Sequence Alignment for Masquerade Detection

By

Scott Eric Coull

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF COMPUTER SCIENCE

Rensselaer Polytechnic Institute
Troy, New York
January 2005
(For Graduation May 2005

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to begin by thanking Joel W. Branch and Dr. Eric A. Breimer for all of the work that they contributed to this intrusion detection research project. I am quite sure it would not have come as far as it has without all of your research experience. You both have not only been excellent role models for an aspiring researcher, but you have also inspired me to reach beyond what I thought was possible and continue my education as a graduate student. I can say with certainty that this thesis would never have been completed without your influences, and for that I am indebted to the both of you.

I would also like to extend special thank to my mother, Anna V. Coull, for her continued patience and encouragement as I continue my education into graduate school. You helped me stay motivated to complete both my Bachelor's and Master's degrees here at RPI, and without that motivation I would have surely failed. Thank you for your unconditional love.

Finally, but certainly not least, I would like to thank Professor Boleslaw K. Szymanski for all the encouragement and help he has contributed to my research over the last three years. Not only have you been an excellent role model and a wonderful teacher, but you have provided me with opportunities that no one else has. I can firmly say that without your guidance and patience when I first approached you as an undergraduate curious about research, I would never have progressed as far as I have. I am thankful that I was able to find an advisor that would both listen to my research ideas and encourage me to pursue them. Any graduate student would be lucky to get you as an advisor.

ABSTRACT

The masquerade attack, where an intruder assumes the identity of a valid user, as well as the insider attack, where a user utilizes their privileges to perform malicious acts, remain among the most challenging and difficult problems in the area of computer security. Many methods have been proposed to compare a user's behavior to their past and expected behavior in an effort to distinguish these malicious activities, but none have achieved a level of accuracy required for wide distribution. This paper continues previous research on the ability of a specially tuned pair-wise sequence alignment technique to detect and classify instances of masquerading in command sequences. Specifically, we evaluate the effects of various scoring systems on the sequence alignment algorithm's ability to classify masqueraders, as well as discussing modifications to the previously published algorithm that would allow it to run in a real-time environment and with a lower computational complexity. These modifications to the sequence alignment algorithm allow it to compete against the more prevalent machine learning techniques, besting all but one. Additionally, we are able to gain insight into the most useful features of the dataset used for testing, and its applicability to more realistic environments.

1. INTRODUCTION

Masquerade attacks are among the most dangerous attacks posed to information systems today, not merely because they are so difficult to detect, but also because they have the ability to undermine some of the most advanced information security technologies available today. In the typical case, security administrators rely on firewalls, network and host-based intrusion detection technologies, and strong authentication protocols to ensure there that no malicious individuals get unauthorized access to sensitive data. The masquerade attack, however, undermines all of these information security technologies by taking advantage of a legitimate user's access to the systems being secured. In the traditional masquerade attack, a malicious user takes over the user account of a legitimate user to use their privileges and rights in the information systems to carry out their malicious agenda. In a slight variant of the masquerade attack, known as the insider attack, a legitimate user chooses to use their privileges for malicious or unauthorized purposes. For the sake of simplicity, we will simply refer to both types of attacks as masquerade attacks. In both situations, the user's account escapes traditional security measures because there is a level of trust imparted upon the account after it has been authenticated. Moreover, the masquerade attack has been shown to be particularly difficult to effectively combat, as shown in the extensive research performed on the topic [4], [5], [8], [12].

In order to facilitate the detection of these masquerade attacks, a detection mechanism, known as anomaly detection, is employed to pick out areas of activity that are significantly different than the expected behavior. In order to provide this anomaly detection, a profile of 'self' is created, sometimes called a user's signature, which provides a characteristic sampling of the user's behavior over a period of time. Additionally, some algorithms make use of a 'non-self' profile, which provides a characteristic sampling of activities and behaviors that are not performed by the user. These profiles are then compared to the data being inspected for masquerade attacks to pick out anomalous behaviors.

This paper aims to extend our previous work in utilizing pair-wise sequence alignment in the detection of masquerade attacks [3] by better understanding the effects of various scoring systems on the accuracy of the sequence alignment technique in masquerade detection, as well as proposing alterations to the algorithm to decrease its computational complexity and to allow the algorithm to run in a real-time environment. In our previous work, we showed that the problem of masquerade detection can be viewed as a generalization of the longest common subsequence problem [15], and that the Smith-Waterman sequence alignment algorithm [13], which is often used in aligning and comparing sequences of biological data, can be utilized in the detection of masquerade attacks. In the initial investigation, we found that the sequence alignment algorithm could compete with the best masquerade detection algorithms published, and that it held promise of improved performance with more complex scoring systems. Here, we propose the use of five scoring systems, utilizing a broad range of features including command frequencies and probabilities, as well as specialized domain knowledge, such as command functionality, to improve upon previously reported results.

We begin by giving a detailed description and analysis of related works in the area of masquerade detection. We continue by describing the dataset used for testing, as well as analyzing its possible weaknesses and their effects on the published results. The sequence alignment algorithm is then described along with modifications to allow the algorithm to run with a reduced computational complexity, thereby making it suitable for real-time masquerade detection. We then describe the scoring systems that will be tested along with the intuitions behind their development. Finally, we conclude with the results of the testing, comparisons to other published masquerade detection algorithms, and a brief discussion of future work and the viability of masquerade detection using sequence alignment.

2. RELATED WORKS

This work on follows from a large and varied body of previous research dealing with both anomaly and masquerade detection. Though machine learning techniques, such as Bayesian classifiers and Support Vector Machines, are quite prevalent in work done in anomaly detection, there have been notable exceptions that, like our algorithm, use statistical and lexicographical techniques to classify anomalies. The earliest and most well-known of these works is the SRI IDES Statistical Anomaly Detector developed by Javitz and Valdes [4]. The IDES Statistical Anomaly Detector uses a myriad of behavioral inputs, such as file access and CPU time, from each user to compare recent metrics to the user's historical profile of behavior. These individual metrics are then summarized into a single value that summarizes the abnormality of the user's behavior in the recent past. The work continues by examining a number of methods to delineate time and gives a thoughtful summary of the strengths and weaknesses of each measure. This method, however, relies heavily on the temporal locality of metrics, and therefore may miss high-level patterns that may emerge over time. Additionally, this method has some reliance on multiple behavioral inputs, which may not necessarily be available at all times.

In a similar vein as the work presented in this paper, Lane and Brodley have presented a number of methods to detect anomalies using sequences of commands [5], [6], [7]. These methods include both statistical and machine learning methods. Their most salient work, with respect to sequences [5], uses the number of consecutive tokens, or commands, matched to create a similarity measure of two compared sequences. In many ways, this method is similar to the sequence alignment method proposed in this paper. Both methods seem to have their roots in the longest common subsequence problem, though perhaps with different semantics for matching. Furthermore, both methods assert that the spatial locality of commands is an important feature of command sequences. Their sequence matching method, however, only considers continuous sequences of matched commands, therefore ignoring patterns that may emerge from these subsequences. Additionally, the sequence matching method proposed by Lane and Brodley considers only lexicographical similarities between commands, whereas the sequence alignment method described in this paper allows for the application of additional considerations, such as command frequency or the functionality of a command, to influence the alignment through its score.

