Statistical Intrusion Detector with Instance-Based Learning

Ivan Verdonik, Bojan Novak
Fakulteta za elektrotehniko in računalništvo
Univerza v Mariboru
Smetanova 17, 2000 Maribor, Slovenija
ivan.verdonik@siol.net

Keywords: intrusion detection, instance-based learning, reduction techniques

Received: [Enter date]

In this paper we are dealing with computer security issues. In this very broad area, we focused on intrusion detection, specifically, on statistical detection. Our statistical intrusion detector, presented in the paper, is based on Instance-based Learning with the k-nearest Neighbours method.

Statistical detector requires a good and small database of regular data to be able to validate the actual traffic correctly and promptly. Therefore we considered reduction techniques of gathered data, based on clustering. We adjusted the k-nearest Neighbours algorithm by comparing a sequence of actual data with sequences of regular data instead of comparing only one actual instance with k-nearest regular instances. For this purpose we explored four similarity measure functions. Finally, our security solution VAL (Varnostni ALarm), consisting of our statistical detector, a SNORT rule-based intrusion detection system, an iptables Linux firewall and a management console, is presented.

1 Introduction
In addition to the great opportunities and benefit for mankind, the emergence of global networking in the previous decade has brought also serious security threats to its users. Intrusion detection can be regarded as a tool that can improve the security of local network and/or individual hosts. Intrusion Detection Systems (IDS) can prevent unauthorized access to system resources and data and catch the attacker at the act. There are two main approaches to intrusion detection [6]. These are: rule-based misuse detection and statistical based anomaly detection. Each of them has its strong and weak points. Rule-based detectors are better for internal security (by that we mean security inside the company intranet). On the other hand, the strongest point of statistical detectors is the detection of novel, previously unknown kinds of attacks while they are weak at internal security. Therefore it is reasonable to combine a rule-based and a statistical detector into a hybrid detector. The latest trend is to block the attacker IP address with a firewall from the intrusion detector. Such systems are called intrusion prevention systems. Our security solution VAL encompasses all these features.

2 Instance-based Learning
Instance-Based Learning (IBL) algorithms consist of simply storing the presented training examples as well as their attribute lists and their outcome (database of regular data). And when a new instance is encountered, a set of similar, related instances is retrieved from the memory and used to classify the actual (new) instance according to the outcome of the majority of related training instances [2]. This kind of classification is called target function. The outcome is in our case either 0 - normal activity or 1 - intrusion. The following are the most common IBL target functions:

- k-Nearest Neighbor
- Locally Weighted Regression
- Radial Basis Function
IBL approaches can construct a different approximation of the target function for each distinct new instance to be classified. Some techniques only construct a local approximation of the target function that applies in the neighborhood of the new query instance and never construct an approximation designed to perform well over the entire instance space. This is an advantage when the target function is very complex, but can still be described by a collection of less complex local approximations [2].

2.1 k-Nearest Neighbour

The k-Nearest Neighbor algorithm is the most basic of all Instance-Based Learning (IBL) methods. The algorithm assumes all instances correspond to points in the n-dimensional space $R^n$. The nearest neighbors of an instance are defined in terms of standard Euclidean geometry (distances between points in n-dimensional space).

More precisely, let an arbitrary instance $x$ be described by the feature attribute list: $< a_1(x), a_2(x), a_3(x), ..., a_n(x)>$, where $a_i(x)$ denotes the value of the $i^{th}$ attribute of instance $x$. In our case attribute list of the instances consists of TCP packet header parameters. The most important parameters are: source and destination IP addresses, source and destination port numbers and status of flags.

The distance between the two instances $x_i$ and $x_j$ [3] is given by equation 1 below. This is the general form for calculating distance in n-dimensional space.

$$d(x_i, x_j) = \left\{ \sum_{r=1}^{n} [a_r(x_i) - a_r(x_j)] \right\}^2$$

Equation 1: Euclidian distance between two instances with n attributes

We do not use this distance equation exactly since we test only the equality between attributes.

In nearest-neighbor learning, the target function may be either discrete-valued or real-valued. The form of the discrete-valued target function is $f : R^n \rightarrow V$, where $V = \{v_1, v_2, v_3, ..., v_k\}$ is a finite set (in our case $V = \{\text{regularity, intrusion}\}$ and $R_n$ is real n-dimensional space. The k-Nearest Neighbours algorithm for approximating a discrete-valued target function [3] is given in algorithm 1 below:

Algorithm 1: k-nearest Neighbours

In the training part we must collect training examples instances. We collect their attribute list as well as their target. In our case instances are TCP packets. We collected the important header parameters as an attribute list.

