Volume 52, issue 8
15 April 2008

ISSN 0167-9473

ELSEVIER

COMPUTATIONAL STATISTICS & DATA ANALYSIS

Incorporating Statistical Software Newsletter

The official journal of iasc

The International Association for Statistical Computing
A Section of The International Statistical Institute

Available online at ScienceDirect
www.sciencedirect.com

Available online at www.sciencedirect.com
ScienceDirect

Computational Statistics and Data Analysis 52 (2008) 4116–4131

COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

Sequence alignment for masquerade detection

Scott E. Coull[a], Boleslaw K. Szymanski[b,*]

[a] *Johns Hopkins University, 3400 N. Charles Street, Baltimore, MD 21218, United States*
[b] *Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, United States*

Received 9 January 2007; received in revised form 21 January 2008; accepted 22 January 2008
Available online 5 February 2008

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 due to the trust imparted to user accounts once they have been authenticated. Many attempts have been made at detecting these attacks, yet achieving high levels of accuracy remains an open challenge. In this paper, we discuss the use of a specially tuned sequence alignment algorithm, typically used in bioinformatics, to detect instances of masquerading in sequences of computer audit data. By using the alignment algorithm to align sequences of monitored audit data with sequences known to have been produced by the user, the alignment algorithm can discover areas of similarity and derive a metric that indicates the presence or absence of masquerade attacks. Additionally, we present several scoring systems, methods for accommodating variations in user behavior, and heuristics for decreasing the computational requirements of the algorithm. Our technique is evaluated against the standard masquerade detection dataset provided by Schonlau et al. [Schonlau, M., DuMouchel, W., Ju, W.H., Karr, A.F., Theus, M., Vardi, Y., 2001. Computer intrusion: Detecting masquerades. Statistical Science 16 (1), 58–74], and the results show that the use of the sequence alignment technique provides, to our knowledge, the best results of all masquerade detection techniques to date.

1. Introduction

To protect information systems from unauthorized use, administrators rely on security technologies such as firewalls, network-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, an attacker who has obtained a user's password can utilize all of the user's privileges without being detected due to the trust placed in the compromised account. Similarly, malicious insiders can choose to use their privileges to perform unauthorized actions. These examples describe the canonical masquerade attack, where 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

[*] Corresponding author. Tel.: +1 518 276 2714; fax: +1 518 276 4033.
E-mail addresses: coulls@cs.jhu.edu (S.E. Coull), szymansk@cs.rpi.edu, szymab@rpi.edu (B.K. Szymanski).

0167-9473/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2008.01.022

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. This detection is difficult in practice, as legitimate daily activity could appear malicious based on its context. In fact, there have been several attempts at creating algorithms for detecting these attacks, though achieving high levels of detection accuracy remains an open problem (Coull et al., 2003; Lane and Brodley, 1997, 1998, 1999; Maxion and Townsend, 2002; Maxion, 2003; Schonlau and Theus, 2000; Schonlau et al., 2001; Szymanski and Zhang, 2004; Wang and Stolfo, 2003).

In this paper, we leverage the pattern matching abilities of sequence alignment algorithms to discover masquerade attacks within sequences of information system audit data (*e.g.,* command line entries). In bioinformatics, these algorithms are typically used to discover areas of similarity between two sequences of biological data (*e.g.,* DNA sequences). Sequence alignment provides several benefits beyond simple lexical matching by incorporating domain knowledge into the alignment process, such as likely mutations in the sequences. Customized scoring systems are used to define ‘good’ and ‘bad’ alignments based on this domain knowledge. Hence, the alignments actually highlight areas of functional similarity between the aligned sequences based on the scoring system utilized.

These sequence alignment algorithms can also 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 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 adaptable to the task of detecting masquerade attacks, most 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 (*e.g.,* 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 because of its dynamic nature. Accordingly, we provide a modification of the Smith–Waterman local alignment algorithm (Smith and Waterman, 1981) to properly accommodate for the alignment of computer audit data. We also investigate two novel scoring systems designed to model mutations in this audit data.

In addition, several challenges are inherent to the task of detecting masquerade attacks. First, the usage patterns of legitimate users can be expected to change over time, perhaps due to new projects or software. The use of static user signatures is therefore prone to label legitimate variations in behavior as attacks. By using the sequence alignment algorithm’s ability to discover areas of similarity, however, we are able to dynamically update the user’s signature as new user behavior is encountered. Second, the Smith–Waterman algorithm is computationally expensive, making it 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 computations required with almost no loss of accuracy. Our modified alignment algorithm, along with our scoring systems and signature updating scheme, were tested on the Schonlau et al. dataset (Schonlau and Theus, 2000), 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, to the best of our knowledge, the best accuracy of any of the known masquerade detection techniques.