Yet another approach, proposed by Wepsi, Dacier and Debar [18], leverages the pattern discovery capabilities of the Teiresias algorithm, which is used in the realm of bioinformatics to discover patterns in DNA and protein sequences, to discover abnormalities in the audit trail of running processes. The algorithm creates a signature for a particular process by choosing significant patterns, or patterns of maximal length, that define its key features at runtime. This signature is then checked against the runtime audit trail for differences in the significant patterns displayed by the given process. A divergence from the significant patterns is labeled an anomaly when more than a predefined number of audit events differ from the signature. Again, there are several similarities between the Teiresias system and the sequence alignment method proposed here. Most notable are the common roots of bioinformatics and pattern discovery. Many areas in the analysis of biological data rely heavily on the use of pattern discovery and matching, so it is not a surprise that two algorithms that originated in the realm of bioinformatics provide similar functionality. Like the sequence matching work from Lane and Brodley, however, the Teiresias system relies on a lexicographical matching of patterns to determine what is anomalous and what is not, thereby using the patterns alone as an indication of anomaly.

There have also been a number of approaches to the specific problem of masquerade detection presented by Schonlau et. al. [12]. Schonlau et. al. propose a number of approaches ranging from simple algorithms, such as comparing the compressed size of the command sequences to determine anomaly, to more complex Markov chain models, in which command transition probabilities create a model which is then used to label anomalies in the testing data. While many approaches were proposed, we will focus on the ones that are applicable to the work presented in this paper, namely the Compression and Uniqueness techniques. In the Uniqueness method [11], each command in the sequence is given a uniqueness value, which increases with the uniqueness of the command among all of the users' commands. The most unique commands are thought to best differentiate between a legitimate user and a masquerader. These uniqueness scores are then added if they have occurred in the user's signature, or subtracted if they have not, to create a score for the sequence of commands. This score is then compared to a static threshold to determine if the activity is anomalous. The Compression method [12] approaches the problem from a different angle by taking advantage of repeated commands in the sequence. In this approach, current command sequences are appended to historical command sequence data and then compressed. The compression ratio is then compared to a threshold, which is determined by the compressibility of the historical data, to discover anomalous behavior. Both of these techniques forgo traditional lexicographical comparison in favor of using novel metrics about the given data to discover anomalies. However, as is evidenced by research mentioned above, as well as the poor performance of the Compression and Uniqueness methods, it seems that it is necessary to take into account patterns of commands rather than individual occurrences of commands alone.

In response to the work by Schonlau et. al., Maxion and Townsend published a paper that included a thoughtful analysis of the Schonlau results, as well as a new approach to the masquerade problem [8]. The approach views the masquerade problem as a type of text classification. The method utilizes a Naïve Bayes classifier, and is currently the best-known method for discerning masquerading users from legitimate users. The Naïve Bayes approach uses independent command probabilities, determined from a training corpus, to determine the overall probability of a command sequence being created by a given user. A threshold is determined by cross-validation of the training corpus, and sequences are classified based on the ratio of the probability that the sequence is from a given user to the probability that the sequence is not from a given user. This particular method, like the Compression and Uniqueness methods mentioned above, takes into account only individual occurrences of commands, rather than the patterns that emerge from these commands. The important difference, however, is that it creates what can be thought of as a pseudo-pattern by creating a probability metric for a given sequence. These metrics would then be similar for similar command sequence patterns. Additionally, this method makes extensive use of the training data by building profiles of both behavior that constitutes being the user, and behavior that constitutes not being the user, and updating these profiles with new data as it is scrutinized to make the system more robust to new and exotic commands issued by the user.

Most recently, Wang and Stolfo [16] have presented interesting work that attacks the masquerading problem with one-class training algorithms. Unlike Maxion et. al., Wang and Stolfo used only the positive training data from each user to create a profile for that user, but they do not view other training data to build a profile of behavior that constitutes not being that user. In their approach, they make slight changes to the traditional two-class Naïve Bayes and Support Vector Machine classifiers so that they can perform masquerade detection with only one-class training. In the case of the Naïve Bayes classifier, they created a user profile using only that particular user's training data, and then performed classification by taking the ratio of the probability that a given command is from that user to the probability that the command is not from that user. Because no explicit profile for not being the user has been created, they simply use an equal probability that any command is not from the user, essentially the inverse of the number of distinct commands. For the one-class Support Vector Machine, they mapped data points, both training and testing, related to the command occurrences onto a high-dimensional space. The Support Vector Machine then found a hyperplane that maximally separates the training data points from the origin of the high-dimensional space. After experimentation, they showed that one-class classification methods perform comparably to two-class methods. More specifically, their one-class Support Vector Machine using binary features bested all previous algorithms aside from the Maxion and Townsend Naïve Bayes classifier that updates command probabilities during processing.

Szymanski and Zhang also made use of the one-class Support Vector Machine technique, but with the addition of a novel recursive data mining method [14]. The method recursively mines a string of symbols, or commands in the case of the masquerade problem, and finds patterns of these symbols that occur with certain frequency. These patterns are then encoded with a new, unique symbol that represents the pattern, and the string is rewritten with the newly encoded pattern symbols. This process continues recursively until a predefined recursive depth is reached. The final strings produced by the recursive data mining method characterize the structure and high-level patterns present in the user's signature and the monitored sessions, and the one-class Support Vector Machine is then used to measure the similarity of these characterizations. This combination of lexicographical pattern discovery and machine learning was able to produce results that were superior to all previous masquerade detection algorithms, aside from the Naïve Bayes classifier with updates.

While these machine learning methods, rooted in probabilistic reasoning, currently provide the best results, we believe other methods, such as sequence alignment, which are based on statistical and lexicographical methods of anomaly discovery, can provide equally successful results.

3. TEST DATASET

3.1 Description

In order to test the performance of our algorithm, a dataset must be chosen to provide results that can be fairly compared to other efforts in the field of masquerade detection. Unfortunately, there are a limited number of standard datasets available, and therefore a limited range of test data that can be used to compare different efforts. In the case of masquerade detection, a dataset created by Schonlau et al. (SEA) has become the de-facto standard. The SEA dataset was created by recording users' commands as they were provided to the UNIX shell. These commands were recorded via the acct utility with all command arguments removed for the sake of user privacy. Commands were recorded for seventy distinct users, fifty of which were chosen as the users that would be used in the dataset. For each of these fifty users, 5,000 commands were recorded to make up the set of commands that are considered free of any anomaly or intrusion - we will call this anomaly-free command sequence the user's signature. An additional 10,000 commands were also recorded from each user to make up the set of commands that are to be tested for anomalies or intrusions – we can call this set the test data. Finally, the commands that have been recorded from the remaining twenty users are randomly interspersed into the fifty users' test data, thereby removing those users' commands and replacing them with those of the twenty users not represented in the dataset. This interspersion is probabilistic in nature, and is discussed in detail in the Schonlau et al. work [12].

The commands, in both the user's signature and the test data, are broken into 100-command groupings, which we will call blocks. We can now think of the dataset as consisting of fifty users, where each user has a user signature made of 50 blocks, and test data made of 100 blocks. Additionally, any of the 100 blocks in the test data may or may not have another user's command(s) embedded within it. The only information about the intrusions that are given to users of the dataset is a marking of which test blocks have at least one command that was created by a different user. No statements are made as to which commands, or even how many commands constitute the masquerade within the test block.

