

Intrusion Detection: A Bioinformatics Approach

Scott Coull¹
coulls@cs.rpi.edu

Joel Branch¹
brancj@cs.rpi.edu

Boleslaw Szymanski¹
szymansk@cs.rpi.edu

Eric Breimer²
ebreimer@siena.edu

¹*Rensselaer Polytechnic Institute*
110 Eighth Street
Troy, NY 12180
(518) 276-8326

²*Siena College*
515 Loudon Road
Loudonville, NY 12211
(518) 786-5084

Abstract

This paper addresses the problem of detecting masquerading, a security attack in which an intruder assumes the identity of a legitimate user. Many approaches based on Hidden Markov Models and various forms of Finite State Automata have been proposed to solve this problem. The novelty of our approach results from the application of techniques used in bioinformatics for a pair-wise sequence alignment to compare the monitored session with past user behavior. Our algorithm uses a semi-global alignment and a unique scoring system to measure similarity between a sequence of commands produced by a potential intruder and the user signature, which is a sequence of commands collected from a legitimate user. We tested this algorithm on the standard intrusion data collection set. As discussed in the paper, the results of the test showed that the described algorithm yields a promising combination of intrusion detection rate and false positive rate, when compared to published intrusion detection algorithms.

Keywords

Intrusion detection, sequence alignment, bioinformatics, masquerade detection, pattern matching

1. Introduction

In the field of computer security, one of the most damaging attacks is masquerading, in which an attacker assumes the identity of a legitimate user in a computer system. Masquerade attacks typically occur when an intruder obtains a legitimate user's password or when a user leaves their workstation unattended without any sort of locking mechanism in place. It is difficult to detect this type of security breach at its initiation because the

attacker appears to be a normal user with valid authority and privileges. This difficulty underlines the importance of equipping computer systems with the ability to distinguish masquerading attacker actions from legitimate user activities.

The detection of a masquerader relies on a user signature, a sequence of commands collected from a legitimate user. This signature is compared to the current user's session. The underlying assumption is that the user signature captures detectable patterns in a user's sequence of commands. A sequence of commands produced by the legitimate user should match well with patterns in the user's signature, whereas a sequence of commands entered by a masquerader should match poorly with the user's signature. Designing algorithms to distinguish legitimate users and masqueraders based on user signatures has been extensively studied [12][14].

In this paper, we propose a new algorithm that uses pair-wise sequence alignment to characterize similarity between sequences of commands. Sequence alignment has been extensively applied in the field of bioinformatics as a tool for comparing genetic material [3][4]. Our algorithm, which is a unique variation of the classic Smith-Waterman algorithm [17], uses a novel scoring scheme to construct a semi-global alignment. The algorithm produces an effective metric for distinguishing a legitimate user from a masquerader.

To provide a self-contained paper, we describe the details of the intrusion detection problem and we introduce the fundamental concepts of sequence alignment. In subsequent sections, we describe the semi-global alignment algorithm, the scoring scheme, and the experimental results. We conclude with a discussion of future work and improvements.

2. Background

2.1 Intrusion Detection

Standard security deployments such as firewalls, patched operating systems and password protection are limited in their effectiveness because of the evolving sophistication of intrusion methods and their increasing ability to break through entry points of a guarded infrastructure [10]. An intrusion detection system (IDS) addresses the layer of security following the failure of the prior devices. This layer usually monitors any number of data sources (i.e., audit logs, keystrokes, network traffic) for signs of inappropriate or anomalous behavior. Since attacks occurring at this level are sophisticated enough to bypass entry point protection, advanced algorithms and frameworks for detection are required to prevent total subversion of critical resources. While no computer or network is entirely secure, intrusion detection is essential for any computer-based infrastructure, in which the value of its assets draws the attention of potential attackers. Traditionally, there have been two main classes of IDSs: host-based and network-based systems. A host-based IDS monitors the detailed activity of a particular host. Depending on the specific IDS implementation, any number of data sources can be used to search for malicious activity. Solaris Basic Security Module (BSM) provides system call traces which are typically used as datasets for host-based IDSs [15]. For instance, when an analysis of the BSM data shows signs of an intrusion, the IDS alerts the system administrator of an attack. In other implementations, host-based systems also use such identifying information as a user's keystrokes and command execution patterns.

Network-based IDSs monitor networks of computers and other devices (i.e., routers and gateways) that are normally subject to attacks. Subsequently, rather than using machine and process-oriented data, such as that from BSM, network-based IDSs primarily use data from network traffic in detecting intrusions. The most popular program used to capture network traffic is tcpdump, which can display or store every field belonging to a TCP packet [7]. Different implementations of network-based IDSs may serve different functions. For instance, some network-based systems may monitor only the traffic activity of a single host, while distributed tools may analyze the aggregate traffic information from a range of devices on the same network. To prevent confusion, we use data-centric definitions in distinguishing between host and network-based IDSs.

