One-Class Training for Masquerade Detection

Ke Wang Salvatore J. Stolfo
Computer Science Department, Columbia University
500 West 120th Street, New York, NY, 10027
{kewang, sal}@cs.columbia.edu

Abstract

We extend prior research on masquerade detection using UNIX commands issued by users as the audit source. Previous studies using multi-class training requires gathering data from multiple users to train specific profiles of self and non-self for each user. One-class training uses data representative of only one user. We apply one-class Naïve Bayes using both the multi-variate Bernoulli model and the Multinomial model, and the one-class SVM algorithm. The result shows that one-class training for this task works as well as multi-class training, with the great practical advantages of collecting much less data and more efficient training. One-class SVM using binary features performs best among the one-class training algorithms.

1. Introduction

The Masquerade attack may be one of the most serious security problems. It commonly appears as spoofing, where an intruder impersonates another person and uses that person's identity, for example, by stealing their passwords or forging their email address. Masqueraders can be insiders or outsiders. As an outsider, the masquerader may try to gain superuser access from a remote location and can cause considerable damage or theft. A simpler insider attack can be executed against an unattended machine within a trusted domain. From the system's point of view, all of the operations executed by an insider masquerader may be technically legal and hence not detected by existing access control or authentication schemes. To catch such a masquerader, the only useful evidence is the operations he executes, i.e., his behavior. Thus, we can compare one user's recent behavior against their profile of typical behavior and recognize a security breach if the user's recent behavior departs sufficiently from his profiled behavior, indicating a possible masquerader.

The insider problem in computer security is shifting the attention of the research and commercial community from intrusion detection at the perimeter of network systems. Research and development is going on in the area of modeling user behaviors in order to detect anomalous misbehaviors of importance to security; for example, the behavior of user-issued OS commands as represented in this paper, and in email communications [17]. Considerable work is ongoing in certain communities to detect not only impersonation, but also author identification. For example, Sedelow [16] and Vel [18] are two examples bracketing the length of time this topic has existed in the literature.

The masquerade problem is a challenging problem. If the masquerader can mimic the user's behavior successfully, he won't be detected. In addition, if the user himself is behaving much differently than his trained profile, the detector will misclassify him as masquerader, which may cause annoying false alarms. There have been several attempts to solve this problem using *command line sequences*, [14] and [9]. The best results so far reported are 60-70% accuracy with a false positive rate as low as 1-2%. The profiles were computed using supervised machine learning algorithms that classify training data acquired from multiple user. These approaches considered training user profiles as a multi-class supervised learning task where data gathered on a user is treated as an example of one-class, i.e. a distinct user.

In this paper, we consider a different approach with substantial practical advantage. We examine the task of profiling a user by modeling his data exclusively, without using examples from other users, and achieving good detection performance and minimal false positive rates. We also consider alternative machine learning algorithms that may be employed for this "one-class" training approach.

One-class training means that we *only* use the user's own legitimate examples of commands they issue to build the user's self profile. Previous work uses both positive and negative examples to build both self and non-self profiles, except for Maxion [9], who considers the problem of determining how vulnerable a user's behavior may be to mimicry attack. Here we extend this technique using one-class SVM. This is important in many contexts, especially when the only information available is the history of the user's activities. If a one-class training algorithm can achieve similar performance to that exhibited by a multi-class approach, we may provide a significant benefit in real security applications; much less data is required, and training can proceed independently of any other user. The study reported in this paper indicates that indeed one-class training algorithms perform equally well as two class training approaches.

This self profile idea is similar to the widely used "anomaly detection" techniques in intrusion detection system [eg. 2, 3]. For example, the anomaly detector of IDES [8] uses established normal usage profiles, which is the expected behavior, to identify any large usage deviation as a possible attack. Several methods have been used to model the normal data, for example, decision trees [7], neural network [4], and sparse Markov Transducers [2], and Markov chains [19]. In this paper, we applied one-class Naïve Bayes and one-class SVM algorithms to the masquerade dataset of UNIX system call sequences.

In previous work, we believe there were several methodological flaws in the manner in which data was acquired and used. The "Schonlau dataset" from [14] presents each user's command line data with a varying number of artificially created masquerade command blocks, ranging from 0 to 24, out of a total of 100 command blocks to be classified. The previous work only considered the average performance of a given method when it is applied to all of the 50*100 blocks of commands issued by the 50 users. However, since the masquerade blocks are "randomly" inserted into each user's data by using some other user's command block, each user's data has a different number of masquerade blocks, and the content of these masquerade blocks all differ. This data is not a good baseline to compare the effectiveness of alternative detection methods because one method might be better at detecting certain forms of masquerade attack while others are not. Unfortunately, since the distribution of such masquerade blocks appear many times in the dataset, some algorithms appear to have better performance over others, while, in practice or in other contexts, this finding may not be true. To better compare the alternative methods proposed in this work, we follow the exhaustive "1v49" evaluation methodology from [9], which will be described in detail in the section about the experimental methodology and results. The *ROC score* [5] is used to compare several one-class training methods under different false positive rate restrictions. This approach provides a better view to indicate which algorithm is better for most users.

