Masquerade Detection Using a Taxonomy-Based Multinomial Modeling Approach in UNIX Systems

Malek Ben Salem and Salvatore J. Stolfo

Computer Science Department, Columbia University

500 West 120th Street, New York, NY, 10027

Abstract

This paper presents one-class Hellinger distance-based and one-class SVM modeling techniques that use a set of features to reveal user intent. The specific objective is to model user command profiles and detect deviations indicating a masquerade attack. The approach aims to model user intent, rather than only modeling sequences of user issued commands. We hypothesize that each individual user will search in a targeted and limited fashion in order to find information germane to their current task. Masqueraders, on the other hand, will likely not know the file system and layout of another user's desktop, and would likely search more extensively and broadly. Hence, modeling a user search behavior to detect deviations may more accurately detect masqueraders. To that end, we extend prior research that uses UNIX command sequences issued by users as the audit source by relying upon an abstraction of commands. We devised a taxonomy of UNIX commands that is used to abstract command sequences. The experimental results show that the approach does not lose information and performs comparably to or slightly better than the modeling approach based on simple UNIX command frequencies.

1. Introduction

The *masquerade attack* is a class of attacks, in which a user of a system illegitimately poses as, or assumes the identity of another legitimate user. Identity theft in financial transaction systems is perhaps the best known example. Masquerade attacks are extremely serious, especially in the case of an insider who can cause considerable damage to an organization. The insider attack detection problem remains one of the more important research areas requiring new insights to mitigate against this threat.

A common approach to counter this type of attack, which has been the subject of prior research, is to develop novel algorithms that can effectively identify suspicious behaviors that may lead to the identification of imposters. We do not focus on whether an access by some user is authorized since we assume that the masquerader does not attempt to escalate the privileges of the stolen identity, rather the masquerader simply accesses whatever the victim can access. However, we conjecture that the masquerader is unlikely to know how the victim behaves when using a system. It is this key assumption that we rely upon in order to detect a masquerader. Thus, our focus in this paper is on monitoring a user's behavior in real time to determine whether current commands issued by a user are consistent with the user's historical behavior. The far more challenging problems of thwarting mimicry attack and other obfuscation techniques, as well as the use of honeypots or honeytokens, are beyond the scope of this paper.

Masquerade attacks 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 password, or through a break in and installation of a rootkit or keylogger. In either case, the user's identity is illegitimately acquired. Another perhaps more common case is laziness and misplaced trust by a user, such as the case when a user leaves his or her terminal or client open and logged in allowing any nearby co-worker to pose as a masquerader. In the first two cases, the identity thief must log in with the victim's credentials and begin issuing commands within the bounds of one user session. We conjecture that legitimate users initiate the same repeated commands each time they log in to set their environment before using it, initiate some set of applications (read email, open a

browser, or start a chat session) and similarly, clean up and shut down applications when they log off. Such repeated behaviors constitute a profile that can be modeled and used to check the authenticity of a user session early before significant damage is done. The case of hijacking a user's session is perhaps a bit more complicated. We presume the preamble commands issued by the legitimate user have already concluded and have no value in detecting the masquerader. In either case, a monitoring system ought to detect any significant deviations from a user's typical profiled behaviors in order to detect a likely masquerade attack. Ideally, we seek to detect a possible masquerader at any time during a session.

In an operational monitoring system, one naturally would be concerned with the error rate of a detector. In a nutshell, we posit that at a minimum a challenge/response mitigation strategy may work well to prevent significant damage to a system by a masquerader. The downside of a false positive is essentially annoyance by a legitimate user who may be challenged too frequently. However, a false positive should do no damage. An interesting problem to study is how to calibrate a detector to balance its false positive rate to ensure its false negative rate is minimized. False negatives in this context, i.e., an undetected masquerader, are far more dangerous than an annoying false positive. A thorough discussion of mitigation strategies is beyond the scope of this paper.

In this paper we extend prior work on modeling user command sequences for masquerade detection. We use one-class support vector machines and introduce the use of 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. Previous work has focused on auditing and modeling sequences of user commands including work on enriching command sequences with information about arguments of commands. [1], [3], and [4].

We propose an approach to profile a user's behavior based on a 'taxonomy' of UNIX commands. The taxonomy abstracts the audit data and enriches the meaning of a user's profile. Hence, commands that perform similar types of actions are grouped together in one category making profiled sequences more abstract and meaningful. Furthermore, modeling sequences of commands is complicated whenever "Never-Before-Seen-Commands" are observed. A command taxonomy reduces this complexity, since any distinct command is replaced by its category, which is very likely to have been observed in the past. Commands are thus assigned a type, and the sequence of command types is modeled rather than individual commands.

One particular type of command is "information gathering" commands, i.e. *search* commands. We conjecture that a masquerader is unlikely to have the depth of knowledge of the victim's machine (files, locations of important directories, available applications, etc.), and hence, a masquerader would likely first perform information gathering and search commands before initiating specific actions. A taxonomic command abstraction helps achieve our goal to model the user's intent in this specific case. A detector may increase its suspicion of an attack in real time if it detects monitored search command actions that deviate substantially from the user's profile on search commands.

