Sequence Alignment for Masquerade Detection

Scott E. Coull[1] coulls@cs.jhu.edu
Joel W. Branch[2] brancj@cs.rpi.edu
Boleslaw K. Szymanski[2] szymansk@cs.rpi.edu
Eric A. Breimer[3] ebreimer@siena.edu

Johns Hopkins University[1]
3400 N. Charles St.
Baltimore, MD 21202
(410) 516-8775

Rensselaer Polytechnic Institute[2]
110 Eighth St.
Troy, NY 12180
(518) 276-8326

Siena College[3]
515 Loudon Rd.
Loudonville, NY 12211
(518) 786-5084

Abstract

The masquerade attack, where an attacker takes on the identity of a legitimate user to maliciously utilize that user's privileges, poses a serious threat to the security of information systems. Such attacks completely undermine traditional security mechanisms, including strong authentication and intrusion detection, because the trust imparted in user accounts once they have been authenticated. Many attempts have been made at detecting these attacks, yet none of them have provided a level of accuracy necessary for practical deployment. In this paper, we discuss the use of a specially tuned sequence alignment algorithm, typically used in the field of bioinformatics, to detect instances of masquerading in sequences of computer audit data. By aligning monitored audit data with sequences known to have been produced by the user, known as the user's signature, the alignment algorithm can discover areas of similarity and ultimately derive a metric which indicates the presence or absence of masquerade attacks. Specifically, we use a specially tuned Smith-Waterman sequence alignment algorithm and investigate the use of various scoring systems on the algorithm's ability to detect masquerade attacks. Additionally, we provide methods to dynamically update the user's signature to accommodate for variations in behavior that occur over time, and describe heuristics for decreasing the computational requirements of the algorithm. Our technique is evaluated against the standard masquerade detection dataset provided by Schonlau et al. [9]. The results show that the use of our sequence alignment technique provides the best results of all known to us masquerade detection techniques.

1 Introduction

To protect information systems from unauthorized use, administrators rely on security technologies such as firewalls, network and host-based intrusion detection systems, and strong authentication protocols. If an attacker can gain access to a legitimate user account, however, these state-of-the-art security technologies are rendered useless. For instance, once a user has his password compromised by an attacker, the attacker can then utilize all the privileges available to the legitimate user without being detected due to the trust placed in the compromised account. Similarly, a malicious insider can choose to use their privileges to perform unauthorized actions. These examples describe the canonical masquerade attack, when an attacker masquerades as a legitimate user of the system to perform unauthorized and malicious actions without being subjected to the scrutiny of traditional security technologies. Clearly, such attacks pose a serious threat, and their detection often occurs long after the damage is done. The key, therefore, is to develop techniques to differentiate this malicious masquerading behavior from legitimate usage of the information system. Of course, this is quite difficult in practice, as legitimate daily activity can easily become malicious based on its context. In fact, there have been several attempts at creating algorithms for detecting these attacks, and yet a level of accuracy required for practical deployment has not been achieved [1, 5, 6, 7, 9, 10, 12, 16].

In this paper, we leverage the pattern matching abilities of sequence alignment algorithms found in the field of bioinformatics to discover masquerade attacks within sequences of information system audit data (e.g., command line entries, mouse movements, system calls, etc.). Sequence alignment algorithms are used in the field of bioinformatics to discover areas of similarity between two sequences of biological data, such as nucleotides in DNA sequences. The true value of these algorithms is their ability to align sequences not simply using lexical matching, such as string matching or longest common substring searches, but to incorporate domain knowledge and allow for certain mutations in the sequences. Customized scoring systems are used to define ‘good’ and ‘bad’ alignments based on knowledge of likely mutations found in research performed on these biological sequences. Hence, these alignments actually discover areas of functional similarity between the aligned sequences based on the scoring system utilized.

The ability of sequence alignment algorithms to find areas of similarity can be used to differentiate legitimate usage from masquerade attacks. To do so, we create a signature of the normal behavior for a given user by collecting sequences of audit data that are known to be created from legitimate use of the information system, known as the *user signature*. This user signature can then be aligned with audit data collected from monitored sessions to find areas of similarity between the two. Areas that do not align properly can be assumed to be anomalous, and the presence of many of these anomalous areas is a strong indicator for masquerade attacks. The ability to encode domain knowledge within the scoring system used by these algorithms allows us to align sequences with similar high-level functionality or behavior, despite the fact that the underlying audit data may differ lexically. This kind of specialization allows the sequence alignment technique to provide more complex pattern matching than previous masquerade detection techniques.

Though the use of sequence alignment appears to be easily adaptable to the task of detecting masquerade attacks, much of the research performed on the use of sequence alignment and its scoring systems focuses on biological applications. Unfortunately, much of this research does not have clear parallels to masquerade detection. For instance, biological sequences are typically made up of some finite alphabet of base symbols (e.g., nucleotides or proteins), while audit data has an effectively infinite alphabet of base symbols, such as the set of all possible command line entries. Moreover, significant research has been performed on the mutations of DNA and RNA, and likely mutations can be codified as probabilities gleaned from the substantial set of samples available, whereas such a model is difficult to derive for computer audit data, due to its dynamic nature. Thus, there is substantial need to investigate appropriate algorithms for aligning audit data and scoring those alignments based on the properties of the audit data sequences. To this end, we provide a modification of the Smith-Waterman local alignment algorithm [11] to properly accommodate for the alignment of computer audit data. We also investigate two novel scoring systems designed to model mutations in audit data.

There are also several problems that are inherent to the task of detecting masquerade attacks. First, the usage patterns of legitimate users can be expected to change, perhaps due to new projects or software. The use of static user signatures is therefore prone to label legitimate variations as attacks. By using the sequence alignment algorithm’s ability to discover areas of similarity, we are able to dynamically update the user’s signature as new user behavior is encountered, thereby avoiding false detection of masquerade attacks. Second, the Smith-Waterman algorithm can be considered computationally expensive, and impractical for use in detecting masquerade attacks on multi-user systems. By selectively performing alignments only on the portions of the user signature that have the highest probability of alignment, we significantly reduce the number of computations required in practice with almost no loss of accuracy in detecting masquerade attacks. Our modified alignment algorithm, along with our scoring systems and signature updating scheme, were tested on the Schonlau et al. dataset [9], which has become the de-facto standard due to its use in nearly all previous masquerade detection work. Results of the evaluation show that our system provides the best accuracy of any of the known masquerade detection techniques. These results provide encouragement for the practical use of our sequence alignment technique, and advance the state of the art in masquerade detection.

We begin by providing a discussion of previous attempts at detecting masquerade attacks, as well as other uses of bioinformatics algorithms within the field of computer security. We then provide a description of the methodology used to evaluate our masquerade detection technique and its various improvements. We continue by describing our modifications of the Smith-Waterman local alignment algorithm [11], the scoring systems developed for use in aligning computer audit data, our user signature updating mechanism, and finally the use of heuristics to reduce the computational requirements of the alignment in detecting masqueraders. For each improvement, we provide its evaluation in-line for ease of reading. We progressively adapt our sequence alignment algorithm from the most simplistic version given in previous, exploratory work [1], to the most sophisticated, taking advantage of the custom scoring systems, signature updating, and reduction of computational requirements. We conclude by assessing the impact of this algorithm on the state of the art in masquerade detection.

2 Related Work

The problem of detecting masquerade attacks was introduced by Schonlau et al. [9,10]. In those works, a number of possible methods for detecting attacks were discussed, though most approaches focused on statistical models. One of the proposed methods, for instance, used the uniqueness of a command in a sequence of command line entries as an anomaly metric. If a particular command was rarely used previously, its score would be proportionally lower than a command that was used more often. Another approach was the use of compression algorithms on the sequence of commands to create a compression ratio that would be used as an indicator for anomalous activity. The underlying idea in the methods proposed by Schonlau et al. was that legitimate sequences of command line data should be consistent with the commands found in the user's signature, and any deviation would indicate possible masquerade attacks. Of course, these approaches have several shortcomings, including ignoring sequencing information by assuming command independence, ignoring command functionality, and ignoring variations in human behavior by unduly punishing any change from past command line entries. Lane and Brodley [3, 4, 5] take a string matching approach by attempting to lexically match subsequences of the user's signature with subsequences of the monitored sequence, and used the number of commands that were matched to create a similarity metric. The method proposed by Lane and Brodley, like Schonlau et al., ignore the underlying functionality of the commands in the sequences, instead relying solely on finding exact lexical matches.