The paper begins with 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 describe the methodology used to evaluate our masquerade detection technique. We continue by describing our modifications of the Smith–Waterman local alignment algorithm (Smith and Waterman, 1981), 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 present 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 (Coull et al., 2003), to the most sophisticated one, 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

Schonlau and Theus (2000) and Schonlau et al. (2001) provided the first techniques for detecting masquerade attacks through the use of a number of statistical methods. 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. 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 (1997, 1998, 1999) take a string matching approach by attempting to lexically match subsequences of the user's signature with subsequences of the monitored session, 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., ignores the underlying functionality of the commands in the sequences, relying instead on finding exact lexical matches.

There have also been several attempts at applying more advanced machine learning techniques to the problem of masquerade detection. Maxion and Townsend (2002) provide the best results of all past techniques by using a two-class, Naïve Bayes classifier to detect masqueraders. The most important contribution made by Maxion and Townsend is the use of updating mechanisms that dynamically update the classifier probabilities as monitored sequences are classified. Thus, this approach adapts to changes in user behavior. However, despite the improved performance of the classifier, sequencing information and the functional semantics of the commands are ignored. Wang and Stolfo (2003) apply one-class Naïve Bayes and Support Vector Machine classifiers, and find that their results are comparable to those of the two-class classifiers. This approach, however, suffers from the same weaknesses as the Maxion and Townsend approach by ignoring sequence and functionality information. Szymanski and Zhang (2004) also use a one-class Support Vector Machine, but 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.

Several other bioinformatics tools have been applied to computer security problems. Wespi et al. (1999) 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 et al. (2004) used the concept of motifs, or conserved areas of recurring behaviors, to discover anomalies within sequences of audit data. Wright et al. (2004, 2006) use Hidden Markov Models, which are typically used to align many biological sequences at once, to detect the presence of various application protocols within encrypted tunnels.

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 (Coull et al., 2003). 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 this paper, 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 obtained by previously published methods using the Schonlau et al. dataset (Schonlau et al., 2001). This 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, thereby making comparison among masquerade detection algorithms straightforward.[1]

The Schonlau dataset was created by recording, via the *acct* utility, users' commands as they were entered into the UNIX shell. All command arguments were 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, 5000 commands were recorded to make up the set of commands that are considered free of any anomaly or intrusion. We will call such anomaly-free command sequences the *user's signature*, or training set. An additional 10,000 commands were recorded from each user in the dataset to make up the set of commands that are to be tested for intrusions. We 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 by Schonlau et al. (2001).

The commands, in both the user's signature and the test data, are broken into 100-command groupings, which we will call *blocks*. Hence, the dataset consists of data for 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 5000-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.

Unfortunately, the Schonlau dataset 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 may 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 truly malicious actions.

The primary weakness of this dataset, though, is its lack of command arguments. It is not only unrealistic, but could also lead to some questions about the applicability of the so-called mimicry attacks (Wagner and Soto, 2002), where an attacker attempts to accurately mimic the valid behavior of the program or user being monitored to escape detection. However, evaluation of the Maxion and Townsend Naïve Bayes classifier (Maxion, 2003) on a proprietary dataset has shown that performance improves significantly when command arguments are present. This indicates that the Schonlau dataset is of value as a lower bound on the performance of masquerade detection techniques, and that augmenting command lines with argument data may be an effective countermeasure for mimicry attacks, though discussion of specific mimicry countermeasures is beyond the scope of this paper.

In evaluating their technique, Maxion and Townsend (2002) created a scoring framework, which we call the Maxion–Townsend score, that rates the overall performance of a masquerade detection algorithm as a function of its false positive rate and its false negative, or miss, rate. The Maxion–Townsend scoring equation is equal to: $6 * \mathit{FalsePositives} + \mathit{Misses}$. The choice of a false positive coefficient of six and a false negative coefficient of one is somewhat arbitrary, but provides a realistic assessment of the overall cost of the detection techniques in low security environments, where false positives are far more costly than false negatives due to their disruptive nature. We use the 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 is used in bioinformatics to find areas of similarity between two biological sequences, such as DNA or protein sequences. It can be viewed as a generalization of the longest common subsequence problem (Wagner and Fischer, 1974) in which we are given two strings, $A = a_1a_2 \dots a_m$ and $B = b_1b_2 \dots b_n$ $(n \leq m)$ over alphabet Σ. The goal is to find the maximal length lexically similar subsequences of A and B (*i.e.,* exact string matches). This

[1] Unfortunately, Wang and Stolfo (2003) do not provide specific false positive and true detection scores, thereby making direct comparison to their technique impossible.

```
-  -  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
```

Fig. 1. Example sequence alignment of DNA sequences.

could be achieved by deleting characters from A and B until two maximal length and lexically similar subsequences of A and B remain. Alternatively, instead of deleting symbols, gaps may be inserted into the sequences so that the matching common subsequences are aligned. For instance, gaps are inserted in both the upper and lower sequences in Fig. 1 to align the matching subsequences.