Network and host-based IDSs, can be further classified based on two methods of detection: anomaly detection and penetration identification. The former method attempts to differentiate "anomalous" activity

from the established normal operating behavior of a computer system, application, or user. Thus, in general, the IDS must first train on data representing normal behavior before it can be deployed in an operative detection mode. The principle advantage of an anomaly detection system is that it can detect previously unknown attacks [8]. Considering this, anomaly-based systems are strongly applicable to masquerade detection, which is the problem of focus in this paper. Penetration identification (often referred to as misuse detection) is the second major detection technique. After a "signature" is defined that identifies a manifestation of an attack, the attack can be discovered in either monitored network traffic or host-based audit trails. Penetration identification systems typically yield fewer false alarms; however, they require continuous updates, as their signature databases may become outdated fairly quickly.

There are many types of host and network intrusion attacks. Intrusion classifications can be based on intent. For instance, the denial-of-service attack aims to either completely shut down or degrade the performance of a network, computer or process. Remote-to-local attacks are used by assailants who attempt to illegally gain access to a machine on which they have no account. These attacks target one specific resource. On the other hand, surveillance (or scan) attacks use distributed software to find vulnerabilities across hundreds of machines. In 1998, a seminal study was performed by DARPA to evaluate the performance of various IDSs in detecting these attacks. Specific details about the attacks and IDS evaluation are available in [9].

In our work, we focus on masquerade attacks in which an assailant attempts to impersonate a legitimate user after gaining access to this legitimate user's account. Masquerade attacks often arise after a successful remote-to-local attack; however, masquerading can also result from far simpler means. An example is a temporarily unattended workstation with a legitimate user logon session still active and unlocked. Anyone can access such a workstation and all resources accessible through the logon account. The broad range of damage that can be performed via masquerade attacks (i.e., stolen documents, data, or email) makes them one of the most serious threats to computer and network infrastructure.

Matching the potentially devastating costs of masquerade attacks is the difficulty in successfully detecting them. As stated previously, masquerade detection falls under the cover of anomaly detection, which already poses a challenge in implementation alone. Masquerade detection adds another layer of complexity to the problem. A masquerader may happen to have similar behavioral patterns as the legitimate user of an account to which he or she is currently logged therefore escaping detection and successfully causing damage under the cover of seemingly normal behavior. Another problem is

caused by computer users' tendency toward concept drift—a change in activity that is not captured strongly in the original user signature. As a result legitimate user command sequences may differ enough from the signature to appear to be an intrusion. In the following, we will refer to missed attacks as false negatives and to false alarms as false positives.

There have been numerous attempts at successfully detecting masquerade attacks (minimizing false negatives) without degrading the quality of a user's session (minimizing false positives). A seminal work by Schonlau et al. [14] analyzes the performance of various masquerade detection methods. Results showed that the method yielding the lowest number of false alarms was *uniqueness*, which had a false positive rate of 1.4%. However, it had a false-negative rate of 60.0%. Another good performer was the *Bayes one-step Markov* with a false positive rate of 6.7% and a false negative rate of 30.7%. In another paper [12], Maxion and Townsend analyzed the sources of error made by the detection mechanisms covered by Schonlau et al. and proposed several improved methods, among which the *Naïve Bayes with updates* yielded excellent 1.3% of false positive rate with a respectable 38.5% of false negative rate.

We choose to depart from Schonlau and Maxion's approach in trying to detect masquerade attacks with greater accuracy by unconventional methods. Most masquerade detection attempts begin with an analysis of a user's command sequences, which is a logical step. This type of data represents a user feature often termed as biometric [1]. The behavioral features of biometrics, in general, include such characteristics as handwriting and speech patterns, inapplicable for computer masquerading. The physiological features include fingerprints and eye color—things that do not change over time but are not available for remote computer sessions. User's command sequences on a computer system will of course change over time, but an adequate record of his or her normal behavior will capture most sequence variations. Nonetheless, the classification of the data used for detection led us to an appropriate class of algorithms and mechanisms for effective detection: bioinformatics.

2.2 Sequence Alignment

Sequence alignment is a well-studied tool used to quantify and visualize similarity between sequences. Sequence alignment has been most prominently applied in the comparison of genetic material such as DNA, RNA, and protein sequences [4]. The applications of sequence alignment include searching sequence databases for specific genes or patterns [3], and discovering phylogenetic relationships through the use of multiple alignments [2].

Sequence alignment is a generalization of the classic longest common subsequence problem (*lcs*). Given two strings $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$, with $m \leq n$, over some alphabet Σ of size s , the *lcs*-problem is to find a sequence of greatest possible length that can be obtained from both A and B by deleting zero or more (not necessarily adjacent) characters. Alternatively, the *lcs*-problem can be described as the problem of aligning two strings in order to maximize the number of matching characters by inserting gaps into either string in order to shift the characters into matching alignment.

