3-17-2008

Intrusion Signature Creation via Clustering Anomalies

Gilbert R. Hendry
Rochester Institute of Technology

Shanchieh Jay Yang
Rochester Institute of Technology

Recommended Citation

Gilbert R. Hendry, Shanchieh J. Yang, "Intrusion signature creation via clustering anomalies", Proc. SPIE 6973, Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008, 69730C (17 March 2008); doi: 10.1117/12.775886; https://doi.org/10.1117/12.775886

Intrusion Signature Creation via Clustering Anomalies

Gilbert R. Hendry and Shanchieh J. Yang

Rochester Institute of Technology, Rochester, NY, USA

ABSTRACT

Current practices for combating cyber attacks typically use Intrusion Detection Systems (IDSs) to detect and block multi-stage attacks. Because of the speed and impacts of new types of cyber attacks, current IDSs are limited in providing accurate detection while reliably adapting to new attacks. In signature-based IDS systems, this limitation is made apparent by the latency from day zero of an attack to the creation of an appropriate signature. This work hypothesizes that this latency can be shortened by creating signatures via anomaly-based algorithms. A hybrid supervised and unsupervised clustering algorithm is proposed for new signature creation. These new signatures created in real-time would take effect immediately, ideally detecting new attacks. This work first investigates a modified density-based clustering algorithm as an IDS, with its strengths and weaknesses identified. A signature creation algorithm leveraging the summarizing abilities of clustering is investigated. Lessons learned from the supervised signature creation are then leveraged for the development of unsupervised real-time signature classification. Automating signature creation and classification via clustering is demonstrated as satisfactory but with limitations.

Keywords: intrusion detection, clustering, adaptive signatures

1. INTRODUCTION

Cyber security today is a problem growing more complex with increasing network sizes, amount of information, and sophistication of attacker methods. One key component in network defense is the use of Intrusion Detection Systems (IDSs). These systems analyze both host and network activity to detect the presence of occurring malicious activity.

The methods that IDSs employ can typically be grouped into two categories: signature-based and anomaly-based systems. Signature-based analyzers compare events against predefined models looking for either specific kinds of events or specific patterns of events. This method typically requires one or more experts defining and constantly updating the signatures. Anomaly-based analyzers attempt to identify unusual events or unusual patterns of events assuming that cyber attacks produce rare or different effects that appear in network or host data. Anomaly detection is focused on attempting to catch new attacks as well as old ones, but it often comes with the cost of being less accurate because of unusual but legitimate user behavior or varying amounts of malicious activity. Compared to signature-based systems, anomaly-based algorithms often require less or no *a priori* knowledge specific to the attacks, and thus may adapt to new attack methods. This work proposes to utilize an anomaly-based algorithm for dynamic signature creation which is updated as new activities are observed.

Signature-based intrusion detection is the predominant method in commercially available IDS systems. [1–3] In a signature-based system, activity is compared to a variety of specific models that are historically labeled as malicious. A signature-based IDS will most often have a very high detection rate of known malicious activity due to the precision of the models and the predictability of many attacks. However, because of the precision of signatures which allows them to be highly accurate, new attacks can often go undetected due to the time and resources necessary for an analyst to update the signature database. Because cyber attacks can change drastically in short periods of time, signature databases must be updated by analysts frequently. Also, signature-based IDSs can have varying degrees of sensitivity which may produce varying amounts of false positives.[4]

Anomaly-based detection schemes aim at finding attacks based on their relative differences to normal activity, and typically use statistical profiling, clustering, or other machine learning techniques. An anomaly-based system may be referred to as supervised if it requires that training data be labeled as malicious versus normal. Many existing clustering algorithms[5–10] are unsupervised methods that attempt to group data based on similarity measures determined over a high dimensionality without ground truth labels. The idea is to isolate interesting groups of data possibly in subspaces not obvious to an analyst. Clustering, one class of anomaly-based systems, aims at capturing new and evolving attacks as quickly as possible by forming groups of activity based on the relative differences between normal and malicious data.

Data Mining, Intrusion Detection, Information Assurance, and Data Networks
Security 2008, edited by Belur V. Dasarathy, Proc. of SPIE Vol. 6973,
69730C, (2008) · 0277-786X/08/$18 · doi: 10.1117/12.775886

Clustering algorithms used offline are not reliant on *a priori* knowledge, and can usually summarize many high-dimensional data points; these attributes make clustering an attractive choice in the intrusion detection community.

Some existing systems attempt to combine the advantages of both anomaly-based and signature-based systems. One approach is to use a supervised anomaly-based system to create signatures that are later used for detection. [11–13] Some systems propose combining separate processes of signature- and anomaly-based methods to decrease false positive rates. [14,15] When trained correctly, these systems can be very accurate. These works do not provide, however, schemes for updating signatures automatically. Other systems propose using artificial intelligence and other machine learning techniques to adapt an intrusion detection model. [16–18] Though these systems are adaptive, not all of them demonstrated their accuracy with a range of heterogeneous dynamic activity.

To take advantage of both signature- and anomaly-based intrusion detection, this work proposes using reliable signatures created from a supervised clustering algorithm, while updating them in real-time based on the results of unsupervised clustering. Signatures must be updated in a way that retains the signatures' useful information, while adapting to changing attack methods. Specifically, we will adopt a simple density-based clustering algorithm, called Simple Logfile Clustering Tool. [19] The choice of the algorithm is due to its simplicity, as opposed to full-fledged machine learning, in an attempt to create a *real-time* signature creation and update algorithm. The proposed work is different from existing methods in that it retains the integrity of proven signature-based methods while attempting to automatically adapt to new attack methods in real time and thus reducing the latency between attack occurrence and signature creation.