Even though one machine learning algorithm may perform better than another in terms of detection and false positive rate, our most important finding is that one-class training can indeed perform as well as multi-class training. Even so, the overall performance of both approaches indicates that there is much room for improvement.

The rest of the paper is organized as follows. Section 2 describes the dataset we used in our research and provides a review of recent work, and Section 3 discusses the one-class Naïve Bayes and one-class SVM methods. Section 4 describes the experimental setting and our results, and Section 5 concludes the paper with our analysis and findings.

2. Recent Work

Schonlau *et al.* [14] and Maxion [9] provide a dataset collected from keyboard commands on a UNIX platform. The dataset is publicly available at http://www.schonlau.net. Unfortunately, there are neither timestamps nor other detailed command line arguments available in the Schonlau dataset. Hence, the dataset in question consists simply of sequential "truncated" commands issued by each of 70 users. Each user is represented by 15,000 commands, collected over time from a few days to a few months. They randomly chose 50 users to serve as intrusion target, and the remaining 20 users as masqueraders. The first 5,000 commands of each user are "clean data" (legitimately issued by the user), and the next 10,000 commands of the target users were randomly injected with commands issued by the 20 masquerade users, apparently none of whom are any of the 50 users. The commands are grouped into blocks, with 100 commands per block. The commands in one block are either all clean or all masquerade attacks, called "dirty blocks". The task studied in [14] is to accurately classify the "dirty" blocks as masquerade commands, while classifying the clean blocks as belonging to the user. From this point of view, this problem is very similar to a text classification problem, where the commands may be regarded as words or tokens in a document that is to be classified. Their work identifies six masquerade-detection schemes. They are: Bayes 1-Step Markov, Hybrid Multi-Step Markov, Incremental Probabilistic Action Modeling (IPAM), Uniqueness, Sequence-Match, and Compression. These masquerade detection methods were directly applied to this dataset; the first 5000 commands were used for training, and the remaining 10000 commands were inserted and used for testing, where each were processed in blocks of 100 commands. Testing and training were performed on a per block basis. The paper by Maxion [9] presents a clear and concise review of these six methods. The strategies behind each of these methods are quite different, but the results show that no method completely dominates another using cluster analysis and ROC curves.

Maxion [9] improved upon Schonlau's result by applying the Naïve Bayes classification algorithm using the "bag of words" features. Naïve Bayes has been used in text classification for a long time and proved to be very efficient in this context as well. Naïve Bayes was also used in the earlier work [15] on classifying malicious code attachments in email messages. Maxion presents a detailed analysis of the origins of the classification error, revealing why some users are good masquerades and others are not. That paper designed another experiment, called "1v49", to perform this error analysis. We also use the "1v49" experimental setting in our work, but here it is

used to compare the performance of different classifiers when applied to multiple classes.

The results for these reviewed methods are displayed in Table 1 and serve as a baseline for comparison.

Table 1. Results of previous classification methods

Method	Hits	False Positives
N. Bayes (updating)	61.5%	1.5%
N. Bayes (no Upd.)	66.2%	4.6%
Uniqueness	39.4%	1.4%
Hybrid Markov	49.3%	3.2%
1-step Markov	69.3%	6.7%
IPAM	41.4%	2.7%
Sequence Matching	36.8%	3.7%
Compression	34.2%	5.0%

3. Machine learning methods

3.1. Learning task

For this masquerade detection problem, the learning task is to build a classifier that can accurately detect the masquerade commands while not misclassifying the user's legitimate commands as a masquerade. Using the Schonlau dataset, which is organized as a set of blocks of 100 commands, the learning task is to compute a binary classifier whose input is a block of 100 commands and whose output is a classification of that block as either generated by a masquerader or not. The target classification is to detect the masquerader's command blocks. Hence, the masqueraders' data are positive examples, while the user's legitimate data are treated as negative examples. Thus, a true positive outcome is a masquerade block of 100 commands, while a false positive outcome is a block of commands legitimately issued by the user but misclassified as a masquerade. In the following description, we call the masquerade blocks *positive* examples and call the legitimate blocks, those issued by the user himself, *negative* examples. One-class training means that a classifier is computed using *only negative* examples of the user himself as training data to build the classifier, which will be used to classify both positive and negative data. Thus, the task is to positively identify masqueraders, but not to positively identify a particular user.

3.2. One-class or two class

Previous work considered the problem as a multi-class supervised training exercise. The dataset contains data for 50 users. For each user, a specific class, the first 5000 commands are treated *as negative examples*, while the data from the other 49 users are treated as *positive examples*. It is reasonable to assume the negative examples, which belong to the same user, were treated consistently, while the positive examples used in training belong to another user. For the masquerade problem, it is probably impossible and unreasonable to estimate how an attacker would behave. Thus, treating sets of other users' data as positive examples provides a substantive bias (to those users' behavior who probably was not behaving maliciously). We next present the means of implementing one-class training for Naïve Bayes classifier and for SVM, using only data from a single user when training a classifier to profile a distinct user.

