Detecting Masqueraders: A Comparison of One-Class Bag-of-Words User Behavior Modeling Techniques*

Malek Ben Salem and Salvatore J. Stolfo
Columbia University
New York, New York, U.S.A.
{*malek, sal*}*@cs.columbia.edu*

Abstract

A masquerade attack is a consequence of identity theft. In such attacks, the impostor impersonates a legitimate insider while performing illegitimate activities. These attacks are very hard to detect and can cause considerable damage to an organization. Prior work has focused on user command modeling to identify abnormal behavior indicative of impersonation. In this paper, we investigate the performance of two one-class user behavior profiling techniques: one-class Support Vector Machines (ocSVMs) and a Hellinger distance-based user behavior profiling technique. Both techniques model *bags of words* or commands and do not model *sequences* of commands . We use both techniques for masquerade detection and compare the experimental results. The objective is to evaluate which modeling technique is most suitable for use in an operational monitoring system, hence our focus is on accuracy and operational performance characteristics. We show that one-class SVMs are most practical for deployment in sensors developed for masquerade detection in the general case. We also show that for specific users whose profile fits the average user profile, one-class SVMs may not be the best modeling approach. Such users pose a more serious threat since they may be easier to mimic.

1 Introduction

According to the 2007 e-crime watch survey [1] conducted by the Computer Emergency Response Team (CERT), 31% of insiders who commited electronic crimes used password crackers or sniffers, 39% compromised an account, while only 35% used their own account to commit the electronic crime. These numbers are validated by Richardson in the 2008 CSI Computer Crime & Security Survey [13] and by Randazzo et al. in their insider threat study of illicit cyber activity in the banking and finance sector [12].

Such attacks, known as *masquerade attacks* represent a class of insider attacks that can occur in several different ways. In general terms, a masquerader may get access to a legitimate user's account either by stealing a victim's credentials through password sniffing and cracking tools, or through a break-in and installation of a rootkit or keylogger. In either case, the user's identity is illegitimately acquired. Another case is obtaining the credentials through a social engineering attack, or by taking advantage of a user's misplaced trust when he or she leaves his or her terminal open and logged in allowing any nearby co-worker to pose as a masquerader. Masquerade attacks are extremely serious, especially in the case of an insider who can cause considerable damage to an organization.

User behavior profiling is the common approach used to counter masquerade attacks. Previous work on this class of attacks has focused on auditing and modeling sequences of user commands including

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 1, number: 1, pp. 3-13

*This material is based on work supported by the US Department of Commerce, National Institute of Standards and Technology under Grant Award Number 60NANB1D0127, the US Department of Homeland Security under grant award number 2006-CS-001-000001-02 and the Army Research Office under grant ARO DA W911NF-06-10151. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the US Department of Commerce, National Institute of Standards and Technology, the U.S. Department of Homeland, or the Army Research Office.

work on enriching command sequences with information about arguments of commands [17, 9]. For modeling, researchers applied statistical and machine learning algorithms to identify abnormal behaviors that may lead to the identification of impostors. Most of this work relied upon two-class modeling approaches, where the private data from multiple users are mixed for training classifiers. We claim that one-class modeling anomaly detection-based is more appropriate for operational systems, since user's private data need not be exposed. In this paper, we focus on comparing two one-class modeling techniques for masquerade detection: one-class Support Vector Machines and and a Hellinger distance-based one-class modeling technique paying attention to practical and operational characteristics of each.

1.1 Motivation

One-class modeling provides several practical advantages over two-class modeling when profiling user behavior. One-class modeling requires only user data from the user whose behavior is being profiled, whereas two-class modeling needs data from all other users. Such data from other users may not always be available, or more often such data is available may not be shared with other users for privacy-related reasons. As user models are trained using "'self"'-data only, *i. e.* only data pertaining to the user to be profiled, less training data is required. This translates into faster training and testing. The use of "'self"' data only for modeling allows for independent training of the models and for decentralized management of the sensors monitoring changes in user behavior based on those models. Such sensors do not have be constantly updated with data from other users who may leave or join the organization over time. Sensors can run autonomously. There is no need to define *a priori* how a *masquerader* behaves. The focus is rather on detecting abnormal user behavior indicating the work of an *impostor*.