The length of the longest common subsequence is an intuitive measure of similarity between two strings [17]. To improve its capabilities as a tool for comparison, a scoring function can be used to rank different alignments so that biologically plausible alignments score higher. The scoring function assigns positive scores to aligned characters that either match or are known to be similar. Negative scores are assigned to both aligned characters that are dissimilar and characters that are aligned with gaps. Typically, the score of an alignment is the sum of the scores of each aligned pair of symbols. The task of optimal sequence alignment is to find the highest scoring alignment for a given scoring function and pair of strings. An efficient dynamic programming algorithm for optimal sequence alignment was first presented by Needleman and Wunsch [13]. Similar to the length of the longest common subsequence, the alignment score serves as a metric for quantifying similarity among input strings. Alignments are not only a useful metric for measuring similarity, but the alignments themselves serve as an important visual tool in assessing the similarity. Figure 1 shows an example of a typical alignment where a dash indicates a gap and a vertical bar indicates a character match.

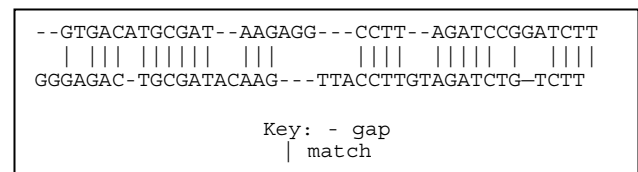


Figure 1: Example of sequence alignment

While computing the optimal alignment of two strings has proven to be a useful tool in the comparison of entire strings, it is often important to identify more subtle types of similarity. While two strings may not possess homogeneity over their entire length, they may contain smaller substrings that are highly similar. To accommodate for this possibility, Smith and Waterman [16] designed a modification of the Needleman-Wunsch alignment algorithm to compute a local alignment. Rather than align two strings over their entire length, the local alignment algorithm aligns a substring of each input

string. Given a scoring function and two strings $A = a_1a_2\dots a_m$ and $B = b_1b_2\dots b_n$, with $m \leq n$, the local alignment problem is to find substrings α and β of A and B , respectively, whose alignment score is maximum over all possible pairs of substrings from A and B .

Previously, an alignment implied that every character from one string had to be aligned with either a character from the other string or a gap. Thus, every character in the two input strings contributed to score of the optimal alignment. This type of an alignment is referred to as a global alignment. In a local alignment, only the characters in the two aligned substrings contribute to the score of the optimal alignment. Thus, for each string, a suffix and a prefix are ignored by the scoring system. Figure 2 shows the difference between a global and local alignment. By allowing a suffix and prefix to be ignored, a local alignment can discover subtle regions of similarity that may go undetected by a global alignment algorithm. While this problem appears to be much more difficult in terms of complexity, the Smith-Waterman local alignment algorithm is only a slight modification of the Needleman-Wunsch global alignment algorithm.

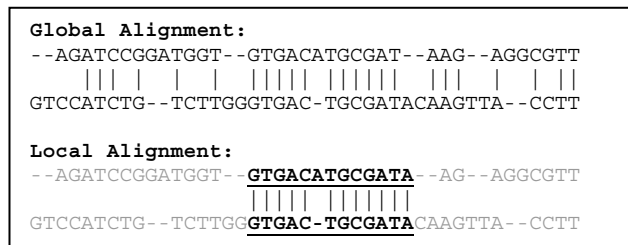


Figure 2: Global alignment vs. local alignment

Global alignment is the tool of choice when comparing two strings that are believed to possess overall similarity. Typically, the two strings are approximately equal in length. Whereas, local alignment is the tool of choice when comparing two strings whose lengths are significantly different. Local alignment allows a substring of the larger input string to be matched to the smaller string. Typically, a global alignment algorithm would fail to identify such similarity since most of the characters from the longer string would have to be aligned with gaps resulting in a negative score.

There are applications where neither local nor global alignment is appropriate for characterizing the types of similarity that may arise. These types of alignments are often referred to as semi-global. In a local alignment, both a prefix and suffix of both input strings can be ignored. Thus, the alignment only involves a substring of each of the two input strings. In a semi-global alignment, you can choose to align only prefixes or suffixes of the original input strings. In Figure 3, the first alignment allows only prefixes to be ignored, whereas the second alignment only allows suffixes to be ignored.

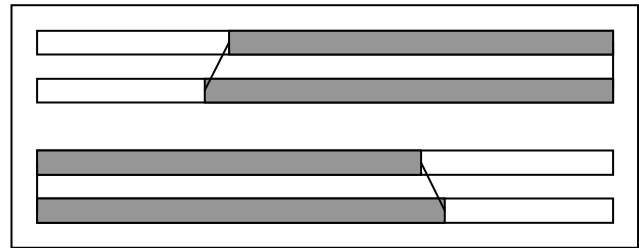


Figure 3: Examples of semi-global alignments

3. Detection Algorithm

3.1 Overview

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

There are a number of factors that predispose these sequence alignment algorithms for use in masquerade detection, namely their abilities to find high-level patterns within the alignment data and the sheer number of parameters that can be changed to suit different types of data. These parameters can be changed to allow for different alignments of the data, which can then bring about new high-level pattern matching. In particular, we can use these properties not only to match commands, but also to match generalized patterns that a user might be prone to over the course of a number of computing sessions. In this way, we are able to more readily judge how indicative a sequence of commands is of a user, not just by the commands themselves, but also by the high-level patterns embedded within the commands.

