RESEARCH ARTICLE

Sequence-Based Masquerade Detection for Different User Groups

S. Sen*

Department of Computer Engineering, Hacettepe University, 06800, Ankara, TURKEY

ABSTRACT

Insider threats are one of the biggest threats that organizations are confronted with today. A masquerader who impersonates another user for his malicious activities has been studied extensively in the literature. The approaches proposed on masquerade detection mainly assume that masquerader behavior will deviate from the typical behavior of the victim. This research presents a rigorous evaluation of sequence-based approaches based on this assumption. The main idea underlying sequence-based approaches is that users type similar commands, in a similar order, every time to do a specific job and, these similarities could distinguish users from others. Sequence-based approaches in the literature only consider commands typed in a specific order, at all times. In this research, we also take into account typing similar commands in a command sequence, but in an unordered way, in the newly proposed method, MUCS. We compare this new technique with another sequence-based approach called MOCS, and a command-based approach called MC. These techniques are evaluated with varying parameters in order to explore how the order of commands, the variations in a command sequence, and the variety of commands affect masquerade detection. Furthermore, the performance of these methods on different types of users and masqueraders is analyzed. We explore what kind of users are easily distinguishable from others, and what kind of masqueraders are difficult to detect. Copyright © 2010 John Wiley & Sons, Ltd.

KEYWORDS

sequence-based approaches ; masquerade detection ; insider threats ; naive Bayes classification ; bag-of-words ; user groups

***Correspondence**

Department of Computer Engineering, Hacettepe University, 06800, Ankara, TURKEY

Received . . .

Security Comm. Networks 2010; **00**:2–23
DOI: 10.1002/sec

1. INTRODUCTION

A malicious insider is a trusted insider who abuses his position of trust to disrupt operations, corrupt data, exfiltrate sensitive information, or compromise an IT system, thus causing loss or damage [1]. It is one of the biggest threats that organizations are confronted with today. 46% of the organizations that participated in a survey conducted by CERT (Computer Emergency Response Team), expressed that damage caused by insider attacks are more severe than outsider attacks [2]. Malicious insiders are also recognized to be within the top threats of cloud computing [3].

Masqueraders are among the insider attacks that have been discussed extensively over the last decade. A masquerader impersonates another user in order to carry out his malicious activities. He might not however, even have full knowledge of the system [4]. In order to detect these attacks, many approaches have been proposed in the literature, such as text-mining, HMM (hidden Markov model), naive Bayes, sequence-based, SVM (support vector machines), information-theoretic based and other similar means [5]. Most of these approaches show comparable results in terms of accuracy. In these approaches, the main underlying assumption is that a masqueraders' behavior will likely deviate from the victim's typical behaviors.

In this research, we investigate the use of sequence-based approaches on masquerade detection. In the literature, there are some sequence-based approaches proposed using the SEA dataset [6], which is widely used in masquerade detection. Even though this dataset consists of an ample amount of commands for 70 different users, these commands might not be typed in order due to the *acct* auditing mechanism used to collect user data. Since this may adversely affect the results of any proposed sequence-based approaches [7], we use both the SEA dataset and the Greenberg dataset [8] in this study.

This research proposes a rigorous investigation of sequence-based approaches for masquerade detection. It explores the effects on masquerade detection of contiguous/non-contiguous and ordered/unordered commands typed by users. We employ naive Bayes classification in three main different settings in order to show the effects of command sequences to differentiate a user from others. One technique, called MC (Matching of Commands) is an application of naive Bayes which uses only self-training data due to privacy issues. Although there are approaches using two-class naive Bayes in the literature [9][10], in this study we have used MC for comparison with the MOCS (Matching of Ordered Command Sequences) and MUCS (Matching of Unordered Command Sequences) methods. While the method MC is employed on user commands, the newer MOCS and MUCS methods use the bag-of-command sequences approach. The main idea underlying these approaches is that users type similar commands in a similar order every time to do a specific job and, these similarities could distinguish users from others.

Sequence-based approaches in the literature only consider commands typed in a specific order at all times. In this research, we also take into account typing similar commands in an unordered way in a command sequence, using the newly proposed method of MUCS. In both the MOCS and MUCS methods, some changes in the commands typed in a sequence are considered with a tolerance parameter. This research presents a rigorous evaluation of these three methods in order to investigate sequence-based approaches for masquerade detection thoroughly. We explore how the order of commands, the tolerance parameter, the variety of commands, and the different user groups affect masquerade detection. Finally we compare the proposed approaches with some other techniques in the literature. As it is shown in the results, the new approach, MUCS with tolerance, outperforms the naive Bayes classification technique, one of the most successful and applied approaches in the literature.

In this research, we use the SEA dataset for comparison with other approaches in the literature. We also employ the Greenberg dataset owing to its appropriateness for sequence-based approaches. This dataset includes four different types of users that differ widely to that used in the Schonlau dataset: novice users, non-programmers, computer scientists and experienced users. Therefore it allows us to analyze the performance of our approaches on different user groups/masqueraders. We explore what kind of users are easily distinguishable from others, what kind of masqueraders are difficult to detect, and such like. The performance of each technique demonstrates that different techniques could be more suitable for different users. As far as we know, this is the first attempt to analyze user groups from the masquerade detection point of view.

The paper is organized as follows. Section 2 presents an overview of the related work in the area of masquerade detection. The details of the datasets used in the experiments are presented in Section 3, and an overview of the naive Bayes classification is given in Section 4. The different experimental settings used in this study are defined in Section 5, and the results are exhaustively discussed in Section 6. Finally, the conclusions of this study are presented in Section 7.

2. RELATED WORK

In the last decade, masquerader detection has been studied using various techniques. Most of these studies use the Schonlau dataset (SEA dataset) introduced in [11], which consists of command lines typed by users. Schonlau *et. al.* applied six different approaches (uniqueness, Bayes one-step Markov, hybrid multistep Markov, compression, sequence-match, and IPAM- Incremental Probabilistic Action Modeling) to this publicly available dataset [6] in order to detect masquerade users. However the results show very low detection rates (between 34.2% and 69.3%), with the false positive rate between 1.4% and 6.7%.

Compression-based approaches [12][13][11] are based on the premise that data from the same user compresses more readily than mixed data from different users [12]. A compression-based classification algorithm called Normalized Compression Distance (NCD) is applied in [12] and is claimed to be used as a universal similarity measure different to the Compression method in [11]. The NCD approach is compared with the six approaches proposed by [11]. The results show that the false alarm rate produced by NCD is generally less than 2%, which is highly desirable in comparison with other approaches. However it costs a relative increase in missing alarms.

In a study by [9], the naiye Bayes classification is applied to the masquerade detection problem and is compared with other detection techniques given by [11]. It is shown that the naive Bayes classification shows a much better performance than other approaches. Moreover, Maxion and Townsend introduce a new experiment setting called 1v49, which also uses the Schonlau dataset. This setting provides a better environment for evaluating proposed approaches, since it includes more masquerader styled data than the SEA dataset. They also discuss why some attackers are more successful and why some users are harder targets. The naive Bayes classification approach has been applied to the Greenberg dataset which includes command lines enriched with flags, arguments, and information about aliases in [10]. The results show that enriched command lines bring about an improvement in masquerade detection. They also apply naive Bayes with updating thresholds in order to overcome the concept drift issue, and to decrease false positives [7]. Sen [43] has recently proposed an approach based on instance weighting in order to update user profiles, which outperforms the instance selection approach [7] in some scenarios. Yung [14] updated user behaviors with feedback from users in order to increase detection performance. A recent approach using naive Bayes is proposed in [15], where the detection of an attacker is deferred for 2-3 blocks.