3.3. Naïve Bayes Classifier

The Naïve Bayes classifier [12] is a simple and efficient supervised learning algorithm, which has been proved to be very effective in text classification, and many other applications. It is based on Bayes' rule,

$$p(u \mid d) = \frac{p(u)P(d \mid u)}{p(d)}$$

which calculates the probability of a class given an example. Applied to the masquerade problem, it calculates the likelihood that a command block belongs to a masquerader (non-self), or some legitimate user. Different commands c_i , which are used as features here, are assumed independent from each other. This is the Naïve part of this method.

There are two common models used in Naïve Bayes Classifier, one is the multi-variate Bernoulli model, and the other is the multinomial model [11]. In the multi-variate Bernoulli event model, a vector of binary attributes is used to represent a document (in our case, a block of 100 commands), indicating whether the command occurs or doesn't occur in the document. The multinomial model uses the number of command occurrences to represent a document, which is called "bag-of-words" approach, capturing the word frequency information in documents. According to McCallurn [11]'s result, multi-variate Bernoulli model performs better for small vocabulary size, and the multinomial model usually performs better at larger vocabulary size. Because the vocabulary size (the number of distinct commands) of this masquerade problem is 856, which is a moderate in size, we want to compare both of these models for this problem.

Multi-variate Bernoulli model

Using the multi-variate Bernoulli Model, a command block d is represented as a binary vector $\vec{d} = (b_1(d), b_2(d), ..., b_m(d))$, with $b_i(d)$ set to 1 if the command c_i occurs at least once in this block. Here m is the total number of features, i.e., the number of distinct commands. Given $p(c_i \mid u)$, which is the probability estimated for command c_i for user u in the training data, we can compute $p(d \mid u)$ of the test block d as:

$$p(d \mid u) = \prod_{i=1}^{m} (b_i(d)\, p(c_i \mid u) + (1 - b_i(d))(1 - p(c_i \mid u))) \qquad (1)$$

where $p(c_i \mid u)$ is estimated with a Laplacean prior:

$$p(c_i \mid u) = \frac{1 + N(c_i, u)}{2 + N(u)} \qquad (2)$$

$N(u)$ is the number of training examples for user u, while $N(c_i, u)$ is the number of documents containing the command c_i for user u.

Multinomial model

Using the standard bag-of-words approach, each command block is represented by a feature vector $\vec{d} = (n_1(d), n_2(d), ..., n_m(d))$, where $n_i(d)$ is the number of times command c_i appears in the command block d. Similarly, given $p(c_i \mid u)$, which is the frequency count computed for command c_i for user u in the training data, we can compute $p(d \mid u)$ of the test block d as:

$$p(d \mid u) = \prod_{i=1}^{m} (p(c_i \mid u))^{n_i(d)} \qquad (3)$$

where $p(c_i \mid u)$ is derived from:

$$p(c_i \mid u) = \frac{\sum_{j=1}^{N(u)} n_i(d_j) + \alpha}{\sum_{i=1}^{m} \sum_{j=1}^{N(u)} n_i(d_j) + \alpha * m} \qquad (4)$$

Here α is used for smoothing, which controls the sensitivity to previously unseen commands. (This implies there is a non-zero probability any command may be issued by any user.) We set it to 0.01 following [9].

One-class Naïve Bayes

Adapting the above algorithm to one-class Naïve Bayes, which uses only positive examples for training, is very simple. We only compute $p(c_i \mid u)$ for user u's self profile. For the non-self profile, we can assume each command has equal probability $1/m$, which is essentially random. Thus, given a test d, we can compare $p(d \mid self)$ with $p(d \mid nonself)$. The larger the *ratio* of $p(d \mid self)$ to $p(d \mid nonself)$, the more likely this command block d is from the user u.

Applying the one-class Naïve Bayes algorithm to our specific dataset is also quite simple. Since each test document (a block of commands) has a fixed number of 100 commands, the probability of "non-self" is the same for all tested blocks. Since we do not have to compute the probability of non-self; we may compare the probability of being "self" to a threshold in order to decide whether this block is a masquerade block or not. Furthermore, we can easily adjust the threshold to control the false positive and detection rate.

3.4. One-class support vector machine

Support Vector Machines (SVM) have been shown to be highly effective in text classification as well [6], among other important learning tasks. They are *maximal-margin* classifiers, rather than probabilistic as is Naïve Bayes. In the two-class formulation, the basic idea is to map feature vectors to a high dimensional space and to compute a hyperplane that not only separates the training vectors from different classes, but also maximizes this separation by making the *margin* as large as possible.