Sequence-based modeling of user command data may not be very scalable considering the potentially unbounded number of possible commands or applications to be modeled especially in Windows-based environments. Scalability becomes a challenge particularly in multiuser environments or computationally limited devices.

For all of these reasons, we believe that one-class bag-of-words-based modeling is more appropriate for operational masquerade detection monitoring systems. In this paper, we focus on comparing the performance of the one-class bag-of-words modeling techniques for masquerade detection. We introduce a one-class modeling approach based on the Hellinger Distance metric to compute a similarity measure between the most recently issued commands that a user types with a model of the user's command profile. We compare this approach with one-class support vector machines proposed by Wang and Stolfo [20].

1.2 Paper Outline

In section 2 of this paper, we briefly present the results of prior research work on masquerade detection. Section 3 presents the modeling techniques used in this work. In Section 4 we evaluate the results of the experiments conducted for both classifiers. The computation cost of both modeling techniques is analyzed in Section 5. Section 6 concludes the paper by summarizing our key findings.

2 Related Work

Several two-class modeling approaches which profile users by the commands they issue have been reported in the literature. A thorough review of these approaches can be found in a prior survey paper [14]. Most of these techniques were tested against the Schonlau data set [16]. Schonlau et al. collected a data set of "truncated" UNIX commands for 70 users collected over a several month period. Each user had 15,000 commands collected over a period of time ranging between a few days and several months. 50

users were randomly chosen to serve as intrusion targets. The other 20 users were used as masqueraders. The first 5000 commands for each of the 50 users were left intact or ”clean”, the next 10,000 commands were randomly injected with 100-command blocks issued by the 20 masquerade users. The commands have been inserted at the beginning of a block, so that if a block is contaminated, all of its 100 commands are inserted from another user's list of executed commands. Furthermore, the data lacks any timing information, and hence any proposed temporal modeling technique cannot be applied to this data. The objective was to accurately detect the ”dirty” blocks and classify them as masquerader blocks. It is important to note that this dataset does not constitute ground truth masquerade data, but rather simulates impersonation.

In this section, we focus on prior work that proposed one-class modeling techniques tested against the Schonlau data set. Wang and Stolfo compared a naïve Bayes Bayes classifier with Support Vector Machines to detect masqueraders [20]. Their experiments confirmed that for masquerade detection, one class training is as effective as two-class training.

Schonlau et al. [17] applied a compression method to the data set based on the premise that test data appended to historical training data compress more readily when the test data stems from the same user rather than from a masquerader. The method was tested using the UNIX tool *compress* which implements a modified Lempel-Ziv algorithm. This approach only achieved 34.2% true positive rate, while the false positive rate reached 5%.

Schonlau et al. also applied IPAM (Incremental Probabilistic Action Modeling) to the same dataset. IPAM, which was originally presented by Davidson & Hirsch to build an adaptive command line interface [7, 6], is based on one-step command transition probabilities estimated from the training data. When a new command arrives, all transition probabilities are updated according to an exponential updating scheme. During testing, IPAM identifies the top four likely commands to be issued by the user. A prediction is labeled ”good”, if the new user command falls within the four predicted commands. The number of good predictions within the 100-command block is compared to a threshold. If it is below the threshold, an alarm is raised.

Lane & Brodley [8] proposed a sequence-match approach, where upon the arrival of each new command, a similarity measure between the most 10 recent commands and a user's profile is computed. A user's profile consists of all 10-command sequences that the user has issued in the past. For the Schonlau data set, the initial user profiles are therefore made up of 4991 command sequences. The number of matches in a command-by-command comparison of two command sequences constitutes the similarity measure. The matches are weighted and adjacent matches are assigned a higher weight. With each new command, the test sequence of the last 10 commands is therefore scored against all 4991 command sequences available in the user's profile. The maximum of all scores computed is then assigned to the test command sequence. As these scores are noisy, and since the masquerade data in the Schonlau data come in blocks of 100 commands, the last 100 scores are averaged out. The average score is compared to a threshold that varies by user model. If the score is lower than the threshold, the 100-command block is classified as a masquerade block.