2. DESIGN AND IMPLEMENTATION

Figure 1 shows the overall architecture of the proposed system. The individual components will be discussed in the following subsections, starting with the baseline algorithm - SLCT. [19]

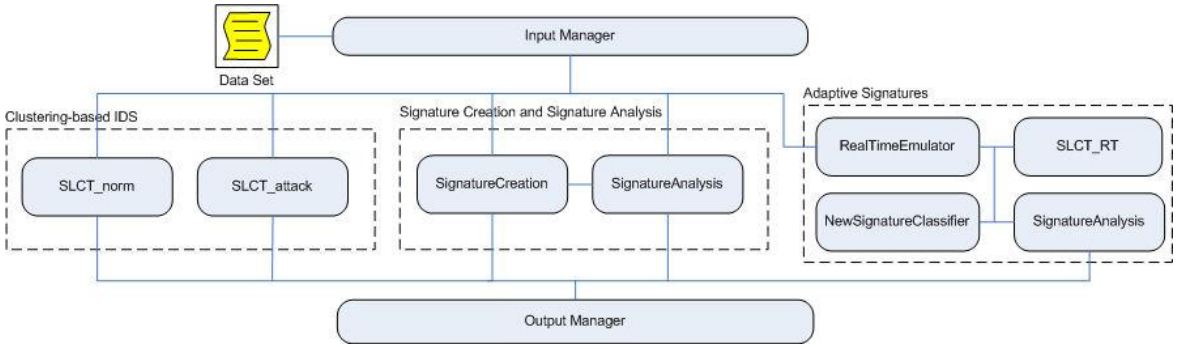


Figure 1. System Diagram

2.1 The SLCT Algorithm

The Simple Logfile Clustering Tool (SLCT) is an implementation of a density-based clustering algorithm proposed by Vaarandi intended to be used as an offline data-mining tool. [19] We will refer to the algorithm itself as the name SLCT, and the following is a description of how it works. Let a data point be a single log file entry and its attributes are the words contained in it. The following definitions are made. The data space has a dimensionality of n, where n is the maximum number of attributes in a data point, and each attribute belongs to a field, f_i. We call Λ the set of all possible fields in the data, or, $\Lambda = \{f_1, ..., f_n\}$. In preliminary experiments, it was determined that not all fields in Λ are necessarily useful for SLCT's operation. Therefore, we call Λ' the subset of Λ that was selected by the user that defines the fields which attributes may be used to define a region S. A region S is a subset of the data space, where certain attributes i_1,...,i_k ($1 \leq k \leq n$) of all points that belong to S have identical values v_1,...,v_k: We call the set (v_1,i_1),...,(v_k,i_k) the set of fixed attributes of region S. If k=1 (i.e., there is just one fixed attribute), the region is called a 1-region. A dense region is a region that contains at least N points (defined as a percentage of the total log lines), where N is the support threshold value given by the user. A region, *i.e.*, a set of fixed attributes, is referred to as a cluster candidate if it contains at least one dense 1-region.

The SLCT algorithm consists of three steps: building a data summary, defining cluster candidates, and refining clusters. The first step identifies dense 1-regions, which are essentially words that occur in at least N percent of lines (the position

in the log file line is also considered). The number of times a [word, position] pair is found in a log line is known as the support value.

The next step, which is concerned with defining cluster candidates, iterates over each line checking for dense 1-regions. If a data point contains one or more of the identified frequent words, a cluster candidate is formed or the existing cluster candidate's support value is incremented. In this way, each log line is assigned to exactly one cluster candidate. Let σ represent the set of these candidates.

The third step, which refines cluster candidates into valid clusters, involves simply iterating over the cluster candidates to check their support value. If the support value, as a percent of total activity, is less than the user-defined support threshold N, then it is not considered a cluster. Vaarandi also proposes an iterative solution to fine tuning the support threshold. Clustering results may heavily depend on the threshold N, and therefore will be chosen based on extensive experimentation. Further details of the SLCT algorithm can be found in the orignal work. [19]

2.2 Using SLCT to Separate Malicious from Legitimate Activity

Intuitively, cyber attacks are assumed to be anomalous. This assumption is made by many related works mentioned in Section 1, and leads to the conclusion that any activity belonging to valid clusters created by SLCT must be normal activity. Any data that does not belong to any cluster must therefore be malicious activity. This method is refered to as SLCT_norm.

Alternatively, clusters created by SLCT may be regarded as malicious activity. At first glance, this may seem counter-intuitive, but there are reasons for its justification. Normal activity is often harder to describe than malicious activity. One potential reason for this is that there are significantly more types of normal activity performed by an average host or network than there are types of known malicious activities that are effective. Cyber attacks are also often automated, making the activity observed very predictable. This automation, which can make an attack more effective and more destructive, can therefore be used to our advantage. Indeed, a signature-based IDS system is built on the concept of detecting the recognizable pattern of malicious activity. If SLCT is concerned with grouping together similar data, then it follows that malicious activity will most certainly form a cluster if the activity forms a significant enough part of the observed data. This phenomenon is indeed observed from the KDD data set, and will be discussed further in Section 3.