3.2 Analysis and Commentary

Many factors make the problem of anomaly detection, or the specific case of masquerade detection extremely difficult, not the least of which is the lack of reliable test data with which to compare results. In the case of masquerade detection, the only widely used dataset is the SEA data that has been described in the previous section. Unfortunately, the manner in which this dataset was created makes the task that much more difficult.

The most significant problem with this dataset is the interspersion of non-anomalous commands. It is difficult, if not impossible, to determine that a common command, used by many users, is anomalous given only the commands with no additional data. For instance, it is easy to imagine the case where all seventy users frequently used the commands *cd* and *ls* to traverse the file systems on their workstations. It is this case where additional command line data, such as the directories that the users are traversing, becomes useful in masquerade detection. Since as little as one command could be interspersed within the test block, it is not hard to see why this is considered such a difficult problem. What should also be remembered, however, is that the results gathered from this data have no direct correlation with results that would be achieved in a more realistic testing environment. Other statistical methods, such as the SRI Statistical Anomaly Detector [4] perform quite well given data with a reasonable amount of information, so it would be shortsighted to assume that the results from this dataset are directly representative of the capabilities of any algorithm tested with it. Maxion and Townsend [8] discuss similar deficiencies that they found in the SEA dataset. Later, we will revisit some of those comments and their relation to our sequence alignment algorithm. Unfortunately, until a more reasonable and reliable dataset is created, the SEA dataset is the most appropriate dataset to use to facilitate comparisons with other efforts in anomaly and masquerade detection.

4. SEQUENCE ALIGNMENT ALGORITHM

4.1 Overview

Sequence alignment is typically used in bioinformatics to determine the similarity between two DNA or protein sequences, in a global, semi-global, or local context, by aligning the nucleotides or amino acids in each sequence. This alignment then produces a score that indicates how well the sequences align with one another, and, consequently, how similar they are. This very same concept can be used to align sequences of commands to produce a score that indicates the similarity of the two command sequences. By aligning a small segment of commands with the user's signature, we can use the score of the alignment as an indicator of the presence of an intrusion within the segment that we are testing.

Sequence alignment algorithms have the ability to lexicographically compare strings to find and score matched patterns, and to take into account specific domain knowledge in this matching. This domain knowledge can be used to allow for different alignments of the data, which can then bring about new high-level pattern matching. For instance, in the domain of biological data matching a user can encode the probabilities of mutation from one nucleotide or amino acid to another in the scoring between two strings. This domain-specific scoring gives even lexicographical mismatches validity in the pattern matching based on the domain knowledge imbued in the scoring. In addition, we can use this technique not simply to match commands, but also to match generalized patterns that a user might be prone to over the course of a number of computing sessions.

4.2 Algorithm Description

We use a modification to the Smith-Waterman local sequence alignment algorithm [13] to compute a semi-global alignment for the command sequences. For intrusion detection, it is critical that we align the majority of the tested block of commands to the user's signature. If we were to allow a large prefix and large suffix of the tested block of commands to be ignored then the intrusion itself might be ignored. The problem with using a purely global alignment is that there may be large portions of the signature that do not necessarily align with a segment of the user's commands. In the case of the Schonlau et. al. data, there is a large disparity in sizes between testing blocks and the user signature, which makes global alignments totally inappropriate because it would unfairly penalize the unused portion of the user's signature. Thus, we want to design a scoring system that rewards the alignment of commands in the user segment but does not necessarily penalize the misalignment of large portions of the signature. In the remainder of this section the signature sequence, which represents the user's typical command behavior, will be referred to as *UserSig*. The monitored command sequence, which may contain a possible subsequence of masquerader commands, will be referred to as *TestBlck*.

```
Input: string UserSig of length m, string TestBlck of length n
Initialize a matrix, D, of type integer
  for i=0 to m
     for j=0 to n
         if(j=0 or i=0)
             D[i][j]=0;
         else
             if(j=n or i=m)
                 top=D[i][j-1];
                 left=D[i-1][j];
             else
                 top=D[i][j-1] – gUserSig;
                 left=D[i-1][j] – gTestBlck;
                 if(top<0) top=D[i][j-1];
                 if(left<0) left=D[i-1][j];
             diagonal=D[i-1][j-1] + matchScore(UserSig$_{i-1}$,TestBlck$_{j-1}$);
             D[i][j]=maximum(top,left,diagonal);
           return D[m][n];
```

Fig. 1. Sequence alignment algorithm applied to masquerade detection

The algorithm, which has previously been described in a paper presented at the Annual Computer Security Applications Conference [3], is shown in Fig. 1. The algorithm starts by initializing a matrix of floats, which is used to store the score throughout the alignment process. Each position *(i, j)* in the matrix corresponds to the optimal score of an alignment ending at $UserSig_i$ and $TestBlck_j$. This optimal score is computed by starting at the upper left corner of the matrix (i.e., at the point *(0,0)*) and then recursively making a step yielding the maximum from the three following options:

Option 1 (diagonal step): The score ending at position *(i-1,j-1)* plus *matchScore*($UserSig_{i-1}$,$TestBlck_{j-1}$), which is a penalty or reward for aligning the *UserSig*'s command at i-1 with the *TestBlck*'s command at j-1.

Option 2 (top-down step): The score ending at position *(i, j-1)* plus *gUserSig*, which is the penalty for introducing a gap into the *UserSig*.

Option 3 (left-right step): The score ending at position *(i-1,j)* plus *gTestBlck*, which is the penalty for introducing a gap into the *TestBlck*.

If Option 1 yields the largest value, then the optimal alignment matches $UserSig_{i-1}$ with $TestBlck_{j-1}$. If Option 2 or Option 3 yields the largest score, then the optimal alignment associates either $UserSig_{i-1}$ or $TestBlck_{j-1}$ with a gap.

There are three essential parameters used in the scoring system. The *matchScore*($UserSig_{i-1}$,$TestBlck_{j-1}$) function returns a negative value if the two commands do not match well and a positive value if they do. The *gUserSig* and *gTestBlck* are negative gap penalties associated with inserting gaps into the *UserSig* and *TestBlck*, respectively.

If Option 2 or Option 3 results in a negative value, then the alignment score is reset to zero. This zeroing of the score allows a prefix of both the *UserSig* and *TestBlck* to have an arbitrary number of un-penalized gaps. The assumption is that a portion of the *UserSig* can be ignored without penalty. Since the *UserSig* is significantly longer than the *TestBlck*, it is expected that most of the commands in the *UserSig* will not participate in the alignment. In addition, a small portion of the *TestBlck* can be ignored. However, there is a difference in ignoring portions of *UserSig* and *TestBlck*, since a high alignment score should not be achievable if a large portion of the *TestBlck* is ignored. Thus, any alignment that ignores a large prefix of the *TestBlck* should have a relatively low score. Similarly, when the algorithm reaches the right-most column or the bottom-most row of the matrix, the gap penalty is not applied. Thus, either a suffix of the *UserSig* or a suffix of the *TestBlck* is ignored. Once again, if the latter is true then the alignment score will be relatively low.

Each gap inserted into the *UserSig* corresponds to a *TestBlck* command that is ignored. Similarly, each gap inserted into the *TestBlck* corresponds to the ignored *UserSig* command. To minimize the number of ignored *TestBlck* commands, the *gUserSig* penalty is set higher than the *gTestBlck* penalty. The overall scoring scheme is designed to reward a *TestBlck* that can be almost entirely aligned to the *UserSig* with a minimal number of gaps and mismatches.