A bioinformatics inspired technique was proposed by Coull et al. [4]. The method, known as semi-global alignment, is a modification of the Smith-Waterman local alignment algorithm. The authors enhanced the method and presented a sequence alignment method using a binary scoring and a signature updating scheme to cope with changes in user behavior [5]. The computational complexity of their algorithm is $O(m*n)$ where where m is the length of the sequence of audit data gathered from the normal user and n is the length of the test sequence.

Oka et al. [11, 10] attempted to capture the dynamic behavior of a user that appears in a command sequence by correlating not only connected events, but also events that are not adjacent to each other while appearing within a certain distance (non-connected events). They have developed the layered networks approach based on the Eigen Co-occurrence Matrix (ECM). While this method provided the

Table 1: Summary of accuracy performance of Anomaly Detectors Using the Schonlau Data Set

Method	True Pos. (%)	False Pos.(%)
Compression [17]	34.2	5.0
IPAM [7, 6, 17]	41.1	2.7
Sequence Match [8, 17]	26.8	3.7
Semi-Global Alignment [4]	75.8	7.7
Sequence Alignment (Updating) [5]	68.6	1.9
Eigen Co-occurrence Matrix [10]	72.3	2.5

best results accuracy results compared to all other one-class modeling techniques, it is computationally intensive. It takes 22.15 seconds to classify a single command sequence as normal or anomalous.

The disadvantage of the last three methods lies in their lack of scalability due to their high computational cost. The size of the user profile grows linearly in the former two and exponentially in the latter with each command used by the user for the first time. Model training and update time as well as command sequence test time grows accordingly. Furthermore, such approaches would be even harder to apply in an operational environment, particularly when using other operating systems such as Windows, where the number of unique applications, processes, and user actions is orders of magnitude higher than the number of commands in Unix.

The results achieved by these specific algorithms for the Schonlau datasets appear in Table1 (with True Positive rates displayed rather than True Negatives). Performance is shown to range from 1.9% - 7.7% False Positive rates, with a False Negative rate ranging from 27.7% to 73.2% (alternatively, True Positive rates from 26.8% to 72.3%). Clearly, these results are far from ideal. The problem of effective and practical masquerade detection remains quite challenging.

3 Modeling Techniques

When dealing with the masquerader attack detection problem, we assume that the attacker has already obtained credentials to access a system. When presenting the stolen credentials, the attacker is then a legitimate user with the same access rights as the victim user. Therefore, monitoring an insider's actions after being granted access is required in order to detect such attacks.

In this paper, we focus on comparing two one-class techniques of normal user command data which are not sequence-based, i.e. they do not model the order in which commands are issued or their conditional probabilities. The two techniques model the frequency of commands and or the co-occurrence of certain commands, i.e. they can be referred to as bag-of-word modeling techniques. We introduce a Hellinger distance-based technique, which models frequencies of user commands and changes of those frequencies over time. The technique is very simple and computationally efficient. The second technique is the one-class SVM tested by Wang and Stolfo [20]. For evaluation, we use the Schonlau dataset described in Section 2.

3.1 Hellinger Distance-Based Modeling Approach

The Hellinger distance computes the change in two frequency tables. Each table is a histogram representing the frequency of some variable at some particular moment in time. In our work, we measure the frequency of commands and the changes in their frequency. The Hellinger distance is defined as: $HD(f_p[], f_t[]) = \sum_{i=0}^{n-1} (\sqrt{f_p[i]} - \sqrt{f_t[i]})^2$ where $f_p[]$ is the array of normalized frequencies for the first set,

$f_t[]$ the one for the second set, and n the number of possible commands. This distance metric is applied whenever a user issues a command. A previous frequency table that modeled the previous commands is compared to a newly updated frequency table by modifying the frequency of the commands. Hence, each command creates a new Hellinger distance score that is subjected to threshold logic. A fixed size histogram has to be maintained during run-time in an operational setting.

3.1.1 Determining window size

In order to determine the commands window size w, we use a metric called the Mahalanobis distance. The Mahalanobis distance is a useful way of measuring the similarity between an unknown sample set and a target distribution. It is defined as $MD(x) = \sqrt{(x-m_x)'C_x^{-1}(x-m_x)}$ where x is the feature vector whose elements are the frequencies of commands within the command window. The vector m_x is the mean vector of frequencies of commands for all the training set. C_x is the covariance matrix for x. It is different from the Euclidean distance in that it takes into account the correlations of the data set. The Mahalanobis distances are computed for consecutive windows of commands using various window sizes.