Scholkopf *et al.* [13] proposed a method to adapt the SVM algorithm for one-class SVM, which only use examples from one-class, instead of multiple classes, for training. The one-class SVM algorithm first maps input data into a high dimensional feature space via a kernel function and treats the origin as the only example from other classes. It then iteratively finds the maximal margin hyperplane that best separates the training data from the origin.

Considering that our training data set $x_1, x_2, ..., x_\ell \in X$, Φ is the feature mapping $X \to F$ to a high-dimensional space, we can define the kernel function as:

$$k(x, y) = (\Phi(x) \cdot \Phi(y))$$

Using kernel functions, the feature vectors need not be computed explicitly, greatly improving computational efficiency since we can directly compute the kernel values and operate on their images. Some common kernels are linear, polynomial, and radial basis function (rbf) kernels:

Linear Kernel: $k(x, y) = (x \cdot y)$

P-th order polynomial kernel: $k(x, y) = (x \cdot y + 1)^p$

rbf kernel: $k(x, y) = e^{-\|x-y\|^2 / 2\sigma^2}$

Now, solving the one-class SVM problem is equivalent to solving the dual quadratic programming (QP) problem:

$$\min_{\alpha} \frac{1}{2}\sum_{ij}\alpha_i\alpha_j k(x_i,x_j)$$

$$\text{subject to } 0 \le \alpha_i \le \frac{1}{\nu\ell}, \sum_i \alpha_i = 1.$$

where α_i is a Lagrange multiplier, which can be thought of as a weight on example x_i, and ν is a parameter that controls the trade-off between maximizing the number of data points contained by the hyperplane and the distance of the hyperplane from the origin.

After solving for α_i, we can use a decision function to classify data. The decision function is:

$$f(x) = \text{sgn}(\sum_i \alpha_i k(x_i, x) - \rho)$$

where the offset ρ can be recovered by $\rho = \sum_j \alpha_j k(x_j, x_i)$.

In our work, we used the LIBSVM 2.4 [1] available at http://www.csie.ntu.tw/~cjlin/libsvm for our experiments. LIBSVM is an integrated tool for support vector classification and regression that implemented Sholkopf's algorithm for one-class SVM. We used the default rbf kernel and the default values of the parameters for one-class SVM.

Another problem to consider for one-class SVM is how to represent the features. We used both a word count representation and a binary representation, which are equivalent to the multinomial model and multi-variate Bernoulli model of Naïve Bayes algorithm, respectively. The vectors are normalized to length 1.

4. Evaluation

We conducted two sets of experiments. The first experiment repeats the experimental methodology of [14]. We show that the performance of one-class training is almost the same as the performance of multi-class training. This is a significant finding on its own.

The second experiment aims to compare the performance of the two one-class training algorithms when applied to multiple users. Following [9], we will call the first the SEA experiment, which is from the authors' names in [14], Schonlau *et al*. The second experiment is called 1v49, because we trained using only one user's data and tested on all other 49 user's data.

4.1. SEA Experiment

Recall that in this experiment, the first 5,000 commands of a user serve as positive examples, and the first 5,000 commands of all the other 49 users serve as negative examples. The resultant classifier is tested on the rest of the 10,000 commands of the user. These have inserted "dirty" command blocks under a probability distribution from other users besides the 50 users whose commands were trained on. For our one-class algorithm, the test data remains the same, but we only use one user's training data (the first 5,000 commands) as negative examples for training; there are no positive examples used at all to compute the classifier. First we compare the Naïve Bayes classifier using the multi-variate Bernoulli model and the multinomial model. Then we compare the one-class training algorithms against the multi-class training algorithms. All the classification algorithms we tested do no updating during the test of any parameters because we want to compare the pure performance of each of them. Maxion [9] used an empirically pre-fixed threshold to update the classifier during the experiment, which is not an intrinsic characteristic of the classification algorithm.

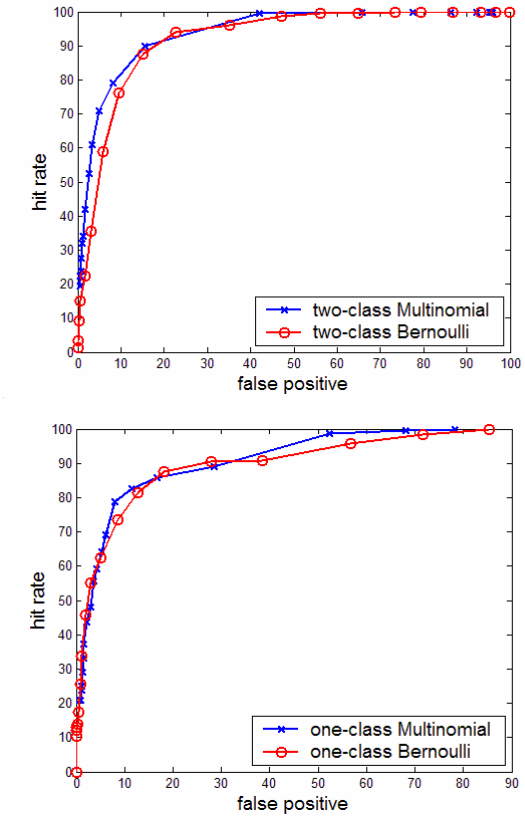

Figure 1. ROC Curves of Naïve Bayes classifiers using multinomial model and multi-variate Bernoulli model, two class and one-class training, respectively.