The proposed approaches to detect masqueraders usually use multi-class training (two classes in this problem). However two-class training data might not always be available due to privacy issues. Therefore Wang and Stolfo apply one-class training to the problem and show that one-class training works as well as multi-class training [16]. This approach has other advantages, such as collecting less data, and more efficient training. Salem and Stolfo have also applied one-class SVM to the problem, and compared it with the Hellinger distance-based user behavior profiling technique [17]. Chen and Dong apply one-class classification using length statistics of emerging patters (OCLEP) [18]. The method is based on the assumption that two command blocks have long emerging patterns if both blocks are typed by the same user. It is shown that OCLEP achieves slightly better performance than one-class SVM. The most recent approach using one-class learning algorithm (Bayesian) is shown to outperform most previous one-class approaches [19].

The main assumption underlying the detection of such malicious users, is that the masqueraders' behavior will deviate from the typical behavior of the normal users. The researchers mainly use command frequencies (multivariate Bernoulli or multi-nominal model) or command transitions in order to define the user behaviors. Some approaches consider command sequences for user profiling. Lane and Brodley use sequence similarity where adjacent matches have a higher score to detect anomalous behavior on their own small dataset [20]. Latendresse generates a context-free grammar by using Sequitur algorithm [22], in order to extract repetitive sequences of commands executed for global user profiling [21]. Zhou *et al.* also use the same method and compare it with the Self-Signature approach, combined with the Uniquness approach in [23]. They show that by using repetitive 2-grams beside command frequencies, an improvement is shown in the statistical methods. Oka *et. al.* propose a new approach, called Eigen co-occurence matrix, which uses principal component analysis (PCA) to model command sequences and extract their significant features [24]. The model correlates two events with their frequency and the distance between them in a given maximum interval size (might not be adjacent to each other). It is shown that the ECM method outperforms many existing approaches. A method which takes into account both frequency and transition characteristics of commands, called n-gram Square Root Term Frequency Inverse Document Frequency (n-gram STF-IDF), is introduced in [25].

Seo and Cha [26] apply SVM with sequence-based string kernel which compares strings according to the non-contiguous sequences of substrings they contain. The results demonstrate that string kernel shows a slightly better performance than the RBF (radial basis function) kernel of SVM. The sequence alignment technique, which aims to find the maximal length of lexically similar subsequences of two strings and is generally used in bioinformatics, is adapted to the masquerade detection domain [27]. Even though the performance of this approach is comparative with the naive Bayes approach with updating, the computational requirements of this approach can be an issue. Another important contribution made by Coull and Szymanski, is the discussion of command grouping where similar commands ending up in the same group can be substituted. Sodiya *et al.* [28] improves the effectiveness of the sequence alignment algorithm by computing the best score for each user, however the results are only discussed on a small dataset. A similar approach, which uses sequence alignments to generate a profile hidden Markov model (PHMM), is proposed in [29]. Since the Schonlau dataset does not include session times, they also generate a synthetic dataset in order to show the performance of PHMM. A recent approach, based on Markov chain with states of variable length sequences, is proposed in [30]. Even though it produces better results by using less time and space than some fixed length approaches, it shows similar results with the naive Bayes approach with low false positive rates.

There are recent approaches which focus on evading masquerade detection. Tapiador and Clark investigate how a resourceful masquerader can successfully evade detection while still accomplishing his goals [31]. They also investigate how to obfuscate user profiles against profiling attacks [32]. Four attacks are introduced in order to evade naive Bayes and HMM-based masquerade detection algorithms [33]. In [34], masquerade attacks are generated synthetically, based on the profile of a target victim, in order to evade detection.

Even though we focus on masquerade detection based on command lines typed by users, as other studies in the literature using the Schonlau and Greenberg datasets have done, there are other useful characteristics of users which might be used for masquerade detection. Garg *et al.* examine user profiling in GUI based systems by considering mouse coordinates, keyboard activities, and mouse clicks, besides just looking at commands [35]. Nguyen *et al.* monitor system call activities in order to detect insider threats and examine the relationship between users and files, users and processes, and processes and files [36]. Salem and Stolfo model user search behaviors, based on the assumption that masqueraders do not know the file system, and have different search behaviors than the victim user being impersonated [37]. A recent approach using data across multiple applications is proposed in [38]. This new dataset will allow researchers to investigate which characteristics best represent the user profile.

3. DATA

3.1. The SEA dataset

The Schonlau dataset is the de-facto standard and widely used in most of the proposed approaches in the literature due to being one of the first datasets publicly available [11]. In this dataset, 70 users are recorded for several months, with 15,000 commands logged for each user. However the arguments of the commands are not logged due to privacy issues.

In the studies of masquerade detection, 50 randomly selected users are used to represent benign users and the remaining 20 users are used to represent masqueraders. In the literature, the first 5,000 commands of each user are used for training, and the remaining 10,000 commands are used for testing. The commands of masqueraders are randomly added into the benign users' testing set in blocks of 100 commands; however these blocks are not equally distributed across each user. If the current block is not belonging to a masquerader, a masquerader block is inserted with 1% probability, otherwise with 80% probability. This experimental setting is called the SEA dataset, and has been employed for comparison in most of the studies in the literature. Therefore, in this study we will also use the SEA dataset, particularly for the sake of comparison with other approaches.

3.2. The Greenberg dataset

The Greenberg dataset [8] is also utilized in this research. The Schonlau dataset is obtained by using the UNIX *acct* auditing mechanism. As stated by the authors in [11], some commands recorded by the system are not explicitly typed by the user such as commands in a shell file, make file, or similar. Moreover, the commands might not be logged in the order they are typed. For example, the commands from different terminals are not combined in order by *acct*, therefore, this dataset might not be the best choice for approaches looking at the order of commands. Since we aim to investigate the effect of command sequences on the detection of masqueraders, the Greenberg dataset [8] is a more plausible choice for this study. Furthermore, this dataset includes more information, such as command arguments, and aliases.

The Greenberg dataset contains commands typed in order by 168 users. In this dataset, the users are divided into four different user groups: novice programmers, experienced programmers, computer scientists, and non-programmers. Unfortunately this dataset does not include as many commands as the Schonlau dataset; with only 6 users in the dataset having more than 5,000 commands. In this study, we chose 60 users who have more than 1,400 commands. The set of users includes novice users, experienced programmers and computer scientists. Since there are insufficient commands for training in the user group of non-programmers, this group has been excluded as a benign user. The usage of commands for each group in the training is given in Table I. The first 1,000 commands of each user are employed for training to base a profile of self, and the next 400 commands are employed for testing in order to evaluate whether each user could recognize themselves. For masquerade data, 60 users out of 112 users are chosen. 400 commands from each of the 60 masqueraders are united, so in total, 24,000 commands are used in testing for evaluating the detection of masqueraders. The masqueraders are selected among user groups equally. Although the dataset also includes flags, grammars and arguments, and aliases, we only consider the commands themselves for the sake of simplicity in this study.