In the classification part we first search for the k-nearest instances from the training examples closest to the actual instance (i.e. to the new instance). Then we classify this instance according to the outcome of the majority of nearest training instances. Our case is a bit specific since all our training examples are considered to be regular, i.e. all of them have only one outcome. Therefore new instance is considered regular, if its attributes are close enough to the attribute lists of training examples.

3 VAL Statistical Detector

As previously said, our statistical detector performs intrusion detection using adapted IBL with k-Nearest Neighbor method. We collected the training examples by recording
the TCP network activity on the computer plugged into university department intranet, for two weeks. So collected training examples were highly redundant and noisy. To improve the quality and to reduce the size of gathered data we first considered clustering methods. Clustering means to partition data space into \( k \) disjoint subsets so that the points in each subset are coherent according to a certain criterion. Our idea was to group TCP network packets into sets of similar packets and to preserve only packets in the center of groups. The methods we have inspected are:

- K-Means
- Mixture of Gaussian distributions used by Expectation-Maximization
- Greedy Clustering Algorithm

### 3.1 K-Means

K-Means is one of the simplest clustering algorithms. It assumes that the clusters are spherical, that every cluster has a center and that other points belonging to the cluster are close around the center [4]. See Algorithm 2 below.

#### Algorithm 2: K-Means clustering

We have \( n \) input data points and \( K \) clusters, while the output is the assignment of data points to clusters and positions of cluster centers. First, cluster centers are initialized with random values. Then, in a loop, the data points are first assigned to the cluster with the center nearest to the data point. In the next step, the cluster centers are recalculated from all the points currently in the cluster. The loop iterates until classification of all the data points to the clusters remains unchanged. The K-Means algorithm fails to find the correct clustering when clusters have different sizes and/or they have (different) elongated shapes [4].

### 3.2 Mixture of Gaussian distributions

Different models have to be used for clusters that aren’t spherical. One of them can be a mixture of Gaussian distributions. A mixture of Gaussian distributions [4] is a probability density given by

\[
f(x) = \sum_{k=1}^{K} \lambda_k f_k(x)
\]

where:

- \( f_k(x) \) are normal densities with parameters \( \mu_k, \sigma_k^2 \) called the mixture components,
- \( \lambda_k \geq 0 \) are real numbers satisfying \( \sum_{k=1}^{K} \lambda_k = 1 \), called mixture coefficients.

Intuitively, adopting a mixture reflects the assumption that there are \( K \) sources which independently generate data \( (f_1, f_2, ..., f_K) \). The probability that data is generated by \( f_k \) is \( \lambda_k \). So \( (\lambda_1, \lambda_2, ..., \lambda_K) \) represent a discrete distribution over the sources. The new data point is generated in two steps: the first source \( f_k \) is randomly picked from \( (f_1, f_2, ..., f_K) \) with a probability given by \( (\lambda_1, \lambda_2, ..., \lambda_K) \), the second data point \( x \) is sampled from chosen \( f_k \). We know \( x \), but we don’t know \( k \), the index of the source that generated \( x \). Therefore \( k \) is called the hidden variable [4].

\( f(x) \) can be rewritten to show the two-step data generation model:

\[
f(x) = \sum_{k=1}^{K} P(k) f(x | k)
\]

where:

\( P(k) = \lambda_k \) for \( k = 1, ..., K \)
\[ f(x \mid k) = f_k(x) \]

In this probabilistic framework, the clustering problem can be translated as follows. Finding the clusters is equivalent to estimating the densities of the \( K \) data sources \( (f_1, f_2, \ldots, f_K) \).

Assigning the data to the clusters means recovering the values of the hidden variable \( k \) for each data point [4].

### 3.3 Expectations-Maximization

The Expectation-Maximization (EM) algorithm [4][5] solves the clustering problem as a Maximum Likelihood estimation problem. It is based on mixture of the Gaussian distributions. It takes the data \( D = \{x_1, x_2, \ldots, x_n\} \) and the number of clusters \( K \) as the input and outputs the model parameters \( \Theta = \{\lambda_1, \ldots, \lambda_k, \mu_1, \ldots, \mu_k, \sigma_1^2, \ldots, \sigma_k^2\} \) and the posterior probability of the clusters for each data point \( \gamma_i(k) \), for \( i = 1, \ldots, n, k = 1, \ldots, K \).