In Section 2 of this paper, we briefly present the results of prior research work on masquerade detection. Section 3 expands on the objective and the approach taken in this work. The experiments conducted to evaluate whether a command taxonomy impacts the efficacy of user behavior models are presented in Section 4. In section 5, we evaluate the results of our experiments while section 6 discusses future work needed to improve and better evaluate our proposed modeling approach. Section 7 concludes the paper summarizing the contributions of this paper.

2. Related Work

In the general case of computer user profiling, the entire audit source can include information from a variety of sources:

- Command line calls issued by users
- System call monitoring for unusual application use/events
- Database/file access monitoring
- Organization policy management rules and compliance logs

The type of analysis used is primarily the modeling of statistical features, such as the frequency of events, the duration of events, the co-occurrence of multiple events combined through logical operators, and the sequence or transition of events. However, most of this work failed to reveal

or clarify the user's intent when issuing commands. The focus is primarily on accurately detecting change or unusual command sequences. In this section, we focus on the approaches reported in the literature that profile users by the commands they issue.

Schonlau *et al.* in [1] applied six masquerade detection methods to 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. The complete data set and more information about it can be found at http://www.schonlau.net. The objective was to accurately detect the "dirty" blocks and classify them as masquerader blocks.

The first detection method applied by Schonlau et al. for this task, called "uniqueness" relies on the fact that half of the commands in the training data are unique and many more are unpopular amongst the users. The second method investigated was the Bayes one-step Markov approach. It is based on one step transitions from one command to the next. The approach, due to DuMouchel (1999), uses a Bayes factor statistic to test the null hypothesis that the observed one-step command transition probabilities are consistent with the historical transition matrix. The two hypotheses modeled are the null hypothesis, which assumes that the observed transitions probabilities stem from the historical transition matrix, and the alternative hypothesis which assumes that they were generated from a Dirichlet distribution.

A hybrid multi-step Markov method has also been used. In order to overcome the high-dimensionality, inherent in multi-step Markov chain, a "mixture transition distribution" (MTD) approach has been used to model the transition probabilities. When the test data contain many commands unobserved in the training data, a Markov model is not usable. Here, a simple independence model w/ probabilities estimated from a contingency table of users versus commands may be more appropriate. The method used automatically toggles between a Markov model and an independence model generated from a multinomial random distribution as needed, depending on whether the test data are "usual", i.e. the commands have been previously seen, or "unusual", i.e. Never-Before-Seen Commands (NBSCs). We note with interest that the proposed taxonomy of commands tends to reduce if not eliminate the problem of modeling "Never-Before-Seen-Commands" since any command is likely to be placed in a category with other similar commands. Hence, although a specific command may never have been observed, members of its class probably were.

The compression method, which was also applied to the Schonlau data set, was based on the premise that test data appended to historical training data compress more readily when the test data stems indeed from the same user rather than from a masquerader, and was implemented through the UNIX tool "compress" which implements a modified Lempel-Ziv algorithm.

IPAM (Incremental Probabilistic Action Modeling), another method applied on the same dataset, and used by Davidson & Hirsch in [5] and [16] to build an adaptive command line interface, is also based on one-step command transition probabilities estimated from the training data. Probabilities are continuously updated using an exponential updating scheme. With arrival of a new command, probabilities are aged by multiplying with *alpha* and *(1-alpha)* is added to the most recent transition.

The sequence-match approach was presented by Lane & Brodley [6]. For 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 command sequences of length 10 that the user has previously used. The similarity measure is a count of the number of matches in a command-by-command comparison of 2 command sequences with a greater weight assigned to adjacent matches. This similarity measure is computed for the test data sequence paired with each command sequence in the profile.

Maxion and Townsend applied a naïve Bayes classifier, which has been widely used in text classification tasks, to the same data set in [3]. Maxion provides a thorough and detailed

investigation of classification errors [7], highlighting why some masquerade victims are more vulnerable than others, and why some masqueraders are more successful than others. Maxion and Townsend also designed a new experiment, which they called the "1v49" experiment, in order to conduct this error analysis.

A method, that is significantly different from other intrusion detection technologies, was presented by Coull et al. [11]. The method is known as semi-global alignment and is a modification of the Smith-Waterman local alignment algorithm, with a scoring system that rewards the alignment of commands in the user segment but does not necessarily penalize the misalignment of large portions of the signature of the user.

Another approach called a self-consistent naïve Bayes classifier was proposed by Yung [13] and applied on the same data set. This method is a combination of the naïve Bayes classifier and the EM-algorithm. The self-consistent naïve Bayes classifier does not have to make a binary decision for each new block of commands. Rather, it assigns a score to it that indicates the probability that the block is a masquerader block. Moreover, this classifier can change scores of earlier sessions as well as later sessions

Oka et al. had the intuition that the dynamic behavior of a user appearing in a sequence can be captured 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). Based on that intuition they have developed the layered networks approach based on the Eigen Co-occurrence Matrix (ECM) in [13] and [14]. The ECM method extracts the causal relationship embedded in sequences of commands, where a co-occurrence means the relationship between every two commands within an interval of sequences of data. This type of relationship cannot be reflected through histograms nor through n-grams.