4.3 Real-time Algorithm Implementation

The current implementation of the sequence alignment algorithm in the context of the Schonlau et. al. dataset, is offline, therefore the data being evaluated is static and is not being generated in real-time. This is an appropriate mechanism for the evaluation of the accuracy of masquerade detection algorithms, but this may not be realistic in discovering intrusions in a timely manner. In order to modify the sequence alignment algorithm for use in a real-time environment, all that would have to be done is to wait for some preset number of commands to be processed, and then simply use a sliding window to perform one alignment for each window of commands of the preset size. For instance, let us assume a preset window size of 100 commands. We wait until 100 commands have been entered by the monitored user, and upon the 100th command being entered, we perform an alignment on that sequence and the user's signature. For each new command that is added to the 100-command sequence, we remove the oldest command from the sequence. Hence, when the 101st command is entered, we remove the first command from the sequence, and when the 102nd command is entered, we remove the second command from the sequence, etc. This will present a consistent 100-command sequence that can be aligned with the user's signature, and provide scoring commensurate with the sequence algorithm as described above.

4.4 Algorithm Complexity and Heuristic Approaches

The longer an intrusion goes undetected, the more damage an intruder can do. Additionally, real-time intrusion detection systems have the burden of keeping up with data as it is being produced, thereby requiring computationally efficient algorithms. Time complexity is of utmost importance in intrusion detection, and is therefore an important part of the discussion of our algorithm. In its basic form, the sequence alignment algorithm has a time complexity of *O(mn)*, where m and n are the sizes of the respective sequences being aligned. While this is not extremely computationally efficient, when considering the sizes of the sequences in the Schonlau et. al. dataset, this is an acceptable time complexity, utilizing only 500,000 iterations per alignment.

While this may be appropriate for this particular dataset or for an offline use of our algorithm, in a real-time environment an *O(mn)* complexity could be too slow. To facilitate the use of our sequence alignment algorithm in a real-time environment, we must investigate the use of heuristic optimizations of the sequence alignment algorithm to reduce the time complexity. All of the optimizations discussed here have the potential to reduce the time complexity of the sequence alignment algorithm to near-linear complexity, depending on the statistical properties of the data, thereby speeding up the algorithm significantly and allowing its use in real-time environments.

FASTA is a heuristic two-stage algorithm for aligning two biological sequences [10]. The name is derived from the term "fast all," which initially implied that it was a fast algorithm for aligning all types of biological sequences. In the first stage, the algorithm searches for all exact matches between two segments of length k, called k-tuples. Fig. 2. shows the location of k-tuples (k=1). The value of k can be determined experimentally through testing or empirically if there is prior knowledge about the input sequences and the expected length of exactly matching segments. Exactly matching segments can be efficiently discovered using a hash table with entries for every possible k-tuple. Thus, there is a practical restriction on the maximum value of k. In the second stage, the locations of these exactly matching segments are used to identify a sub-space of the dynamic programming matrix. In highly similar sequences, this sub-space will consist of a few closely spaced diagonals, as shown in Fig. 3. There are a variety of different ways to define and refine the sub-space [10]. Computing an alignment within the sub-space is likely to produce a near optimal alignment but requires significantly less computation than computing the entire dynamic programming matrix. If no exactly matching k-tuples are found there will be no sub-space to compute. However, if k is carefully selected, the lack of matching k-tuples indicates that there is no significant similarity. While this strategy has proven effective for biological sequences, it is not clear if this strategy would work for the types of event sequences inherent in the masquerade detection problem. One of the main issues is whether exactly matching segments are prevalent in different event sequences taken from the same user. A closer examination of the underlying statistics could reveal whether or not a heuristic approach such as FASTA is appropriate for masquerade detection.

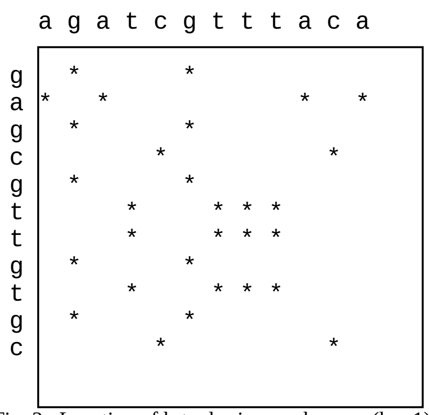

Fig. 2. Location of k-tuples in search space (k = 1).

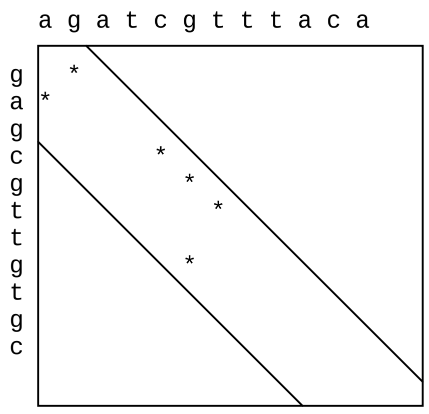

Fig. 3. Sub-region of search space (k = 2).

BLAST, the Basic Local Alignment Search Tool, uses a k-tuple based heuristic similar to that of FASTA [1]. The algorithm generates a gapped alignment by first finding the maximum matching segment pair without gaps. Maximal segment pair alignments have the very valuable property that their statistics are well understood [17]. The maximal segment match and its significance probability can be computed efficiently [1]. In turn, the un-gapped alignment can be used to help pin-point the search for a high scoring gapped alignment. The tradeoff of efficiently computing a significance probability is that the initial alignment cannot have gaps. Thus, there is the implication that two similar sequences must have rather long segments that align without introducing any insertions or deletions. While this turns out to be a reasonable assumption given biological sequences, it is once again unclear whether or not this heuristic approach is appropriate for other types of input sequences. For example, the modifications to the heuristic that improve the sensitivity for protein sequences do not work as well for nucleic acid sequences. While a BLAST-like algorithm could be applied to the masquerade detection problem, the algorithm may require significant modification.

A more generic approach to improving the efficiency of alignment algorithms is to reduce the size of the dynamic programming matrix through direct learning. This approach was successfully applied to reducing the computational complexity of the longest common subsequence problem [2]. The alignment problem is merely a generalization of the longest common subsequence problem and the approach can be applied to many different dynamic programming algorithms. Given a pair of sequences and an algorithm for computing an optimal alignment, we can generate a solution trace, i.e., a computational path through the dynamic programming matrix that will “realize” the optimal solution while performing the minimum number of computations. In the alignment algorithm introduced in the paper, the solution trace would be the chain of adjacent matrix entries needed to compute the final alignment score for a particular input. After accumulating several solution traces, any number of clustering algorithms can be used to identify a sub-space of the dynamic programming matrix that includes a high percentage of solution traces and excludes a large portion of the matrix. After a period of learning, we can use only the discovered sub-space to compute future alignments.

The intrusion detection problem is an ideal application for this type of learning. It has been observed that the solution traces of similar sequences cluster near the center diagonal of the dynamic programming matrix [2], which can yield a sub-space that is significantly smaller than the entire dynamic programming matrix. Since the intrusion detection problem only requires the alignment score, which is used as a similarity metric, it is not essential to obtain an optimal alignment. Rather, it is only important that the alignment algorithm consistently returns an alignment score that is near optimal. In fact, if a high scoring alignment does not exist close to the center diagonal then the two sequences are likely to be globally dissimilar [2]. Thus, by restricting the search for an optimal alignment to the center diagonal, we do not explicitly diminish the algorithms ability to differentiate similar sequences from dissimilar ones. Finally, since we repeatedly compare the signature sequence with various user sequences, we can learn customized sub-spaces for each signature sequence as we tune our masquerade detection system.