Beyond simple lexical matching, sequence alignment uses a scoring system to promote certain matches and to discourage others. In bioinformatics, such scoring is based on observed mutations from one symbol to another (*e.g.,* from one nucleotide in a DNA sequence to another). 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. Fig. 1 depicts an example where mismatched symbols (T and A) are aligned despite lexical dissimilarity because the possibility of mutation from one to the other is encoded in the scoring system used.

In general, there are several modes of alignment, including global, semi-global, and local alignments. Global alignment, known as the Needleman–Wunch algorithm (Needleman and Wunsch, 1970), tries to maximize the length of the subsequence over the entire length of both strings. It is useful when both strings are of approximately the same length and the entirety of both sequences should be similar. Local alignment, known as the Smith–Waterman algorithm (Smith and Waterman, 1981), focuses instead on finding the best aligned substrings of the two sequences over all possible substrings, rather than over the entire sequence. Local alignment is useful when searching for areas of functional similarity between sequences, or when one sequence is significantly longer than the other. In such cases the majority of the sequence can be assumed to be dissimilar except for the area of functional similarity, and so global alignment would provide a poor alignment. Finally, semi-global alignment allows for large areas of the sequences to be aligned as in global alignments, but also allows dissimilar prefixes and suffixes of the sequences to be ignored. Semi-global 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. Recent work on pair-wise sequence alignment has focused on improving computational requirements while maintaining the optimality of the underlying alignments (Altschul et al., 1990, 1997; Pearson and Lipman, 1988; Zhang et al., 2000; Bray et al., 2003).

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, like the Schonlau command line data. In the masquerade detection problem, we are given two strings $Signature = a_1 a_2 \ldots a_m$ and $Test = b_1 b_2 \ldots b_n$ with $n \leq m$. The string *Signature* represents the sequence of audit data gathered from normal usage of the system by a given user. 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 the use of sequence alignment with biological data. Our goal is to find the areas of 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 that alignment with gaps or lexical mismatches may be indicative of possible masquerade attacks.

The alignment algorithm, shown in Algorithm 1, uses dynamic programming to discover the optimal alignment among all possible alignments. It begins by initializing an $m + 1$ by $n + 1$ matrix, called D. Starting at position (0, 0) (*i.e.,* the upper left corner) in the matrix, we iterate through each position whose value is determined through a choice of three transitions to that position:

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
          $top \leftarrow 0$
        end if
        $left \leftarrow D[i-1][j] + gapTest$
        if $left < 0$ then
          $left \leftarrow 0$
        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 $D[m][n]$
```

1. Diagonal step: Indicates an alignment between the $i-1$ symbol in *Signature* with the $j-1$ symbol in *Test*. The alignment score added to the value of the matrix position at $(i-1, j-1)$ measures the level of alignment of the symbols defined in the scoring system, denoted as *diagonal*.
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 $(i, j-1)$, denoted as *top*. 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 $(i-1, j)$, denoted as *left*. As with the vertical step, 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 current 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, 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. We use this score as an indicator for masquerade attacks.

Note that in our masquerade detection problem, the *Signature* sequence must necessarily be larger than the *Test* sequence because 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 0th column and the 0th 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 mth column and the nth row are set to zero for both sequences. Thus, after the end of one of the sequences is reached, we align the remainder 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

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

A B _ C D
A B E C D

Fig. 2. Example alignment using semi-global Smith–Waterman alignment.

areas of similarity within the sequences, and use only the largest contiguous area of similarity in the final score calculation.

The alignment scoring system itself is made up of four variables that define the behavior of the alignment. 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*. This variable can represent very complex models of mutation, and has a significant amount of power in defining the alignment. Finally, 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, or 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. The use of gaps is certainly the worst possible case of alignment, as it means that the symbols in the sequences are significantly different according to the scoring scheme used. An example alignment with a match score of $+2$, a mismatch score of 0, and a gap penalty of -2 is given in Fig. 2.

4.2. Masquerade detection threshold

The alignment scores can range from zero, indicating that the entire sequence was ignored as a prefix or suffix, to the length of the test data sequence multiplied by the match score, indicating an exact match. However, in the context of masquerade detection, these scores depend 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 distinctive, and therefore setting the same threshold for all users is inappropriate. Moreover, the consistency of the user's behavior may actually change over time due to project 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, 1000-command subsequences of the user's *Signature* sequence, and aligning them to 20 non-overlapping, 100-command subsequences of the same 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 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 high sensitivity value would catch a lot masquerade attacks, but may expose the system to a large number of false positives created by normal user activity. Conversely, a low sensitivity would catch only the most onerous masquerade attacks and would rarely raise false alarms, but more subtle attacks might go undetected.

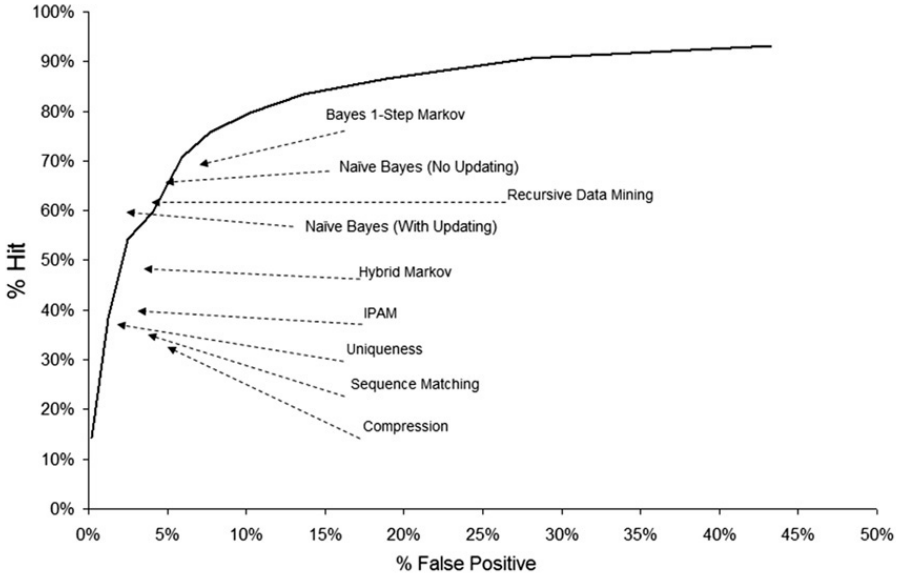

Fig. 3. Receiver operator characteristic curve for the previous result.

4.3. *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 (Coull et al., 2003). The scoring system for this initial study was set 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 by 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 $(0, 1]$. If the frequency is less than the average, the mismatch is given a negative score in the range $[-1, 0)$. 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.