The objective is to find the window size, such that the standard deviation of distance is not too low, which allows for differentiation between the user profiles, and simultaneously not too high, so that an excessive number of false positives is avoided. The best window size w that helped achieve both of these objectives was a size of 100 commands.

3.1.2 Determining the test threshold

Once the window size w was determined, we needed to determine the threshold used to classify a block as a masquerader block or a normal user block. For that, we have computed the Hellinger distance between consecutive blocks with a window of 50 commands, shifting the window by 1 command only. We have done this for the first 5000 commands, and determined the maximum Hellinger distance value for each user. In order to limit the false positive rate, we scaled the threshold value to 95% of the maximum value for each user. So each user has a distinct threshold. Note that no inter-user data is mixed for this process. This constitutes the ”one-class” modeling approach over user commands.

3.2 One-Class Support Vector Machines

Support Vector Machines (SVMs) are linear classifiers used for classification and regression. They are known as maximal margin classifiers rather than probabilistic classifiers. Schölkopf et. al [15] proposed a way to adapt SVMs to the one-class classification task. The one-class SVM algorithm uses examples from one class only for training. Just like in multi-class classification tasks, it maps input data into a high-dimensional feature space using a kernel function, such as the linear, polynomial, or Radial Basis Function (RBF) kernels. The origin is treated as the only example from other classes. The algorithm then finds the hyper-plane that provides the maximum margin separating the training data from the origin in an iterative manner. The kernel function is defined as: $k(x,y) = (\Phi(x).\Phi(y))$,where $x,y \in X$, X is the training data set, and Φ is the feature mapping to a high-dimensional space $X \rightarrow F$.

We used the LIBSVM package [2] to conduct our SVM experiments. It supports both multi-class classification and one-class classification. The one-class SVM function provided by this tool uses the RBF kernel.

4 Evaluation

After computing one classifier for each user, the detection task includes computing the Hellinger distance in the same manner as above, but for the remaining 10,000 commands. The test of self-recognition is the same as in the Schonlau et al. paradigm, with between 76 and 100 blocks of self data presented to the detector for each user. Since the masquerader commands have been injected into blocks of 100 commands, we calculate one value out of all computed Hellinger distances for one block and we compare it with the classification threshold. Because the Hellinger distance scores are very noisy as can be noticed in Figure 1, we take the average score over the 100-command block as the overall score. If the average score is above the user threshold, the block is classified as a masquerader block. The results are shown in Table 2. We note that we do not use any blocks that have been classified as normal to update the user profile or recalculate the threshold. Significant improvement may be possible if models are updated.

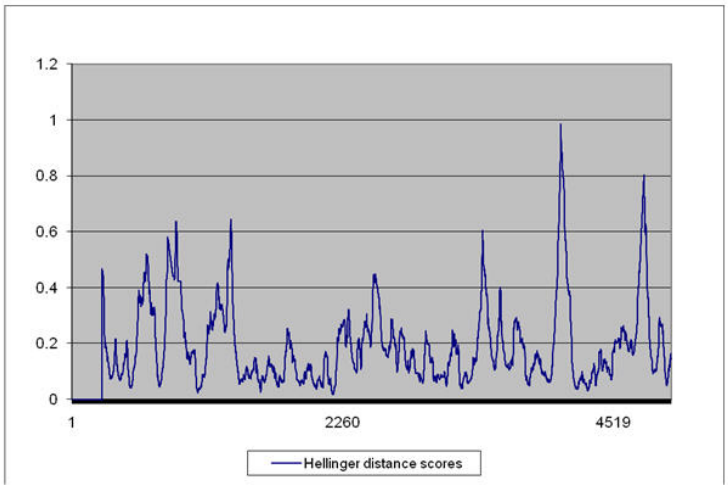

Figure 1: Hellinger distance scores for user 1

For the one-class SVM experiment, we modified the LIBSVM package, so that the one-class prediction models output the probability that a vector belongs to the "self" class, rather than output the classification value "self" or "non-self". We modeled commands using a binary vector indicating whether the command is present in the data set or not. We followed the methodology described in [20] and achieved comparable results. Table 2 clearly shows that one-class SVM classifiers outperform Hellinger distance-based classifiers both in terms of the number of false positives and the number of true positives.