5. SCORING SYSTEMS

5.1 Overview

In order to properly score two aligned command sequences, we must develop a general concept of the characteristics that make a good alignment and the characteristics that make a poor alignment. The most obvious characteristic of a good alignment is a multitude of lexicographically matched commands in the two sequences being aligned. If all commands match, then the alignment is the best that can possibly occur. It isn't always the case that we can find perfectly matching commands, so the next characteristic would be mismatched commands that are in some way similar to each other. Additionally, if two mismatched commands are in no way similar to each other, then a gap may be introduced in either sequence. It is important to note, however, that a gap in the sequence of commands being tested is much better than a gap inserted into the user's signature sequence due to the importance placed on the groupings of commands in the user's signature.

From these characteristics, we are able to develop a rough concept that matched, and similar mismatched commands should be rewarded, while dissimilar mismatched commands and gaps should be penalized. Furthermore, gaps in the user's signature should be more heavily penalized than gaps in the sequence of commands to be tested. The techniques used in this paper for scoring alignments focus on the use of the various features of the SEA dataset in the scoring of matched and mismatched command alignments. The penalties for gaps in the tested blocks and in the user's signature remain constant at -2 and -3, respectively. The match and mismatch scoring, however, takes advantage of such features as command frequencies, groupings of commands based on their functions, and command probabilities. There is also a temporal relation among commands in the SEA dataset, which is defined by the chronological ordering of commands in the sequences. The sequence alignment technique takes into consideration this temporal relation in the sequences by providing a linear alignment of commands. Each of these features is given due consideration in the various techniques described below.

To facilitate proper detection, a threshold score must be determined to define at which point a score is indicative of an attack. Rather than choosing an arbitrary and static threshold score, we have decided instead to determine the initial threshold score for each user by cross-validating the user's signature against itself. We do this by taking 20 randomly chosen, 100-command sections of the user's signature and aligning them to a randomly chosen 1,000-command sections of the same user's signature. This allows us to create an initial average score that is similar to the score that the user's testing data should produce. Additionally, we update this average while new testing blocks are checked by averaging the current test block's score, and all tested block scored before it, with the initial average produced by the training data. We then take a percentage of that average as the threshold score. This allows us to customize the threshold for each user so that if a particular user did not have consistently high scoring alignments with their signature, this user's testing blocks will not be unduly flagged as intrusions. This, in particular, allows our algorithm to be somewhat forgiving of concept drift.

We are also able to choose a threshold percentage that is proportionate with the amount of sensitivity that we would like to express in the detection process by specifying the percentage of the threshold value to use as the cut off for determining what is and what is not a masquerade attack. For instance, if we are more concerned with keeping a secure environment, then we would not mind an additional amount of false positive alarms in exchange for increased masquerade detection, so we can then use a higher percentage threshold. With such a threshold, the required alignment score would have to be much closer to the user's average score to be considered a non-intrusion. Conversely, we can choose a lower threshold percentage, which would allow for a more lax security environment with less intrusive alerts by allowing the score to be significantly lower than the average score of the user.

5.2 Techniques

5.2.1 Expected Frequency

With the expected frequency technique, we provide a blanket reward for commands that match, as well as an additional reward or penalty for how often the test block's command occurs in the user's signature. More specifically, we reward the match or mismatch if the test command occurs more often than the average in the user's signature and penalize the match or mismatch if the test command occurs less often than the average. Additionally, we limit the reward that can be gained in such a way that a match will always score higher than a mismatch, despite the number of occurrences of the test command in the user's signature.

This technique draws upon the command frequencies found in the user's signature to boost the score of a sequence alignment, despite the fact that it may not have a large number of the user signature's most frequent commands. For instance, a user may have utilized a few commands many times during the creation of their user signature, but in the command sequence currently being tested they may have used that same small set much less, perhaps due to a change in a project they were working on. If we were to rely solely on matched commands, the two sequences would score very poorly in an alignment because the test command sequence does not contain these same few commands occurring many times. With the expected frequency technique, however, the matches of these commands are weighted more than other matches or mismatches, therefore raising the score of the overall alignment. Many permutations of this technique were tested, including use of command frequencies in the test command sequence, and similarity of frequencies in the two sequences being aligned. The best performing expected frequency formula for matches and mismatches is given in Fig. 4.

```
UserSig_i = User signature command at i
TestBlck_j = Test block command at j
score = Current score of the sequence alignment
freqscore = Frequency score for test block command
UserFreq_k = Frequency of command k in the user's signature
|UserFreq| = Cardinality of set UserFreq

$if\ (UserSig_i = TestBlck_j)$
        $score = score + 2;$
$freqscore = \left( \frac{UserFreq_{TestBlckj}}{\left( \frac{5000}{|UserFreq|} \right)} \right) - 1;$
$if\ (freqscore > 1)$
        $freqscore = 1;$
$score = score + freqscore;$
```

Fig. 4. Expected frequency formula.

In this formula, we use the average frequency of occurrence of commands in the user's signature as a measure of how much weight should be given to the scoring of a particular match or mismatch. Specifically, we find the number of times the test block command occurs in the user's signature, and divide it by the average frequency, which is found by dividing the size of the user's signature, 5,000 commands, by the cardinality, or the number of distinct commands, that the *UserFreq* set contains. This will then provide us with a ratio that we subtract one from. Finally, we limit the upper bound of the scale to one to provide a scoring scale of -1 to +1. In the case where the test block's command occurs more frequently than the average we add to the alignment score, and in the case where the test blocks' command occurs less frequently than the average we subtract from the score. This allows us to give preference to alignments that include test block commands that occur very frequently in the user's signature and to discourage alignments that include commands that are rarely or never used in the user's signature.

5.2.2 Odds Ratio

The log-odds ratio is a method often used for the scoring the alignments of biological data, as well as in the field of statistical analysis. It is, therefore, a natural choice for usage in the alignment of command sequences. The log-odds ratio, as the name suggests, takes the log to some arbitrary base of the odds ratio for the actual probability versus the expected probability of matching any two characters, or commands in the sequences being aligned. The odds ratio is found by dividing the actual probability of matching the two characters, or commands being aligned by the probability of matching any two characters, or commands in the sequences at random, also known as a null hypothesis.

Unfortunately, the log-odds ratio does not provide a method of scoring that is directly useable in our current scoring system due to its tendency to assign very large values for scoring, thus outweighing the static gap penalties and undermining the scoring scheme. Fortunately, a method developed by Yule [19], known as Yule's Q, provides us with the ability to utilize the odds ratio of matching commands while keeping the scoring of these commands in the range of -1 to +1, which fits quite nicely with our current scoring system framework. Yule's Q takes the ratio of the odds ratio minus one with the odds ratio plus one to give a positive score to any alignment that has a higher probability of aligning than the null hypothesis, and a negative score to any alignment that has a lower probability of aligning than the null hypothesis. Many features were used to determine the best probabilities to use for the odds ratio, including the total number of distinct commands among all user signatures, the number of distinct commands in the particular user's signature, the probability of finding a command in the user's signature, and the probability of finding a command in the test sequence. The best method is shown in Fig. 5.