Figure 1 displays plots comparing the multi-variate Bernoulli model and the multinomial model of Naïve Bayes classifier. When using multi-class training, the multinomial model is obviously better than the Bernoulli model. But the difference is not so obvious in one-class training, especially when the false positive rate is low. We

thus compare both models in the following 1v49 experiment.

To compare the performance of the one-class training algorithms against the multi-class training algorithm on the same test data, we plot the ROC curves as displayed in Figure 1. For the multi-class training algorithm, we only use the multinomial model Naïve Bayes algorithm as the baseline for comparison, which is better than Bernoulli model and has been proved to the best among the variety of methods as described in [9]. For the one-class SVM, we compare both the binary and word count representations. From Figure 2, we can see that only one-class SVM using the word count representation is a little bit worse than the other three methods. One-class SVM using the binary representation and one-class Naïve Bayes achieved almost the same performance as the two class Naïve Bayes algorithm.

We also compare in Figure 3 the performance of all the previous algorithms from Table 1 to one-class SVM algorithm using binary features, which is best one among the one-class training algorithms. One-class SVM-binary is better than most of the previous algorithms except the two-class multinomial Naïve Bayes algorithm with updating.

This experiment confirmed our conjecture that for masquerade detection, one-class training is as effective as two class training.

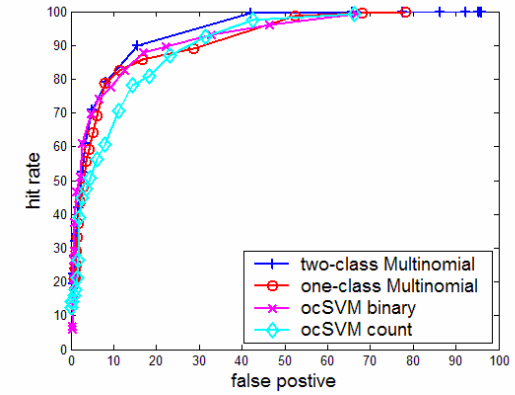

Figure 2. ROC curves of different classification algorithms in SEA experiment.

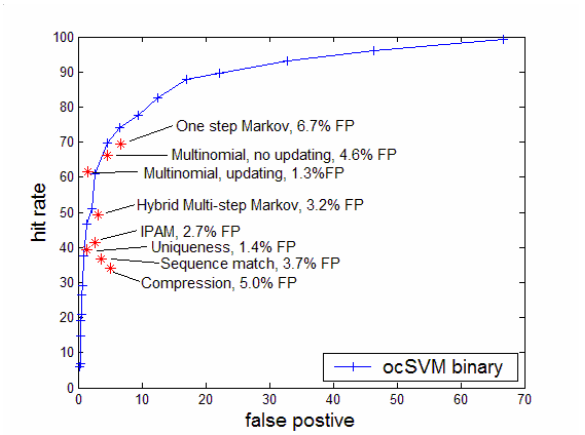

Figure 3. ROC curve for one-class SVM using binary feature. The best-outcome results from other methods are also included for comparison.

4.2. 1v49 Experiment

As we have pointed out, since the dataset used had randomly inserted masquerade blocks in each user's test commands (10,000 commands following the first 5,000), each user has a different number of "dirty" blocks and the origins of these "dirty" blocks also differ. So the result of the SEA experiment may not illustrate the real performance of a classification algorithm. (There are too many unfixed parameters.) To better evaluate the performance of a classification algorithm, we can treat these 50 users as our selected sample of common users. If we can prove algorithm A is better than algorithm B for most of the 50 users, we can infer A is better than B in a general sense.

To meet this requirement, we follow the "1v49" experiment, but for a different purpose. We use one user's first 5,000 commands as *negative* training data to compute a classifier without any positive training data. For test data, we use the non-masquerade blocks from the 10,000 additional commands of the same user as negative test data, and the other 49 users' first 5,000 commands as positive test data. This data is also organized in blocks of 100 commands.

As we mentioned before, the same algorithm might perform quite differently for different users. Figure 4 illustrates the difference. Figure 4 shows the ROC curve for user 2, 20 and 40 using one-class SVM with the binary feature representation. Such a difference occurs no matter which algorithm has been used; the difference is determined by the characteristic of each user.