Table 2: Experimental Results

Method	True Pos. (%)	False Pos. (%)
Hellinger distance model	19.5	7.9
ocSVM (binary model)	72.7	6.3

4.1 User by User Comparison

A typical means to visualize the performance of any classification algorithm is the Receiver Operating Characteristic (ROC) curve which plots the sensitivity rate against 1- specificity rate. In order to build the ROC curve, we need to count the number of true positives (TP), true negative (TN), false positives

(FP), and false negatives (FN) for a set of cutoffs according to a classification rule. A true positive is a masquerader command block that has been correctly identified by our algorithm. A false positive is a normal user's command block that was misclassified as a masquerader block. Similarly a true negative is a normal user's block that our algorithm classifies as normal, and a false negative is a masquerader block that our algorithm fails to detect, the latter being perhaps the worst case of failure. Then we can calculate the specificity defined as $\frac{n_{TN}}{n_{TN}+n_{FP}}$ and the sensitivity defined as $\frac{n_{TN}}{n_{TN}+n_{FP}}$ where n_{TN}, n_{FP}, n_{TP}, n_{FN} are the numbers of true negatives, false positives, true positives, and false negatives respectively.

The Area Under Curve (AUC), also known as the ROC score, which is a measure of the area under the ROC curve, reflects the performance of the detection method used. The higher the AUC is, the better the performance of the method. Figure 2 shows a user-by-user AUC comparison for all users whose files have been contaminated. Some users had no masquerader blocks injected, and therefore it was not possible to plot a ROC curve for them. Note one-class SVM clearly dominates in nearly every case.

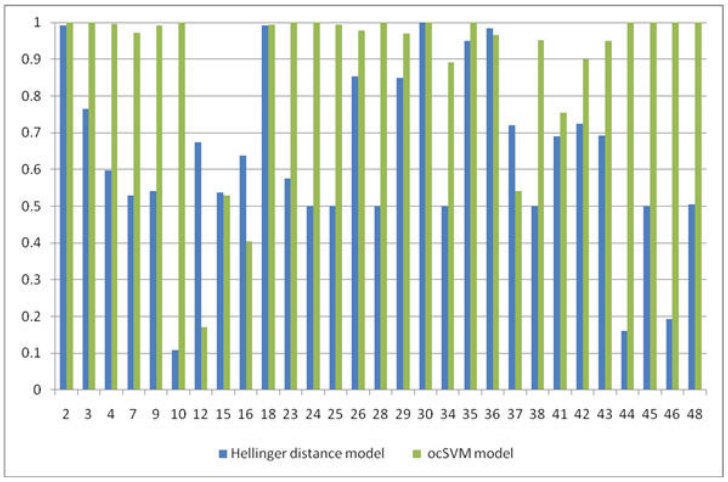

Figure 2: User-by-user comparison of ROC scores (AUCs)

One interesting observation is the performance of both techniques for user 12. For this particular user, the general results achieved were reversed. The Hellinger distance-based modeling technique outperforms one-class SVM. If we compute the histogram of the frequencies of commands used by each user individually in their training data, and compute the same histogram averaged over all users, we find that out of all users tested, user 12 has the closest histogram to the average case. The sparsity of the data for this user and the "'average"' of his or her profile makes it difficult for one-class SVM to build a classifier that can separate this user's data from other users' data. The user essentially emits many distinct commands and hence their command distribution is more uniform than all other users.

In contrast, the Hellinger distance method is able to detect differences in the command frequency distributions between this user and other users who may not fit the average (or uniformly distributed) profile. User 12 may be regarded as a more serious security threat in the case where an SVM model is deployed. A masquerader can more easily mimic this user.

5 Computational Cost Comparison

We compare the computational efficiency of both algorithms by analyzing the time complexities of the feature extraction, model training, command block testing, as well as the model update steps. We discuss the computational complexity of each of these steps for a single user model.