One clear problem with this method is that not only malicious but also normal activity will form clusters. However, if we assume that a cluster contains either mostly malicious or mostly normal activity, the problem then simplifies to differentiating between clusters. This assumption will later be verified, justifying our simplification of the problem. One characteristic of a cluster is that the number of fixed attributes in a cluster describes how similar the data in the cluster is: a higher number of fixed attributes indicates a higher degree of similarity. Given the automated, and therefore patterned nature of malicious activity, we assert that clusters formed from malicious activity often contain more fixed attributes than clusters formed from normal activity. If this is the case, then separating malicious clusters from normal clusters becomes a problem of separating clusters with more fixed attributes from the ones with less.

For this purpose, a new user parameter, M, is introduced to SLCT which specifies the percent of fixed attributes out of the maximum possible that a valid cluster is required to have. An M value of 0 will allow a cluster to be formed regardless of the number of fixed attributes. Setting M to a higher value will ideally eliminate all normal clusters leaving only the malicious ones, and thus classifying the original data. We will refer to this method using the M parameter as SLCT_attack. The only differences between SLCT_norm and SLCT_attack are additional contraints on valid clusters as well as the interpretation of the members of clusters.

Using SLCT in these two different methods should provide insight into the usefulness of a simple density-based clustering algorithm as an Intrusion Detection System. Because a density-based clustering algorithm simply classifies data based on relative differences, it may also be useful for creating IDS signatures from a labeled data set. In other words, if the classification of data has already been performed, SLCT may be useful in summarizing the characteristics of both normal and malicious activity, instead of trying to distinguish between the two.

2.3 Signature Analysis and Signature Creation

Signature-based Intrusion Detection Systems make use of misuse models, or attack signatures instead of a relative approach. They rely on a set of signatures that attempt to describe every known cyber attack that could be carried out. We propose a signature creation algorithm based on an iterative application of SLCT to a labeled training set, found in Figure

2. For a labeled training set, the user parameter N is found iteratively to maximize percent clustered, and the field selection Λ' is determined to minimize false positives during validation. For our purposes, a signature is similar to a cluster in that it contains a number of [value,position] attributes that describes activity. Activity matches a signature when all of the attributes in the signature match the corresponding values in the fields indicated by the positions in the signature. Other methods of matching signatures to activity can be investigated in future work.

```
FOR EACH(oneTypeOfData IN trainingData)
   Λ' = initial selection
   WHILE (data is not validated OR FalsePos ≥ FP_threshold)
      N = N_initial
      WHILE (90% is not clustered)
         N = N/2
         sigs = SLCT(oneTypeOfData, N)
      END WHILE
      FalsePos = validate(sigs, trainingData)
      signatureSet += sigs
      Λ' = next field selection
   END WHILE
END FOR
return signatureSet
```

Figure 2. Signature Creation Algorithm

2.4 Adaptive Signatures

Signatures used in a commercial IDS must be constantly updated to reflect changes in attacker methods. This process is often performed by an analyst, with the latency commonly on the order of days, or a week. [20] Extending the algorithm shown in Figure 2, we attempt to incorporate attack information into the signatures immediately after the attack has occurred by using clustering. Unlike some algorithms which attempt to use clustering as an anomaly-based sensor, we propose to use its summarizing abilities like those demonstrated in the signature creation algorithm. In order to take advantage of density-based clustering, we utilize our previous assertion that clusters contain either mostly malicious or mostly normal activity. We also assume that there is a working set of signatures that must be updated to reflect a new type of activity present in the data.

New data enters the system in chunks, or windows. The size of this window can be important, which we will refer to as ψ. This could be achieved in a real system by periodically polling at a set rate in time. Each window is first clustered. The supports for new and existing words and cluster candidates are updated to form new or modified clusters. These clusters become the new set of signatures, which are then used to analyze the same window. This process can be found in Figure 3. A set of challenges arise when considering this process.

```
signatures = createSignatures(trainingdata) //see Figure 2
clusters = SLCT(trainingdata)
FOR EACH (window)
   newClusters = SLCT_RT(window, clusters, attributes)
   signatures += classify(newClusters, attributes) //see Figure 4
   maliciousActivity = signatureAnalysis(window, signatures)
   clusters += diff(clusters, newClusters)
END FOR
```

Figure 3. Signature Update Process

First, clustering is by nature a post-processing, or data-mining application. For our purposes, we must adapt SLCT to real-time use while retaining a low computational complexity. We achieve this by introducing smoothing constants, α and

β, to both the words' support and cluster candidates' support, respectively. In a window, the new support for a word and cluster candidate become:

$$s_w = s_w\alpha + s'_w(1-\alpha) \quad (1)$$

$$s_c = s_c\beta + s'_c(1-\beta) \quad (2)$$

where s_w and s_c are the current supports, and s'_w and s'_c are the new supports calculated from the new window. In this way we can continuously cluster incoming cyber data without having to process the entire history of data, and we refer to this modification of the algorithm as SLCT_RT (real time).

Another problem with real time signature creation and adaptation is the classification of new signatures. After a window of data has been clustered, the set of fixed attributes for each cluster makes up the set of signatures. Many of these signatures will already have a label from the training phase. Any signature that has not been previously discovered is classified in the following way. Each word, or 1-region, in the dictionary has what is called an *attack support*. This value corresponds to the amount of data out of the total whose signature has contained the particular word and has already been classified as an attack. For instance, if the data is:

Red, **blue**, green : attack
yellow, **blue**, white : normal
red, grey, red : attack
purple, **blue**, black: attack

then the attribute (Blue, 2) would have an attack support of 0.667 because two of three pieces of data that contained the attribute were labeled as attacks. A classification for an unknown signature can therefore be given by taking the average of the attack supports for the signature. If this average is greater than a specified *confidence*, then the signature is for a new attack, else, it is normal. The nominal value of this confidence is of course 0.5, though a higher value may be used to possibly reduce false positives. Given a cluster that is yet unclassified, this process can be found in Figure 4.