Receiver operator characteristic (ROC) curves are used to depict the overall performance of our technique throughout this paper. These ROC curves are plotted with hit rate (*i.e.,* true positive rate) as a function of the miss rate (*i.e.,* false negative rate). The respective aggregate rates are derived by averaging over the individual rates of each user in the dataset, and the ROC curve is plotted by varying the sensitivity of the masquerade threshold from 0% to 100% of the user's average at 5% increments. Additionally, we provide the Maxion–Townsend Score for each masquerade detection algorithm as a means of comparison. We choose the Maxion–Townsend score for our techniques as the minimum score across all sensitivity levels of the masquerade threshold, and scores from previous techniques are quoted from the cited references.

Despite the simplicity of the scoring system, the approach yielded encouraging results, as depicted in the ROC curve of Fig. 3. According to the ROC curve, the sequence alignment technique is only bested by the Naïve Bayes approaches of Maxion and Townsend (2002), and the Support Vector Machine of Szymanski and Zhang (2004). However, the use of Maxion–Townsend score, shown in Table 1, provides a very different picture of the best performance when compared to other techniques. When the algorithms are sorted by Maxion–Townsend Score, the sequence alignment technique has 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.

5. Scoring systems

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. (1978), for instance, use the wealth of research performed

Table 1
Comparison of published masquerade detection technique sorted by Maxion–Townsend score

Technique	% Hit	% False Positive	Maxion–Townsend score
Naïve Bayes (With Updating) (Maxion and Townsend, 2002)	61.5	1.3	46.3
Recursive Data Mining (Szymanski and Zhang, 2004)	62.3	3.7	59.9
Naïve Bayes (No Updating) (Maxion and Townsend, 2002)	66.2	4.6	61.4
Uniqueness (Schonlau et al., 2001)	39.4	1.4	69.0
Hybrid Markov (Schonlau et al., 2001)	49.3	3.2	69.9
Sequence Alignment (Schonlau et al., 2001)	**75.8**	**7.7**	**70.4**
Bayes 1-Step Markov (Schonlau et al., 2001)	69.3	6.7	70.9
IPAM (Schonlau et al., 2001)	41.1	2.7	75.1
Sequence Matching (Lane and Brodley, 1997; Schonlau et al., 2001)	36.8	3.7	85.4
Compression (Schonlau et al., 2001)	34.2	5.0	95.8

on sequences known to be closely related to create probabilistic models of mutations. 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. Unfortunately, in the case of audit data, such longitudinal ground truth is not available. Since the alphabet of audit data symbols is essentially infinite, it is difficult to create observations for all, or even most, possibilities of mutation. Hence, these probabilistic models for mutations most likely would have little semantic meaning in terms of audit data. Instead, we must look for semantically meaningful models of mutation for computer audit, rather than biological, data.

5.1. Modeling mutation in computer audit data

Here, 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. Thus, when mutation occurs within the audit data, 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 another (*e.g.,* replacing *vi* with *emacs*). The second model of mutation draws upon the conclusions of Wang and Stolfo (2003), which suggest that a legitimate user session is indicated by a fairly consistent set of audit data symbols. 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 the remainder of this paper, penalties for gaps in the *Test* and *Signature* sequences remain constant at -2.

Command grouping. Any useful model of mutation for audit data must be able to find which commands can be interchanged without altering the high-level function of the patterns 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. 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 shells. 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

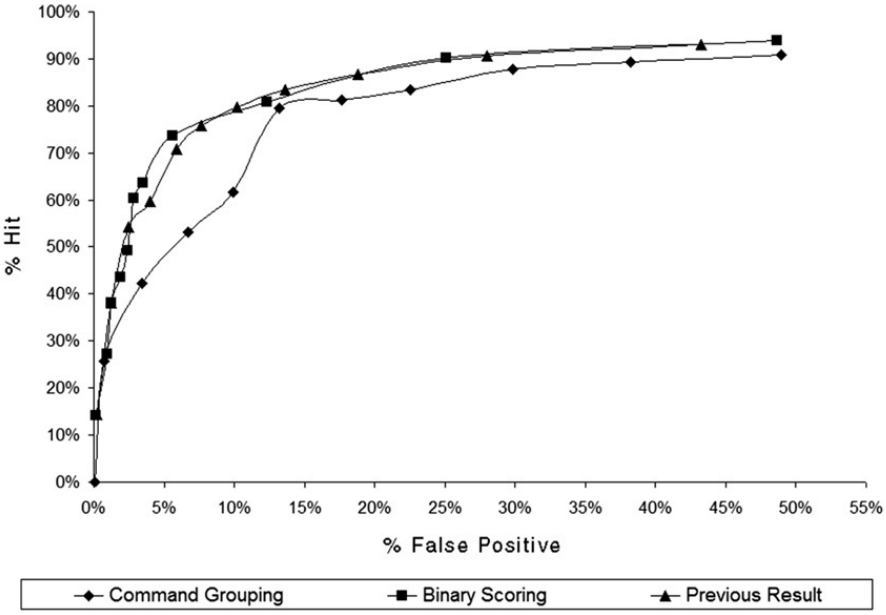

Fig. 4. Receiver operator characteristic curve for previous result, binary and command group scoring.

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.

Binary scoring. Previous work by Wang and Stolfo (2003) compared various features of the audit data sequences, and found that binary information about the presence or 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 Wang and Stolfo (2003) 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 been 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 other lexical mismatches are scored as −1. Thus, any command from a given user's signature can replace any other previously observed command from the signature, but the 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.

5.2. Scoring results