Forrest et al. proposed a real-time on-line anomaly detection system [15] that mimicked the mechanisms used by the natural immune systems. This was done by monitoring system calls of running privileged processes (profiles were built using normal runs of such programs), rather than sequences of user commands, and therefore used a different data set than the Schonlau data set. The modeling was limited to privileged root processes since they have more access to computer resources than user processes, and they have a limited range of behavior that is quite stable and predictable. A separate database of normal behavior is built for each privileged process. The database was specific to a particular architecture, software version and configuration, local administrative policies, and usage patterns, providing a unique definition of "self".

Table 1 presents the results of all methods described above, that were applied to the Schonlau data set and which are based on a two-class training approach of a self model and a non-self model. In a real-word setting, it is probably more appropriate to use a one-class training approach; as users join and leave the organizations, keeping the non-self model up-to-date can be really challenging. Wang and Stolfo tried such an approach in [4] by using both, a naïve Bayes classifier and a Support Vector Machine (SVM), to detect masqueraders. With the naïve Bayes classifier, they have used the multinomial model, as well as the multivariate Bernoulli event model. Their work has shown that the difference in detection accuracy between the two models is not so obvious in one-class training, especially when the false positive rate is low, unlike the case of multi-class training, where the multinomial model performs better than the Bernoulli one.

Method	False Alarms (%)	Missing Alarms (%)
Uniqueness	1.4	60.6
Bayes one-step Markov	6.7	30.7
Hybrid multi-step Markov	3.2	50.7
Compression	5.0	65.8
Sequence Match	3.7	63.2
IPAM	2.7	58.9
Naïve Bayes (Updating)	1.3	38.5
Naïve Bayes (No Upd.)	4.6	33.8
Semi-Global Alignment	7.7	24.2
Eigen Co-occurrence Matrix	3.0	28.0
Naïve Bayes + EM	1.3	25.0

Table 1: Summary of accuracy performance of Two-Class Based Anomaly Detectors Using the Schonlau Data Set

Wang and Stolfo have also investigated SVMs using the binary features and frequency-based features. The one-class SVM algorithm using binary features was the best one among all four one-class training algorithms that were analyzed. It also performed better than most of the two-class algorithms listed above, except the two-class multinomial naïve Bayes

algorithm with updating. In summary, Wang and Stolfo's experiment confirmed that for masquerade detection, one-class training is as effective as two class training. The practical significance of this is important. In prior work, a masquerade detector is trained as a classifier by mixing labeled data from a number of different users. Besides the privacy implications of this approach, whenever a new user enters or leaves an organization each detector would necessarily be retrained in order to maintain its accuracy. In the case of "one-class" modeling, an anomaly detector is trained for each user on their own data. It is this approach that sets the stage for our subsequent work described next.

3. Objective and Approach

When dealing with the masquerader attack detection problem, it is important to remember that the attacker has already obtained credentials to access a system. When presenting the stolen credentials, the attacker is then a legitimate user to any access control system. Ideally, monitoring a user's actions after being granted access is required in order to detect such attacks. Furthermore, if we can determine the user's intent, we may automatically determine if actions of a user are malicious or not. We have postulated that certain classes of user commands reveal user intent. For instance, "search" should be an interesting behavior to monitor. Hence, we defined a taxonomy of commands to readily identity and model search behavior. Another behavior that is interesting to monitor is remote access to other systems and the communication or egress of large amounts of data to remote systems, which may be an indication of illegal copying or distribution of sensitive information. Once again, the taxonomy defined allows a system to automatically audit and model this behavior as well. However, user behavior naturally varies from each user. We believe there is no one model or one easily specified policy can capture the inherent vagaries of human behavior. Instead, we aim to automatically learn a distinct user's behavior, much like a credit card customer's distinct buying patterns.

Our objective is to model the normal pattern of submitted commands of a certain user in a UNIX environment assuming that the masquerader will exhibit different behavior from the legitimate user and this deviation will be easily noticed. In order to detect the deviations we compute the Hellinger distance between the frequencies of recent commands or command categories that show up in one block of commands of window size *w* and a second block of the same window size shifted by only one command. Hence, this approach essentially tracks a user's behavior and measures any changes in that behavior. Any significant change will raise an alarm. In the following we present the command taxonomy that we have developed as well as the Hellinger distance applied to blocks of issued commands.

3.1. UNIX Command Taxonomy

We abstract the set of Linux/Unix commands into a taxonomy of command categories as presented in Figure 1. In particular, we are interested in identifying the specific set of commands that reveal the user's intent to search, to change access control privileges, and to copy or print information. Once these commands are identified, we can extract features representing such behavior while auditing the user's behavior.