There have also been several attempts at applying machine learning techniques to the problem of masquerade detection. Maxion and Townsend [6] provide the best results of all past techniques by applying a two-class Naïve Bayes classifier to the problem, one class being the commands entered by the user and one being the commands entered by all other users. The most important contribution made by Maxion and Townsend is the use of updating mechanisms which dynamically update the classifier probabilities as monitored sequences are classified. Thus, this approach adapts to changes in user behavior. However, despite the performance of the classifier, several features including sequencing information and functionality are ignored. Wang and Stolfo [16] apply one-class Naïve Bayes and Support Vector Machine classifiers, and find that their results are comparable to the results of the two-class classifiers. This approach suffers from the same issues as the Maxion and Townsend approach by ignoring sequence and functionality information. Szymanski and Zhang [12] also use a one-class Support Vector Machine, but also implement a novel recursive data mining strategy to perform dimensionality reduction. Unlike the Support Vector Machine of Wang and Stolfo, Szymanski and Zhang do provide some consideration for sequencing information in their dimensionality reduction technique, but functionality is ignored as is the possibility of variation in user behavior.

There have been several other applications of bioinformatics tools to problems in the field of computer security. Wespi, Dacier, and Debar [17] were among the first to consider the use of bioinformatics techniques beyond biological data when they applied the TEIRESIAS pattern discovery algorithm to sequences of system call data. This algorithm finds recurring patterns of maximal length sequences and uses these recurring patterns to build a database of valid system call sequences. More recently, Tandon, Chan, and Mitra [13] used the concept of motifs, or conserved areas of recurring behaviors, to define behavior in sequences of audit data and define anomalies within the data. Wright et al. [18, 19] use Hidden Markov Models, which are typically used to perform alignment of many biological sequences at once, to detect the presence of various application protocols within an encrypted tunnel.

Previous work on using sequence alignment techniques to detect masquerade attacks focused on an exploration of the technique and the design choices in tuning the algorithm for use in masquerade detection [1]. This prior work, however, ignored the use of domain knowledge in the development of scoring systems, and made no attempt to address issues involved in dynamically updating user signatures to variations in behavior. In our current work, we focus on formalizing the insights gleaned from exploration of the sequence alignment techniques, and provide an evaluation of methods for updating the user's signature and using domain knowledge in scoring alignments. Through this deeper exploration, we are able to provide a system which performs substantially better than those previously proposed.

3 Evaluation

To evaluate our sequence alignment method we compare our results to those of previously published methods by using the Schonlau et al. dataset [10]. The Schonlau et al. dataset has become the de-facto standard due to its wide use and public availability. In fact, all previous masquerade detection techniques were evaluated, at least in part, on the Schonlau dataset, making comparison among masquerade detection algorithms straightforward. The Schonlau 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 make up the dataset. For each of these fifty users in the dataset, 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*, or training set. An additional 10,000 commands were also recorded from each user in the dataset to make up the set of commands that are to be tested for intrusions. We can call this set the *test data*. The commands that have been recorded from the remaining twenty users are randomly interspersed into the fifty users' test data, thereby replacing the commands of the fifty users with those of the twenty users not represented in the dataset. This replacement is probabilistic in nature, and is discussed in detail in the Schonlau et al. work [10].

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 signature made of fifty blocks, and test data made of one hundred blocks. Additionally, any of the one hundred blocks in the test data may or may not have another user's command(s) embedded within it. Our task, therefore, is to align the 100-command test blocks to the 5,000-command user signature sequence, and determine if the resultant alignment is indicative of a masquerade attack. The only information about the intrusions that are given by the dataset is a marking of which test data blocks have at least one command that was inserted from a different user's recorded sequence. No statements are made as to which commands, or even how many commands, constitute the masquerade attack within each test block.

The Schonlau dataset obviously has several weaknesses which limit its realism. The lack of detailed information about which commands were intrusions in the test blocks makes a thorough analysis of masquerade detection techniques difficult at best. Also, the manner in which masquerade attacks are interspersed among the true command data is far from realistic. In some cases, only a single command could be inserted, which could be as innocuous as a command to change the working directory. Certainly, such an 'attack' is far less dangerous and realistic than one that performs a longer sequence of malicious actions. The primary weakness of this dataset, however, is its lack of command arguments. This makes it difficult to perform a realistic evaluation of the dataset. Since the masquerade detection techniques are typically not designed to use arguments, augmenting them to include arguments may have unknown effects on their performance. There is also some concern that an attacker could perform a mimicry attack [15] on detection systems that do not use argument data. In the mimicry attack, an attacker utilizes knowledge of the user's signature to craft their system usage in such a way that it would evade detection by the masquerade detection techniques. If, however, command argument data is included, such techniques become much more difficult as the attacker would also have to use similar arguments, not simply similar commands.

Though there are several negative aspects to this dataset, it is the most widely used and generally accepted dataset in the area of masquerade detection, and is necessary to facilitate comparison of performance across a variety of techniques[1]. There is is also anecdotal evidence provided by an eval-

[1]We regret that the results of Wang and Stolfo [16] do not provide specific false positive and true detection scores,

uation of a proprietary dataset by Maxion [7] which suggests the Schonlau dataset provides adequate evaluation of masquerade detection techniques. Maxion found that by including command argument information with his existing Naïve Bayes machine learning technique actually achieved an increase of approximately 20% in performance when compared to the Schonlau data. This result indicates that the Schonlau dataset offers some benefit as a lower bound on the performance of the masquerade detection techniques. The results are bolstered by the fact that the sequence alignment technique is easily augmented to include argument data simply by including both the command and its arguments as base symbols in the sequences being aligned. Finally, while the Schonlau dataset considers the use of command line entries as the base symbols of the signature and test sequences, we reiterate the fact that the sequence alignment technique can just as easily be made to detect anomalies within a variety of computer audit data.

In previous work, Maxion and Townsend created a scoring framework that rates the overall performance of a masquerade detection algorithm as a function of its false positive rate, or the percentage of legitimate user blocks that were incorrectly labeled as masquerade attacks, and its false negative rate, or the percentage of masquerade attack blocks that were incorrectly labeled as legitimate [6]. Such a framework is quite useful in providing a summary of the performance of a detection technique. The framework provided by Maxion and Townsend calculates the overall cost, or score, of a masquerade detection algorithm as six times the false positive rate plus the miss rate. The choice of coefficients should give some indication of the relative negative costs for raising false alarms, and failing to catch masquerade attacks when they occur. For the sake of consistency with previous work, we keep the coefficients at six and one, respectively. The choice of a false positive coefficient of six and a miss coefficient of one is somewhat arbitrary, but provides a realistic assessment of the various masquerade detection algorithms, where false positives are far more costly to administrators than misses, due to their disruptive nature. We use the so-called Maxion-Townsend score, as well as the more traditional receiver operator characteristic (ROC) curve, to compare the overall performance of previously published techniques to our sequence alignment algorithm.

4 Sequence Alignment Algorithm

Sequence alignment algorithms are used in the field of bioinformatics to find areas of similarity between two biological sequences, such as DNA or protein sequences. The problem of sequence alignment, however, can be viewed as a generalization of the longest common subsequence problem [14]. In the longest common subsequence problem, we are given two strings, $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ with $n \leq m$, created over alphabet Σ. The goal is to find the maximal length subsequence of A and B such that the subsequences are lexically similar (i.e., exact string matches). This is achieved by deleting characters from A and B in such a way that the resultant strings are the maximal length sequence that A and B have in common. Similarly, we can also consider this as a type of alignment, where instead of deleting characters we simply insert gaps into the sequences such that the areas that are matched are the common subsequence between the two sequences. For instance, notice the gaps that have been inserted in both the upper and lower sequences found in Figure 1 so that the subsequences can be properly aligned.

```
-  -  G  T  G  A  C  A  T  G  C  G  A  T  -  -  A  A  G  A  G  G
      |     |  |  |     |  |  |  |  |  |        |  |  |
G  G  G  A  G  A  C  -  T  G  C  G  A  T  A  C  A  A  G  -  -  -
                    - = Gap , | = Match
```

Figure 1: Example sequence alignment of DNA sequences

Beyond simple lexical matching, sequence alignment allows for a scoring system that can be used to provide preference to certain matches while discouraging others. In bioinformatics, such scoring systems are based on observed mutations from one symbol to another (e.g., from one nucleotide in a

and thus direct comparison is not possible. One can assume their results are aligned closely with those of Maxion and Townsend's naïve Bayes approach, however

DNA sequence to a different nucleotide). Thus the discovery of the maximal subsequence between two biological sequences is not simply based on lexical matching, but on the biological plausibility of two sequences having similar functionality, or having been descended from a similar predecessor sequence, also known as phylogenetic similarity. An example can be seen in Figure 1, where mismatched symbols are aligned despite lexical dissimilarity due to the fact that mutation from one to the other is allowable in the scoring system used.