3.2 Alignment Algorithm

To use a sequence alignment in the detection of a masquerading user, we use a modification of the Smith-Waterman local alignment algorithm to compute a semi-global alignment. The problem with using a purely local alignment to characterize similarity between command sequences is that both a prefix and suffix can be ignored in both sequences. For intrusion detection, it is critical

that we align the majority of the tested block of commands to the user's signature. If we were to allow a large prefix and large suffix of the tested block of commands to be ignored then the intrusion itself might be ignored. The problem with using a purely global alignment is that there may be large portions of the signature that do not necessarily align with a segment of the user's commands. Thus, we want to design a scoring system that rewards the alignment of commands in the user segment but does not necessarily penalize the misalignment of large portions of the signature. In the remainder of this section the signature sequence, which represents the user's typical command behavior, will be referred to as the *UserSig*. The monitored command sequence, which may contain a possible subsequence of masquerader commands, will be referred to as the *IntrBlck* (tested block).

The algorithm, shown in Figure 4, starts by initializing a matrix of floats, which is used to store the score throughout the alignment process. Each position (i, j) in the matrix corresponds to the optimal score of an alignment ending at *UserSig_i* and *IntrBlck_j*. This optimal score is computed by starting at the upper left corner of the matrix (i.e., at the point $(0,0)$) and then recursively making a step yielding the maximum from the three following options:

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

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

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

If Option 1 yields the largest value, then the optimal alignment matches *UserSig_i* with *IntrBlck_j*. If Option 2 or Option 3 yields the largest score, then the optimal alignment associates either *UserSig_i* or *IntrBlck_j* with a gap.

There are three essential parameters used in the scoring system. The $matchScore(UserSig_i, IntrBlck_j)$ function returns a negative value if the two commands do not match well and a positive value if they do. The $gUserSig$ and $gIntrBlck$ are negative gap penalties associated with inserting gaps into the *UserSig* and *IntrBlck*, respectively.

If Option 1 or Option 2 results in a negative value, then the alignment score is reset to zero. This zeroing of the score allows a prefix of both the *UserSig* and *IntrBlck* to have an arbitrary number of un-penalized gaps. The assumption is that a portion of the *UserSig* can be ignored without penalty. Since the *UserSig* is significantly longer than the *IntrBlck*, it is expected that most of the commands in the *UserSig* will not participate in the alignment. Also, a small portion of the *IntrBlck* can be ignored. However, there is a difference in ignoring portions of *UserSig* and *IntrBlck*, since a high alignment score should not be achievable if a large portion of the *IntrBlck* is ignored. Thus, any alignment that ignores a large prefix of the *IntrBlck* should have a relatively low score. Similarly, when the algorithm reaches the right-most column or the bottom-most row of the matrix, the gap penalty is not applied. Thus, either a suffix of the *UserSig* or a suffix of the *IntrBlck* is ignored. Once again, if the latter is true then the alignment score will be relatively low.

```

Input: string UserSig of length m, string IntrBlck of length n
1. Initialize a matrix, D, of type integer
2. for i=0 to m
3.     for j=0 to n
4.         if (j=0 or i=0)
5.             D[i][j]=0;
6.         else
7.             if (j=n or i=m)
8.                 top=D[i][j-1];
9.                 left=D[i-1][j];
10.            else
11.                top=D[i][j-1] - gUserSig;
12.                left=D[i-1][j] - gIntrBlck;
13.                if (top<0) top=D[i][j-1];
14.                if (left<0) left=D[i-1][j];
15.                diagonal=D[i-1][j-1] + matchScore(UserSigi-1, IntrBlckj-1);
16.                D[i][j]=maximum(top, left, diagonal);
17. return D[m][n];

```

Figure 4: Semi-global alignment algorithm

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

In order to understand this algorithm’s viability in a real-time environment, we must consider its time complexity, so that we can determine how quickly it will be able to run. Our algorithm, like the original Smith-Waterman algorithm, has a time and space complexity of $O(mn)$, where m is the size of the *UserSig* and n is the size of the *IntrBlck*. In general, this is not a very quick algorithm; however, in the case of the masquerade problem, our set of data is relatively small, and, therefore, doesn’t hamper the real-time discovery of intruders. In specific, we have a *UserSig* size of 5000 and an *IntrBlck* size of 100 for each test, so we then have 500,000 iterations, which a modern computer could perform in less than a second. Additionally, we must consider that the use of commands by a user will not occur at such a fast rate as to cause the time complexity of this algorithm to become a factor in the detection of intruders.