$newSignature = \Phi$
FOR ($a \in \Phi.attributes$)
  $sum = sum + a.attackSupport$
END FOR

IF($sum/\Phi.attributes.Count \geq confidence$)
  $newSignature.status = newattack$
ELSE $newSignature.status = normal$

Figure 4. Signature Classification Algorithm

This method of classifying new signatures is contingent upon the assertion that attributes in the data are indicative of the status of the data. In other words, the process relies on an attribute occurring in mostly malicious or mostly normal activity. For our purposes, we will use ground truth to train the attributes' attack supports, and maintain those values throughout real time execution. The update of those attack supports could be done by an analyst or automatically based on the new signatures created.

3. EXPERIMENT SETUP

3.1 The KDD Data Set

The KDD data set[21] is a set of 'connection' records formed from low level data on MIT's DARPA virtual network. These connection records, numbering about five million, are labeled as either being normal or one of 21 different types of attacks from the following four categories: Denial of Service (DOS), Remote to Local (R2L), User to Root (U2R), and Surveillance. The KDD data set contains 41 fields partitioned into three categories: TCP (0-8), content (9-21), and traffic data (22-40). Traffic data is computed over 2 second windows. Also, besides being conveniently labeled for use in ground truth comparison, the KDD data set is widely used in intrusion detection studies.

3.2 Test Data Modifications

In order to properly test the proposed work described above, modifications were made to the KDD data. The modifications retain the characteristics of the data, such as the relative amounts of different types of attacks. The new data sets include those that contain different amounts of malicious activity, those intended for training and testing, and data sets that attempt to emulate zero-day attacks. A 10% subset of the KDD is also available which contains similar content as the original. Because the 10% subset requires much less resources than the full data set and is still representative of the content of the original, it will be used for our testing purposes. For the remainder of this paper, any reference to the KDD data set will refer to the 10% subset just mentioned.

Many of the IDS systems and proposed algorithms discussed above gauge their performance on the KDD data set. None, however, present their results in terms of varying amounts of malicious activity. Very few mention the introduction of new kinds of attacks into the data. For IDS studies, these should be important characteristics of a proper test data set. The KDD data set was modified to reflect varying amounts and types of malicious activity. To test our proposed work on a range of malicious volume, data sets were constructed with 0, 1, 5, 10, 25, 50, and 80 percent attacks by iteratively removing every other attack line until the desired amount of malicious activity was reached. This preserved the relative amounts of different kinds of attacks existing in the original set, which is essential to testing algorithms such as density-based clustering that are concerned with the relative amounts of data in the set.

To test parts of the proposed work, such as signature creation, data sets for training, validation, and testing were needed. A method similar to Clare's work[22] was used to randomly select trainig, validation, and testing sets. The size of the training set needed for the Signature Creation algorithm in Figure 2 is determined experimentally.

The third set derived from the KDD set was an attempt to allow the system to examine zero-day attacks. A data set was needed to introduce attacks that have not yet been processed by the system to determine its ability to adapt to new kinds of attacks. This was done in the following way. If there are T attack types, let Ω be the set of all attack types $\{t_1, t_2, \ldots, t_T\}$, where t_i is the i^{th} type. A subset of these attacks, Ω', is selected from Ω to represent the attacks that would 'enter' the system. Let Ω' be equal to $\{j_1, j_2, \ldots, j_L\}$, and L be the number of attack types in Ω'. Each instance of every attack type in Ω was removed from the original KDD data set, leaving only the normal lines. This new set with attack lines removed was divided into L+1 subsets. Let $\Delta = \{\delta_0, \delta_1, \ldots \delta_L\}$, where δ_i is the i^{th} of these subsets. Attack types from Ω' were added incrementally to each successive subset, starting with δ_1. Therefore, δ_1 contains only attack type j_1. The subset δ_2 contains attack types j_1 and j_2, and so on. The attack types found in Ω but not Ω' are added to δ_0, which is used for training SLCT and forming a baseline of signatures. Note that the relative amounts of the attacks found in the original set were preserved in each subset. This process was repeated for different selections of Ω'. Recall that there are four categories of attacks. Each of these categories was designated seperately as Ω', as well as a mix of attack types from different categories. We call the dynamic sets with different selections of Ω' as KDD_dynamic_DoS, KDD_dynamic_R2L, KDD_dynamic_U2R, KDD_dynamic_surv, and KDD_dynamic_mix.

Malicious content, training and testing, and dynamic attack modifications to the KDD were used to test the three design components described in Section 2.

3.3 Real-Time Emulator

In order to approximate a real-time flow of data, further modifications to the method that reads in data was incorporated to adjust for the amount of data produced by a single attack. We previously defined a window as representing the data collected in a set amount of time. Therefore, some types of attacks which produce a large amount of data points could skew a large number of windows. To correct for this, the method that reads in the data was modified so that the window size only applies to normal lines. This confines consecutive attack lines completely to one window. It could be argued that this method is both realistic and unrealistic, depending on the interpretation of one window, and how this approximation relates to a real-world scenario. If the system polled every one second, for instance, this would be an unrealistic approximation because it is unlikely that an entire attack would occur within this time frame.