Moreover, the sequence alignment algorithm has several modes of alignment: global, semi-global, and local alignments. These various alignments allow the algorithm to search for subsequences with different characteristics. The global alignment, known as the Needleman-Wunch algorithm, tries to maximize the length of the subsequence over the entire length of both strings [8]. This type of alignment is useful in situations where both strings are approximately the same length, and where it is believed that the entirety of both sequences should be similar, such as the case in phylogenetic similarity searches on DNA sequences. The local alignment, known as the Smith-Waterman algorithm, instead focuses on finding the best aligned substrings of the two sequences over all possible substrings, rather than over the entire sequence [11]. The local alignment is useful when searching for areas of functional similarity between sequences, such as protein sequences, or when one sequence is significantly longer than the other. Since the majority of the sequence can be assumed to be dissimilar except for the area of functional similarity, global alignment will provide a poor alignment, whereas local alignments can focus the alignment on the conserved area that has the best functional similarity. This local alignment is achieved by allowing the algorithm to ignore both prefixes and suffixes of the sequences to find the best possible matching subsequence. Finally, semi-global alignment allows for large areas of the sequences to be aligned as in global alignments, but take advantage of the conserved nature of local alignments. Like the local alignment, the semi-global alignment allows both prefixes and suffixes to be ignored, but rather than only aligning the conserved area with the maximal similarity, the entire sequence is still aligned as in the global alignment case. In essence, the semi-global alignment is a slight modification to the traditional Smith-Waterman algorithm which aligns the entire sequence based on the alignment of several conserved areas of similarity. Such an alignment is particularly useful in situations where the alignment of the entire length of the sequences should be dictated primarily by the alignment of several small, conserved subsequences.

4.1 Detecting Masquerade Attacks

We can draw several parallels between searching for similarity within sequences of biological data and searching for signs of masquerade attacks within computer audit data, and the Schonlau command line data specifically. Like bioinformatics sequence alignment, we can define the masquerade detection problem as a form of sequence alignment. In the masquerade detection problem, we are given two strings $Signature = a_1a_2...a_m$ and $Test = b_1b_2...b_n$ with $n \leq m$. The string $Signature$ represents the sequence of audit data gathered from normal usage of the system by a given user, or the user's signature. The string $Test$ represents the sequence of audit data gathered from the currently monitored session in which we wish to detect masquerade attacks. These sequences are made from an alphabet, Σ, which is defined by the type of audit data being recorded. In the case of the Schonlau data, this alphabet consists of commands that can be entered at the command line, not including their arguments. Note that this alphabet is effectively infinite for our purposes, and this represents a significant difference from use of sequence alignment with biological data. Our goal is to find the areas dissimilarity between $Signature$ and $Test$, and determine if the extent of this dissimilarity indicates a masquerade attack. This is in fact equivalent to aligning the two strings to find the areas of similarity, and assuming areas of dissimilarity, as indicated by alignment with gaps or lexical mismatches, are indicative of possible masquerade attacks.

Aligning Sequences The alignment algorithm, shown in Algorithm 1, uses the principle of dynamic programming to discover the optimal alignment over all possible alignments. The algorithm begins by initializing an $m + 1$ by $n + 1$ matrix. Starting at position $(0, 0)$ (the upper left most corner) in the matrix, we iterate through each position in the matrix and determine its value through a choice of three

Algorithm 1 *Align*(*Signature* of length m, *Test* of length n)

```
for $i = 0$ to $m$ do
    for $j = 0$ to $n$ do
        if $i = 0$ or $j = 0$ then
            $D[i][j] \leftarrow 0$
        else
            if $i = m$ or $j = n$ then
                $top \leftarrow D[i][j-1]$
                $left \leftarrow D[i-1][j]$
            else
                $top \leftarrow D[i][j-1] + gapSignature$
                if $top < 0$ then
                    $top \leftarrow D[i][j-1]$
                end if
                $left \leftarrow D[i-1][j] + gapTest$
                if $left < 0$ then
                    $top \leftarrow D[i-1][j]$
                end if
            end if
            if $Signature[i-1] = Test[j-1]$ then
                $diagonal \leftarrow D[i-1][j-1] + match$
            else
                $diagonal \leftarrow D[i-1][j-1] + mismatch$
            end if
            $D[i][j] \leftarrow max(top, left, diagonal)$
        end if
    end for
end for
return $D[m][n]$
```

transitions to that position:

1. Diagonal Step: Indicates an alignment between the $i-1$ symbol in *Signature* with the $j-1$ symbol in *Test*. The alignment score is added to the value of the matrix position at $(i-1, j-1)$. The alignment score is dependent on the score provided to the alignment of the symbols by the scoring system, and whether the symbols being aligned are a lexical match or mismatch.

2. Vertical Step: Indicates the insertion of a gap into *Signature*, and alignment of the gap with the $j-1$ symbol in *Test*. The gap penalty is added to the value of the matrix position at $(i, j-1)$. The gap penalty for this transition is dependent on the scoring system used.

3. Horizontal Step: Indicates the insertion of a gap into *Test*, and alignment of the gap with the $i-1$ symbol in *Signature*. The gap penalty is added to the value of the matrix position at $(i-1, j)$. The gap penalty for this transition is dependent on the scoring system used.

The maximum value of these three possible transitions is used as the value for the matrix position and indicates the actual alignment made. Thus, given the dynamic programming principle, each position, (i, j), in the matrix represents the score of the optimal alignment of all symbols up to location $i-1$ in *Signature* and $j-1$ in *Test*. By induction, one can see that the score given in position (m, n) represents the score of the optimal alignment of the two sequences given the scoring system, and by tracing the transitions made in deriving that score we can recreate the alignment of the two sequences. The resultant score at the (m, n) position of the matrix represents a metric for the similarity of the two strings according to the scoring system used, and we use this as an indicator of the presence of masquerade attacks.

Note that in our masquerade detection problem, the *Signature* sequence must necessarily be larger than the *Test* sequence due to the fact that it must record the user's behavior over long periods of time to create a useful profile of behavior. Thus, it is prudent to choose a type of alignment that allows the entire *Test* sequence to be aligned, but whose alignment is dictated by conserved areas of similarity within the sequences — semi-global alignment is an excellent choice for such behavior. In our semi-global implementation, there are some cases where the transitions given above are altered. To allow for a prefix of the sequences to be ignored, the 0^{th} column and 0^{th} row have a gap penalty of zero for gaps in either sequence. Therefore, the prefixes can be ignored simply by inserting gaps into either sequence, with no penalty to the resultant scoring when alignment between symbols begins in earnest. Similarly, the gap penalties for the m^{th} column and n^{th} row are set to zero for both sequences. Thus, after the end of one of the sequences is reached, we align the remained of the opposite sequence with gaps with no loss of score. Also, while performing the alignment, if at any point the score of the alignment becomes negative due to gap penalties within the central positions of the matrix, the score at that position is reset to zero. This allows us to delineate the areas of similarity within the sequences, and use only the largest contiguous area of similarity in the final score calculation.

Scoring Alignments The scoring system itself is made up of four variables that can be defined to alter the behavior of the alignment, and create alignments that are based on domain knowledge, such as known mutations in biological sequences. The *match* variable determines the value added to the score at a given matrix position when a diagonal step is made and the two symbols being aligned are an exact lexical match. The *mismatch* variable determines the value added to or subtracted from the score based on the plausibility of mutation from the symbol in *Signature* to the symbol found in $Test$. Of course, this variable can be made to take into account very complex models of mutation, and has a significant amount of power in defining the alignment. The *gapSignature* and *gapTest* variables determine the values subtracted from the score due to the introduction of a gap within the *Signature* or $Test$ sequence, respectively.