3.3 Scoring Scheme Determination

The goal of our alignment algorithm is to match characteristic groups of commands in a tested block with similar groups in the user’s signature. This requires that we heavily penalize any gaps that may be inserted into the user signature, as we do not want commands in the tested block to be aligned with gaps in the user’s signature. Similarly, we would like to be able to insert gaps into the tested block to simulate the insertion of commands between characteristic groups of commands in the user’s signature. This requires that we provide a slightly lesser penalty for gaps in the tested block. Matches should positively influence the score of an alignment, and should be chosen so that matches are preferred to gaps. Mismatches are kept at a constant score of 0, as a blanket reward or penalty for any mismatch would unfairly favour certain alignments, and would not disallow concept drift.

Given the above criteria, we chose scores of +1 for a match between two aligned commands, -2 for a gap placed in the tested block, -3 for a gap placed in the user’s signature, and, of course, 0 for a mismatch between aligned commands. This scoring scheme appears to provide very reasonable detection and false positive rates, and is intuitively suited to the requirements of our problem.

4. Experiment Overview

4.1 SEA Data

To facilitate comparison with other masquerade detection algorithms, we have chosen to use the masquerade data provided by Schonlau et al. [15], abbreviated to SEA, as a basis for our experimentation. The SEA data was created using the UNIX *acct* auditing utility, which records user’s commands augmented with other metrics of interest. For our use, we only concern ourselves with the command entries that have been produced by this utility. The SEA data provides 50 blocks of 100 commands each (5000 total commands) for each user, which can be assumed to be intrusion-free and are used as training data for our system. In addition, we are provided with 100 blocks of 100 commands each (10000 total commands) for each user, in which we must determine if a masquerade attack has occurred. To create this data, commands were taken from 70 individual users, and separated into two groups. One group, made up of 50 users, was used as our test subjects, while the other group, made up of the remaining 20 users, had their commands interspersed into the data of the 50 user test group. The data from the 20 users was to be used as the masquerade data to be detected. The SEA data has been the de facto standard for masquerade detection algorithm testing thanks to its wide-spread use and the difficulty of obtaining alternative data due to privacy concerns. As a result, SEA data is the obvious choice for our tests.

4.2 Experiment Metrics and Parameters

Our experimentation focuses on the effects of changing the various parameters of the alignment algorithm on the false positive and false negative rates. One of the benefits of this particular approach is the sheer number of tunable parameters. These parameters include: reward for matches, penalties for gaps inserted into the user’s signature or into the tested blocks, rewards or penalties for mismatches, the threshold score for detection of intrusions, user signature length, and tested block length.

To best facilitate comparison with other masquerade detection algorithms, we use false positive rate, false negative rate, and hit rate metrics to determine how well our alignment algorithm performed. A false positive is a non-intrusion block that the algorithm labeled as containing an intrusion. A false negative is an intrusion block that the algorithm has labeled as non-intrusion. Finally, a hit is an intrusion block that the algorithm has properly labeled as containing an intrusion. False

positives, false negatives and hits are computed for each user, transformed into corresponding rates, which are then summed and averaged over all 50 users. Figure 5 summarizes the metric calculations used by the algorithm.

<p>f = number of false positives n = number of non-intrusion command sequence blocks u = number of users (50 in our case) false positive_{overall} = $(\sum_{users} (f_i/n_i)/u) * 100$</p> <p>fn = number of false negatives n = number of intrusion command sequence blocks c = number of users who have at least one intrusion block false negative_{overall} = $(\sum_{users} (fn_i/n_i)/c) * 100$</p> <p>hit rate_{overall} = 100 - false negative_{overall}</p>

Figure 5: Metric calculations

5. Results

5.1 Threshold Determination

To facilitate proper detection, a threshold score must be determined to define at which point a score is indicative of an attack. Rather than choosing an arbitrary and static threshold score, we decided instead to determine the initial threshold score for each user by cross-validating the user’s signature against itself. We do this by taking 20 randomly chosen, 100 command sections of the user’s signature and aligning it to a randomly chosen 1000 command section of the same user’s signature. This allows us to create an initial average score that is similar to the score that the user’s testing data should produce. Additionally, we update this average as new testing blocks are checked by averaging the current testing block’s score, and all tested block scores previous to it, with the initial average produced by the training data. We then take a percentage of that average as the threshold score. This allows us to customize the threshold for each user so that if a particular user did not have consistently high scoring alignments with their user signature, this user’s testing blocks will not be unduly flagged as intrusions. This, in particular, allows our algorithm to be somewhat forgiving of concept drift.