4. NAIVE BAYES MASQUERADE DETECTION

In this research, we use naive Bayes which performs well on masquerade detection [9][10]. The naive Bayes classifier is a supervised learning approach which is easy to implement and fast. It has been successfully employed to a range of applications, such as text classification. In naive Bayes, the probability of a text, x belongs to a class, y is computed as the probability P(y|x) and the highest probability predicts the class in which the text belongs to. In our problem, given a command sequence block s, the probability that the block belongs to user x (u_x) can be computed as:

User Groups	Avg. Number of Different Commands	Min. Number of Different Commands	Max. Number of Different Commands
Novice Programmers	53.2	24	112
Experienced Programmers	78.9	57	120
Computer Scientists	79.75	49	110

Table I. User Groups in the Dataset

$$P(u_x|s) = \frac{P(s|u_x)P(u_x)}{P(s)}. \quad (1)$$

P(s) is the probability of that specific command block occurring and it is usually omitted based on the assumption that each command has equal probability [16][31]. Naive Bayes assumes that all commands/command sequences in a block are independent of each other. This is the naive Bayes assumption. Based on this assumption, the probability that a command block is typed by a particular user u_x can be computed as:

$$P(s|u_x) = \prod_{i=1}^{|s|} P(s_i|u_x). \quad (2)$$

Hence, the formula which calculates the probability that the block belongs to user x becomes (the prior P(u_x) is also ignored):

$$P(u_x|s)_{log} = log(\prod_{i=1}^{|s|} P(s_i|u_x)) = \sum_{i=1}^{|s|} log(P(s_i|u_x)). \quad (3)$$

P_{s_i,u_x} is the probability of command sequence i for a particular user x. In this research, four different sequence sizes are employed: 1, 2, 3, 4. When the sequence size is 1, it means that the probabilities of one command line for a particular user x are considered. In this research, we use the multinominal event model (bag-of-words approach) for naive Bayes, which usually outperforms the multivariative Bernoulli model at large vocabulary sizes [39]. The performance of these two models on masquerade detection is compared in the literature [16]. Therefore, the probability of a command or command sequence for a particular user is computed based on the frequency of the command or command sequence in the training data with the given formula below [9]:

$$P(s_i|u_x) = \frac{Training\ Count_{s_i,u_x} + \alpha}{Training\ Data\ Length + (\alpha\ x\ Alphabet\ Size)}. \quad (4)$$

Here, $Training\ Count_{s_i,u_x}$ is the number of the command or command sequence (s_i) observed in the training data. α is a pseudo count to ensure that there are no zero counts for unseen commands or command sequences. The lower the α is, the more sensitive the classifier is. Therefore, it is chosen as 0.01 as in [9]. Alphabet size is determined separately for each user. Training data length is the same and fixed for each user. If the command sequence size is 1, the training data length is 1,000.

In [9][10], besides the self-probability given in Equation 3, the non-self-probability that a command block generated by non-self users is evaluated by using non-self-training data, which consists of command blocks generated by the remaining 49 legitimate users. Then, an anomaly score is calculated as the ratio of self-probability to non-self-probability. The effect of non-self-users on anomaly score is discussed in [40]. It is shown that when the number of never-seen-before-commands is large in a block, the probability that the test command block is generated by the self-user is more likely than the non-self-probability. However in this research, we use self-probability to evaluate a score. If the score is higher than a threshold, it indicates an illegitimate user.

In this research, naive Bayes classifier training with self-data (called one-class naive Bayes in some studies) is used due to privacy issues. Hence, a user only employs its self-data for training. This design is believed to be more appropriate for the problem. Moreover, it decreases the training time as a result of using less training data than multi-classification.

5. EXPERIMENTAL SETTINGS

In previous studies which employ naive Bayes classification for masquerade detection [9][10][16], only the frequency/occurrence of commands is taken into account. The order of commands and thus the frequency of command sequences are not addressed. However in this research we aim to analyze the effect of command sequences in user profiling. For that reason, we introduce three different experimental settings. All settings are implemented and evaluated with Java and MATLAB.

5.1. Matching of Commands (MC)

In this setting, only the commands and their probabilities are considered to build a model of self. The naive Bayes (one-class) equation given in Equation 3 are employed.

5.2. Matching of Ordered Command Sequences (MOCS)

In this research we aim to investigate the effect of command sequences in order to classify user behaviors as self or non-self. The main hypothesis is that repeated sequences of user actions can differentiate users from others. For example, when a user (called user Bob) logs on to the system at work, he might always do the same actions in order such as opening an e-mail client, then logging onto his home computer remotely. Or a system administrator could run the same commands and scripts in order to get and analyze system logs. Moreover, users respond in a similar manner to similar situations, which in turn lead to repeated user actions [20]. These are the reasons that the probabilities of ordered sequences in different sizes are examined in this setting.

The parameters of *sequence size* are defined as 1, 2, 3, 4. Ordered sequences of each user are obtained separately from each user's training data. When the sequence size is 1, the setting is equal to the MC setting. As given in Equation 2, the probability of a block generated by self-user is calculated by multiplying the probabilities of the sequences in the block. The *block size* is defined as 100 in the experiments.

In the MOCS approach, we assume that the usage of commands in order could be used to distinguish users from others. However users might not type the exact command sequence all the time, but use similar command sequences. For example, an experienced user might use different editors (vim, gedit, etc.), hence different commands for similar jobs from time to time. Moreover, users have variations in their behavior. This characteristic of a user is defined by a tolerance parameter in our experiments. The Tolerance parameter shows the number of commands in a sequence that a user might change, while the order of the commands in the MOCS approach is preserved. The *tolerance parameter* is selected as 1 for 2-command sequences, and as 1, 2 for 3-command sequences. For example, if we have "abc", "abd" sequences in the training set with the probabilities p_1 and p_2 respectively, when the user types the sequence "abz", the average of probabilities p_1 and p_2 is taken for this sequence with the tolerance of 1.

5.3. Matching of Unordered Command Sequences (MUCS)

As in the MOCS setting, the probabilities of sub-sequences in a block are considered. However, the sequences do not necessarily have to be in order in this setting. For example, user Bob sometimes logs onto his home computer remotely when he first logs onto his work computer, then he opens an e-mail client (in an order differing from his usual behavior). The main assumption is that even users behave similarly and type the same commands in similar situations, although the commands might not be typed in the same order every time (where the order of commands does not affect the outcome).

Therefore, we propose a new approach that considers the frequency of command sequences which might be similar to other sequences typed by the same user. Furthermore, the users might add some other commands into the sequence. In this setting, we take into account this situation by defining a *tolerance parameter*. Please note that the order of commands in a sequence is not preserved in this approach, which is different from the MOCS approach.