In the following evaluation, results from sequence alignment with the 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. Fig. 4 shows that the binary scoring system vastly outperforms the command grouping system. Table 2 provides the specific Maxion–Townsend scores for the three scoring systems, confirming the clear advantage of the binary scoring system. The results lend further credence to results, such as those provided by Wang and Stolfo, which suggest that the best indicator of legitimate user behavior is the previous observance of base symbols in the user's signature (Wang and Stolfo, 2003). In fact, when we examine several of the high scoring alignments under the binary scoring system, we find that many of the sequences that occur are simply permutations of previously seen commands into new sequences.

6. 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. The static signature will have no way of adapting to new user behavior, and therefore much of the variation will be considered to be masquerade attacks. As an example,

Table 2
Comparison of sequence alignment scoring systems

Technique	% Hit	% False Positive	Maxion–Townsend score
Binary Scoring	**60.3**	**2.9**	**56.8**
Previous Result (Coull et al., 2003)	75.8	7.7	70.4
Command Grouping	42.2	3.5	78.4

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 punish each mismatch created by the command calling this new program.

To overcome this complication, 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) augmenting areas in the *Signature* sequence to include the new behavioral information, (ii) adding new commands to the lexicon, and (iii) ensuring 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 (Maxion and Townsend, 2002). To choose which class 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 update its probabilities to reflect the commands found in this sequence. Similarly, the non-user classifier would be updated to reflect the new commands of non-user sequences. Note that since their technique assumes that commands appear independently of each other, 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.

6.1. Updating signatures with aligned sequences

To determine the way in which the *Signature* sequence and the user's lexicon should be augmented, we look to the scoring matrix that is created during the alignment process. This scoring matrix 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, however, 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 since the correct symbol already exists in the *Signature* sequence. 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. 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.

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

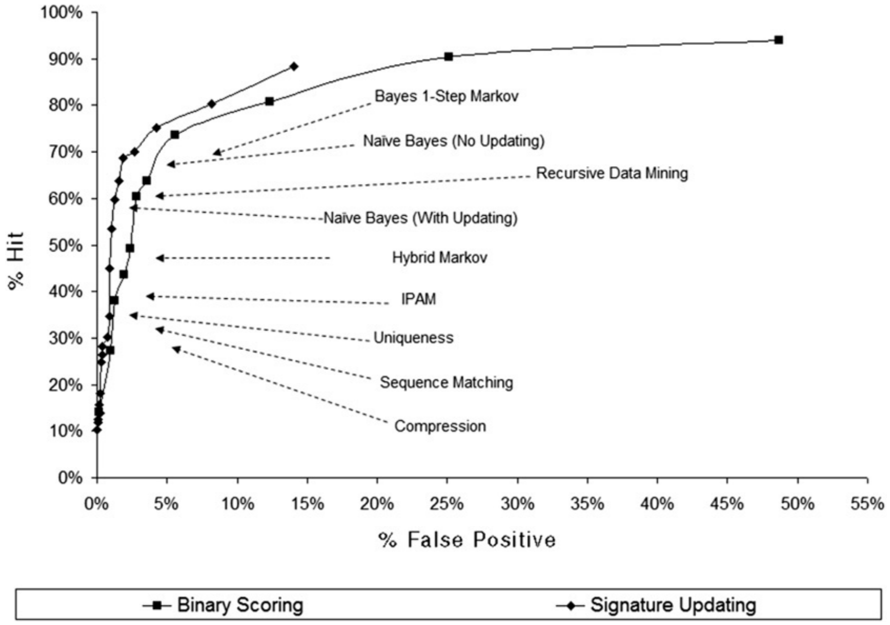

Fig. 5. Receiver operator characteristic curve for signature updating compared to binary scoring and previous results.

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. 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. Like Maxion and Townsend (2002), we perform our update whenever the alignment score is greater than our masquerade detection threshold, indicating that our detection system believes the sequence to belong to the user.

6.2. *Signature updating results*

In this evaluation, we use the sequence alignment algorithm with the binary scoring system as evaluated in Section 5.2, 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*.

Fig. 5 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, Fig. 5 and Table 3 indicate that the use of signature updating provides performance that surpasses all previous masquerade detection algorithms, in terms of both 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.

7. 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 its computational requirements. The computational efficiency of the sequence alignment algorithm is crucial for real-time masquerade detection deployments, especially when detection needs to be performed for multiple users utilizing an information system simultaneously. Depending on the granularity of the audit data and its rate of generation, the computational requirements of such detection could easily become

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

Technique	% Hit	% False Positive	Maxion–Townsend score
Sequence Alignment (Updating)	**68.6**	**1.9**	**42.8**
Naïve Bayes (With Updating) (Maxion and Townsend, 2002)	61.5	1.3	46.3
Sequence Alignment (Binary Scoring)	60.3	2.9	56.8