UserSig$_i$ = User signature command at i
TestBlck$_j$ = Test block command at j
score = Current score of the sequence alignment
or = Odds ratio for the current commands being aligned
UserFreq$_k$ = Frequency of command k in the user's signature
|UserFreq| = Cardinality of set UserFreq

$$if\,(User\mathrm{Sig}_{\mathrm{i}} = TestBlck_{j})$$

$$score = score + 2\,;$$

$$or = \frac{\left(\dfrac{UserFreq_{UserSigi}}{5000}\right) * \left(\dfrac{UserFreq_{TestBlckj}}{5000}\right)}{\left(\dfrac{1}{|UserFreq|}\right)^{2}}\,;$$

$$score\ = score\ + \frac{(or\ -1)}{(or\ +1)}\,;$$

Fig. 5. Odds ratio formula.

With this odds ratio formula, we utilize only the frequency with which commands occur in the current user's signature. If an exact match between the commands is found, an immediate reward is given. We then use the frequencies of the two commands being aligned to determine their individual probabilities by dividing them by the size of the user's signature, 5,000 commands. We then multiply these probabilities to get the probability of both aligning, and divide this by the null hypothesis, that is, the average probability of aligning two commands in the user's signature. The result then provides us with the odds ratio, to which we can then apply Yule's Q formula by subtracting one and dividing it by the odds ratio plus one. This scoring system allows us to reward alignments of two commands that have a high combined probability of aligning based on the user's signature, and penalize the alignment of two commands that have a low combined probability of aligning. Essentially, we reinforce the natural tendency of the alignment algorithm to match commands that occur very frequently in the user's signature, despite the spatial constraints of the commands themselves, while discouraging the alignment of commands that occur very rarely, or not at all.

5.2.3 Binary Scoring

To test the importance of frequencies and probabilities are in the definition of good and bad alignments, a simpler approach was chosen as a base-line against which we can test the other scoring methods. In the binary scoring method, command alignments are scored solely on exact lexicographical matches, and the membership of the test block's command in the user's signature. This method rewards an exact match by adding one to the score for the alignment. One is also added to the score if the command occurs in the user's signature, and one is subtracted from the score if the command does not occur at all in the user's signature. This means that an exact match adds two to the score, a mismatch in which the test sequence command occurs in the user's signature adds one to the score, and a mismatch where the test sequence command does not occur in the user's signature subtracts one from the score. Simply put, this system discourages alignments of commands that are not contained in the user's signature while encouraging alignments that have occurred at some point in the user's signature. While this seems extremely simple, it provides the same type of scoring as the previous two scoring techniques without the granularity of defining how often the commands should occur in the user's signature to be considered good.

5.2.4 Command Grouping

In another diversion from the usage of frequencies and probabilities, the command grouping technique utilizes an intuitive approach to scoring command alignments. Unlike the other techniques, the command grouping technique applies only to mismatches, and not to exact command matches. In this technique, a static reward is given to exact matches. Commensurate with the other scoring techniques, this reward is set to two. During a mismatch, however, the groups to which the two commands belong are compared to determine their scoring. These groups were created from a set of common Unix commands found among all of the commands in the dataset. Each of these groups reflects the general usage of the commands within it. For instance, a group with the commands *sh*, *tcsh*, *ksh*, *csh*, and *bash* would be representative of various Unix shell programs.

With these command groups, we now have a model of mutation from one command to another, namely from one command in a group to another in the same group. Furthermore, we can say that an alignment of two commands is good when a command found in the user's signature aligns with a mismatched command in the test sequence that is in the same group, thereby representing an expected mutation from the user's command sequence. If a mismatch occurs in which the commands do not have the same grouping, then an unexpected mutation has occurred and this should be penalized. In our system, we reward mismatched commands that are in the same group by adding one to the alignment score, and penalize commands that are in different groups by subtracting one from the alignment score. This provides us with an intuitive system of dealing with the possibilities of a user utilizing different programs with similar functionality as their daily usage changes from their user signature.

5.2.4 Split Scoring

The final technique presented here draws upon the idea that it is quite possible that the properties that make for a desirable exact match may vary from the properties that describe a desirable mismatch. For instance, a good exact match may be characterized by how frequent the command occurs in the user's signature, while a good mismatch may be measured by the functional similarity between the commands. This method provides us with the ability to better distinguish between good and bad matches and mismatches, thereby allowing us to give preference to alignments that contain not only good matches, but also which contain good mismatches, as defined by the methods used to score them.

Each of the previously mentioned techniques were tried, as applicable, to score both the matches and mismatches. Furthermore, some intuition was used to develop additional techniques used to score mismatches based on the two commands' similarity to one another based on frequency of occurrence, probability, or function. The best of these combinations, described in Fig. 6., utilizes the already shown expected frequency technique to score matches, while utilizing an additional technique to score the mismatches. The additional technique essentially rewards two commands that occur with a similar frequency in the user's signature. This mixture of techniques ensures that matching commands that are especially prevalent in the user's signature will be given preference over other matching commands during alignment. Additionally, mismatches are considered good when both commands' frequencies are similar, and they are considered bad when those frequencies are substantially.

```
UserSig_i = User signature command at i
TestBlck_j = Test block command at j
score = Current score of the sequence alignment
freqscore = Frequency score for test block command
mismatch = Mismatch score for the current commands being aligned
UserFreq_k = Frequency of command k in the user's signature
|UserFreq| = Cardinality of set UserFreq

$if\ (UserSig_i = TestBlck_j)$
    $score = score + 2$;
    $freqscore = \left( \frac{UserFreq_{TestBlckj}}{\left( \frac{5000}{|UserFreq|} \right)} \right) - 1$;
    $if\ (freqscore > 1)$
        $freqscore = 1$;
    $score = score + freqscore$;
else
    $score = score + 1$;
    $mismatch = \left| \left( \frac{UserFreq_{TestBlckj}}{UserFreq_{UserSigi}} \right) - 1 \right|$;
    $if\ (mismatch > 2)$
        $mismatch = 2$;
    $score = score - mismatch$;
```

Fig. 6. Split scoring formula.

In the formula presented in Fig. 6., matches receive an immediate reward by adding two to the alignment score. This is then modulated by the ratio of the command's frequency in the user's signature to the average frequency of commands in the user's signature, just as in the expected frequency technique discussed earlier. This score is then limited to a -1 to +1 scale, thereby allowing exact matches to add between one and three to the alignment's score. For mismatches, we add one to the alignment score, and then take the ratio of the frequencies of both commands in the user's signature. We then subtract one from this ratio and take the absolute value, and, finally, limit the ratio to a maximum of two. This provides us with a score that is commensurate with the similarity of the frequencies of both commands, as found in the user's signature. If both commands have the same similarity, then we don't subtract anything from the alignment score, thereby raising the alignment score by one. If the frequencies of the two commands are significantly different, the alignment score would be lowered by a maximum of one.