In addition, to emulate varying amounts of activity, the size of each window is randomly chosen which centers around the mean window size ψ, specified by the user. Both randomly choosing a window size and making attacks atomic within one window make up the component we will refer to as the Real-Time Emulator. Applying a varying window size to only normal lines attempts to describe the process of collecting data from a set amount of time. Though this process may not be entirely accurate, it is an approximation that must be made given the unavailability of a suitable data set.

3.4 Performance Metrics

An essential part of performing an experiment is determining the measure of success and failure, and how the outcome can be used for other purposes. The two most common and useful metrics for intrusion detection are detection rate and false positive rate. Receiver Operating Characteristic (ROC) curves may be generated to summarize the sensitivity of the system to user parameters. ROC curves are commonly used to show the performance of binary classifiers by showing detection rate versus false positive rate for different input parameter configurations. Though detection rate and false positive rate are the most commonly reported metrics for an IDS, other metrics may be used to provide insight into the system rather than reporting performance.

$$ClusterIntegrity = \chi = \frac{\sum_{i=0}^{C} |a_i - n_i|}{C} \tag{3}$$

$$TotalClusterIntegrity = \kappa = \frac{\sum_{i=0}^{C} |a_i - n_i|}{C} \times \rho \tag{4}$$

where a_i is the percentage of attack lines in cluster i, n_i is the percentage of normal lines in cluster i, C is the number of clusters, and ρ is the percentage of total lines clustered. Using Cluster Integrity (χ) allows us to verify our previous assertion that clusters contain mostly malicious or mostly normal activity, and not a mix of the two. Total Cluster Integrity (κ) allows us to assess the clustering algorithm's ability to summarize the most amount of data while seperating malicious activity from normal activity.

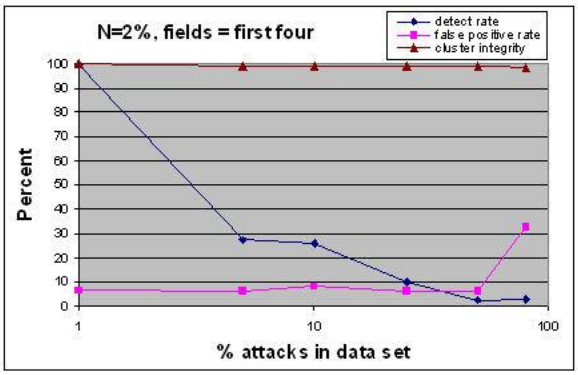
4. SIMULATION RESULTS

4.1 Clustering-based IDS

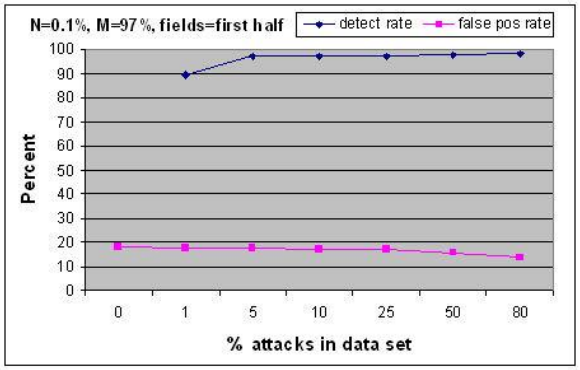
SLCT_norm was first tested as an IDS to classify each data set containing different amounts of malicious activity. A large range of user parameters, N, M, and Λ' is used to determine the configuration that yields the best detection rate and false positive rate. In general, SLCT_norm exhibits a wide range of performance in terms of detection rate and false positive rate across the different data sets. The algorithm was reliant on the parameters N and field selection, as well as the amount of malicious activity observed. Figure 5(a) shows SLCT_norm's performance when N=2% and field selection is *first four*, which yielded typical results of most other configurations. The detection rate varies considerably across different amounts of malicious activity. Despite the sensitivity to malicious content, Figure 5(a) also shows that cluster integrity is very high regardless of the relative content of the data. This indicats that SLCT performs very well at seperating malicious and normal activity into seperate clusters, which will be illustrated later in this section. In any case, SLCT_norm was very sensitive to its parameters, making it a poor overall IDS itself. And, although Figure 5(a) shows only one parameter configuration, the results shown were typical of most other configurations used.

SLCT_attack, the complementary approach, uses another user parameter, M, to place a threshold on the number of fixed attributes required for a valid cluster. A parameter configuration was chosen, namely N=0.1 and Λ'=*first half*, which corresponded to a high percent clustered as well as a larger number of attributes. This configuration will help to better illustrate the effect of the parameter M. Figure 5(b) shows the performance achieved when applying M. It shows that the detection rate is high and the false positive rate is low regardless of the malicious activity present in the data. This is the best configuration we have found for SLCT as a clustering-based IDS. Considering the high detection rate and the relatively low false positive rate of current signature-based systems, however, this is not sufficient performance. The false positive rate is much too high to be of any real use, and the detect rate is probably not indicative of how many kinds of attacks were correctly labeled. However, Figure 5(b) does support our assertion that malicicous clusters often contain more attributes than normal ones because the M parameter was in fact able to distinguish between malicious and normal clusters.