Intuitively, the choice of functions for defining these variables should represent the preference of the respective alignments (e.g., match, mismatch, gap) within the optimal alignment of the sequences. In our case, it is clear that an exact lexical match of all symbols is the optimal situation — one in which the user repeats his behavior exactly as it is captured in his signature. A lexical mismatch between symbols could indicate either a positive or negative alignment based on the specific symbols being aligned. In the following section, we examine the ways in which we can define 'good' and 'bad' mismatches through custom scoring systems. Finally, the use of gaps is certainly the worst possible case of alignment, as it means that the sequences are completely different according to the scoring scheme used. In the case of masquerade detection, however, there is a slight distinction between gaps found in the *Signature* sequence and those found in the $Test$ sequence. The *Signature* sequence represents the known behavior of the user and thus altering the temporal properties found in this sequence by inserting gaps is certainly much more detrimental to an alignment than the insertion of gaps into the $Test$ sequence. Therefore, a gap in the $Test$ sequence is preferable to a gap that alters the known behavior of the user in the *Signature* sequence.

Example As a concrete example, consider the alignment of the two strings in Figure 2. This example shows the semi-global alignment of the strings "ABCD" and "ABECD" with a match having a value of +2, a mismatch having a score of +1, a gap in the "ABCD" sequence having a score of -2, and a gap in the "ABECD" sequence having a score of -3. This scoring system represents the relative importance of each type of alignment (match, mismatch, and gap) in determining the similarity of the sequences. The bold numbers within the matrix shows the transitions made by the optimal alignment, which is depicted below the scoring matrix. Notice the lack of gap penalties for the prefix and suffix columns and rows, as described above. Also, note that the score provided in the lower right position of the matrix represents the score of the optimal alignment. The maximum possible score is 8, based on the fact that the minimum sequence length is 4 and exact matches represent an addition of 2 points to the score each. Therefore, this alignment has scored very well and the two sequences can be considered similar under the provided scoring system.

Masquerade Detection Threshold It remains to be seen how exactly the score provided by the alignment algorithm can be used to detect masquerade attacks. These scores can have a range from zero, indicating that the entire sequences were ignored as a prefix or suffix, to the minimum length of the two sequences multiplied by the match score, indicating that the entire minimum length sequence

		A	B	C	D
	0	0	0	0	0
A	0	**2**	1	1	1
B	0	1	**4**	2	2
E	0	0	**2**	3	3
C	0	1	2	**4**	4
D	0	1	2	4	**6**

```
A B _ C D
A B E C D
```

Figure 2: Example alignment using semi-global Smith-Waterman alignment and the resultant alignment

was matched exactly. However, in the context of masquerade detection, these scores are dependent on the consistency of the user's behavior. For instance, if the user were to use only a single command, then one would expect the alignment of his monitored sequences to his signature to produce the maximum score at all times. Of course, user behavior is very much distinct, and therefore setting a static threshold is inappropriate. Moreover, the consistency of the user's behavior may actually change over time due to projects requirements, or other individual considerations.

Given these constraints, we choose to create a dynamic threshold for each user. This threshold is calculated by taking 20 random, 1,000-command subsequences of the user's *Signature* sequence, and aligning them to 20 non-overlapping, 100-command subsequences of the user's *Signature* sequence. The average of the resultant scores of these alignments represents the typical consistency of the user's behavior as depicted in his signature. As alignments to the monitored audit data are made, this average is updated with the latest scores, thus allowing for slight changes to the user's consistency over time. To determine the threshold for detecting masquerade attacks, we take a percentage of this average. Thus, the percentage represents the sensitivity of the detection mechanism. For example, if we set the percentage, or sensitivity, to 50% then any alignment that scores lower than half the user's average is classified as containing a masquerade attack. This sensitivity allows administrators to tune the security provided by the system. A higher sensitivity value would catch more masquerade attacks, but may expose the system to a number of false positives created by normal user activity. Conversely, a lower sensitivity would only catch the most onerous masquerade attacks without raising false alarms, but more subtle attacks may go undetected.

4.2 Sequence Alignment Results

To provide context for the evaluation of our custom scoring system, signature updating strategy, and heuristic speed up method, we provide the results from our previous, cursory study of the sequence alignment technique's ability to detect anomalies in sequences of audit data. In our exploratory study, we implemented the Smith-Waterman semi-global alignment algorithm with a simplistic scoring system [1]. The scoring system for this initial study was set simply to provide a score of +1 to exact matches, -2 for gaps created in the *Test* sequence, and -3 for gaps created in the *Signature* sequence. The mismatch score was derived through calculating the frequency of occurrence of the mismatched commands in the *Signature* sequence. If the frequency of the commands within the *Signature* sequence is greater than the average frequency of occurrence of commands in the *Signature*, the mismatch is given a positive score in the range [1,0). If the frequency is less than the average, the mismatch is given a negative score in the range (0,-1]. Of course, if the frequency is exactly the average, then the mismatch score is 0. This scheme represents a very simplistic model where all matches are weighted more heavily than mismatches, which are weighted more heavily than gaps in the test sequence, and so on.

Despite the simplicity of the scoring system, the approach yielded encouraging results, provided in the ROC curve of Figure 3 and the table of results in Table 1. According to the ROC curve, the technique is only bested by the Naïve Bayes approaches of Maxion and Townsend [6], and the Support Vector Machine of Szymanski and Zhang [12]. Though the use of Maxion-Townsend score provides a

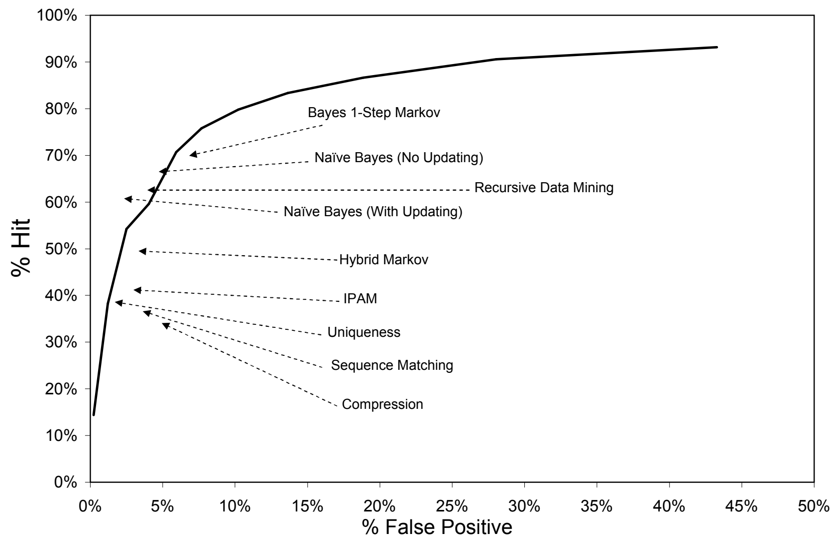

Figure 3: Receiver operator characteristic curve for previous result

very different picture of the best performance when compared to other techniques, showing the sequence alignment technique as having only middling performance. We shall use these results as a baseline for the improvement of our system as we examine each improvement to the system, in turn.

Technique	% Hit	% False Positive	Maxion-Townsend Score
Naïve Bayes (With Updating) [6]	61.5	1.3	46.3
Recursive Data Mining [12]	62.3	3.7	59.9
Naïve Bayes (No Updating) [6]	66.2	4.6	61.4
Uniqueness [10]	39.4	1.4	69.0
Hybrid Markov [10]	49.3	3.2	69.9
Sequence Alignment [1]	**75.8**	**7.7**	**70.4**
Bayes 1-Step Markov [10]	69.3	6.7	70.9
IPAM [10]	41.1	2.7	75.1
Sequence Matching [3,10]	36.8	3.7	85.4
Compression [10]	34.2	5.0	95.8

Table 1: Comparison of published masquerade detection technique sorted by Maxion-Townsend Score

5 Scoring Systems

In order to properly score two aligned command sequences, we must first 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 lexically matched commands in the two sequences being aligned. If all commands match, then the alignment is the best that can possibly occur. It is not always the case that we can find perfectly matching commands, however, so the next characteristic would be lexically mismatched commands that are in some way similar to each other based on the scoring metrics used. Additionally, if two mismatched commands are in no way similar to each other, then a gap may be introduced in either sequence to represent an insertion or deletion in that sequence of commands.