5.1 Feature extraction time complexity

Let o be the total number of commands in the input data. We use this data to compute and output the training vectors $x_i \in R^n, i = 1,...,l$ and testing vectors $x_j \in R^n, j = 1,...,m$ for each user u, where n is the number of features used for modeling. When using one-class SVMs, this step requires reading all training data (5000 out of 15000 commands, i.e. $(1/3) * o$) in order to get the list of unique commands in the dataset. The number of unique commands used determines the number of features n. This step is not required for the Hellinger distance based approach, as the already computed Hellinger scores do not change with the appearance of "'Never-Before-Seen-Commands"' as modeling progresses through the training data. In other words, we do not have to know all the commands that a user could be possibly use in advance.

For SVMs, another pass through the data is required to build the feature vectors by grouping commands in one block, calculating and and outputting n features for that block. This operation has a time complexity of $O(o + n \times (l + m))$ for one-class SVMs. For this dataset, both l and m are dependent on o ($l = (1/3) * (1/100) * o$ and $m = (2/3) * (1/100) * o$. Therefore, for SVMs, the overall time complexity for this step is $O(n \times o)$.

For the Hellinger distance-based method, we slide a window of size w across the user's data, and compute the Hellinger distance for adjacent windows. The number of sliding windows is $O(o)$, the time complexity for computing the Hellinger distance is $O(n)$, and updating the command frequency distribution as we slide the windows through the data requires constant time, *i.e.* $O(1)$. Therefore, the overall time complexity for this step is $O(n \times o)$ as well.

5.2 Training time complexity

Chang and Lin [3] show that the computational complexity of the training step for one user model is $O(n \times l) \times$#Iterations if most columns of Q are cached during the iterations required ; Q is an l by l semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$; $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel; each kernel evaluation is $O(n)$; and the iterations referred to here are the iterations needed by the ocSVM algorithm to determine the optimal supporting vectors.

For the Hellinger-distance based method, all we need to do is select the maximum Hellinger distance determined in the previous step. This step can be combined with the previous one, so that the largest Hellinger distance is maintained when the Hellinger distances are computed.

5.3 Testing time complexity

For SVMs. the computational complexity of the testing step is $O(n \times m)$ as the kernel evaluation for each testing vector y_j is $O(n)$. For the Hellinger distance, this step requires constant time to compare the command block Hellinger score against the model threshold. Hence, the testing phase has an $O(m)$ time complexity.

5.4 Model update time complexity

Updating an SVM model requires re-training the SVM with feature vectors for the newly collected user data. It is straightforward and efficient. Prior data may be expunged and the support vectors computed from that data are retained and used to compute a new update model using the new data [19, 18]. The update time complexity is therefore equivalent to the train time complexity.

Updating the Hellinger distance-based model can be done in constant time. The newly computed Hellinger score is compared with the user model threshold, if it is higher, the model threshold value is updated with this new maximum Hellinger score.

The time complexities for all steps of both methods are summarized in Table 3. We experimentally validate the computational complexity analysis in the next section.

Table 3: Computational complexity comparison between the Hellinger distance-based modeling approach and one-class SVM

Step	**Hellinger distance-based method**	**one-class SVM**
Extracting Features	$O(n \times o)$	$O(n \times o)$
Training	-	$O(n \times l) \times$#Iterations
Testing	$O(m)$	$O(n \times m)$
User Model Update	$O(1)$	$O(n \times l) \times$#Iterations

5.5 Performance Results

We ran our experiments on a regular desktop with a 2.66GHz Intel Xeon Dual Core processor and 24GB of memory in a Windows 7 environment. We measured the average running time of each step of the experiment over three runs. We have not applied a model updating scheme in our experiments. Therefore, no running times are reported for model updates. The results are recorded in Table 4. Recall, that we combined the training step for the Hellinger distance based method with the feature extraction step. The results show that the feature extraction step dominates all other steps. For this step, both methods have the same time complexity. So even though, for the training and testing steps, the computational complexity for SVMs is polynomial in n, the overall runtime for both methods is comparable, making SVMs the better modeling technique considering their far better accuracy results.