5.4 Results

Based on common-sense knowledge of how a user utilizes various commands, it would stand to reason that the command grouping scoring scheme would perform the best out of the techniques presented above due to its use of a mutation model that allows the user to utilize different commands with similar functionality to create the best alignment of command sequences. Conversely, it would make sense that the binary scoring model would perform poorly because it does not use the various features, such as command frequencies or probabilities, available in the dataset to make its determination of what constitutes a good or bad match or mismatch. The other techniques should perform similarly due to their use of similar types of features in the dataset, with the split scoring technique edging out the others due to its ability to provide a more granular description of good and bad matches and mismatches. The performance of these techniques in relation to one another should provide some insight into the features and concepts which best differentiate good and bad matches and mismatches in the Schonlau et. al. dataset. To properly compare these techniques to one another, as well as to other masquerade detection algorithms, a scoring framework must be utilized. Maxion and Townsend [8] created a scoring framework that rates the overall performance of a masquerade detection algorithm as a function of its false positive rate and its false negative rate, or miss rate. In this scoring framework, a false positive is considered substantially more costly than missing a masquerade attack. In fact, the framework calculates the overall cost, or score, of a masquerade detection algorithm as six times the false positive rate plus the miss rate, as shown in Fig. 7. This score provides a realistic assessment of the various masquerade detection algorithms, with lower scores indicating a good mix of masquerade detection and false positives.

$$\boxed{Cost = (6 * FalsePositive) + Misses}$$

Fig. 7. Maxion-Townsend scoring system formula

To choose the best sensitivity for each technique being compared, we simply choose the sensitivity setting that provided the lowest Maxion-Townsend cost. We then took the best costs for each of the techniques mentioned above and compared them to each other with some very interesting results. The bar graph shown in Fig. 8. depicts the comparison of the best costs from each technique. Surprisingly, the command grouping technique that we anticipated would perform quite well performed the worst of all techniques compared. Maybe even more surprising is that the binary scoring system performed very near the best technique, split scoring. As predicted, the expected frequency and odds ratio techniques performed similarly to each other. These results imply some interesting features of the Schonlau et. al. dataset, namely that utilizing the frequency and probabilities available in the dataset provide no distinct benefit in the determination of good and bad matches and mismatches. In fact, these results seem to imply that the most reliable indicator of good matches and mismatches are the commands' membership in the user's signature. The split scoring technique seems to outshine the binary scoring technique only because of its more granular analysis of both matches and mismatches. It is important to note, however, that the training process used to create the threshold scores chooses random data, so the difference can very well be attributed to the difference in data used to create the threshold. In fact, upon closer inspection, the binary scoring method actually achieves a lower average cost, as seen in Tables I and II, among many of its sensitivity levels than the split scoring technique, thereby rendering it the best technique discussed in this paper.

TABLE I

SPLIT SCORING – FULL RESULTS

SENSITIVITY	% MISS	% FALSE POSITIVE	%HIT	COST
0%	85.8%	0.1%	14.2%	86.7
10%	84.8%	0.4%	15.2%	87.3
20%	80.6%	0.6%	19.4%	84.5
30%	75.2%	1.0%	24.8%	81.1
40%	58.7%	1.4%	41.3%	67.2
50%	41.2%	2.4%	58.8%	55.7
60%	29.5%	4.9%	70.5%	59.0
70%	18.6%	10.9%	81.4%	83.8
80%	11.5%	20.2%	88.5%	132.6
90%	9.5%	31.9%	90.5%	200.9
100%	7.4%	47.6%	92.6%	292.8

AVERAGE COST: 112.0

TABLE II

BINARY SCORING – FULL RESULTS

SENSITIVITY	% MISS	% FALSE POSITIVE	%HIT	COST
0%	85.8%	0.2%	14.2%	86.8
10%	72.8%	1.0%	27.2%	78.6
20%	61.9%	1.3%	38.1%	69.4
30%	56.3%	1.9%	43.7%	67.7
40%	50.7%	2.4%	49.3%	65.0
50%	39.7%	2.9%	60.3%	56.8
60%	36.3%	3.5%	63.7%	57.5
70%	26.4%	5.6%	73.6%	60.1
80%	19.2%	12.3%	80.8%	93.1
90%	9.7%	25.1%	90.3%	160.3
100%	6.1%	48.7%	93.9%	298.1

AVERAGE COST: 99.4

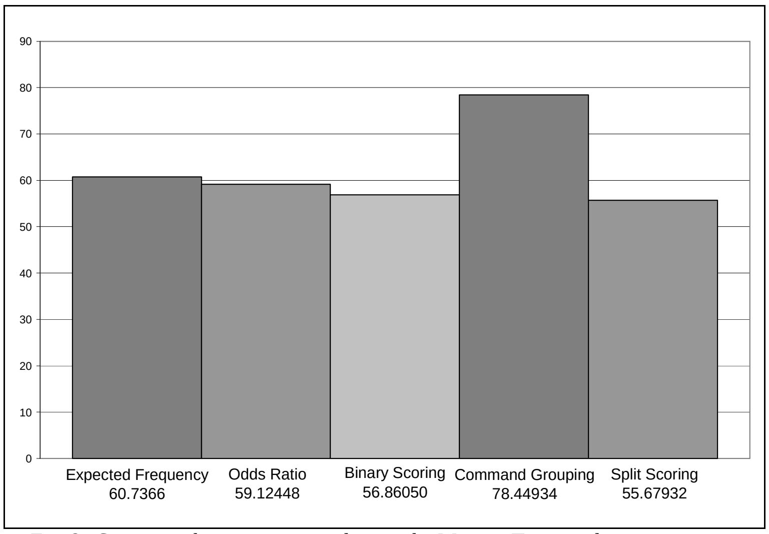

Fig. 8. Scoring techniques compared using the Maxion-Townsend scoring system.

Use of the binary scoring system also allows the sequence alignment algorithm to overtake a number of competing masquerade detection algorithms, falling short of only the Naïve Bayes method, which updates its command probabilities based on the classifications it makes. Table III provides a listing of masquerade detection algorithms, ranked by the Maxion-Townsend score, as provided by the works of Schonlau et. al. [12], and Maxion and Townsend [8]. Exact results could not be gathered from the Support Vector Machine work by Wang and Stolfo [16], but the receiver operator characteristic curve that they provide seems to be fairly close to the curve for our sequence alignment algorithm when utilizing the binary scoring system, as provided in Fig. 9. These results rank our sequence alignment algorithm near the top of all masquerade detection algorithms.

TABLE III
COMPARISON OF MASQUERADE DETECTION ALGORITHMS

ALGORITHM	% HIT	%FALSE POSITIVE	COST
NAÏVE BAYES (WITH UPDATING)	61.5%	1.3%	46.3
SEQUENCE ALIGNMENT	**60.3%**	**2.9%**	**56.8**
RECURSIVE DATA MINING	62.3%	3.7%	59.9
NAÏVE BAYES (NO UPDATING)	66.2%	4.6%	61.4
UNIQUENESS	39.4%	1.4%	69.0
HYBRID MARKOV	49.3%	3.2%	69.9
BAYES 1-STEP MARKOV	69.3%	6.7%	70.9
IPAM	41.1%	2.7%	75.1
SEQUENCE MATCHING	36.8%	3.7%	85.4
COMPRESSION	34.2%	5.0%	95.8