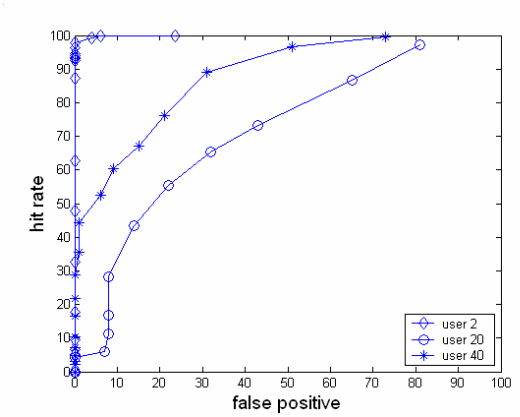

Figure 4: ROC curves for different users using one-class SVM-binary algorithm.

To compare the different methods for multiple users, we compute the *ROC score* for each user. In general, a ROC score is the fraction of the area under the ROC curve, the larger the better. A ROC score of 1 means perfect detection without any false positives. Figure 5 below shows the ROC scores for users 20 and 40 using the one-class SVM-binary algorithm.

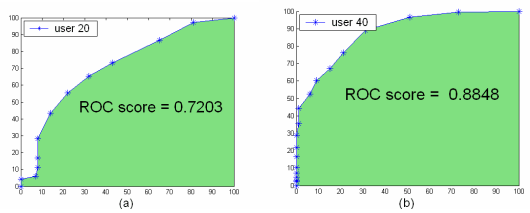

Figure 5: Example for ROC score using user 20 and user 40: the fraction of shaded area under ROC curve.

Figure 6 illustrates the performance of several one-class training algorithms as measured by ROC scores. The figure includes results for all 50 users. From Figure 6, we can see that one-class SVM using word-count features is the worst among the four algorithms. At the high ROC score region, with a ROC score higher than 0.8 (which is what we prefer) one-class SVM using binary features performs best among all. There is no big difference between Naïve Byaes using the multinomial model or the multi-variate Bernoulli model.

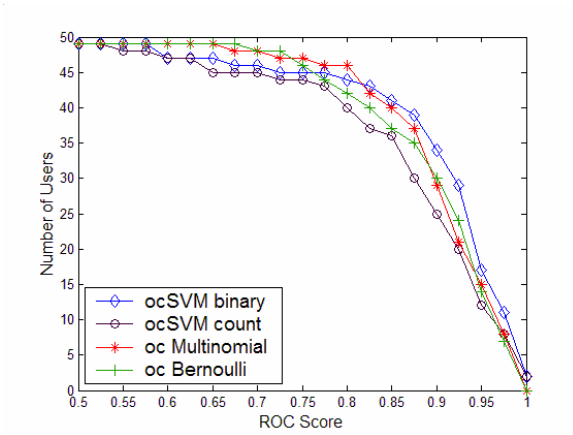

Figure 6: Comparison of four masquerade detection methods. The graph plots the total number of users for which a given method exceeds a ROC score threshold.

For the masquerade problem, we are more interested in the region of the ROC curve with a low false positive rate; otherwise, the "annoyance level" of false alarms would render the detector useless in practical use. Therefore, we restrict the ROC scores to the curves with false positive lower than P, which is called the ROC-P score. For example, if we want to restrict the false positives to be lower than 5% of all command blocks, we can compute ROC-5. Similar to the general ROC score, the ROC-P score is the fraction of the area under the ROC curve where the false positive rate is lower than P%. Figure 7, displays an example of ROC-10, based on the ROC-curves of users 20 and 40. Only part of the ROC curve is drawn here to highlight the plots.

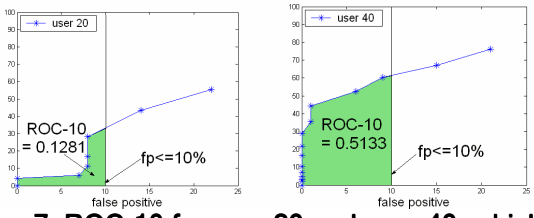

Figure 7: ROC-10 for user 20 and user 40, which is the fraction of the shaded area left to fp<=10% threshold line.

Since we can see that one-class SVM using the binary feature is generally better than one-class SVM using the word count feature, as depicted in Figure 6; here we only compare the one-class SVM using the binary representation with the multinomial model Naïve Bayes and Bernoulli model Naïve Bayes in the following ROC-P comparison. Figures 8 plots the comparison for ROC-5 and ROC-1, which means false positives are below 5% and 1%, respectively. From these two plots, we can

determine that one-class SVM using the binary feature is almost always better than the other two one-class Naïve Bayes methods.

