrival rate of routine alerts might be symptoms of large scale attacks, but are hard to detect. However, this is an inherent weakness of alert classification and filtering systems (e.g., see [3, 5, 6] for a related discussion). For the future work, we plan to research our classification method further, and study various statistical algorithms (e.g., regression analysis and time series analysis) for detecting unexpected fluctuations in the arrival rates of routine alerts.

REFERENCES

[1] 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.

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

[3] 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.

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

[5] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen. "Processing intrusion detection alert aggregates with time series modeling," in Information Fusion Journal, 2009, to appear.

[6] 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.

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

[8] 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.

[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," in Computer Communications, vol. 32(2), 2009, pp. 419-430.

[11] R. Vaarandi. "Mining Event Logs with SLCT and LogHound," in Proc. of 2008 IEEE/IFIP Network Operations and Management Symposium, pp. 1071-1074.

[12] 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.

[13] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. "Knowledge Discovery from Telecommunication Network Alarm Databases," in Proc. of 1996 International Conference on Data Engineering, pp. 115-122.

[14] B. Goethals. "Survey on Frequent Pattern Mining," Technical Report, University of Helsinki.

[15] M. J. Zaki and C.-J. Hsiao. "CHARM: An Efficient Algorithm for Closed Itemset Mining," in Proc. of 2002 SIAM International Conference on Data Mining, pp. 457-473.