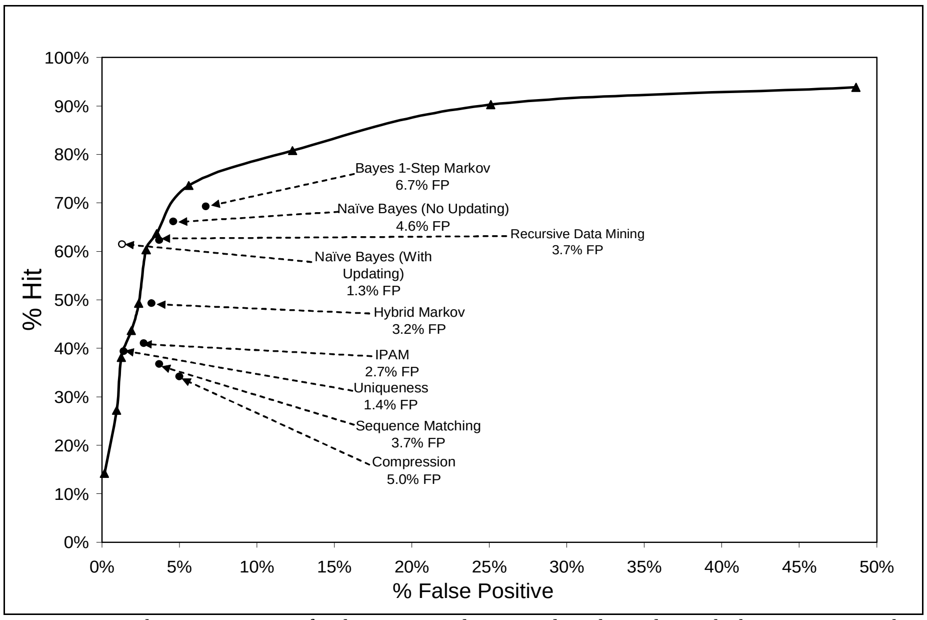

Fig. 9. Receiver operator characteristic curve for the sequence alignment algorithm utilizing the binary scoring technique plotted against other masquerade detection algorithms.

6. CONCLUSION

Through our testing of the five scoring systems, we have found the binary scoring system to perform the best at masquerade detection with a hit rate of 60.3% and a false positive rate of 2.9%. Interestingly, Wang and Stolfo [16] also used both command frequency and binary representational data in their Support Vector Machine approach and found that the use of binary scoring data actually performed better than the use of a one-class Naïve Bayes or a Support Vector Machine with the command frequency representation. It is also interesting to point out that the receiver operator characteristic (ROC) curve created by the Support Vector Machine with the binary scoring approach is very similar to the ROC curve provided by our sequence alignment method. This correlation in results seems to indicate that the binary representation of commands in the user's signature is the best feature available in the Schonlau et. al. (SEA) dataset, even though it is not the intuitive choice for feature selection.

We have also found that the pair-wise sequence alignment technique described in our previous work [3] has bested all but the Naïve Bayes classification technique with updating when using the Maxion-Townsend scoring system. One important note, however, is that while our algorithm provides a fluctuating threshold for each user, we do not update the original 5,000 command user signature, while the best Naïve Bayes classifier updates its command probabilities after each block is classified. This distinction may very well explain the difference in performance between the Naïve Bayes classifier with updating and our sequence alignment technique. An important area of future research with respect to the use of our sequence alignment technique in masquerade detection is the development of a technique for updating the user's signature as the algorithm classifies the blocks of commands being tested. If a reliable updating method can be found, it may help to provide better results than the Naïve Bayes classifier with updating.

Finally, we have found that the use of the SEA dataset continues to provide lackluster results despite the numerous and varied approaches attempted. This lackluster performance seems to stem from the inconsistent interspersion of masquerade attacks and the limited amount of information provided by the truncated command lines. Maxion and Townsend [8] provide a detailed analysis of the SEA dataset, including a description of why certain users in the dataset perform so poorly and why some users perform so well in masquerade detection, essentially echoing our beliefs about the shortcomings of this dataset. They also point out the fact that the lack of command line information could very well hamper the process of masquerade detection. Maxion has since performed research on the effects of additional command line parameters on masquerade detection [9] and found that the additional information constitutes a 9.1% decrease in the Maxion-Townsend score, as described above. This helps to confirm the opinion that additional data about the commands entered would help to pick out masquerading users, and to make masquerade detection viable in more realistic scenarios.

REFERENCES

[1] ALTSCHUL, S. F., MADDEN, T. L., SCHAFFER, A. A., ZHANG, J., ZHANG, Z., MILLER, W., AND LIPMAN, D. J. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research* 25(17), 3389-3402.

[2] BREIMER, E., GOLDBERG, M., AND LIM, D. 2003. A Learning Algorithm for the Longest Common Subsequence Problem. *ACM Journal of Experimental Algorithms* 8, Article 4.

[3] COULL, S. E., BRANCH, J. W., SZYMANSKI, B. K., AND BREIMER, E. A. 2003. Intrusion Detection: A Bioinformatics Approach. In *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, NV, December 2003, 24-33.

[4] JAVITZ, H. S., AND VALDES, A. 1991. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1991, 316-326.

[5] LANE, T., AND BRODLEY, C. E. 1997. Sequence Matching and Learning in Anomaly Detection for Computer Security. In *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, Providence, RI, 43-49.

[6] LANE, T., AND BRODLEY, C. E. 1998. Approaches to Online Learning and Concept Drift for User Identification in Computer Security. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, New York, NY, August 1998, 259-263.

[7] LANE, T., AND BRODLEY, C. E. 1999. Temporal Sequence Learning and Data Reduction for Anomaly Detection. *ACM Transactions on Information and System Security* 2(3), 295-331.

[8] MAXION, R. A., AND TOWNSEND, T. N. 2002. Masquerade Detection Using Truncated Command Lines. In *Proceedings of the International Conference on Dependable Systems and Networks*, Washington, D.C., June 2002, 219-228.

[9] MAXION, R. A. 2003. Masquerade Detection Using Enriched Command Lines. In *Proceedings of the International Conference on Dependable Systems and Networks*, San Francisco, CA, June 2003, 5-14.

[10] PEARSON, W. R. 1990. Rapid and Sensitive Sequence Comparison with FASTP and FASTA. *Methods in Enzymology* 183, 63-98.

[11] SCHONLAU, M. AND THEUS, M. 2000. Detecting Masquerades in Intrusion Detection Based on Unpopular Commands. *Information Processing Letters* 76, 33–38.

[12] SCHONLAU, M., DUMOUCHEL, W., JU, W., KARR, A. F., THEUS, M., AND VARDI, Y. 2001. Computer Intrusion: Detecting Masquerades. Statistical Science 16(1), 58-74.

[13] SMITH, T. F., AND WATERMAN, M. S. 1981. Identification of Common Molecular Subsequences. *Journal of Molecular Biology* 147, 195-197.

[14] SZYMANSKI, B., AND ZHANG, Y. 2004. Recursive Data Mining for Masquerade Detection and Author Identification. In *Proceedings of the 5th IEEE System, Man and Cybernetics Information Assurance Workshop*, West Point, June 2004, 424-431.

[15] WAGNER, R. A., AND FISHER, M. J. 1974. The String-to-String Correction Problem. *Journal of the ACM* 21, 168-173.

[16] WANG, K., AND STOLFO, S. J. 2003. One Class Training for Masquerade Detection. In *Proceedings of the 3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security*, Florida, November 2003.

[17] WATERMAN, M. S., AND VINGRON, M. 1994. Sequence Comparison Significance and Poisson Approximation. *Statistical Science* 9, 367-381.

[18] WEPSI, A., DACIER, M., AND DEBAR, H. 1999. An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. *EICAR 1999 Best Paper Proceedings*, 1-15.

[19] YULE, G. U. 1912. On the Methods of Measuring Association Between Two Attributes. *Journal of the Royal Statistical Society* 75, 579-642.