For any given set of model parameters \( \Theta \), we compute the probability \( P(k \mid x_i) \) that observation \( x_i \) was generated by the \( k \)-th source \( f_k \) using the Bayes formula

\[
P(k \mid x_i) = \frac{P(k) f(x_i \mid k)}{\sum_{k'} P(k') f(x_i \mid k')} = \frac{\lambda_k f_k(x_i)}{\sum_{k'} \lambda_k f_k(x_i)} = \gamma_i(k)
\]

The values \( \gamma_i(k), k = 1, \ldots, K \) sum to 1. They are called the partial assignments of point \( x_i \) to the \( K \)-clusters - see Algorithm 3.

It can be proved that the EM algorithm converges. The parameters \( \Theta \) obtained at convergence represent a local maximum of the likelihood \( L(\Theta) \). The complexity of each iteration is \( O(Kn) \).

Clustering methods based on EM are popular because they are general and often highly effective. However when many local optima are present in the likelihood space the quality of the solution produced can be sensitive to the initial assignment of points to clusters. A larger difficulty for the anomaly detection domain is that \( K \), the number of clusters to be sought must be known a priori, yet it is not clear how to determine the number of natural clusters in a set of network packets with their parameters. Furthermore, for large \( K \) search time can be prohibitive [1].

**Algorithm 3: Expectations-Maximization**

| Input \( \{x_1, x_2, \ldots, x_n\} \) the data points, \( K \) the number of clusters |
| Output \( \gamma_i(k) \) for \( i = 1, \ldots, n, k = 1, \ldots, K \) |
| \( \mu_k, \sigma_k^2 \) for \( k = 1, \ldots, K \) the parameters of the \( K \) mixture components |
| \( \lambda_k \) for \( k = 1, \ldots, K \) the mixture coefficients |
| Initialize \( \mu_k, \sigma_k^2, \lambda_k \) for \( k = 1, \ldots, K \) with random values |
| Do |
| E step |
| for \( i = 1, \ldots, n \) |
| \( \gamma_i(k) = \frac{\lambda_k f_k(x_i)}{\sum_{k'} \lambda_k f_k(x_i)} \) for \( k = 1, \ldots, K \) |
| M step |
| for \( k = 1, \ldots, K \) |
| \( n_k = \sum_{i=1}^{n} \gamma_i(k) \) |
| \( \lambda_k = \frac{n_k}{n} \) |
| \( \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} \gamma_i(k)x_i \) |
| \( \sigma_k^2 = \frac{1}{n_k} \sum_{i=1}^{n} \gamma_i(k)(x_i - \mu_k)^2 \) |
| until convergence |

### 3.4 Greedy Clustering Algorithm

Greedy clustering algorithm [1] builds individual clusters consecutively attempting to minimize the criterion:

\[
val(C) = \frac{\sum_{x \in C} \sum_{y \in C} Dist(x, y)}{|C|^2}
\]

for each cluster \( C \). Beginning with the initial point, the cluster grows by including points, which increases \( val(C) \) the least. Growth is stopped when the value reaches a local minimum. When the cluster is complete we define its center, i.e. the point, which has the minimum distance to all other points in the cluster. Finally, the cluster is represented only by the center point and the mean radius. The complete clustering algorithm is similar to the single cluster construction. We
sequentially select individual clusters by their ability to maximize the mean intra-cluster distance:

\[ \text{val}(C_1, C_2, ..., C_n) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \text{Dist}(C_{i,\text{cent}}, C_{j,\text{cent}})}{n^2} \]

We halt the clustering process when the inter-cluster value falls below a certain threshold. This parameter defines when the clustering process will be halted and how many clusters will be created. A small threshold results in many clusters and a large one in few clusters.

3.5 Our Algorithm

After considering all of these clustering methods and a number of network packets collected by recording a network traffic, which was greater than 200,000, we had to find a computationally less demanding algorithm. First, we decided to discard all the packets whose source IP, destination IP, port number and TCP flags combination appeared only once in the collection of packets. After that, we further reduced our collection by preserving only one packet among all which had the same source IP, destination IP, port number and TCP flags combination. In this way, we reduced the number of packets to only about 500 packets.