We are also able to choose a threshold percentage which is appropriate with the amount of sensitivity which we would like to express in the detection process. For instance, if we are more concerned with keeping a secure environment, then we would not mind an additional amount of false positive alarms in exchange for increased masquerade detection, so we can then use a higher percentage threshold so that the required alignment score would need be much closer to that user’s average score to

be considered a non-intrusion. Conversely, we can choose a lower threshold percentage, which would allow for a more lax security environment with less intrusive alerts by allowing the score to be significantly lower than the average of the user.

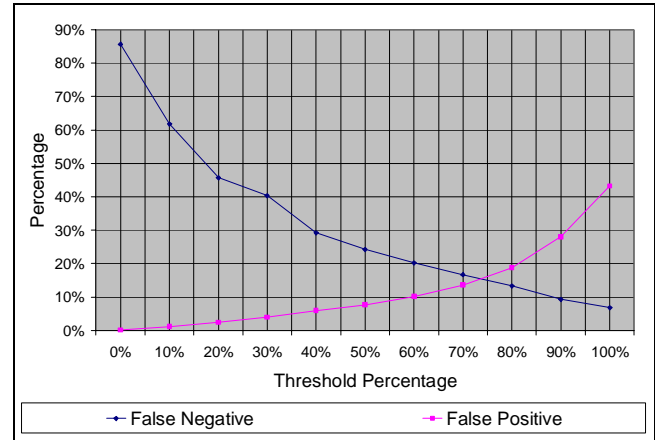


Figure 6: False negative and false positive vs. threshold percentage

5.2 Comparison to Local Alignment

As previously discussed, our semi-global alignment algorithm is actually a modification of the Smith-Waterman local alignment algorithm [17]. By comparing our semi-global alignment algorithm to the original Smith-Waterman algorithm, we are able to identify the unique ability of our modified algorithm to detect masquerade attacks in the SEA data. This comparison also gives us an indication of an appropriate length for the user’s signature. Good results for the local alignment algorithm, which were not achieved, would indicate that the tested block could be better aligned with a subsequence of the full 5000 command user signature sequence, rather than the full user signature.

While the local alignment algorithm performs comparably with our modified semi-global algorithm in areas of low sensitivity (low false positive rates and low hit rates) and high sensitivity (high false positive rates and high hit rates), it falls significantly below the performance of our algorithm for median sensitivity, arguably the most significant area of detection because it provides the best trade-off between detection hit rates and false positive rates. This indicates that using subsequences of the user’s signature provides no benefit to the detection process. Additionally, breaking the 5000 command user signature into subsequences introduces additional logistical problems for patterns which may cross subsequence boundaries. It is, therefore, most intuitive to keep the 5000 command user signature as one sequence, and to change the parameters of the alignment algorithm to

discourage gaps in the user signature, as we mentioned above, to provide an accurate alignment of the tested block to the user's signature. Similarly, it is intuitive to use a tested block size of 100 commands because the SEA data marks each 100 command block as an intrusion or a non-intrusion, and provides no information on which specific commands make up the intrusion. This limits the tested block size to 100 commands, as larger or smaller block sizes could not be checked for correctness.

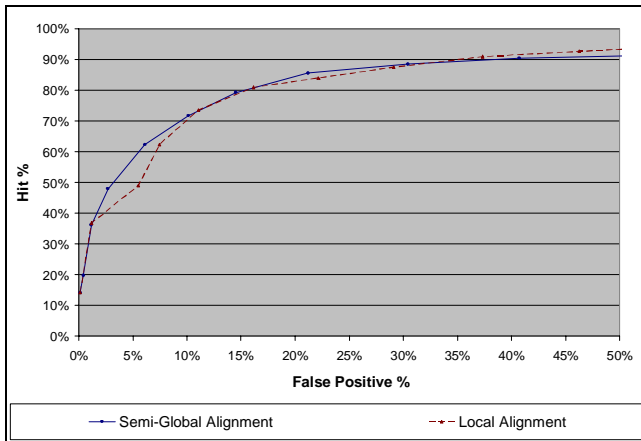


Figure 7: Hit rate as a function of false positive rate for semi-global and local alignment methods on SEA data

5.3 Command Mismatch Scoring

While the semi-global alignment algorithm works fairly well by not rewarding or punishing mismatches, these mismatches can be used to better determine how well the tested block aligns to the user's signature, and therefore better tailor our algorithm to the problem of masquerade detection. We can use a customized mismatch scoring system to allow for the possibility that the legitimate user may have interchanged one command with another in a particular alignment. This allows us to punish commands that are not as likely to be interchanged while rewarding commands that have a good likelihood of being interchanged with each other. Figure 8 summarizes the mismatch score calculation.