Recursive Data Mining (Szymanski and Zhang, 2004)	62.3	3.7	59.9
Naïve Bayes (No Updating) (Maxion and Townsend, 2002)	66.2	4.6	61.4

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 executed to test for masquerade attacks. Certainly, this is far too much computation to require for each user session, especially in multi-user environments or computationally limited devices, and thus a method for reducing the computational requirements is highly desirable.

7.1. A heuristic for aligning audit data

In essence, we must find some way of reducing the number of computations during the alignment without reducing the quality of the alignments or allowing masquerade attacks to escape detection. Our method of reducing the computations per alignment relies 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 $\leq n$. A high scoring alignment will always find that the *Test* sequence matches exactly with a subsequence of the *Signature* sequence, with few or no gaps inserted. This implies that, during any particular alignment, only subsequences of size $\leq 2n$ are ever used. That is, any subsequence that is larger than $2n$ in size would necessarily score poorly because the number of gaps being inserted equals or exceeds the number of matches. Therefore, we can focus on aligning the *Test* sequence with 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 $2n$ *Signature* subsequence. We need only choose the $2n$ subsequence(s) for which the sum of possible matches is the largest. We then align the *Test* sequence only with the maximum subsequence(s), 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 masquerade detection session.

7.2. Heuristic results

We evaluate the heuristic improvement in computational requirements of the detection process by: (i) empirically evaluating the average number of alignments needed to perform detection, and (ii) measuring the impact that this heuristic has on the detection performance of the sequence alignment algorithm. Unfortunately, since the previously published methods do not provide their computational requirements, our evaluation is limited to the methods introduced in this paper. In our evaluation, we augment the previously described sequence alignment algorithm using binary scoring and signature updating by implementing the heuristic as described above. For comparison, we refer to sequence alignment with the heuristic as *Heuristic*, and the alignment algorithm without it as *Signature Updating*.

Through our evaluation, we found that our proposed heuristic reduces the computational requirements of the sequence alignment algorithm significantly in the average case. Without the use of the heuristic, the sequence

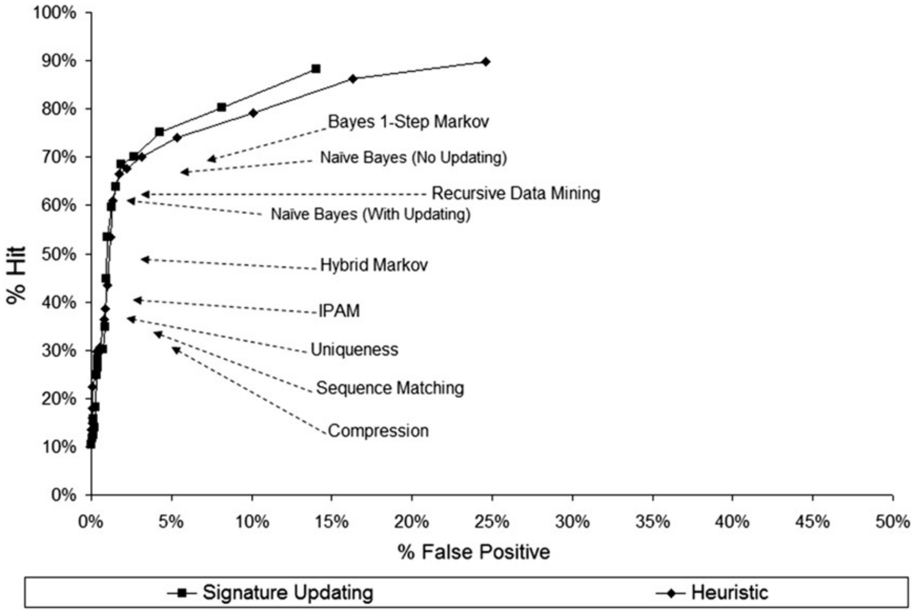

Fig. 6. Receiver operator characteristic curve for heuristic compared to signature updating.

Table 4
Comparison of the top five masquerade detection techniques sorted by Maxion–Townsend score

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) (Maxion and Townsend, 2002)	61.5	1.3	46.3
Sequence Alignment (Binary Scoring)	60.3	2.9	56.8
Recursive Data Mining (Szymanski and Zhang, 2004)	62.3	3.7	59.9

alignment algorithm requires 500,000 computations (one 5000 × 100 alignment). With our heuristic, however, the algorithm performs only 90,000 computations (4.5 200x100 alignments) on average, and a worst case of 980,000 computations (49 200 × 100 alignments). Concretely, when the algorithm is run on a commodity single-core, 2.4 GHz processor, the unmodified algorithm took 7319.5 s to complete the analysis of all users in the Schonlau dataset while the algorithm with the heuristic took only 528.74 s. Thus, our heuristic translates to a 10-fold improvement in the computational performance of the algorithm. More importantly, as Fig. 6 and Table 4 show, the use of this heuristic has minimal impact on the algorithm's ability to detect masquerade attacks.

8. 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 (Lane and Brodley, 1997, 1998, 1999; Maxion and Townsend, 2002; Maxion, 2003; Schonlau and Theus, 2000; Schonlau et al., 2001; Szymanski and Zhang, 2004; Wang and Stolfo, 2003). However, improvements to the accuracy of these techniques is highly desirable in providing practical masquerade detection. 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 (Coull et al., 2003) 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 task of masquerade detection.