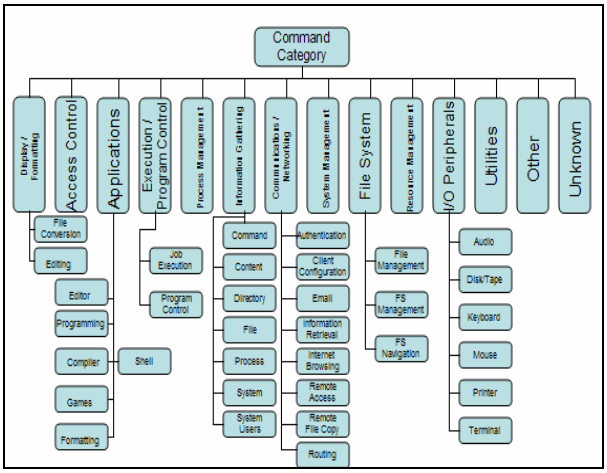


Figure1: Taxonomy of Linux and Unix Commands

The taxonomy has 14 different categories: Access Control, Applications, Communications / Networking, Display / Formatting, Execution / Program Control, File System, I/O Peripherals, Information Gathering, Other, Process Management, System Management, Unknown, and Utilities. Most categories were further classified into sub-categories, however some did not require more granularity, such as the "Resource Management" category. The "Information Gathering" category

includes commands such as "find" and "fgrep". Examples of commands in the "Process Management" category include "kill", "nohup", and "renice". "date", "clock" and "cal" are examples of commands that fall in the "Utilities" category. The "Other" category includes commands that have been recognized but could not be classified under any other category. However, the "Unknown" category includes commands that were not identified or script names that are not recognizable.

3.2. Hellinger Distance

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. Here, we measure the frequency of commands, and thus one can develop a detector of abnormal behavior by modeling user command frequencies and the changes in that frequency. The Hellinger distance is defined as:

$$HD(f_p[], f_t[]) = \sum_{i=0}^{n-1} (\sqrt{f_p[i]} - \sqrt{f_y[i]})^2$$

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/ command categories. 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 command types. Hence, each command creates a new Hellinger distance score that is subjected to threshold logic. Each bin of the frequency table is any chosen category of command we wish to model. In the most general case all command categories would be tracked. The method is straightforward and efficient to implement. It remains to me seen how accurate it may be, and whether modeling categories of command significantly reduces the information available when modeling sequences of commands. In other words, is it more accurate to model sequences of commands or frequencies of commands' categories. It is that question we address next.

3.3. One-Class Support Vector Machines

Support Vector Machines (SVMs) are linear classifiers used for classification and regression. They are known as maximal margin classifiers, as opposed to probabilistic classifiers, thanks to their ability of minimizing empirical classification error while maximizing geometric margin.

SVMs are typically used in multi-class classification tasks. Scholkopf et. al. proposed a way to adapt SVMs to the one-class classification task [16]. 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 suing 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 hyperplane 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 $x,y \in \boldsymbol{X}$, $\boldsymbol{X}$ is the training data set, and Φ is the feature mapping to a high-dimensional space $X \rightarrow F$. The RBF Kernel is defined as

$$k(x,y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}.$$

4. Experiments

As mentioned in previous sections, the task here is to monitor changes in user command behavior, and detect any deviations from normal behavior. We use the Schonlau data set presented in section 2, comprised of sequences of 15,000 commands for 50 users. The first 5,000 have all been issued by the legitimate user, however the remaining 10,000 have been injected at random locations with blocks of 100 commands issued by other users (or simulated masqueraders). For each user, there are between 0 and 24 masquerade blocks. In the first experiment, we apply the Hellinger distance to detect changes in user behavior. In the second experiment, we use the Support Vector Machine (SVM) for detection on the Schonlau data set, and we call it the SEA experiment, as in [3]. Finally in the third experiment, we also use SVMs, but with a different experimental setup where we train the model using one user's data and test on all other 49 users' data, which is why the experiment is called the 1v49 experiment.

4.1. Hellinger Distance Experiment

In order to detect the changes in user behavior, we compute the Hellinger distance between two blocks of commands of size w. With each new command, we count the frequencies of the

commands or command categories within the block of the last *w* commands. We shift the window of *w* commands by one command, and we count the same frequencies for the new block. Then we compute the Hellinger distance between the two sets of commands. However, we need to determine what the most appropriate window size of *w* should be. We describe below how we determined the value of *w*.

4.1.1. Determining the window size *w*

We conducted a first experiment where the profiles of all users have been modeled using several window sizes. For each user, we divide the first 5000 commands into blocks of size *w*, and we count the frequencies of command categories within each block. Then we calculate the mean and the standard deviation for each of those frequencies. The results for users 1 and 2 are shown in Figure 2 for window sizes of 50, 100, and 200 commands, respectively. By looking at both profiles, we notice that the two users have indeed different profiles. This shows the potential for detection of the command blocks that have been contaminated by another user's commands. The window size has been selected, such that the standard deviation 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. This tradeoff serves as the core "calibration" process for a detector using this method. The window size *w* that helped achieve a tradeoff between these two objectives was a size of 50 commands.

4.1.2. Determining a threshold

Once the window size 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 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 command categories, and the method is relatively easy and efficient to implement. (A fixed size histogram would be maintained for each user at run-time in an operational setting.)