Table 4: Computational complexity comparison between the Hellinger distance-based modeling approach and one-class SVM

Step	**Hellinger distance-based method**	**one-class SVM**
Extracting Features	8	7.5
Training	0	2.5
Testing	0.05	2
Total (min)	**8.05**	**12**

6 Conclusion

Masquerade attacks are a serious computer security problem. In this paper, we evaluated the operational characteristics of two one-class user behavior profiling techniques used for masquerade attack detection: one-class Support Vector Machines (ocSVMs) and a Hellinger-distance based user behavior profiling technique. We have compared both accuracy results and computational efficiency of the two *bags of word*-based modeling methods. OUr goal was to identify the method that is most suitable for use in an operational monitoring system. The experimental evaluation shows that one-class SVMs are more effective for masquerade detection. However, we caution that for certain users, who display an "'average"' profile or a relatively more uniform distribution of commands than other users, detection is extremely hard. Such users, which may be easier to mimic, pose a real security threat when an SVM model is deployed.

References

[1] CERT. 2007 e-crimes watch survey, 2007.

[2] C.-C. Chang and C.-J. Lin. Libsvm: http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[3] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. http://www.csie.ntu.edu.tw/ cjlin/-papers/libsvm.pdf, 2001.

[4] S. E. Coull, J. Branch, B. Szymanski, and E. Breimer. Intrusion detection: A bioinformatics approach. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 24–33, 2001.

[5] S. E. Coull and B. K. Szymanski. Sequence alignment for masquerade detection. *Computational Statistics and Data Analysis*, 52(8):4116–4131, 2008.

[6] B. D. Davison and H. Hirsh. Toward an adaptive command line interface. In *Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI97)*. Elsevier Science Publishers, 1997.

[7] B. D. Davison and H. Hirsh. Predicting sequences of user actions. In *Working Notes of the Joint Workshop on Predicting the Future: AI Approaches to Time Series Analysis, 15th National Conference on Artificial Intelligence/15th International Conference on Machine Learning*, pages 5–12. AAAI Press, 1998.

[8] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *In AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49. AAAI Press, 1997.

[9] R. A. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 219–228. IEEE Computer Society, 2002.

[10] M. Oka, Y. Oyama, H. Abe, and K. Kato. Anomaly detection using layered networks based on eigen co-occurrence matrix. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection*, 2004.

[11] M. Oka, Y. Oyama, and K. Kato. Eigen co-occurrence matrix method for masquerade detection. In *Publications of the Japan Society for Software Science and Technology*, 2004.

[12] M. R. Randazzo, M. Keeney, E. Kowalski, D. Cappelli, and A. Moore. Insider threat study: Illicit cyber activity in the banking and finance sector, 2004.

[13] R. Richardson. Csi computer crime and security survey, 2008.

[14] Malek Ben Salem, Shlomo Hershkop, and Salvatore J. Stolfo. A survey of insider attack detection research. In *Insider Attack and Cyber Security: Beyond the Hacker*. Springer, 2008.

[15] B Schölkopf, J. C. Platt, J. Shawe-taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7), 2001.

[16] M. Schonlau. Schonlau dataset: http://www.schonlau.net.

[17] M. Schonlau, W. Dumouchel, W. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16:58–74, 2001.

[18] Nadeem Ahmed Syed, Huan Liu, Syed Huan, Liu Kah, and Kay Sung. Handling concept drifts in incremental learning with support vector machines. In *In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99*, pages 317–321. ACM Press, 1999.

[19] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 1999.

[20] K. Wang and S. J. Stolfo. One-class training for masquerade detection. In *Proceedings of the 3rd IEEE Workshop on Data Mining for Computer Security*, 2003.

Malek Ben Salem is a PhD candidate in Computer Science at Columbia University. Her research interests include developing novel data mining and machining learning techniques applied to intrusion detection, and combining trap-based defenses with anomaly detection for more accurate intrusion detection. Ben Salem holds a M. Sc. in Computer Science from Columbia University and Dipl. Ing. in Electrical Engineering from Universitaet Hannover, Germany. Contact her at *malek@cs.columbia.edu*.

Salvatore J. Stolfo is Professor of Computer Science at Columbia University. He received his Ph.D. from NYU Courant Institute in 1979 and has been on the faculty of Columbia ever since. He has published extensively in the areas of parallel computing, AI knowledge-based systems, data mining and most recently computer security and intrusion detection systems. His research has been supported by DARPA, NSF, ONR, NSA, CIA, IARPA, DHS and numerous companies and state agencies over the years while at Columbia.