The *sequence size* is selected as 1, 2, 3, 4 and the *block size* is selected as 100 in the experiments. Again, when the sequence size is 1, the setting is equal to the MC setting. The *tolerance parameter* is selected as 1 for

2-length command sequences, and as 1, 2 for 3-length command sequences. For example, if we have “abc”, “bad” sequences in the training set with the probabilities p_1 and p_2 respectively, when the user types the sequence “zab”, the average of probabilities p_1 and p_2 is taken for this sequence with the tolerance 1. The basic parameters for each setting is summarized in Table II. Even though more experiments are carried out by using more parameters (sequence size and tolerance parameters) for the MUCS method, the parameters in Table II are enough to show the performance of each method. Hence other parameters employed are omitted for simplicity in the results.

6. RESULTS

6.1. Analysis of the Proposed Sequence-Based Approaches

We performed a number of experiments in order to evaluate the performance of the proposed methods on masquerade detection. Firstly, we compared our newly proposed methods with some of the approaches in the literature, mainly comparing our approach with recent and well-accepted studies. In this experiment, the SEA dataset is employed, since it is accepted as the de-facto standard. Figure 1 shows the ROC curve of the MUCS,3,1 method on the SEA dataset which performs with one of the best results. To clarify, the style of representation for methods shown in this paper includes the method, the sequence size, and the tolerance. For example, MUCS,3,1

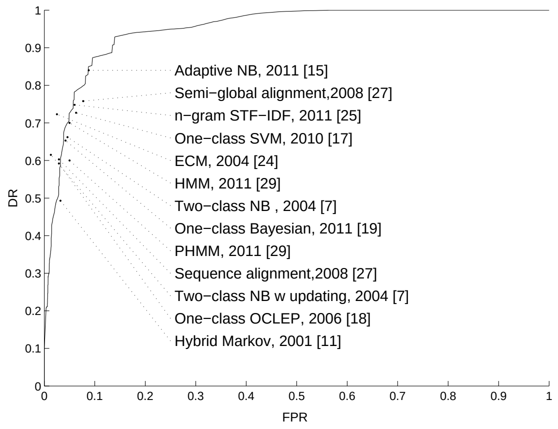


Figure 1. The Performance of MUCS,3,1 on the SEA Dataset

represents the method MUCS with sequence size 3 and a tolerance of 1. The ROC curve plots detection rate versus false positive rate. Detection rate (DR) shows the ratio of correctly detected attacks to the total attacks. False positive rate (FPR) shows the ratio of normal activities that are incorrectly marked as masquerade attacks to the total normal activities. The results are the average results of all users computed by employing threshold averaging [41]. As it is seen in the figure, the MUCS technique outperforms both the one-class and two-class naive Bayes techniques. The MUCS,3,1 method achieves a detection rate of 70% with a false positive rate of 4.6%. Please note that the updating mechanism is not applied here for simplicity.

In order to assess and compare the proposed approaches, the Greenberg dataset is applied in the subsequent experiments. First of all, the ROC curves of the exact matching methods (with no tolerance) at different sequence sizes (1, 2, 3, 4) are given in Figure 2. MOCS,2 represents the method MOCS with sequence size 2. The

Experiment Setting	Sequence Size	Tolerance
MC	1	N/A
MOCS/MUCS	2	1
	3	1,2
	4	1,2,3

Table II. Parameters of the Experiment Settings

results are the average results of all users at block size 100 for compatibility with the application of naive Bayes. As seen in the figure, the performance of the methods MC and MOCS,2 (exact matching of 2-length command sequences) are quite close. The users' behaviour of typing the same command pairs could distinguish them from masqueraders as the command-based approaches. When the sequence size increases, users do not type the commands in the same order, as so the performance of the MOCS method decreases as a result. To conclude, users could also be profiled with consecutive command pairs they type. In the future, command-based and sequence-based approaches could be employed together to achieve a better performance.

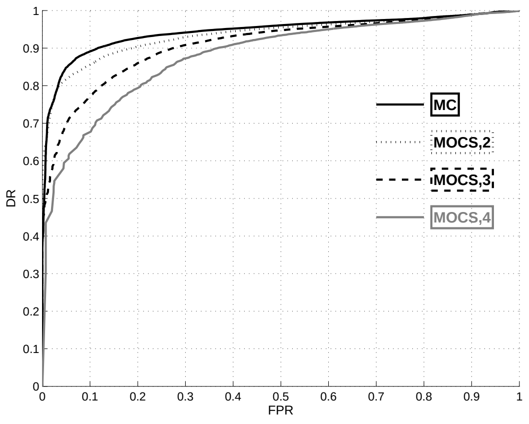


Figure 2. The Performance of MOCS on the Greenberg Dataset

6.2. Effect of Order of Commands on Masquerade Detection

In this section, the effect of the order of commands is investigated, but in order to do that, we first compared the MOCS and MUCS methods, with no tolerance, as shown in Figure 3. The best performance was from MUCS,2. When the order of 2-length command sequences is ignored, it shows a better performance than other sequence-based approaches on masquerade detection. In general, non-sequential methods (MUCS) always show better results than the corresponding sequential methods (MOCS).

A comparison of the MOCS and MUCS methods with tolerance is shown in Figure 4. Using the same style, MOCS,2,1 represents the performance of the MOCS method, with sequence size 2 and tolerance 1. It is observed that when the tolerance is taken into account at small sequence sizes, the order of commands does not

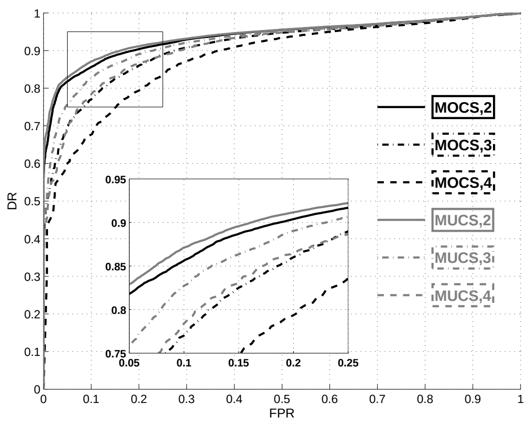


Figure 3. Effect of the Order of Commands on Masquerade Detection

show any difference on the performance. The accuracy of both methods (MOCS,2,1 and MUCS,2,1) at the same tolerance is quite close to each other. We could say that the effect of tolerance is much more influential than the order of commands.