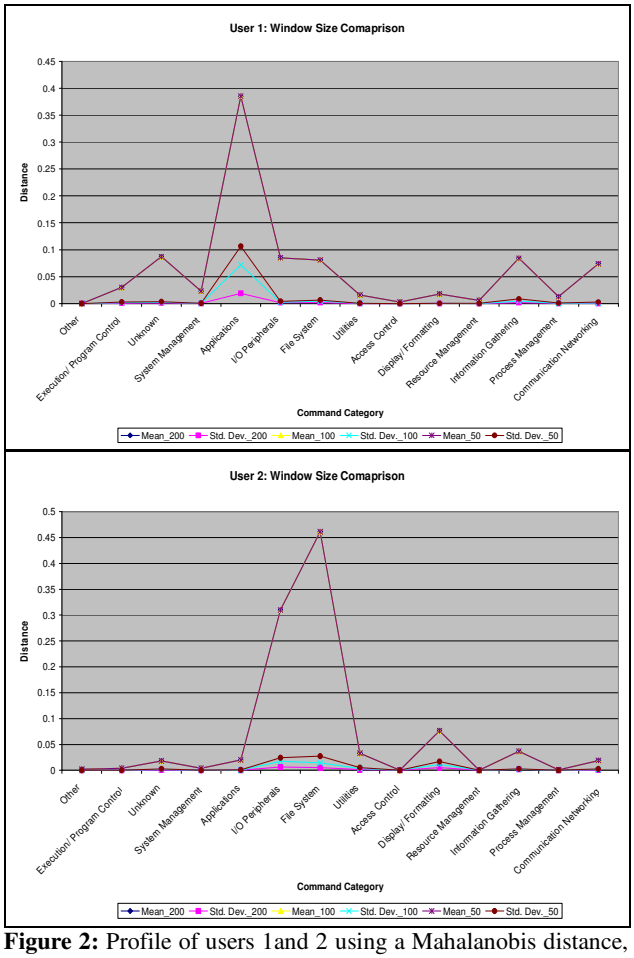


Figure 2: Profile of users 1and 2 using a Mahalanobis distance, the commands taxonomy, and window sizes of 200, 100, and 50 commands.

4.2. SVM Experiments

We have used the LIBSVM package [19] to conduct our SVM experiments. LIBSVM is a tool for support vector classification, regression, and distribution estimation. It supports both multi-class classification and one-class classification. The one-class SVM function provided by this tool uses the RBF kernel. We have applied this kernel with the default settings to conduct the SEA and 1v49 experiments. We have created a new version of the LIBSVM code, 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 have used two different ways to represent features. The first is frequency-based where we count the number of times a simple command or a

command category, retrieved using the command taxonomy, shows up in the data set. The second approach is binary where we indicate whether the command or command category is present in the data set.

4.2.1. SEA Experiment

In this experiment we follow the methodology described in [1] and [4], and we show that the performance of one-class SVMs (ocSVM) using command categories per our taxonomy is almost the same as the performance of ocSVM that uses simple commands. We use the first 5000 commands of a user as positives examples for training the model. No negative examples are used for training. Then we test the classifier using the remaining 10,000 commands of the user, which may have injected command blocks from other users under a probability distribution described in [2], which we will refer to as the "dirty" blocks.

4.2.2. 1v49 Experiment

The 1v49 was introduced by Maxion and Townsend to avoid some of the inherent methodological problems with the Schonlau data set. One major shortcoming of the experimental setup that Schonlau used is that each user in the data set has a different number of "dirty" blocks than the remaining 49 users. Moreover, these blocks originate from different users. This may prevent the comparison of the performances of the classification algorithms on solid ground. In order to better evaluate and compare the performance of these one-class classification algorithms, a different data configuration is used: the first 5,000 commands of a user are used as negative training data, no positive examples are used. The test data is made up of the remaining 49 users' first 5,000 commands, which make the positive test examples, s well as the non-masquerade data from the additional 10,000 commands of the same user.

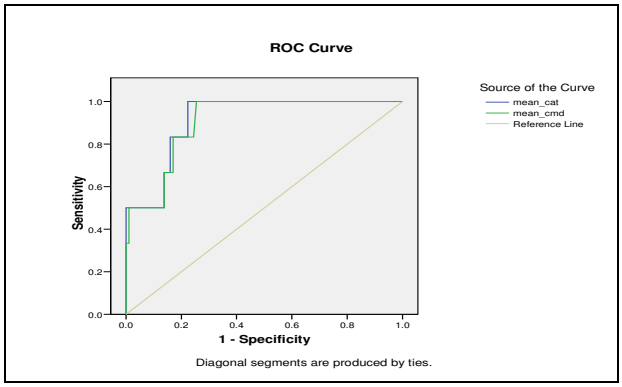
5. Evaluation

5.1. Hellinger-Distance Method 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. It is important to note that our approach is a one-class modeling approach. No positive examples of masquerade blocks were used for training, and no non-self models were computed for each user. 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, we take the average score over the 100-command block as the overall score. We have also experimented with using the maximum value of those scores. However, better classification accuracy has been achieved by using the average scores. If the average score is above the user threshold, the block is classified as a masquerader block. We do not use any blocks that have been classified as normal to update the user profile or recalculate the threshold.

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 $\frac{n_{TP}}{n_{TP} + n_{FN}}$ where n_{TN}, n_{FP}, n_{TP}, n_{FN} are the numbers of true negatives, false positives, true positives, and false negatives respectively. Figure 3 displays the ROC curves for users 2 and 6 for the Hellinger distance-based approach using the frequencies of simple commands and the same approach using the frequencies of command categories.

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.