From these characteristics, we are able to develop a concept that matched and similar mismatched commands should be rewarded, while dissimilar mismatched commands and gaps should be penalized. The user's signature should be a representation of the user's overall usage patterns, and therefore should not be subject to insertions, represented by gaps in the *Signature* during alignment, as this would imply that additional commands were used, but were somehow excluded from the signature. Conversely, a gap in the *Test* sequence indicates that a command that occurred in the *Signature* pattern is deleted from the *Test* sequence, which is possible if a user were to use a subset of the pattern represented in the *Signature*. A simple example would be a user that typically changes directory, performs a directory listing, and then edits a file. If the user were already familiar with the directory, then there would be no need for the listing, so that could be deleted in the test sequence. Therefore, gaps in the *Signature* sequence should be more heavily penalized than gaps in the *Test* sequence. While the relations among the scoring variables are certainly important, the scoring scale itself is somewhat arbitrary. As long as matches produce a score that is proportionally higher than mismatches, and mismatches score proportionally higher than gaps, then the exact scoring scale is inconsequential. This is due to the fact that the alignment algorithm chooses from among all possible scoring possibilities, so the absolute magnitude of these scoring parameters is not as important as the relative magnitude of how much they vary from one another. This scoring is what allows us to give preference to certain types of alignments over others.

6 Modeling Mutation in Computer Audit Data

In bioinformatics, scoring systems are created based on years of research regarding the forms mutation can take in a variety of biological sequence types. Dayhoff et al., for instance, use the wealth of research performed on sequences known to be closely related to create probabilistic models of mutation [2]. Such methods are widely used in practical applications of bioinformatics algorithms, but their creation relies heavily on extensive ground truth from which to create the probability models from. Unfortunately, in the case of audit data, such longitudinal ground truth is not available. The nature of the audit data, that is that the alphabet from which symbols in the sequences are chosen is essentially infinite, makes it impossible to create observations for all, or even most, possibilities of mutation. Moreover, such probabilistic models for mutations most likely would have little semantic meaning in terms of audit data. Instead, we must look for models of mutation which provide semantic meaning to the mutation and which can be justified given the use of computer audit rather than biological data.

In this study, we explore two possible models of mutation in computer audit data sequences. Our first model considers the possibility of mutation through semantic, or functional, equivalence with the command grouping scoring system. This model of mutation assumes that the user's behavior falls into well defined patterns of functional behaviors, such as patterns of always opening an e-mail client before opening a web browser. Thus, when mutation occurs within the audit data, and in the case of Schonlau the commands that the user runs, we can assume that these changes retain the functional properties of the patterns in the *Signature*, such as replacing the command for one text editor with that of an equivalent text editing program (e.g., replacing *vi* with *emacs*). The second model of mutation draws upon the conclusions of previous work in masquerade detection that suggest that a legitimate user session is indicated by a fairly consistent set of audit data. In our case, each user is prone to use the same set of commands that were encountered in their signature. We refer to this set of commands known to be from the legitimate user as the *user's lexicon*. The model of mutation, therefore, allows mutation from any command in the lexicon to any other command in the lexicon without penalty. These models of mutation offer differing explanations for the mutations that occur in sequences of audit data, and through evaluation of these systems on the Schonlau dataset we empirically determine which provides the best performance. For the tests performed in this remainder of this paper, penalties for gaps in the *Test* sequence and in the *Signature* sequence remain constant at -2 and -3, respectively.

Command Grouping In this scoring system, a static reward of +2 is given to exact matches. During a mismatch, the groups to which the two commands belong are compared to determine their scoring.

These groups were manually created from a set of common UNIX commands found among all of the commands in the dataset. Each of these groups reflects the general function 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 *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 a value of +1 to the alignment score, and penalize commands that are in different groups by adding the value -1 to the alignment score. For our needs, we must be able to find which commands can be interchanged without changing the high-level function of the pattern created by the user. The command grouping technique does this by keeping the functional high-level definition consistent while allowing for changes in the low-level representation of that functionality.

Binary Scoring Previous work by Wang and Stolfo [16] compared various features of the sequences, and found that binary information about the presence of absence of a command from the user's lexicon performed better than features that took into account frequency of occurrence. Our binary scoring method therefore builds upon the results presented in [16] by implementing a simple scoring system where lexical mismatches that have previously occurred in the user's lexicon are given a positive score, and lexical mismatches that have not previously be observed in the lexicon are given a negative score. Specifically, the binary scoring system rewards exact matches by adding +2 to the score for the alignment. Lexical mismatches where the symbol in the *Test* sequence previously occurred in the user's lexicon are scored as +1, while lexical mismatches are scored as -1. Thus, any previously observed command from a given user can replace any other previously observed command, but use of previously unseen commands is labeled as anomalous behavior. This model of mutation is roughly equivalent to allowing various permutations of previously observed patterns without reducing the score significantly.

6.1 Scoring System Results

We have developed two scoring systems based on distinct conjectures about the form that mutations take within the audit data. In the command grouping scoring system, the conjecture states that the mutations in sequences of audit data occur as a result of functionally similar base symbols replacing the original symbols found in *Signature*. For instance, in the case of commands, one text editor can be replaced with another text editor with no loss of functionality. In the binary scoring system, the conjecture follows the results provided in previous masquerade detection work whereby mutation does not replace base symbols found in the user's lexicon, these symbols are in fact a strong indicator of the user being legitimate, but that these original base symbols can be permuted in some fashion. In being permuted, the only strong indicator for mutation is whether or not the symbol has been seen in the user's signature. The sequence alignment algorithm used in our evaluation in this section builds upon the algorithm described in Section 4 by using binary scoring and command grouping scoring systems rather than the simple scoring scheme previously described. In the following evaluation, results from the sequence alignment with simple scoring system described in Section 4 are referred to as previous result, while the results of the sequence alignment algorithm with our two novel scoring systems are referred to as command grouping and binary scoring, respectively.

Signature:	*sendmail*	*sendmail*	*mailx*	*sendmail*	*rm*	*rm*	*rm*	*p*	*sh*	*sendmail*	*sendmail*	*sendmail*
Test:	*sendmail*	*sendmail*	*sendmail*	*p*	*sh*	*p*	*sh*	*p*	*sh*	*p*	*sh*	*mailx*

Figure 4: Example of differences between User 23's signature and test sequences

Interestingly, as the ROC curve comparing the previous, simplistic scoring system to the two scoring systems proposed in this paper shows, the binary scoring system vastly outperforms the command grouping system. This lends further credence to results such as those provided by Wang and Stolfo, that suggest that the best indicator of legitimate user behavior is the previous observance of base symbols in the user's signature [16]. When we closely examine a subsequence of the *Signature* and *Test* sequences for user number 23 in our dataset, we find additional evidence to support the claim that the mutations that do occur within these sequences of audit data are in fact permutations of previous behaviors. Thus, the most viable model for mutation is clearly the binary scoring method whereby mismatches of commands that have previously been observed in the user's lexicon are weighted more heavily than those mismatches that have never been observed previously. Table 2 provides the Maxion-Townsend scores for the three scoring systems, again showing the clear advantage of the binary scoring system.