Though SLCT may not be well suited as a stand-alone IDS, it can be used for other purposes. As briefly mentioned before and displayed in Figure 5(a), the Cluster Integrity (χ) was high regardless of the amount of malicious activity present in the data. This affirms our assertions that clusters contain either mostly malicious or mostly normal activity. Eventually,



(a) SLCT_norm



(b) SLCT_attack

Figure 5. Performance Across Varying Malicious Content

we are interested in SLCT's ability to summarize data. In other words, it must be able to cluster as much data as possible while still maintaining χ. This we defined as Total Integrity (κ). It was experimentally determined that a N value of 2% yields fairly good results for integrity and number of clusters. We can now determine the field selection that yields the best κ. Figure 6 shows the Total Integrity for different field selections for $N = 2\%$. The error bars show one standard deviation of the Cluster Integrity for each cluster created, weighted by percent clustered, or ρ. A high standard deviation would mean that some clusters had high Integrity, while others were much lower than the mean. It is clear that *first four* is a good selection having the highest average and a low standard deviation. The field selection *first four* is also convenient because it contains only four fields which is less computationally expensive.

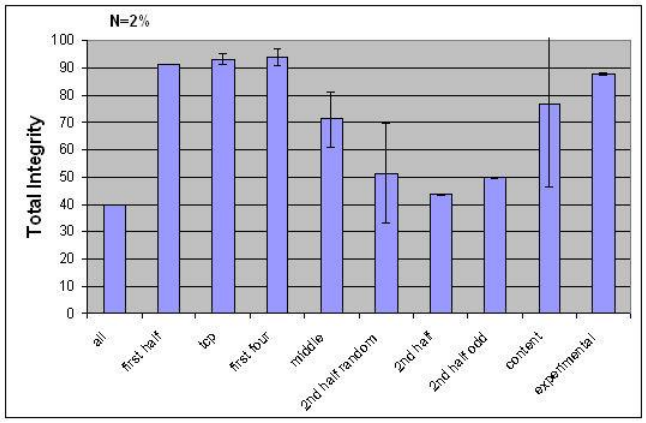


Figure 6. Total Cluster Integrity by Λ' Selection

4.2 Signature Analysis and Signature Creation

The second experiment aims at finding acceptable conditions for signature creation using SLCT with regard to the size of the training and testing sets used. The Signature Creation algorithm is run on different sizes of training sets which produce sets of signatures. We can measure the success of this endeavor by looking at the detection rate of the attack signatures when the Signature Analysis method is run on the test data. Recall that the Signature Creation algorithm purposefully attempted to reduce false positives using the validation set, so these are not expected to be significant. Overall, the detection rate was around 99%, and the false positive rate much less than 1%. However, this is largely due to the flood of DoS-type attacks present in the data. For example, instances of '*neptune*' and '*smurf*' make up 95% of the malicious activity in the KDD set. Because of this large variance in the amounts of different attack types, it is more beneficial if we look at the performance for each attack type's signatures individually. Figure 7 shows the detect rates for each attack type for different sizes of training sets.

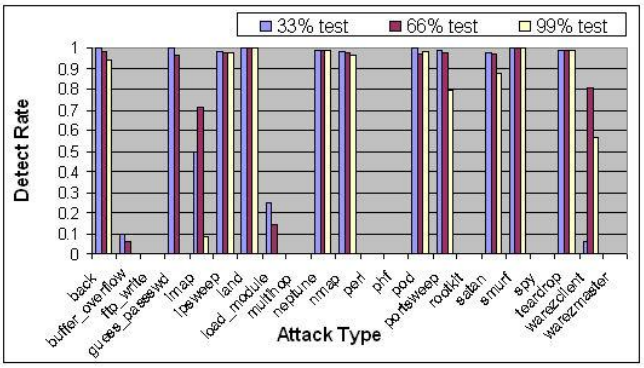


Figure 7. Signature Detect Rates

In general, there was some success demonstrated for various types of attacks, and certain cases are more interesting than others. The first case to consider is the DoS and Surveillance attack types. These include '*back*', '*ipsweep*', '*land*', '*neptune*', '*nmap*', '*pod*', '*portsweep*', '*satan*', '*smurf*', and '*teardrop*'. These attacks are by nature very predictable, and therefore created very successful signatures.

Another case are the attack types that had 0% detect rates. These include '*ftp_write*', '*multihop*', '*perl*', '*phf*', '*rootkit*', '*spy*', and '*warezmaster*'. Some of the training sets contained no lines for some of the attacks, and therefore did not yield any signatures. In some cases, because of the small amounts of some of these attacks in the original data set, the relative amounts were not preserved in the random selection of the training subsets. This made it difficult to determine the effect of training set size on signature performance. Other types, such as '*warezmaster*', are made up of activity that is too heterogeneous to be properly summarized by SLCT, and therefore difficult to create signatures for.

Finally, there were cases which did not seem to exhibit a direct relationship between detect rate and training set size, including '*warezclient*' and '*imap*'. This phenomenon is probably related to the iterative nature of the Signature Creation algorithm as well as the randomly selected training and validation sets. Because the algorithm iterates over different values of Λ' until a low enough false positive is reached, signatures could use completely different fields from one training set to another. This simple fact alone could create the unpredictable differences in detect rate we see for these attack types.

It is clear that the Signature Creation algorithm proposed can create signatures with some success, but needs modification. A process which is able to more finely select better signatures would be superior to the one implemented. Clustering may not be the best way to achieve this despite its ability to summarize multidimensional data sets.

4.3 Adaptive Signatures

The goal of this section is to demonstrate the ability of SLCT to classify new signatures in a changing data set. To do this, we use the dynamic sets described earlier. Recall that Ω' describes the set of attacks that enter the data as emulated zero-day attacks, where the different selections were split into the categories: DoS, R2L, U2R, surveillance, and a mix. For each dynamic set, the first subset is used to create a base of signatures, as well as to train SLCT_RT. The training phase is also used to collect attributes' attack supports based on the ground truth in the training set. This process is only done once for each data set, though it could be a real time process based on the classifications given to new activity based on the set of signatures. SLCT_RT as well as the signature classification and anaylsis methods were run on each of these dynamic data sets to determine if the attack types modeled as zero-day attacks could be detected in real time.