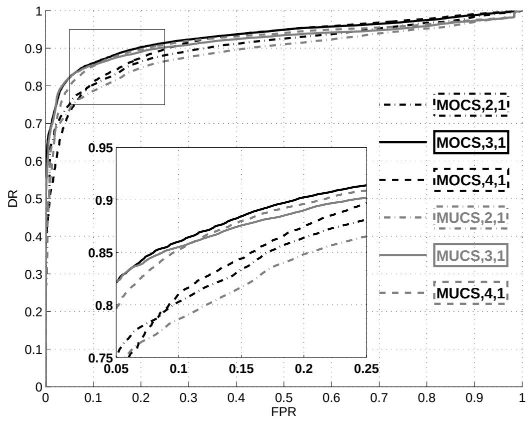


Figure 4. Effect of the Order of Commands with Tolerance on Masquerade Detection

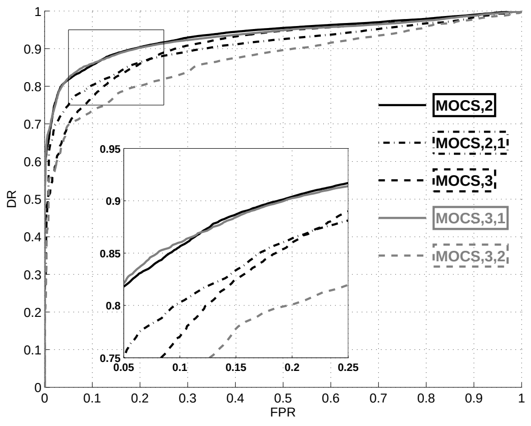
6.3. Effect of Tolerance on Masquerade Detection

The performance of the MOCS method at different sequence sizes (2, 3) and different tolerance parameters (1, 2) is demonstrated in Figure 5a. The effect of tolerance on the MOCS method at the sequence size 4 is given in Figure 5b for the purposes of adding clarity. It is shown that the methods MOCS,3,1 and MOCS,4,1 give a much better performance than the methods MOCS,3 and MOCS,4 respectively. It could be concluded that users frequently type the same sequences, but with minor changes.

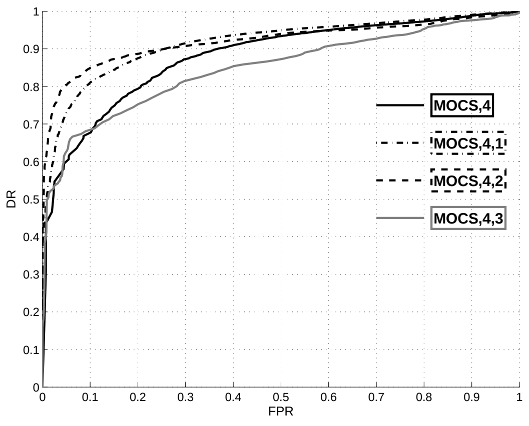
The performance of the MUCS method at different sequence sizes (2, 3, 4) and different tolerance parameters (1, 2, 3) is demonstrated in Figure 6a and Figure 6b. It is observed that the tolerance parameter for MOCS method is much more effective than for MUCS. Both methods (sequential and non-sequential) show that an increase in the change tolerated in a sequence (more than half of the sequence) negatively affects the performance.

6.4. Effect of Variety of Commands on Masquerade Detection

We have shown the results on average for all users so far. However there are different types of users with different behaviors in the data set. We show the performance of the MC method for different user groups in Figure 7. The user groups are constructed according to the number of different commands they use. MC 50-75 represents the users whose average command size is between 50

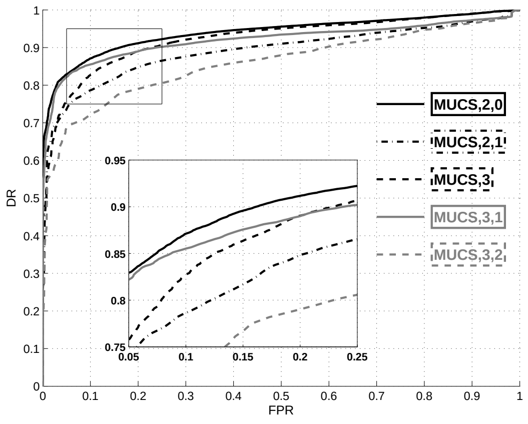


(a) Sequence size: 2, 3

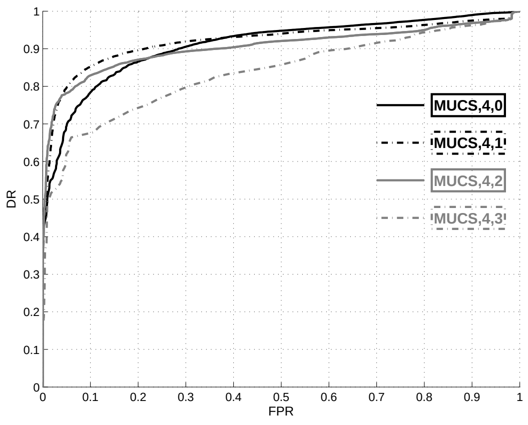


(b) Sequence size: 4

Figure 5. The Performance of MOCS Method at Different Tolerance Size



(a) Sequence size: 2, 3



(b) Sequence size: 4

Figure 6. The Performance of MUCS Method at Different Tolerance Size

and 75. The performance decreases when the number of commands increases. The false positive rate increases for users with more than 75 commands. Especially few users with a high diversity of commands could affect the results. Please note that each command group could contain different types of users (novice, experienced or computer scientists).

To conclude, we show the performance of the MC, MOCS, and MUCS methods in this section. The newly proposed MUCS method outperforms both the one-class and two-class naive Bayes approaches, which are some of the most successful and well-known methods in the literature. The performance of the MUCS method could be increased with the updating mechanism in the future.

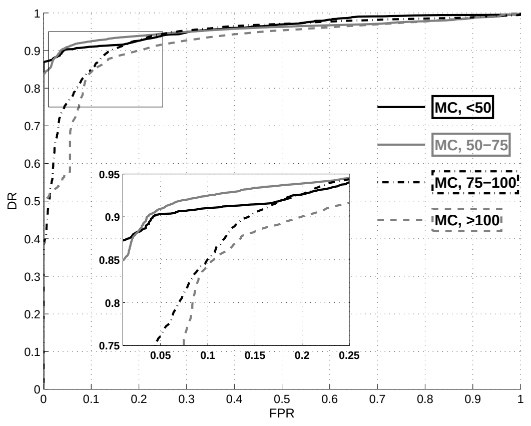


Figure 7. The Performance of MC Methods for Different User Groups

6.5. Effect of User Groups

In this section, we analyze the results for different types of users and masqueraders. Since there is insufficient data for non-programmers in the dataset, we have excluded such users from this analysis. The detection rate of each method for different user groups at false positive rate 3%, is given in Table III. The system is trained with the given user group and the given method in each cell and evaluated for all types of masqueraders. A matrix showing the performance of each method on different types of masqueraders for each trained user group is given in detail in Appendix A - Results Expanded.