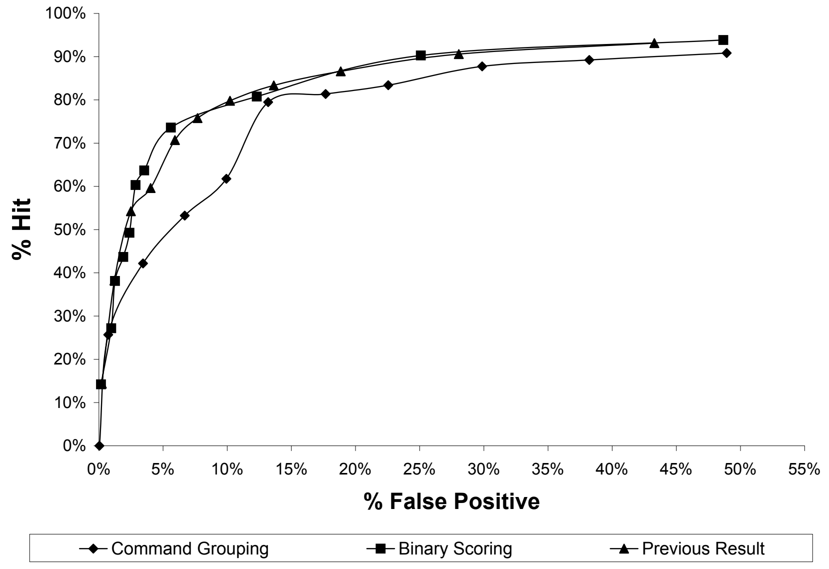

Figure 5: Receiver operator characteristic curve for previous result, binary and command group scoring

Technique	% Hit	% False Positive	Maxion-Townsend Score
Binary Scoring	**60.3**	**2.9**	**56.8**
Previous Result [1]	75.8	7.7	70.4
Command Grouping	42.2	3.5	78.4

Table 2: Comparison of sequence alignment scoring systems

7 Signature Updating

As usage of the information system progresses, it is likely that the users will alter their behavior to varying degrees based on changes to projects, or installation of new programs. This can become a serious problem if a static *Signature* sequence is used to detect anomalies in monitored sessions where user behavior may change over time. The static signature will have no way of adapting to the new behavior, and therefore much of the variation will be considered to be masquerade attacks, thereby creating false positives. As a concrete example, consider the case where a new program has been installed for the users of the information system. Since this new program does not exist within the user's lexicon of previously used commands, the binary scoring system that we have previously examined in Section 5 will always punish any mismatch with this new program. Clearly, Such issues can undermine the confidence in the results of the masquerade detection and should therefore be addressed.

To overcome such issues, the *Signature* must be updated dynamically as user behavior changes. Of course, there are several challenges involved in such updating mechanisms, namely maintaining the temporal properties of the *Signature* sequence (i.e., maintaining useful patterns) and preventing tainted masquerade commands from entering the user's lexicon or *Signature* sequence. These challenges can be distilled into three tasks that the updating mechanism must perform: (i) finding areas in the existing *Signature* sequence that require updating and augmenting them to include the new behavioral information, (ii) determining when a user has begun using a new command not present in his existing lexicon and adding that command to the lexicon, (iii) choosing thresholds that ensure that both the *Signature* sequence and user lexicon remain free of tainted commands from masquerade attacks. Previously, Maxion and Townsend performed a similar procedure on their Naïve Bayes classifier by updating the class probabilities of a command every time a $Test$ sequence was classified [6]. To choose which class, user or non-user, was augmented, they simply used the classification provided by their Naïve Bayes approach. Thus, if their classifier determined a particular $Test$ sequence to be from the user, the user's classifier would be have its probabilities updated to reflect the commands found in this sequence. Similarly for non-user sequences, the non-user classifier would be updated to reflect the new commands. Note that since their technique utilized an independence assumption among commands, there is no need to maintain the integrity of the temporal characteristics of the sequences. Thus, while the Maxion and Townsend method provides some insight into how to characterize which commands should be used for updating, our sequence alignment approach requires a far more complex updating scheme.

Updating Signatures with Sequence Alignment To determine the way in which the *Signature* sequence and the user's lexicon becomes augmented, we look to the scoring matrix that is created during the alignment process. This scoring matrix, as previously shown in Figure 2, can be used to recreate the optimal alignment by tracing back through the matrix from the (m, n) position to the $(0, 0)$ position. This alignment provides us with the areas where the $Test$ sequence has aligned well with the *Signature* sequence. In fact, this is an intuitive and natural way to determine which areas of the *Signature* sequence should be augmented, and when new commands should be entered into the user's lexicon. Note that when a gap is encountered in this alignment, in either the *Signature* or $Test$ sequence, it indicates poor alignment for the opposite symbol. This is a clear indication that symbols that are aligned with gaps should not be considered in the updating process since they are not determined to be similar.

When two symbols do align, they can align as an exact lexical match, a 'good' mismatch where the mutations from the *Signature* symbol to the $Test$ symbol is expected by our scoring system, or a 'bad' mismatch where the mutation is unexpected. In the case of a match, no updating needs to be done, as the alignment properly utilized the existence of the symbol within the pertinent subsequence to find the optimal alignment. Information is gained by the appearance of mismatches, however. For 'good' mismatches, we know that the symbol in the $Test$ sequence must have previously existed in the user's lexicon since our binary scoring system uses that information to define 'good' mismatches. The fact that the command existed in the user's lexicon and participated in a conserved alignment indicates the creation of a new permutation of the user's behavior. Similarly, in the case of 'bad' mismatches, we know that the user has never used the command before since it does not exist in his lexicon, but its presence within a conserved, high scoring alignment means that it participates in an alignment that was most likely created by the legitimate user. Hence, such 'bad' mismatches could indicate the introduction of a new symbol into the user's lexicon, but only if the resultant alignment has a very high score when compared to the user's average alignment score.

Updating the signature, therefore, becomes a simple process of finding the 'good' and 'bad' mismatches in the alignment between the *Signature* and $Test$ sequence. When a 'good' mismatch is encountered, we can augment the *Signature* sequence by adding the $Test$ symbol to the *Signature* sequence at the aligned position. For instance, if the alignment produced a 'good' mismatch between *vi* in the *Signature* sequence and *cd* in the $Test$ sequence, the *Signature* sequence would be augmented such that the position that contained *vi* can now match with *vi* or *cd*. Notice that this augmentation does not destroy previous information encoded in the *Signature* sequence, it simply embeds observed variations into the sequence. Likewise, when a 'bad' mismatch is encountered, we can simply add the *Test* symbol to the user's lexicon, but make no changes to the *Signature* sequence. Essentially, we let this newly introduced symbol be used as a 'good' mismatch in future alignments.

Updating Threshold The question remains as to how we choose what score an alignment must have in order to implement these updating features. Recall that our threshold system for the sequence alignment technique, given in Section 4, provides two distinct measures of normal alignment scores for each user — the user's average score and the threshold score for labeling masquerade attacks based on a percentage of the user's average score. In choosing the appropriate threshold for updating, we must consider the relative dangers in each of the updating operations, namely augmenting the *Signature* sequence and inserting new commands into the lexicon. Since the *Signature* sequence updating mechanism only updates those symbols, or commands, that have previously been seen in the user's lexicon, it is a relatively safe procedure. Conversely, adding a command into the user's lexicon that has never been seen before could introduce a tainted command that would prevent our detection of masquerade attacks. Therefore, since updating the user's lexicon is much more dangerous, we require that an alignment score higher than the user's average score to perform the lexicon updating procedure. To perform the *Signature* sequence augmentation, however, we require only that the alignment score above the user's masquerade threshold (i.e., some percentage of the user's average score determined by the sensitivity level).

Example We provide a concrete example of this updating approach in Figure 6. Let us assume that the provided alignment has scored above the user's average score, and therefore both the lexicon and the *Signature* sequence will be updated. Furthermore, let use assume that the symbol 'C' occurs within the user's lexicon, while the symbol 'F' does not. Therefore, by recreating the alignment from the scoring matrix, we see that we must augment the *Signature* sequence with the symbol 'C' at the position where it has been mismatched, represented by | in the new *Signature* sequence. Likewise, the symbol 'F' has now been added to the user's lexicon, and in future alignments it will be treated as a 'good' mismatch. Since it is a 'good' mismatch, the 'F' symbol may be augmented, as the 'C' symbol was, in future alignments. This alignment process allows us to control the symbols which enter the user's lexicon and *Signature* sequence, while maintaining the temporal properties of the *Signature* and adapting to changing user behavior.

Signature:	A	B	E	_	_	J	A	B
Test:	A	C	F	H	K	J	A	B
New Signature:	A	B|C	E	J	A	B		
X|Y = Match X or Y								

Figure 6: Example of signature updating

7.1 Signature Updating Results

In their application of Naïve Bayes machine learning to masquerade detection, Maxion and Townsend showed that updating the signature of the user provides a significant increase in performance. This is certainly true in cases where a user's behavior may vary over time due to changes in work patterns, or installed software. Above, we provided a method of using the alignment created by the tuned Smith-Waterman algorithm to find areas of similarity, and augment the user's signature with additional commands, as well as increasing the user's lexicon with newly introduced symbols. In this evaluation, we use the sequence alignment algorithm with the binary scoring system as evaluated in Section 6.1, and augment it with the addition of the signature updating procedure outlined above. The following discussion refers to the sequence alignment algorithm with binary scoring and no updating as binary scoring, while the algorithm with updating is referred to as signature updating.