<p>M = Mismatch score S = # of occurrences of the intrusion block command in the user's signature C = # of distinct commands in the user's signature</p> $M = \lfloor S / (5000 / C) \rfloor - 1$ <p>If $(M > 1) \{ M = 1 \}$</p>

Figure 8: Mismatch score calculation

We use the ratio of the number of times a particular command in the tested block actually occurs in the user's signature to the expected number of occurrences a command in the user's signature. We then subtract 1 from this ratio and limit the maximum score to 1. This essentially puts the mismatch score on a real number scale from -1 to 1, such that if the tested block's command never occurs, or occurs fewer times than the average command, we penalize the mismatch, but if the tested block's command occurs more times than the expected average number of occurrences per command, we reward the mismatch. Meanwhile, if the particular command has the same number of occurrences as the expected average number of occurrences per command, then we neither reward, nor penalize this mismatch, as it does not definitively indicate whether that command was entered by the legitimate user or from a masquerader.

After implementing this mismatch scoring scheme, our results drastically improved over the previous semi-global algorithm where mismatches were neither rewarded, nor penalized. Our selective reward and penalty of mismatched command alignments based on command frequency allows us to differentiate between a user and a masquerader by taking into account concept drift in our tested block sequences and allowing small variations in user activity based upon their past activity.

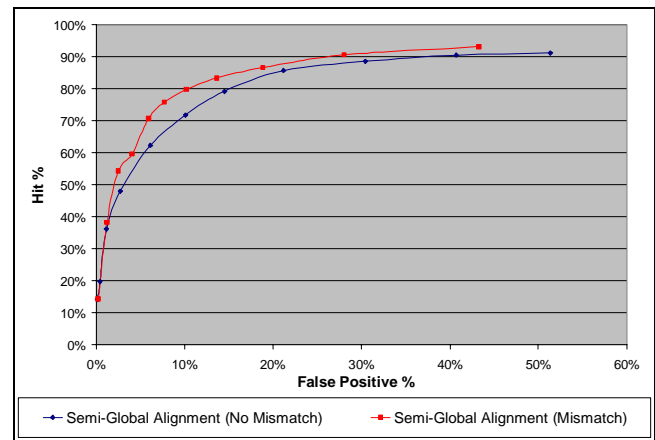


Figure 9: Hit rate as a function of false positive rate for semi-global with mismatch scoring and semi-global without mismatch scoring

5.4 Overall Results

After tuning the algorithm, as described above, we have produced a hit rate and false positive rate that are extremely competitive with other top masquerade detection algorithms. The only algorithms that perform comparably with our current results are the *Naïve Bayes* algorithms. All other algorithms perform somewhat worse than our current best results, and though they may fall

near our Receiver Operator Characteristic (ROC) curve, their detection abilities are clearly far below our 75.8% hit rate [13].

Table 1: Comparisons to other algorithms

Algorithms	Hit Rate	False Positive
Semi-Global Alignment	75.8%	7.7%
Bayes 1-Step Markov	69.3%	6.7%
Naïve Bayes (no updating)	66.2%	4.6%
Naïve Bayes (updating)	61.5%	1.3%
Hybrid Markov	49.3%	3.2%
IPAM	41.1%	2.7%
Uniqueness	39.4%	1.4%
Sequence Matching	36.8%	3.7%
Compression	34.2%	5.0%

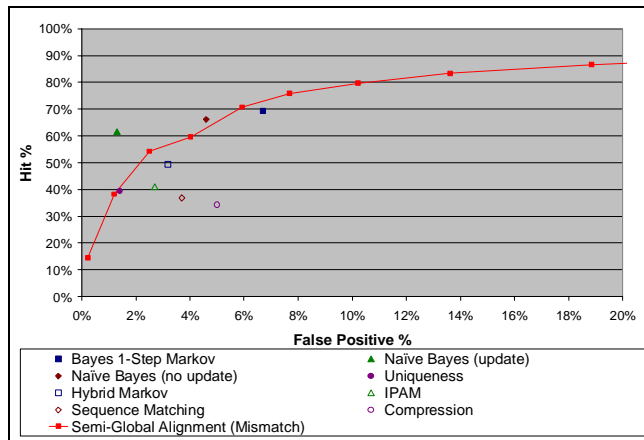


Figure 10: ROC curve with comparison points

6. Discussion & Future Work

Bioinformatics, as an area of study, is peculiarly suited to create algorithms that can be applied in a myriad of fields. The sequence alignment algorithms, as we discussed here, are actually specialized pattern matching algorithms, which, with some tuning, can be duly applied to many different fields in which pattern matching is applicable, intrusion detection in our case. Our particular system is also equally applicable to graphic user interface (GUI) interactions, as those interactions can be broken down to various system calls, which would produce a set of system calls. This set of system calls could then just as easily be analyzed by this algorithm to determine intrusions from GUI interactions.

Wepsi et al. have also noticed this peculiar and novel use of bioinformatics algorithms in their use of the *Teiresias* pattern matching algorithm to flag abnormal Unix system calls that might indicate an attack on a Unix process [18]. While both our alignment algorithm and the

Teiresias algorithm originated in the domain of bioinformatics, their approaches to detection vary considerably. In particular, we use sequence alignment to score similarity between command sequences whereas Wespi et al. use dominant patterns to classify abnormality in Unix processes.