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 contributions, therefore, focused on new methods for applying sequence alignment to 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 that were inspired by 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 the best of all previous techniques, including Maxion and Townsend's Naïve Bayes classifier. 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.[2] 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 anomaly detection.

Acknowledgements

The authors would like to thank Joel Branch and Eric Breimer for their insightful comments on earlier drafts of this work. Boleslaw Szymanski's research is continuing through participation in the International Technology Alliance sponsored by the US Army Research Laboratory and the UK Ministry of Defence.

References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., 1990. Basic local alignment search tool. Journal of Molecular Biology 215 (3), 403–410.

Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J., 1997. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. Nucleic Acids Research 25 (17), 3389–3402.

Bray, N., Dubchak, I., Pachter, L., 2003. AVID: A Global Alignment Program.

Coull, S., Branch, J., Szymanski, B., Breimer, E., 2003. Intrusion detection: A bioinformatics approach. In: Proceedings of the 19th Annual Computer Security Applications Conference, pp. 24–33.

Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C., 1978. A model of evolutionary change in proteins. Atlas of Protein Sequence and Structure 5 (Suppl. 3), 345–352.

Lane, T., Brodley, C.E., 1997. Sequence matching and learning in anomaly detection for computer security. In: Proceedings of the AAAI-97 Workshop: AI Approaches to Fraud Detection and Risk Management, vol. 49, pp. 43–49.

Lane, T., 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, pp. 259–263.

Lane, T., Brodley, C.E., 1999. Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security 2 (3), 295–331.

Maxion, R.A., 2003. Masquerade detection using enriched command lines. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 5–14.

Maxion, R.A., Townsend, T.N., 2002. Masquerade detection using truncated command lines. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 219–228.

Needleman, S.B., Wunsch, C.D., 1970. A general method applicable to the search for similarity in the amino acid sequence of two proteins. Journal of Molecular Biology 48, 443–453.

Pearson, W.R., Lipman, D.J., 1988. Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences 85 (8), 2444–2448.

Schonlau, M., DuMouchel, W., Ju, W.H., Karr, A.F., Theus, M., Vardi, Y., 2001. Computer intrusion: Detecting masquerades. Statistical Science 16 (1), 58–74.

[2] Example C++ code illustrating the various techniques described in this paper can be found at: http://www.cs.jhu.edu/~coulls/seqalignment.zip.

Schonlau, M., Theus, M., 2000. Detecting masquerades in intrusion detection based on unpopular commands. Information Processing Letters 76 (1), 33–38.

Smith, T.F., Waterman, M.S., 1981. Identification of common molecular subsequences. Journal of Molecular Biology 147, 195–197.

Szymanski, B.K., Zhang, Y., 2004. Recursive data mining for masquerade detection and author identification. In: Proceedings of the 5th Annual IEEE System, Man, and Cybernetics Information Assurance Workshop, pp. 424–431.

Tandon, G., Chan, P., 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, pp. 16–25.

Wagner, D., Soto, P., 2002. Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 255–264.

Wagner, R.A., Fischer, M.J., 1974. The string-to-string correction problem. Journal of the ACM 21 (1), 168–173.

Wang, K., Stolfo, S.J., 2003. One-class training for masquerade detection. In: Proceedings of the 3rd IEEE International Conference on Data Mining Workshop on Data Mining for Security Applications.

Wespi, A., Dacier, M., Debar, H., 1999. An intrusion-detection system based on the teiresias pattern-discovery algorithm. In: EICAR 1999 Best Paper Proceedings, pp. 1–15.

Wright, C., Monrose, F., Masson, G.M., 2004. HMM profiles for network traffic classification. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 9–15.

Wright, C.V., Monrose, F., Masson, G.M., 2006. On inferring application protocol behaviors in encrypted network traffic. Journal of Machine Learning Research 7, 2745–2769.

Zhang, Z., Schwartz, S., Wagner, L., Miller, W., 2000. A greedy algorithm for aligning DNA sequences. Journal of Computational Biology 7 (1–2), 203–214.