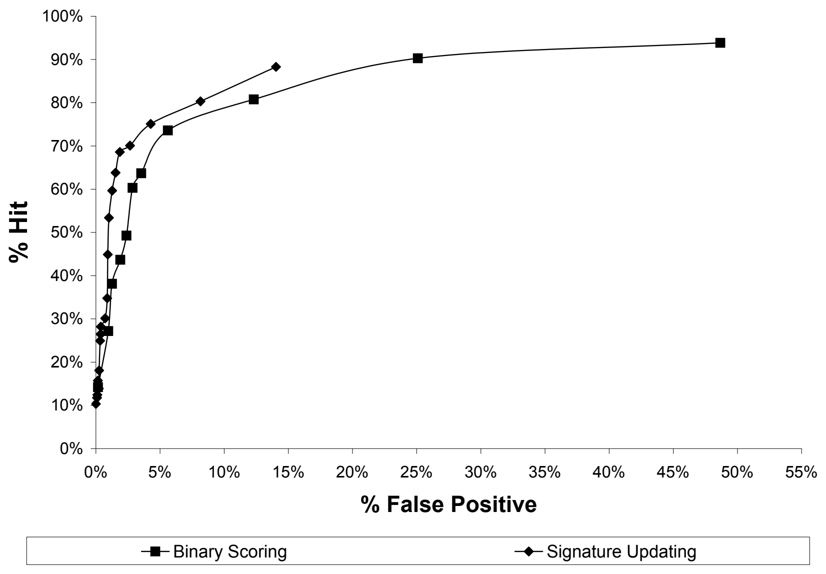

Figure 7: Receiver operator characteristic curve for signature updating compared to binary scoring alone

Figure 7 shows that the binary scoring system with the signature updating method significantly improves the results of our sequence alignment using the binary scoring method alone. Notice that the entire ROC curve for the alignment algorithm using signature updating is far more conserved with respect to false positives — all sensitivity levels for the signature updating approach lie in the area below 15% false positives while the binary scoring approach lies in the area below 50%. Moreover, Figure 8 and Table 3 indicate that the use of signature updating provides performance that surpasses all previous masquerade detection algorithms, both in terms of ROC curve and Maxion-Townsend score. This result is a significant improvement to the state of the art in masquerade detection, and underscores the significance of using sequencing information, custom mutation models, and signature updating in the development of masquerade detection algorithms.

Technique	% Hit	% False Positive	Maxion-Townsend Score
Sequence Alignment (Updating)	**68.6**	**1.9**	**42.8**
Naïve Bayes (With Updating) [6]	61.5	1.3	46.3
Sequence Alignment (Binary Scoring)	60.3	2.9	56.8
Recursive Data Mining [12]	62.3	3.7	59.9
Naïve Bayes (No Updating) [6]	66.2	4.6	61.4

Table 3: Comparison of the top five masquerade detection techniques ranked by Maxion-Townsend score

8 Computational Requirements

Thus far, we have shown that the sequence alignment algorithm is proficient at detecting masquerade attacks in computer audit data through evaluation on the Schonlau dataset. This evaluation, however, only tests the detection performance of the algorithm without regard to the practical implementation issues, such as computational requirements. In realistic deployments, the masquerade detection technique would have to perform detection for multiple users utilizing an information system at the same time. Depending on the granularity of the audit data and its rate of generation, the computational

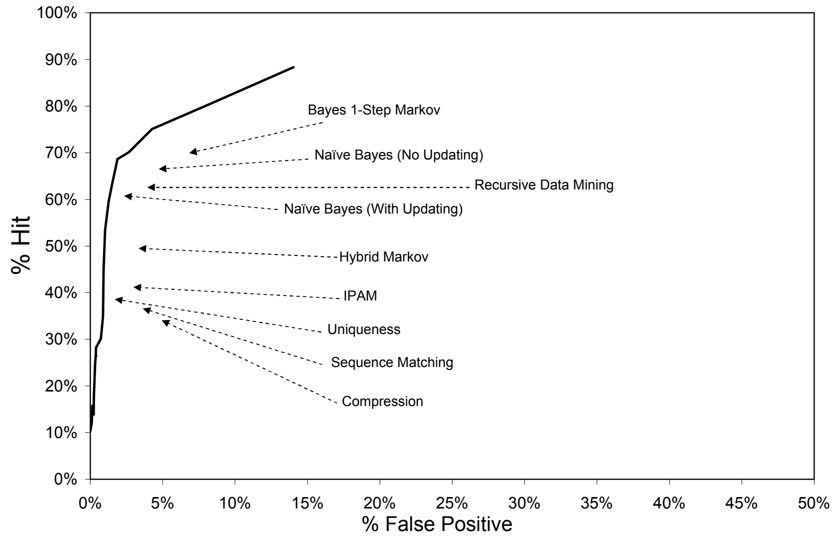

Figure 8: Receiver operator characteristic curve for signature updating compared to previous results

requirements of such detection could easily become overwhelming for a modern information system. The computational complexity of the sequence alignment algorithm is a rather slow $O(mn)$, where m is the length of the *Signature* sequence and n is the length of the *Test* sequence. In the case of the Schonlau data, this means that there are 500,000 computations (asymptotically) per monitored user session that needs to be tested for masquerade attacks. Certainly, this is far too much computation to require for each user session, especially in multiuser environments or computationally limited devices, and thus a method for reducing the computational requirements is necessary

A Heuristic for Aligning Audit Data To reduce this computational requirement, we must find some way of reducing the number of computations that are required without reducing the quality of the alignments, or allowing masquerade attacks to escape detection. Our method of reducing the computations per alignment rests on using the heuristic that high scoring alignments are typified by a large number of exact lexical matches in a very conserved area. In fact, this conserved area can be expected to always be of size $\leq n$, since n is always less than or equal to m. In other words, a high scoring alignment will always find the *Test* sequence matching exactly with a subsequence of the *Signature* sequence, with few or no gaps inserted, expect for prefixes and suffixes which are ignored by the semi-global alignment. This implies that during any particular alignment, only subsequences of size $\leq 2n$ are ever used. Any subsequence that is larger than $2n$ in size would necessarily score poorly because of the number of gaps being inserted, and therefore we can focus our alignments only on the subsequences of size $2n$ within the *Signature* sequence.

Given this observation, we split the original *Signature* sequence into overlapping blocks of size $2n$ each, such that the last n symbols of block i are also the first n symbols of block $i+1$. This ensures that we have all possible adjacent pairs of the subsequences of size n. Since high scoring alignments require matches, we simply need to estimate the number of possible matches between the *Test* sequence and the blocks of size $2n$. A simple way to do this is to take each distinct symbol within the *Test* sequence and determine its number of occurrences within the *Test* sequence and within the *Signature* sequence, respectively. The minimum of these two numbers is the maximum number of times that particular symbol can be matched. By summing this number over all distinct symbols in the *Test* sequence, we have a heuristic for the number of matches possible between the *Test* sequence and the particular $2n$

subsequence. We need only choose the $2n$ subsequence(s) for which the sum of possible matches is the largest from among all $2n$ subsequence(s). Essentially, we choose those subsequences of the *Signature* sequence which have the maximum number of possible matches with the *Test* sequence. We then align the *Test* sequence only with those maximum subsequences, rather than blindly attempting alignment on all subsequences. This has the effect of, in the average case, reducing the number of computations performed since only a very small portion of the *Signature* sequence is being utilized for any given monitored session.

Example For concreteness, let us consider the example given in Figure 9. In this example, we have a *Signature* sequence of length 12, and a *Test* sequence of length 4. Notice that the best alignment of the *Test* sequence falls between the final two blocks of size 4. As described above, the *Signature* sequence is partitioned into overlapping blocks of size $2n$, in our case blocks of size 8, given as Subsequence 1 and 2, respectively. Since Subsequence 2 has the largest number of possible matches, only Subsequence 2 will be aligned with the *Test* sequence. This reduces the number of computations necessary from 48 to 36. In cases of computer audit data where $n \ll m$ due to long observation periods for the *Signature* sequence, this improvement in the computational requirements can be expected to be much larger, on average.