We have presented a novel implementation of a modified sequence alignment algorithm for the detection of masqueraders, and shown that, with appropriate customization and tuning, it performs competitively when compared to the top masquerade detection algorithms. While a significant amount of customization has been done to the generic Smith-Waterman local alignment algorithm to produce a good masquerade detection algorithm, there are still a number of additional metrics which we could use for improvements in our algorithm's performance.

One great advantage of using alignments to characterize similarity between command sequences is that the alignment can produce many different statistics. These statistics include the density of the alignment (alignment score divided by alignment length), the maximum, minimum, and average gap length, the total number of matching and mismatched commands, and the total number of gaps in each of the aligned subsequences. These statistics measure different aspects of the similarity and they can be used to further distinguish user subsequences from intruder subsequences.

Though this algorithm's initial false positive rate is somewhat lackluster, we much consider that this is a new method of intrusion detection, and as such, initial testing was needed to determine its viability. While the alignment score is effective in identifying intruders, it often misidentifies user subsequences as an intruder. This may be the result of uncharacteristic user behavior, which can be identified and ignored. Fortunately, there may be subtle differences between uncharacteristic user behavior and intruder behavior, which can be detected by looking at the alignment statistics in a multidimensional space. A multidimensional approach using several different alignment statistics could be a more powerful and robust mechanism for decreasing the false positive rate of our algorithm. Additionally, the parameters of the scoring algorithm itself (gap penalties, mismatch scoring, and match scoring) can be tuned even further to allow for a more dynamic scoring system, similar to what has already been done with the mismatch scoring, to better separate legitimate user activity from malicious attack. Furthermore, this method is significantly different from other intrusion detection technologies, and it is, therefore, particularly well suited to coupling with existing intrusion detection technologies in a hybrid system that could provide even more impressive results.

7. References

- [1] Ashbourn, J. Biometrics: advanced identity identification: the complete guide. Springer-Verlag, London, 2000.
- [2] Carrillo, H., and Lipman, D. The multiple sequence alignment problem in biology. *SIAM Journal of Applied Math* 48(5): 1073-1082, 1988.
- [3] Gelfand, M., Mironov, A., and Pevzner, P. Gene recognition via splices sequence alignment. *Proc. Natl. Acad. Sci. USA*, 93: 9061-9066, 1996.
- [4] Goad, W., and Kanehisa, M. Pattern recognition in nucleic acid sequences: a general method for finding local homologies and symmetries. *Nucleic Acids Research*, 10: 247-263, 1982.
- [5] Hirshberg, D. S. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24: 664-675, 1977.
- [6] Hubbard, T., Murzin, A., Brenner, S., and Chothia, C. Scop: a structural classification of proteins database. *Nucleic Acids Research*, 25: 236-239, 1997.
- [7] Jacobson, V., Leres, C., and McCanne S. Tcpcdump, June 1989. Available via anonymous FTP from ftp.ee.lbl.gov.
- [8] Kemmerer, R. A., and Vigna, G. Intrusion detection: a brief history and overview. *IEEE Security & Privacy*, 2002. URL <http://computer.org/computer/sp/articles/kem/>
- [9] Lippmann, R., Cunningham, R. K., Fried, D. J., Graf, I., Kendall, K. R., Webster, S. E., and Zissman, M. A. Results of the darpa 1998 offline intrusion detection evaluation. *Recent Advances in Intrusion Detection, 1999, Second International Workshop, RAID 1999*.
- [10] Loshin, P. Intrusion detection. *Computer World*, April 2001. URL <http://www.computerworld.com/hardwaretopics/hardware/story/0,10801,59611,00.html>.
- [11] Marchette, D. J. Computer intrusion detection and network monitoring. Springer-Verlag New York, Inc., New York NY, 2001.
- [12] Maxion, R. A. and Townsend, T. N. Masquerade detection using truncated command lines. *Proceedings of the International Conference on Dependable Systems and Networks (DSN-02)*, 219-228, Washington, D.C., June 2002. IEEE Computer Society Press, Los Alamitos, California.
- [13] Needleman, S. and Wunch, C. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48: 444-453, 1970.
- [14] Schonlau, M., DuMouchel, W., Ju, W., Karr, A. F., Theus, M., and Vardi, Y. Computer intrusion: detecting masquerades. *Statistical Science*, 16(1): 58-74, February 2001.
- [15] S.M. Inc. Sunshield basic security module guide, solaris 7. Sun Part No. 8052635-10, October 1998.
- [16] Smith, T. F. and Waterman, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147: 195-197, 1981.
- [17] Wagner, R. A. and Fisher, M. J. The string-to-string correction problem. *Journal of the ACM*, 21:168-173, 1974.
- [18] Wepsi, A., Dacier, M., and Debar, H. (1999) 'An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm' EICAR 1999 Best Paper Proceedings