3.6 Similarity Measure

Decision about intrusion based on only one packet is certainly unreliable. Therefore, we decided to base the decision whether there is intrusion or not by considering a sequence of packets.

We considered different kinds of similarity functions \[1\] to compare the sequences of packets. Since an exact match between the involved sequences isn’t likely, we examined four variants of loosely matching similarity functions. Furthermore, we don’t require all header data between two packets (one from actual sequence and the other from training examples sequence) to be the same but at least the source IP, the destination IP, the port and flags. First of the functions, denoted as MC-P (Match Count Polynomial), simply counts the number of matching positions between the sequences.

The next similarity function is denoted as MC-E (Match Count Exponentially). This function doubles its value for each matching position between sequences.

The next two similarity functions are based on the feeling that adjacent matches should have stronger weight. Therefore we explored the MCA-P (Match Count Adjacency Polynomial) and the MCA-E (Match Count Adjacency Exponential) function.

Similarity measure computation is the same in all four cases (only functions are different) - see Algorithm 4 below.

Set an adjacency counter \(c\) to one \((c = 1)\) and the initial value of the similarity measure to \(i, Sim = i\).

For each position \(j\) in the sequence length \(l\):

- If \(X_j = Y_j\) then \(Sim = f(Sim, c)\) and \(c = u(c)\) otherwise \(c = 1\).

After all positions are examined return the measure value.

Algorithm 4: Similarity measure computation

We have a sequence of \(l\) actual packets \(X = (x_1, x_2, ..., x_l)\) and sequence of \(l\) training examples \(Y = (y_1, y_2, ..., y_l)\). Finally, there is table with \(f(Sim, c)\) and \(u(c)\) definitions for all four types of similarity measure – see Table 1 below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-P</td>
<td>(Sim + 1)</td>
</tr>
<tr>
<td>MCA-P</td>
<td>(Sim + c) (c+1)</td>
</tr>
<tr>
<td>MC-E</td>
<td>(2\times Sim)</td>
</tr>
<tr>
<td>MCA-E</td>
<td>(Sim + 2\times c)</td>
</tr>
</tbody>
</table>

Table 1: Functions for different similarity measure computations

It was found that statistical significance of similarity functions is indistinguishable, so we used MC-P.

3.7 Other parts of VAL

We combined our VAL statistical detector with GNU licensed rule-based lightweight intrusion detector SNORT. It is used not only for rule-based detection but it serves also for
TCP network traffic capture. Traffic is stored into MySQL database. From there is accessed by the statistical detector written in GNU C. The original database schema defined with SNORT is adjusted and extended. In this way, we produced a hybrid intrusion detection system. Furthermore, we incorporated Linux iptables personal firewall to block hostile activities detected either with SNORT or with the statistical detector. Additionally, e-mail is sent to the security administrator if intrusion is detected. We also built a web management console written in PHP with access to the same MySQL database for administrative and informative purposes.

4 Results
To test the statistical detector, we used Nessus Vulnerability Scanner and generated the attacks ourselves. We executed a whole range of attacks and obtained the following results:
Total number of packets was 249003  
Number of captured packets was 234213 or 94.060%  
Number of not captured packets was 14790 or 5.940%  
Number of detected intrusive packets was 217103 or 95.273% (among captured)  
Number of undetected intrusive packets was 10771 or 4.727% (among captured)  
The results are relatively satisfying. However, at a greater regular traffic load, the result would probably deteriorate. Also, if the attacker goes slow and low, most likely nothing would be detected. However, no statistical detector performs better in similar conditions.

5 Conclusion
Security threats to our computer systems can be reduced, with the help of an intrusion detection system. A statistical detector performs intrusion detection by comparing a current activity with a knowledge base of regular activity. Our VAL statistical detector, uses Instance-based Learning with the k-Nearest Neighbor method. To improve the quality of gathered data and to reduce its quantity, we examined various clustering algorithms and finally used our own. Then we inspected functions for similarity measure computation. Since it has been found that there is no significant difference in their quality we used MC-P. We completed our solution with SNORT, the firewall and the management console. The testing has shown that our detector, combined with other components, secures computer connected to Internet quite well despite its simple construction.

Acknowledgement
Authors are thankful to Miha Strehar for sharing his experience about intrusion detection, his help at network traffic acquisition and solution testing.

References