SLCT_RT has three more user parameters α, β, and ψ. These were determined through experiment, and can be seen below in Table 1. Table 2 shows the average detection rate and false positive rate for each of the Ω' selections. For three of the four main categories of attack types, a detection rate of 70 to 80% was achieved. Considering that these attacks are being detected on the fly, this is considered relatively good performance. To provide more detailed insight into the system's performance, Figure 8 shows the detection rate of each attack type within the different data sets.

One feature to note is the difference between the DoS results and the rest of the results. Both detection rate and false positive rate are significantly lower for DoS than for the other Ω' selections. It was determined that this was caused by the signatures created for the DoS activity being mislabeled as normal. One explanation for this, is that the signatures created

Parameter	Value
N	2%
Λ'	*first four*
α	0.2
β	0.2
ψ	100 lines

Table 1. SLCT_RT Parameter Values

Data Set	Detect Rate	False Positive Rate
KDD_dynamic_DoS	29.65	0.47
KDD_dynamic_R2L	72.72	7.54
KDD_dynamic_U2R	71.43	8.16
KDD_dynamic_surv	80.82	5.97
KDD_dynamic_mix	39.35	7.28

Table 2. Average Performance for Final System

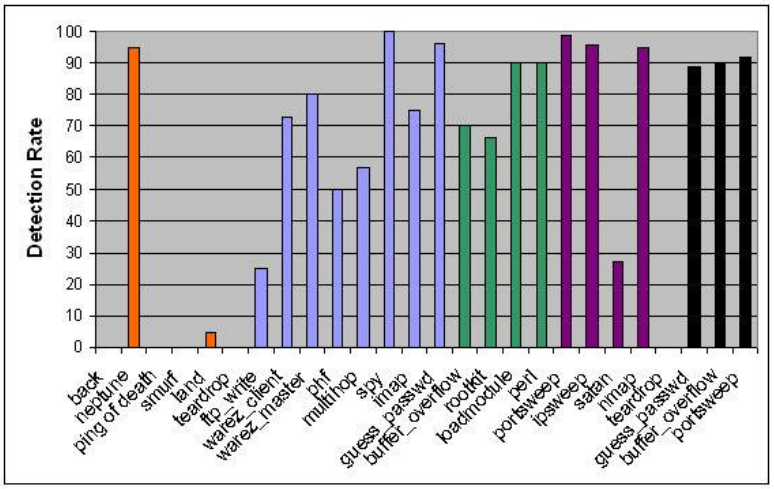


Figure 8. Detection Rate by Attack Type

for DoS attacks are too similar to normal activity in the first four fields to be classified as attack signatures. For example, the signature for '*smurf*' can be easily determined by looking at the raw KDD data. Using the [word, position] format for attributes, this signature can be written as follows.

(0, 0) (icmp, 1) (ecr_i, 2) (SF, 3)

We also know from raw data inspection that the attributes (0, 0) and (SF, 3) are common in both malicious and normal activity. The success of this signature's classification therefore depends on the relative amounts of malicious and normal activity in the training set: more malicious activity will give these attributes higher attack supports. Recall that all attack lines that did not belong to the Ω' selection in the dynamic data sets were placed in the training subset, δ_0. Therefore, we know that the data set KDD_dynamic_DoS contains much less malicious activity in its δ_0 set than the other dynamic sets. This is because there is much more DoS activity than the other types of attacks in the original KDD data. We can therefore conclude that one possible cause for the low detection rate of DoS attacks is the low attack support in the training set for common attributes such as (0, 0) and (SF, 3). Conversely, the same principle can be applied to explain the high false positive rates in the other selections of Ω' due to the large amount of DoS activity in the training sets.

The results obtained were similar to those collected in related works. Bojanic's work [16] reported an average of 66.6% detection rate for unknown attacks, though on a much smaller scale than our own. The results we obtained can be considered fairly good. The classification has obvious flaws, but shows promise because a good number of attack types were more or less successfully detected. The false positive rate for many of the attack categories was too high, though maybe more analysis could be performed to reduce this. Though the New Signature Classification algorithm did have some success in detecting new attacks, it is apparent that performance is reliant on the content of the training set, δ_0. This means that attributes' attack supports in only the first four fields can not reliably classify all new signatures.

5. CONCLUSION

Clustering algorithms have been used in a wide range of experiments spanning many areas of computing. The main contribution of this work was to investigate the use of SLCT, a density-based clustering algorithm, as it can be applied to Intrusion Detection. One conclusion we drew was that density-based clustering does have the ability of summarizing a

fairly large high-dimensional data set. It was also shown that clustering can effectively seperate malicious from normal activity in the KDD data set into different clusters both offline and in real time. These characteristics show promise for clustering as a tool that can create adaptive signatures as attacks are occurring.

Weaknesses were shown mainly when SLCT was responsible for identifying *which* clusters exactly were malicious. This was true for both offline and real time analysis. The process was, however, able to detect on the fly 70 to 80% of most types of attacks with no *a priori* knowledge of those attacks. This could be very useful in creating signatures that are used in the short term to block potential malicious activity until more reliable but time consuming methods can be applied.