For novice users, the method MUCS,4,1 shows the best performance. In general, novice users depend on the tolerance less than other types of users, and non-sequential methods (MUCS) always give slightly better results than its corresponding sequential methods (MOCS) when the tolerance is not taken into account. This type of users have the smallest set of commands and have a tendency to use similar 2-length command sequences in different orders (MUCS,2). Hence this characteristic of novice users might be used to distinguish these users from insider threats. When the matrix is analyzed in Appendix A, novice users detect all kinds of masquerader groups with a high degree of accuracy, except novice masqueraders, in all methods. Since all masqueraders except novice ones, use bigger

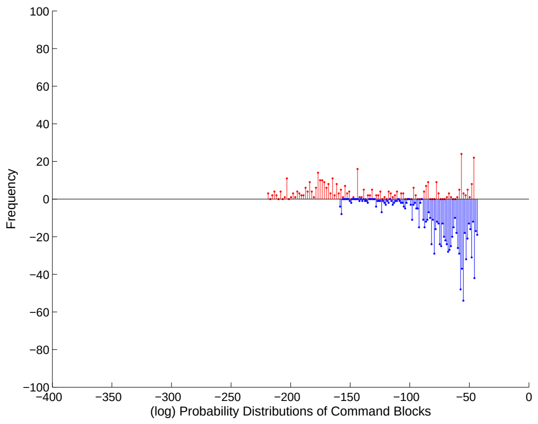
Method/User	Novice Users	Experienced Users	Computer Scientists
MC	76.76%	75.85%	**86.91**%
MOCS,2	77.45%	67.40%	82.90%
MOCS,3	70.42%	50.81%	70.69%
MOCS,4	61.95%	46.37%	55.31%
MUCS,2	78.83%	79.31%	82.81%
MUCS,3	76.57%	57.75%	78.19%
MUCS,4	73.31%	52.02%	68.50%
MOCS,2,1	63.40%	74.57%	72.96%
MOCS,3,1	76.79%	77.09%	84.40%
MOCS,3,2	56.06%	70.40%	62.75%
MOCS,4,1	71.26%	52.34%	76.32%
MOCS,4,2	74.49%	79.12%	78.54%
MOCS,4,3	52.00%	61.46%	56.12%
MUCS,2,1	63.41%	74.83%	69.65%
MUCS,3,1	78.02%	**82.00**%	79.01%
MUCS,3,2	56.24%	68.64%	59.39%
MUCS,4,1	**79.13**%	56.34%	75.86%
MUCS,4,2	72.09%	80.04%	75.47%
MUCS 4,3	51.85%	58.43%	54.66%

Table III. The Comparison of Proposed Methods for Different User Groups

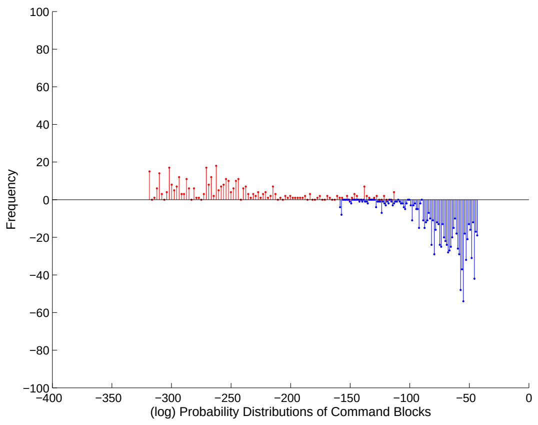
command sets than novice users, they are easily detected when trained with novice users.

Probability distributions of command blocks of a novice user by employing MC method are given in Figure 8a, which is shown below the x-axis, with distributions above the x-axis belonging to the probability distributions of command blocks of novice masqueraders. Since they use similar commands from a small set of commands, the distributions are quite similar as well. On the other hand, the same user is compared with the experienced masqueraders in Figure 8b. It is clearly observed that employing commands from a different and larger set of commands decreases the probabilities, hence detection is achieved with a high rate of accuracy.

For experienced users, the best performance is obtained by the method MUCS,3,1. When the tolerance is smaller than 50% of the sequence size, there is a considerable difference between sequential and non-sequential methods. It could be concluded that these users employ unordered command sequences with small variations. The methods applied for experienced users need more tolerance than the methods applied for novice users. Although both experienced users and computer scientists have a greater diversity of commands on average, computer scientists are easier to distinguish from masqueraders. As seen in the matrix, the systems trained with experienced users cannot detect experienced masqueraders as well as expected, however they do show a slightly better detection



(a) A Novice User vs. Novice Masqueraders



(b) A Novice User vs. Experienced Masqueraders

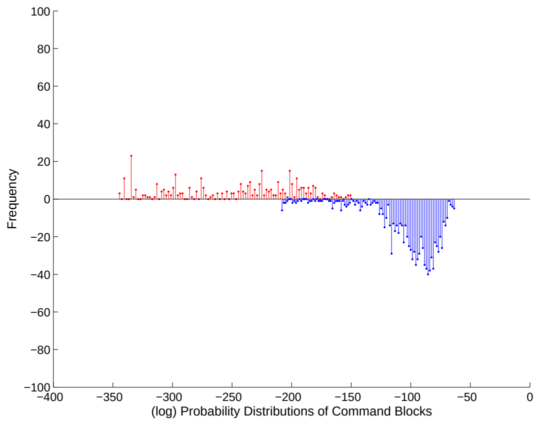
Figure 8. Probability Distributions of Command Blocks

rate than for the novice users (train)-novice masquerader (test) relation.

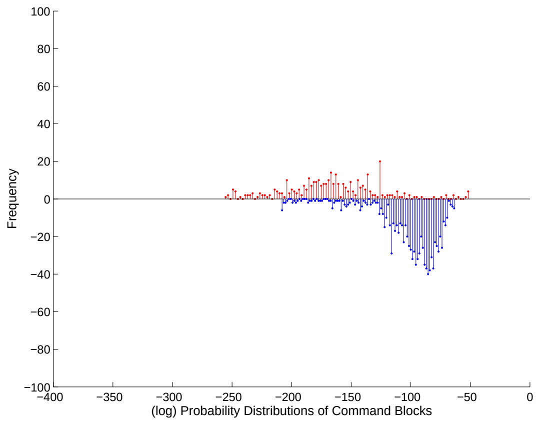
The performance of the best method for each user groups on different masquerader groups is shown in Table IV. It is observed that the novice user group shows very low accuracy on masqueraders with the same type. This is an expected result since they show similar behaviors. An interesting observation is that computer scientist masqueraders can be easily distinguished even from computer scientist users. Computer scientist users have the greatest diversity of commands on average. The usage of specialized commands (such as commands for compiling programs) can distinguish computer scientist users from all other types of masqueraders with a high degree of accuracy.

MC method performs reasonably well for each users, especially for computer scientists. Specific commands used by each computer scientist makes them easier to distinguish from others. When the detection rates are analyzed from the point of masqueraders as in Table IV, computer scientist masqueraders are the easiest to detect among all types of masqueraders. On the other hand, experienced masqueraders are harder to detect. Even computer scientists and experienced users employ various commands, specific commands used by each computer scientists make them easier to distinguish from others.

Probability distributions of commands of a computer scientist user by employing MC method are given in Figure 9a. The distributions above the x-axis belong to the probability distributions of commands of computer scientist masqueraders. As it is seen, one computer scientist can be easily differentiated from other computer scientists. The same user is compared with the experienced masqueraders in Figure 9b. Even though experienced users have a large set of commands, they usually employ common commands. On the other hand, specific command blocks that each computer scientist masquerader types decrease the probability of a block typed by the same computer scientist (below the x-axis), then the multiplication of probabilities stays below the threshold and is marked as a masquerader.