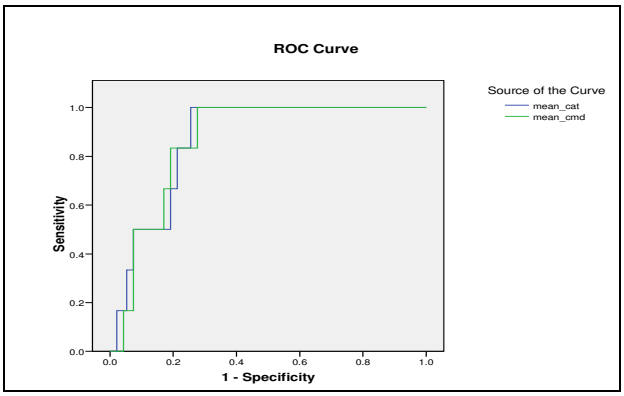


Figure 3: ROC curves for users 18 and 36 respectively using the Hellinger distance metric. *mean_cat* represents the model where the Hellinger distance is computed using frequencies of command categories. *mean_cmd* stands for the model where the distance computed using frequencies of simple commands.

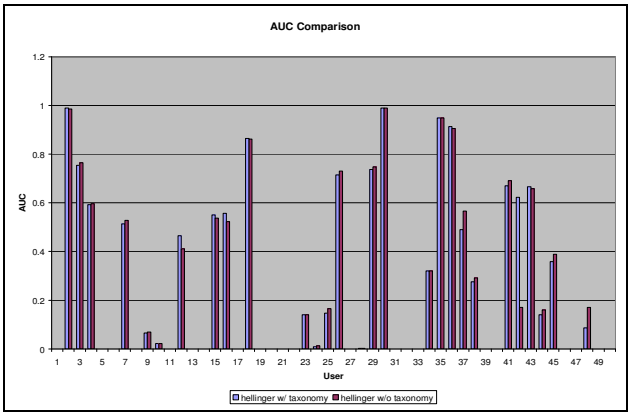


Figure 4: User-by-user comparison of ROC scores (AUCs)

The AUCs displayed in Figure 3 of the taxonomy-based model for user 18 and 36 are 0.865 and 0.913 and those of the command-based model are 0.862 and 0.905 respectively. For some users the taxonomy-based model performed better than the command-based one, for others it did not. Figure 4 shows a user-by-user AUC comparison or all users whose files have been contaminated. Some users had no masquerader blocks injected, and therefore it was not possible to build a ROC curve for them. Comparing the average AUC for those scores, the taxonomy-based modeling method achieves more than a 1.8% improvement over the command-based one, with less information.

Figure 5 shows a comparison of the total number of users whose files have been contaminated and whose AUC's are higher than a certain AUC x versus the value of x. The taxonomy-based approach outperforms the simple-commands-based approaches in most cases, and it only underperforms the other approach for very small AUC values.

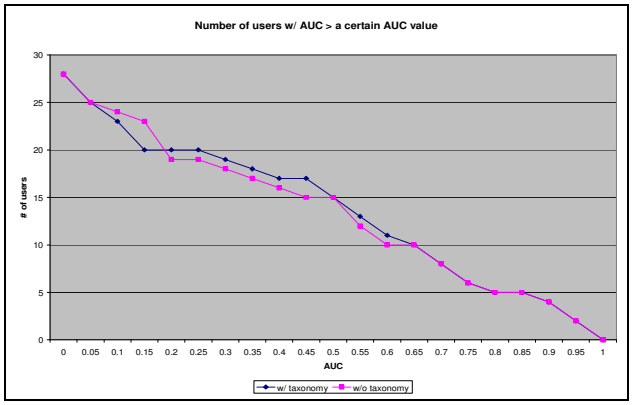


Figure 5: Number of users with ROC scores higher than x versus x

5.2. One-class SVM Evaluation

5.2.1. SEA Experiment Evaluation

Table 2 shows the results achieved by the one-class SVM classifiers. The ocSVMs that use the command taxonomy achieve better false alarm in both cases, when using the frequency-based model and when using the binary model. With the frequency-based model, the ocSVM using the command taxonomy also achieves comparable results with the ocSVM using simple commands. This is not true for the ocSVM using the binary feature. This can be explained by the difficulty to build representative models for the users based on a

binary vector that only has 14 components, since the taxonomy is only made up of 14 different categories, particularly when the training data set includes 5,000 commands. With such a number of commands, the chances that all 14 categories of commands are represented are relatively high, and therefore the binary models built for the users will be very similar, i.e. the classifier will be unable to correctly decide whether a test command block is a self block or a masquerader block.

Method	Hit Rate	False Positive Rate
ocSVM using simple cmds (freq.-based model)	98.7%	66.47%
ocSVM using taxonomy (freq.-based model)	94.8%	60.68%
ocSVM using simple commands (binary model)	99.13%	66.8%
ocSVM using taxonomy (binary model)	86.58%	56.9%

Table 2: SEA experiment results

As mentioned earlier, we have modified the LIBSVM code so that the one-class prediction models output a probability that a vector belongs to the "self" class, rather than output the classification value "self" or "non-self". We have used these prediction values to build ROC curves for each model/user, and we show the corresponding AUC scores in figure 6. We have averaged out the performance of all user models to build a single ROC curve for each method in figures 7-10. The ROC curves and corresponding AUC values confirm that, when using the frequency-based model to build the feature vectors, using the command taxonomy achieves comparable results to those achieved when modeling simple commands.