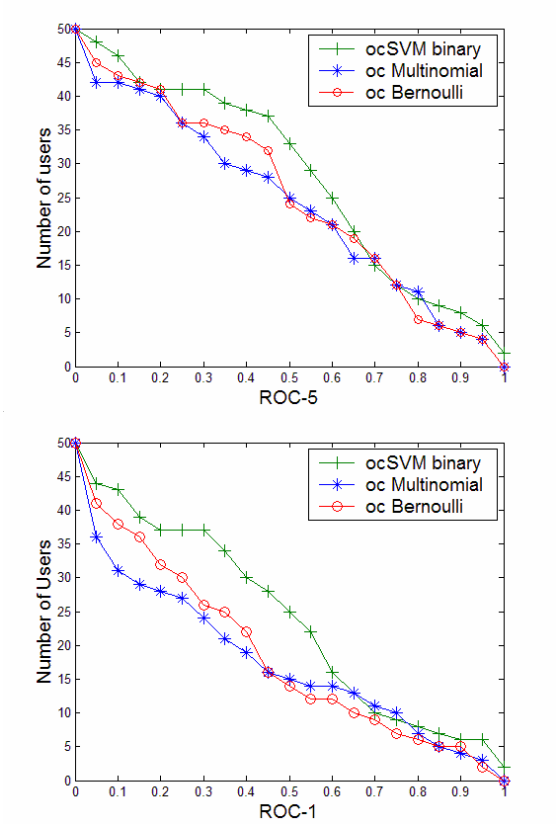

Figure 8: Comparison of ocSVM-binary with one-class Multinomial NB and one-class Bernoulli NB, restricted to 5% and 1% false positive rate, respectively. The graph plots the total number of users for which a given method exceeds an ROC-5, ROC-1 score threshold.

To compare the performance of different algorithms on an individual user basis, we compare the ROC-P score user by user. Figure 9 shows a user-by-user comparison of one-class SVM using the binary feature representation and one-class Naïve Bayes using the multinomial model, when the false positive rate is lower than 1%. Again we can see, for most of the 50 users, one-class SVM with binary features is better than one-class Naïve Bayes using the multinomial model. However, there are still some users whose data exhibit better performance using the one-class Naïve Bayes. This suggests that we can choose the best algorithm to use for an individual user to improve the whole system's performance.

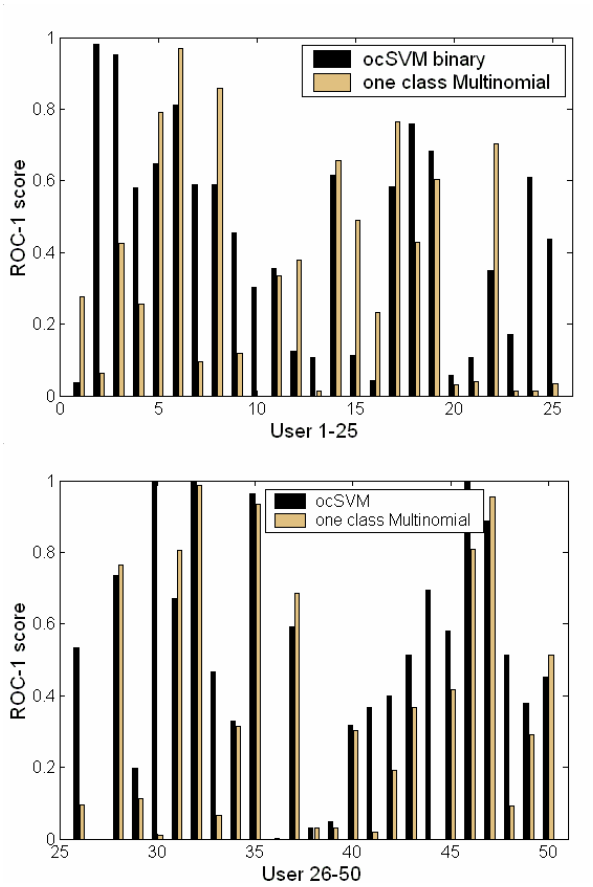

Figure 9: User-by-user comparison of ocSVM-binary and one-class Naïve Bayes using the multinomial model, restricted to 1% false positive.

5. Discussion

From our work we can see that one-class SVM using binary features performs better than one-class Naïve Bayes and one-class SVM using word count features.

Even so, masquerade detection is a very hard problem, and all three algorithms did not achieve very high accuracy with near to zero false positive rates for every user. This is partly caused by the inherent nature of the data available and the difficulty of this problem. We would like to reapply these methods using a richer set of data as described by Maxion [10], incorporating command arguments. We also believe that temporal data associated with each user's sequential commands will provide considerable value as well to improve performance.

Another problem to consider for the practical utility of these approaches is resiliency to direct attack; i.e. how could we protect the models that were computed from, for example, a mimicry attack by the masquerader?

In the experiments performed, we did not evaluate feature selection. We tested one-class SVM using 100,

200, and 300 of the most frequently used UNIX commands. Each of the results is worse than had we used all of the available UNIX commands, whose total number is around 870. We also conjectured that 2-gram features (adjacent pairs of commands) would perform better than individual commands (1-grams) as a feature. However, we found that the results were worse when we used all of the 2-grams. In further work, we would evaluate some feature selection methods to improve performance. For example, we believe a selection of some features using both 1-gram and 2-grams may improve the quality of the user profiles, and thus the accuracy of the detector.