(a) A Computer Sci. User vs. Computer Sci. Masqueraders



(b) A Computer Sci. User vs. Experienced Masqueraders

Figure 9. Probability Distributions of Command Blocks

In real life scenarios, most insider attacks are planned [42]. Some of the primary motivations of these attackers is to steal sensitive information from an organization, or to disrupt critical business operations. In such

Users/Masqueraders	Method	Non Programmers	Novice Users	Experienced Users	Computer Scientists
novice users	MUCS,4,1	96.45%	32.21%	92.02%	96.13%
experienced users	MUCS,3,1	73.96%	98.57%	70.80%	84.45%
computer scientists	MC	81.08%	99.31%	83.07%	84.09%

Table IV. The Performance of Proposed Methods on Different Types of Masqueraders

advanced attack scenarios, attackers could target critical servers which contain sensitive data or perform business operations, and therefore we expect the attacker to be as experienced as the system administrator responsible for critical servers. Hence, we could choose a technique which performs best on differentiating experienced users from experienced masqueraders (MUCS,3,1).

The protection/detection mechanisms are mainly based on the value of the information to the organization. Therefore, we could use multiple protection mechanisms and, correlate logs of these mechanisms for protecting critical resources. In order to protect less valuable resources in an organization, we could consider each group/user within the organization based on their usage behaviors such as the number of commands they use, or the complexity of the commands, in order to make a decision on the most appropriate technique. The method MUCS, 3, 1 usually performs better than other methods as shown in the Figure 1. MC also performs well enough for different user groups, so we could prefer MC over MUCS,3,1 due to its simplicity and efficiency for protecting non-critical resources.

7. CONCLUSION

In this research, sequence-based approaches are investigated for masquerade detection. We propose two sequence-based approaches namely MOCS and MUCS, and compare them with MC, the command-based naive Bayes approach. All these methods employ the bag-of-words approach and use only self-data for training due to privacy issues. The performance of each method and the comparison of these methods with other recent techniques in the literature are demonstrated. In the results, one of the best performances is shown by the MUCS method. The newly proposed method MUCS,3,1 outperforms both the one-class and two-class naive Bayes approaches, which are some of the most successful and well-known methods in the literature. In the future, naive Bayes with updating, as in [7][43] could be applied to overcome concept drift issue and to increase the performance of the MUCS method. Monitoring novice users for a sufficient length of time could also allow us to observe changes in user behaviors over time.

The sequence-based approaches in the literature consider sequences only in order. The effects of the order of commands, the variety of commands and the tolerance on the methods, are also explored in this study. Non-sequential methods usually perform considerably better than the corresponding sequential methods. When the tolerance is taken into account, and at less than half of a command sequence, it improves the results. Furthermore, it is noteworthy that the tolerance parameter is more influential than the order of commands. Command variety is another parameter affecting the results in a negative way. The similarity of user commands is sufficient to differentiate those users with small command sets from masqueraders. For users with a high variety of commands, sequence-based approaches with tolerance might perform better than MC at high detection rates. It could be concluded therefore, that different methods could be more suitable for different types of users.

The effects of different user groups/masqueraders on the results were investigated. Novice users detect all kinds of masquerader groups except novice ones with a high accuracy. Their general behaviors, which use similar unordered command sequences from the smallest set of commands is the best way to model this type of user. Computer scientist masqueraders are easily distinguishable from all types of users, even from computer scientist users. Experienced masqueraders also employ commands from a large set of commands like computer scientists, however they are more difficult to detect than computer scientists due to the common commands they use.

To conclude, we propose sequence-based approaches which consider both the order of commands and small variations in a command sequence. On the other hand, approaches in the literature mainly assume commands in a sequence are ordered at all times. We show the technique based on unordered similar command sequences (MUCS) outperforms the technique based on ordered command sequences (MOCS)/commands (MC) in many cases. Among MUCS techniques, MUCS,3,1 shows the best performance on average for all user groups. We also use the Greenberg dataset which is more appropriate for sequence-based approaches in this research. The approaches were evaluated rigorously by considering different parameters: the order of commands, the variations in command sequences, the variety of commands and the user groups. And finally, it can be stated that we have analyzed user groups from the masquerade detection point of view and thoroughly analyzed sequence-based approaches in this research.

REFERENCES

1. Cummings A, Lewellen T, McIntire D, Moore P, Trzeciak R. Insider threat study: Illicit cyber activity involving fraud in the u.s. financial services sector. *Cert report*, Carnegie Mellon University 2011.
2. Cybersecurity watch survey 2011. URL `http://www.cert.org/insider_threat/`.

3. Cloud security alliance : Top threats to cloud computing v1.0 2010. URL https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf.
4. Salem M, Hershkop S, Stolfo S. A survey of insider attack detection research. *In Advances in Information Security*, vol. 39, Springer, 2008; 69–90.
5. Bertaccihini M, Fierens P. A survey on masquerader detection approaches. *In Proc. of Congreso Iberoamericano de Seguridad Informatica, Universidad de la Republica de Uruguay*, 2008; 46–60.
6. Schonlau dataset 2001. URL http://www.schonlau.net.
7. Maxion R, Townsend T. Masquerade detection augmented with error analysis. *IEEE Transactions on Reliability* 2004; **53**:124–147.
8. Greenberg S. Using unix: Collected traces of 168 users. *Research Report 88/333/45*, Department of Computer Science, University of Calgary 1988.
9. Maxion R, Townsend T. Masquerade detection using truncated command lines. *In Proc. of the International Conference on Dependable Systems & Networks*, 2002; 219–228.
10. Maxion R. Masquerade detection using enriched command lines. *In Proc. of the International Conference on Dependable Systems & Networks*, 2003; 5–14.
11. Schonlau M, DuMoucel W, Ju H, Karr A, Theus M, Vardi Y. Computer intrusion: Detecting masqueraders. *Statistical Science* 2001; **16**(1):58–74.
12. Bertaccihini M, Fierens P. Preliminary results on masquerader information-theoretic detection using compression-based similarity metrics. *Electronic Journal of SADIO* 2007; **7**(1).
13. Evans S, Eliand E, Markham S, Impson J, Laczo A. Mdlcompress for intrusion detection: Signature inference and masquerade attack. *In Proc. of Military Communications Conference*, 2007.
14. Yung K. Using feedback to improve masquerade detection. *In Proc. of the International Conference on Applied Cryptography and Network Security, LNCS*, vol. 2846, Springer, 2003; 48–62.
15. Dash S, Reddy K, Pujari A. Adaptive naive bayes method for masquerade detection. *Security and Communication Networks* 2011; **4**:410–417.
16. Wang K, Stolfo S. One-class training for masquerade detection. *In Proc. of the 3rd IEEE Workshop on Data Mining for Computer Security*, 2003; 398–400.
17. Salem M, Stolfo S. Detecting masqueraders: A comparison of one-class bag-of-words user behavior modeling techniques. *In Proc. of the 2nd International Workshop on Managing Insider Security Threats*, 2010; 3–13.
18. Chen L, Dong G. Masquerader detection using oclep: One-class classification using length statistics of emerging patterns. *In Proc. of the Seventh*

International Conference on Web-Age Information Management Workshops, 2006.