The methods proposed may also be of use in other domains besides intrusion detection. Identifying and classifying a new group of data which may have similar but not identical characteristics to historical data could be applied to any real-time binary classifier or data collection and summarization method. The concepts proposed in this work are entirely experimental. Rigorous experimentation and fine-tuning would be necessary for systems similar to the ones proposed to be implemented in real system. The idea of attempting to classify a new signature shows promise, which could be augmented with the items addressed in the next section.

Areas for possible future work were identified throughout the experimentation. Other clustering algorithms, particularly distance-based ones, may provide other insights into clustering's uses for signature creation and adaptation. This would be an easily performed extension of the work, replacing SLCT and it's definitions of clusters and signatures with another algorithm that used a cluster centroid-based approach. We also saw that attributes alone were not wholly indicative of a siganture's status. The relationships between attributes, such as temporal ones, may provide further insight into a signature's classification.

The modifications made to the KDD data set were intended to produce data more representative of real network data characteristics. Having a data set or actual implementation made for testing varying amounts of malicious activity and detecting zero-day attacks would provide more accurate insight into the benefits of clustering in intrusion detection.

ACKNOWLEDGMENTS

This work is funded through the National Center for Multisource Information Fusion (NCMIF) grant under the technical supervision of AFRL/IFEA.

REFERENCES

1. Sourcefire, "Snort: An Open Source Network Intrusion Prevention and Detection System." `http://www.snort.org`, 2007.
2. Enterasys Networks Inc., "Dragon Intrusion Detection System." `http://www.enterasys.com/products/ids/DSHSS7/`.
3. S. International, "NIDES (Next-Generation Intrusion Detection Expert System)." `http://www.csl.sri.com/projects/nides/`.
4. S. Patton, W. Yurcik, and D. Doss, "An achilles' heel in signature-based IDS: Squealing false positives in Snort," 2001.
5. Y. Guan, A. A. Ghorbani, and N. Belacel, "Y-means: a clustering method for intrusion detection," in *Proceedings of CCECE 2003 - Canadian Conference on Electrical and Computer Engineering Toward a Caring and Humane Technology*, **2**, pp. 1083–6, (Montreal, Que., Canada), May 2003.
6. J.-L. Zhao, J.-F. Zhao, and J.-J. Li, "Intrusion detection based on clustering genetic algorithm," in *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, **6**, pp. 3911–14, (Guangzhou, China), August 2005.
7. S.-H. Oh, J.-S. Kang, Y.-C. Byun, G.-L. Park, and S.-Y. Byun, "Intrusion detection based on clustering a data stream," in *Proceedings of Third ACIS International Conference on Software Engineering Research, Management and Applications*, pp. 220–7, (Mount Pleasant, MI, USA), August 2005.
8. K. Sequeira and M. Zaki, "ADMIT: Anomaly-based Data Mining for Intrusions," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 386–395, 2002.
9. H. Shah, J. Undercoffer, and A. Joshi, "Fuzzy clustering for intrusion detection," in *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, **2**, pp. 1274–8, (St Louis, MO, USA), May 2003.

10. S. Jiang, X. Song, H. Wang, J.-J. Han, and Q.-H. Li, "A clustering-based method for unsupervised intrusion detections," *Pattern Recognition Letters* **27**, pp. 802–10, May 2006.
11. N. Ye and X. Li, "A scalable clustering technique for intrusion signature recognition," in *Proceedings off 2001 IEEE Workshop on Information Assurance and Security*, pp. 1–4, 2001.
12. Z. Li, A. Das, and J. Zhou, "USAID: Unifying Signature-based and Anomaly-based Intrusion Detection," in *Proceedings of 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD 2005*, pp. 702–712, (Hanoi, Viet Nam), May 2005.
13. D. F. Gong, "White paper: Deciphering detection techniques: Part ii anomaly-based intrusion detection," *Network Associates* , 2003.
14. I.-Y. Weon, D. H. Song, and C.-H. Lee, "Effective intrusion detection model through the combination of a signature-based intrusion detection system and a machine learning-based intrusion detection system," *Journal of Information Science and Engineering* **22**(6), pp. 1447–1464, 2006.
15. O. Depren, M. Topallar, E. Anarim, and M. K. Ciliz, "An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks," *Expert Systems with Applications* **29**(4), pp. 713–22, 2005.
16. I. Bojanic, "On-line adaptive IDS scheme for detecting unknown network attacks using hmm models," 2005.
17. S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proceedings of 2002 International Joint Conference on Neural Networks (IJCNN)*, **2**, pp. 1702–7, (Honolulu, HI, USA), May 2002.
18. S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal, "Adaptive neuro-fuzzy intrusion detection systems," in *Proceedings of ITCC 2004. International Conference on Information Technology: Coding and Computing*, **1**, pp. 70–4, (Las Vegas, NV, USA), April 2004.
19. R. Vaarandi, "A Data Clustering Algorithm for Mining Patterns from Event Logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations and Management (IPOM 2003)*, pp. 119–26, (Kansas City, MO, USA), October 2003.
20. "Case Study: Defending the Nation's Lawmakers Against Cyber Threats and Attacks," *Network Associates* , 2003.
21. S. Hettich and S. D. Bay, "The UCI KDD Archive." `http://kdd.ics.uci.edu`, 1999.
22. A. Clare, "Machine learning and data mining for yeast functional genomics, phd thesis," Department of Computer Science University of Wales, Aberystwyth, 2003.
23. MIT Lincoln Laboratory, "1999 DARPA Intrusion Detection Evaluation Data Set," 1999. http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html.