A system to detect masqueraders as described in this paper should not be viewed as a single detector, but rather as evidence to be correlated with other sensors and other detectors. Thus, although the performance of the detectors described herein and in prior work seemingly are not accurate enough, when one wishes to limit false positives, it may be wise to relax the threshold to generate higher true positive rates. If the output of the detector were combined with other evidence (for example, file system access anomaly detection, or other sensors), it may be possible to raise substantially the bar in protecting hosts from malicious abuse.

6. Conclusion

In this paper, to solve the masquerade detection problem, we use one-class training algorithms which only train on a user's clean data. It has been demonstrated that one-class training algorithms can achieve similar performance as multiple class methods, but require much less effort in data collection and centralized management. Besides masquerade detection, we believe one-class training is also good for some other intrusion detection problems where sample intrusion data are hard to get or too variable to cluster.

We also give a detailed comparison of the performance of different one-class algorithms as applied to multiple users. The results show that for most users one-class SVM using the binary feature representation is better than one-class Naïve Bayes and one-class SVM using the word count representation, especially when we want to restrict the false positive rate to a relatively low level.

In our future work, we plan to include command arguments, not only truncated commands, as features to improve the accuracy of masquerade detection. As the number of features increase, we also plan to do feature selection to find the most informative features and to discard those features that have no value for the target task.

Acknowledgments

This work was partially supported by DARPA contract No. F30602-02-2-0209. We also thank Prof. Tony Jebara for helpful suggestions and valuable comments.

Reference:

[1] Chih-Chung Chang and Chih-Jen Lin, "LIBSVM: a library for support vector machines", 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[2] Eleazar Eskin, Wenke Lee and Salvatore J. Stolfo, "Modeling System Calls for Intrusion Detection with Dynamic Window Sizes", *Proceedings of DISCEX II*, June, 2001.

[3] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff, "A sense of self for UNIX processes", In *Proceedings of IEEE Symposium on Security and Privacy*, 1996.

[4] Anup K. Ghosh and Aaron Schwartzbard, "A study in using neural networks for anomaly and misuse detection", In *Proceedings of USENIX Security Symposium* 1999

[5] M. Gribskov and N. L. Robinson, "Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching", *Computers and Chemistry*, 20(1):25–33, 1996.

[6] Thorsten Joachims, "Text categorization with support vector machines: Learning with many relevant features", In *Proc. of the European Conference on Machine Learning (ECML)*, pp. 137-142, 1998.

[7] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection", In *Proceedings of USENIX Security Symposium* 1998

[8] T. Lunt, A.Tamaru, F. Gilham, R. Jagannathan, C. Jalai, H.S. Javitz, A. Valdes, and P.G. Neumann, "A Real-Time Intrusion Detection Expert System," *SRI CSL Tecnical Report*, SRI-CSL-90-05, June 1990.

[9] Maxion, Roy A. and Townsend, Tahlia N, "Masquerade Detection Using Truncated Command Lines", *International Conference on Dependable Systems and Networks (DSN-02)*, pp. 219-228, Washington, D.C. 23-26 June 2002.

[10] Maxion, Roy A. "Masquerade Detection Using Enriched Command Lines", In *International Conference on Dependable Systems & Networks (DSN-03)*, pp. 5-14, San Francisco, California, 22-25 June 2003. IEEE Computer Society Press, Los Alamitos, California, 2003.

[11] A. McCallurn, K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification", *AAAI-98 Workshop on Learning for Text Categorization*, 1998

[12] T. M. Mitchell, Bayesian Learning, Chapter 6 in *Machine Learning*, pp. 154-200. McGraw-Hill, 1997.

[13] B. Scholkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson, "Estimating the support of a high-dimensional distribution". *Technique report, Microsoft Research*, MSR-TR-99-87, 1999.

[14] M. Schonlau, W. DuMouchel, W. -H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades", *Statistical Science*, 16(1):58-74, February 2001.

[15] Matthew G. Schultz, Eleazar Eskin, and Salvatore J. Stolfo, "Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables", *Proceedings of USENIX Annual Technical Conference - FREENIX Track,* Boston, MA: June 2001.

[16] S. Y. Sedelow, "The Computer in the Humanities and Fine Arts", *ACM Computing Surveys* 2(2): 89-110 (1970)

[17] Salvatore J. Stolfo, Shlomo Hershkop, Ke Wang, Olivier Nimeskern, and Chia-Wei Hu, "Behavior Profiling of Email", *1st NSF/NIJ Symposium on Intelligence & Security Informatics (ISI 2003),* June 2-3, 2003, Tucson, Arizona.

[18] O. De Vel, A. Anderson, M. Corney, and G. Mohay, "Mining Email Content for Author Identification Forensics", *SIGMOD: Special Section on Data Mining for Intrusion Detection and Threat Analysis*, December 2001.

[19] Nong Ye, "A Markov Chain Model of Temporal Behavior for Anomaly Detection", *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, 2000.