Signature:	A	C	C	E	F	G	I	A	B	A	C	D
Test:	G	I	A	B								
Subsequence 1:	A	C	C	E	F	G	I	A				
Subsequence 2:	F	G	I	A	B	A	C	D				

Number of Possible Matches (Subsequence 1): 4
Number of Possible Matches (Subsequence 2): 5

Figure 9: Example of computational heuristic

8.1 Heuristic Results

We evaluate the heuristic improvement in computational requirements of the detection process. In this heuristic, the *Signature* sequence is broken into blocks of size $2n$. The number of lexical matches between any given $2n$ block and the *Test* sequence is estimated by the number of symbols that the two sequences have in common. The $2n$ signature blocks with the maximum value for this heuristic are then be used in the detection alignments. In the worst case, all of the alignments still need to be computed, but in the average case a far smaller subset of alignments need be computed to perform the detection. The evaluation of this heuristic speed up, therefore, is primarily concerned with (i) empirically evaluating the number of alignments needed on average to perform detection and (ii) the impact that this heuristic has on the detection performance of the sequence alignment algorithm. In our evaluation, we augment the previously described sequence alignment algorithm using binary scoring and signature updating by implementing the heuristic speed up as described above. For comparison, we refer to the sequence alignment with the heuristic as heuristic, and the alignment algorithm without it as signature updating.

Through our evaluation, we found that there was an average of 4.5 alignments required per detection. This is a huge reduction from the maximum number of alignments, 49, and represents a substantial increase in speed for realistic detection scenarios. In terms of asymptotic computations, this reduces the workload of the sequence alignment algorithm with binary scoring and signature updating from 980,000 computations (49 200x100 alignments) in the worst case to an average of 90,000 computations (4.5 200x100 alignments). More importantly, as Figure 10 and Table 4 show, the use of this heuristic has a minimal impact on the performance in terms of ROC curve and Maxion-Townsend score.

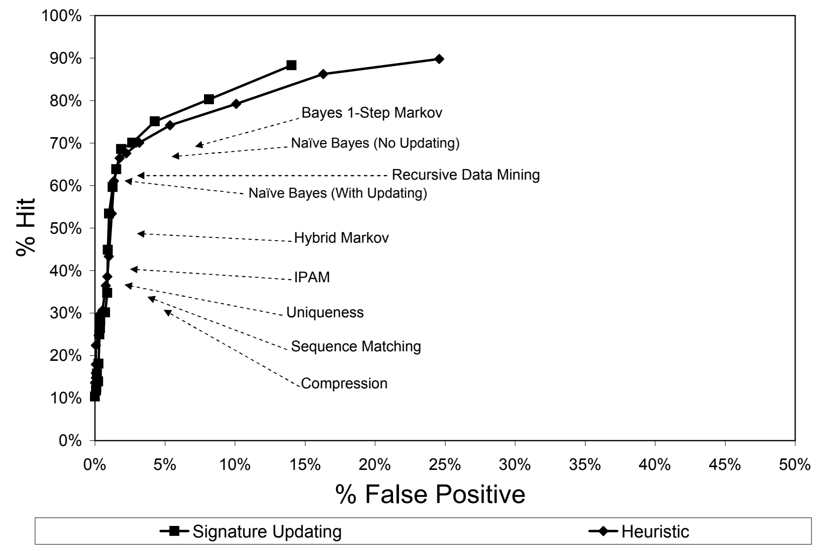

Figure 10: Receiver operator characteristic curve for heuristic compared to signature updating

Technique	% Hit	% False Positive	Maxion-Townsend Score
Sequence Alignment (Updating)	68.6	1.9	42.8
Sequence Alignment (Heuristic)	**66.5**	**1.8**	**44.3**
Naïve Bayes (With Updating) [6]	61.5	1.3	46.3
Sequence Alignment (Binary Scoring)	60.3	2.9	56.8
Recursive Data Mining [12]	62.3	3.7	59.9

Table 4: Comparison of the top five masquerade detection techniques

9 Conclusion

The masquerade attack poses a serious threat to the security of information systems due to its ability to completely undermine even state of the art security technologies. To minimize the risk of these attacks compromising the security of the information system, an automated method for detecting masquerade attacks is necessary. Previous approaches to detecting masquerade attacks take advantage of statistical models, machine learning techniques, and string matching [5, 6, 7, 9, 10, 12, 16]. However, none of these previous approaches offers the level of accuracy necessary for practical deployment. Interestingly, while a wide range of features are explored separately by these previous methods, no technique utilizes all of these features in one mechanism. An exploratory study [1] of the use of sequence alignment algorithms in detecting masqueraders showed that such algorithms can be made to take advantage of all of these features, and that they can be adapted to the general task of intrusion detection with tuning, thereby indicating that they may be able to provide the best performance of all available techniques.

Here, we have explored the intricacies involved in adapting the Smith-Waterman local sequence alignment algorithm for use in masquerade detection. Though there has been significant previous work in the field of bioinformatics, little of it is applicable to the domain of computer security. Our development, therefore, focused on new methods for applying sequence alignment to sequences of audit data from information systems. In doing so, we have presented methods for tuning the Smith-Waterman algorithm by creating semi-global alignments of the sequences. Additionally, we discussed two scoring systems which were inspired from functional mutations found in bioinformatics and models derived from previous work, respectively. Interestingly, our results found that functional mutation was not a useful model for detecting masquerade attacks, and that user behavior is typified by reusing the same symbols from within their lexicon in varying permutations.

We were also able to develop methods for dynamically updating the user signature to accommodate changes in the user's command usage. By recreating the alignment, our technique was able to pinpoint the areas of the user's signature which should be augmented with the new usage information, without violating the temporal properties of the sequence. The implementation of this signature updating technique allowed our sequence alignment approach to best all previous techniques, including Maxion and Townsend's Naïve Bayes classifier which also performed signature updating. Finally, we addressed the computational expense of the alignment algorithm by describing methods for choosing the best subsequences of the user signature to align with the monitored session. Through empirical evaluation, we found that this heuristic speed up in computation imposed only a minor loss of accuracy. Overall, both with and without the heuristic speed up, our sequence alignment technique provides significant advancement to the field of masquerade detection, and opens the possibility of using such alignment techniques in other areas of intrusion detection.

References

[1] 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.
[2] Dayhoff, M. O., Schwartz, R. M., and Oreutt, B. C. 1978. A Model of Evolutionary Change in Proteins. In Atlas of Protein Sequence and Structure (Dayhoff, ed.). Vol. 5, Suppl. 3, 345-352.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] Maxion, R. A. 2003. Masquerade Detection using Enriched Command Lines. In International Conference on Dependable Systems (DSN-03). San Francisco, CA, June 2003, 5-14.
[8] Needleman, S., and Wunch, C. 1970. A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins. Journal of Molecular Biology, 48: 444-453.
[9] Schonlau, M., and Theus, M. 2000. Detecting Masquerades in Intrusion Detection Based on Unpopular Commands. Information Processing Letters 76, 33-38.
[10] 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.
[11] Smith, T. F., and Waterman, M. S. 1981. Identification of Common Molecular Subsequences. Journal of Molecular Biology 147, 195-197.
[12] 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.
[13] Tandon, G., Chan, P., and Mitra, D. 2004. MORPHEUS: Motif Oriented Representations to Purge Hostile Events from Unlabeled Sequences. In Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, Washington, D.C., October 2004, 16-25.
[14] Wagner, R. A., and Fisher, M. J. 1974. The String-to-String Correction Problem. Journal of the ACM 21, 168-173.
[15] Wagner, D. and Soto, P. 2002. Mimicry Attacks on Host-based Intrusion Detection Systems. In Proceedings of the 9th ACM conference on Computer and Communications Security. Washington, D.C.,

255-264.
[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] Wespi, 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.
[18] Wright, C., Monrose, F., and Masson, G. 2004. HMM Profiles for Network Traffic Classification. In Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, Washington, D.C., October 2004, 9-15.
[19] Wright, C., Monrose, F., and Masson, G. 2006. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. Journal of Machine Learning Research, Special Topic on Machine Learning for Computer Security. To appear.