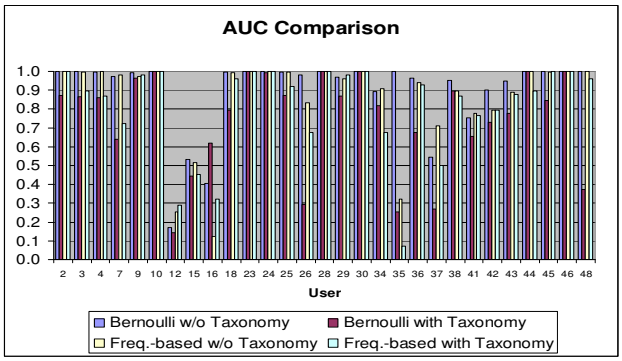


Figure 6: Comparison of AUC scores achieved using the 4 models in the SEA experiment

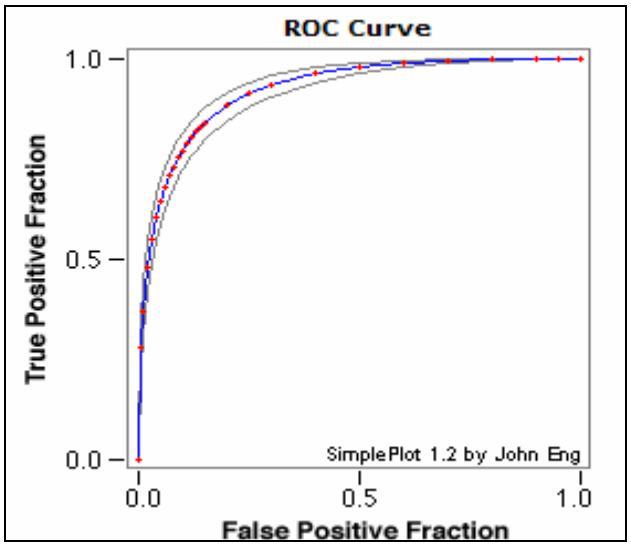


Figure 7: ROC curve for ocSVM with binary model using simple commands (AUC=0.925)

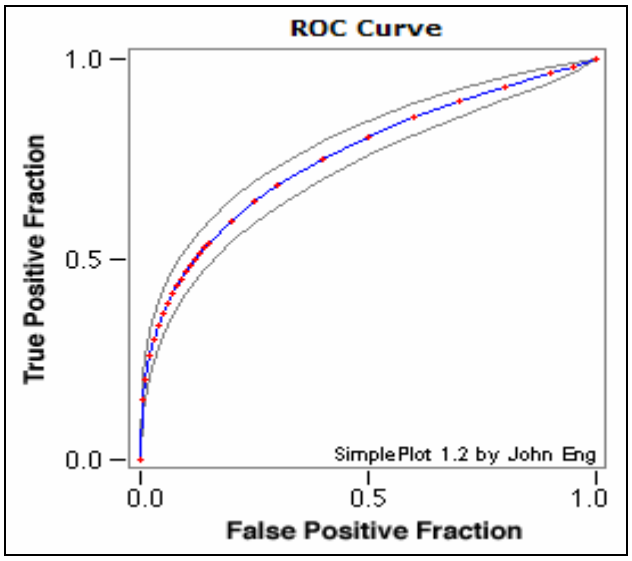


Figure 8: ROC curve for ocSVM with binary model using command taxonomy (AUC=0.756)

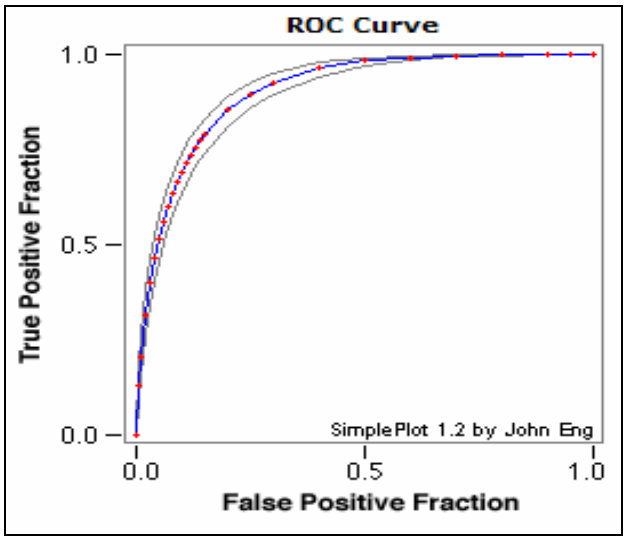


Figure 9: ROC curve for ocSVM with frequency-based model using simple commands (AUC=0.905)