19. Wang Q, Si L. A robust one-class bayesian approach for masquerade detection. *In Proc. of Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011; 111–112.
20. Lane T, Brodley C. Sequence matching and learning in anomaly detection for computer security. *In Proc. of AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997; 43–49.
21. Latendresse M. Masquerade detection via customized grammars. *In Proc. of the 2nd International Conference on Detection of Intrusion and Malware, and Vulnerability Assessment, LNCS*, vol. 3548, Springer, 2005; 141–159.
22. Nevill-Manning C, Witten I. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research* 1997; **7**:67–82.
23. Zhou J, Shirai H, Kuroiwa J, Odaka T, Ogura H. Analysis of command frequency and command sequence grammar in ids. *In Proc. of IEEE Conference on Soft Computing in Industrial Applications*, Springer, 2008; 113–118.
24. Oka M, Oyama Y, Abe H, Kato K. Anomaly detection using layered networks based on eigen co-occurrence matrix. *In Proc. of the 7th International Symposium on Recent Advances in Intrusion Detection, LNCS*, vol. 3224, Springer, 2004; 223–237.
25. Geng D, Odaka T, Kuroiwa J, Ogura H. An n-gram and stf-idf model for masquerade detection in a unix environment. *Journal in Computer Virology* 2011; **7**:133–142.
26. Seo J, Cha S. Masquerade detection based on svm and sequence-based user commands profile. *In Proc. of the 2nd ACM symposium on Information, Computer and Communications Security*, 2007; 398–400.
27. Coull S, Szymanski B. Sequence alignment for masquerade detection. *Computational Statistics and Data Analysis* 2008; **52**(8):4116–4131.
28. Sodiya AS, Folorunso O, Onashoga SA, Ogunderu OP. An improved semi-global alignment algorithm for masquerade detection. *International Journal of Network Security* 2011; **13**(1):31–40.
29. Huang L, Stamp M. Masquerade detection using profile hidden markov models. *Computers & Security* 2011; **30**:732–747.
30. Xiaoa X, Tianc X, Zhaib Q, Xiaa S. A variable-length model for masquerade detection. *The Journal of Systems and Software* 2012; **85**:2470–2478.
31. Tapiador J, Clark J. Masquerade mimicry attack detection: A randomised approach. *Computers & Security* 2011; **30**:297–310.
32. Tapiador JE, Hernandez-Castro JC, Peris-Lopez P. Online randomization strategies to obfuscate user behavioural patterns. *Journal of Network and Systems Management* 2012; **20**:561–578.

33. Kothari A. Defeating masquerade detection. Msc thesis, Department of Computer Science, San Jose State University 2012.

34. Razo-Zapata I, Mex-Perera C, Monroy R. Masquerade attacks based on user's profile. *The Journal of Systems and Software* 2012; **85**:2640–2651.

35. Garg A, Rahalkar R, Upadhyaya S, Kwiat K. Profiling users in gui based systems for masquerade detection. *In Proc. of IEEE Information Assurance Workshop*, Springer, 2006; 48–54.

36. Nguyen N, Reiher P, Kuenning G. Detecting insider threats by monitoring system call activity. *In Proc. of IEEE Workshop on Information Assurance*, 2003; 45–52.

37. Salem M, Stolfo S. Modeling user search behavior for masquerade detection. *In Proc. of the 14th International Conference on Recent Advances in Intrusion Detection, LNCS*, vol. 6961, Springer, 2011; 181–200.

38. Saljooghinejad H, Bhukya WN. Layered security architecture for masquerade attack detection. *In Proc. of Data and Applications Security and Privacy XXVI, LNCS*, vol. 7371, Springer, 2012; 255–262.

39. McCallum A, Nigam K. A comparison of event models for naive bayes text classification. *In Proc. of AAAI Workshop on Learning for Text Categorization*, 1998.

40. Killourhy K, Maxion R. Investigating a possible flaw in a masquerade detection system. *Technical Report 869*, Newcastle University 2004.

41. Fawcett T. An introduction to roc analysis. *Pattern Recognition Letters* 2006; **27**:861–874.

42. Mills RF, Grimaila MR, Peterson GL, Butts JW. A scenario-based approach to mitigating the insider threat. *ISAA Journal* 2011; **9**(5):12–19.

43. Sen S. Using instance-weighted naive bayes for adapting concept drift in masquerade detection. *International Journal of Information Security* 2014; .

A. RESULTS EXPANDED

Method	Users	Non-Programmer Masqueraders	Novice Masqueraders	Experienced Masqueraders	Computer Scientist Masqueraders
MC	novice users	95.59%	25.24%	91.09%	94.99%
	experienced users	65.82%	97.70%	65.15%	75.15%
	computer scientists	81.08%	99.31%	83.07%	84.07%
MOCS,2	novice users	95.14%	29.54%	90.61%	94.65%
	experienced users	57.91%	91.24%	52.39%	68.35%
	computer scientists	78.96%	96.82%	75.04%	80.46%
MOCS,3	novice users	88.17%	24.56%	81.12%	88.16%
	experienced users	43.22%	77.75%	33.88%	48.76%
	computer scientists	68.26%	87.66%	58.32%	68.44%
MOCS,4	novice users	77.37%	23.95%	69.81%	76.65%
	experienced users	39.91%	72.31%	30.11%	40.10%
	computer scientists	53.98%	74.94%	41.83%	49.90%
MUCS,2	novice users	96.42%	31.29%	91.91%	95.62%
	experienced users	71.38%	96.09%	67.26%	82.40%
	computer scientists	78.84%	96.89%	75.26%	80.15%
MUCS,3	novice users	94.04%	29.15%	89.10%	94.10%
	experienced users	50.69%	82.82%	40.39%	57.21%
	computer scientists	75.44%	92.08%	68.70%	76.31%
MUCS,4	novice users	91.17%	30.36%	82.94%	89.80%
	experienced users	47.64%	76.48%	34.55%	49.17%
	computer scientists	67.28%	85.29%	55.10%	66.26%
MOCS,2,1	novice users	83.37%	8.37%	79.23%	82.69%
	experienced users	62.59%	98.78%	64.30%	72.19%
	computer scientists	59.54%	96.89%	69.24%	66.47%
MOCS,3,1	novice users	95.09%	25.65%	90.87%	95.52%
	experienced users	66.53%	96.94%	65.11%	79.67%
	computer scientists	78.40%	97.90%	78.49%	82.80%
MOCS,4,1	novice users	88.32%	23.07%	83.54%	89.81%
	experienced users	41.26%	81.18%	35.50%	51.34%
	computer scientists	72.01%	93.98%	66.27%	73.18%
MUCS,2,1	novice users	83.56%	8.51%	79.24%	83.09%
	experienced users	62.67%	98.79%	64.68%	72.92%
	computer scientists	56.67%	91.97%	65.99%	63.48%
MUCS,3,1	novice users	95.91%	28.32%	92.07%	96.43%
	experienced users	73.96%	98.57%	70.80%	84.45%
	computer scientists	71.54%	93.48%	73.92%	77.10%
MUCS,4,1	novice users	96.45%	32.21%	92.02%	96.13%
	experienced users	45.12%	86.68%	40.20%	53.35%
	computer scientists	70.86%	91.68%	68.39%	72.52%

Table V. Users/Masqueraders Matrix