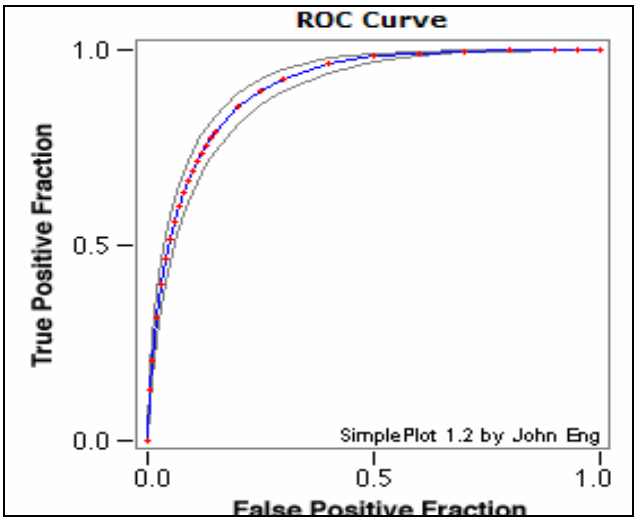


Figure 10: ROC curve for ocSVM with frequency-based model using command taxonomy (AUC=0.89)

5.2.2. 1v49 Experiment Evaluation

The results of the 1v49 experiment shown in table 3 and in figure 11 confirmed the results of the SEA experiment. In particular, it proved that high hot rates can be achieved with the taxonomy, and that when using the frequency-based modeling approach, building features while using of the taxonomy allows for comparable results to those achieved when the features are extracted just from simple commands.

Method	Hit Rate	False Positive Rate
ocSVM using simple cmds (freq.-based model)	96.56%	66.47%
ocSVM using taxonomy (freq.-based model)	87.54%	60.66%
ocSVM using simple commands (binary model)	97.81%	67.03%
ocSVM using taxonomy (binary model)	81.91%	57.11%

Table 3: 1v49 experiment results

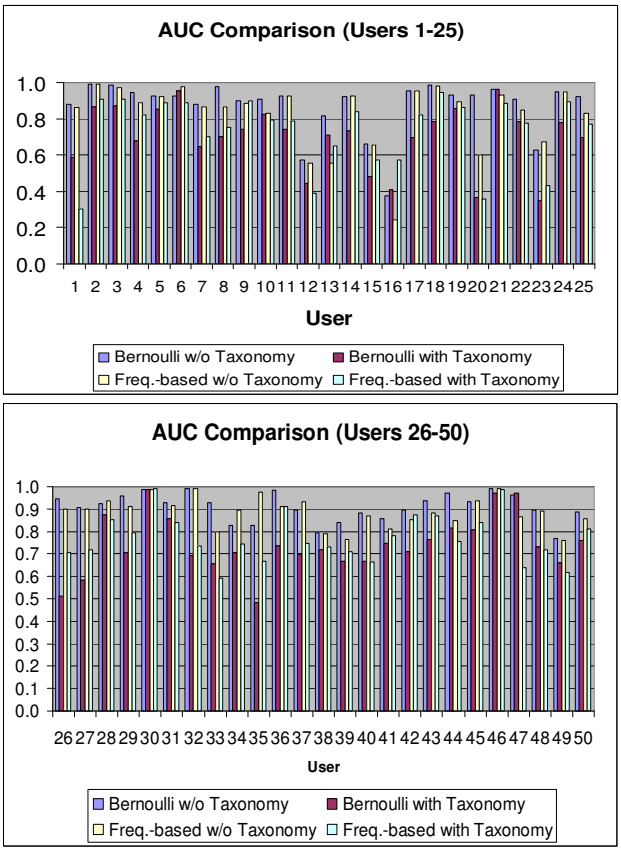


Figure 11: Comparison of AUC scores achieved using the 4 models in the 1v49 experiment

6. Discussion and future work

Unlike a modeling approach based on frequencies of simple commands, the taxonomy-based approach should not raise an alarm for a masquerader if, for instance, the same legitimate user starts running a different C compiler than what he/she normally uses. Both compilers used should be under the "Applications" category. So if the user continues doing the same things he has been doing before, except for the change of compilers, the user model does not change if we use our taxonomy-

based approach. However, using the simple commands approach might raise an alarm for a masquerade. Therefore, our approach is expected to limit the occurrences of false positives. Moreover, the taxonomy-based approach tends to reduce the problem of modeling "Never-Before-Seen-Commands" since any command is likely to be placed in a category with other similar commands, i.e., although a specific command may never have been observed, members of its class probably were.

The results shown above confirm that the information that is lost by compressing the different user shell commands into a few categories does not affect the masquerader detection ability significantly. We expect our modeling approach to achieve even better results when using real masquerader data. This is a crucial observation. The Schonlau datasets are not "true Masquerader" data sets. The data from different users were randomly mixed standing as a simulation of a masquerader attack. A willful act of malfeasance after identity theft is yet to be tested, albeit there is no generally available data set of this nature for scientific study. Hence, Schonlau resorted to simulating this malfeasance in as simple a fashion as possible, monitoring different users and mixing their data. The specific test to determine whether modeling search command behavior is, thus, not possible with this dataset. Figure 12 clearly shows that the distribution of the categories of commands for normal users and for masqueraders are very similar. Moreover, "information gathering" or "search" commands only make up between and 10 and 12% of all the commands for both sets. This suggests that the Schonlau data set is not good enough for evaluating our hypothesis that masqueraders exhibit unusual amounts of search behavior that deviates substantially from their victim's behavior. .

In our future work, we plan to tackle specific areas of work: a) developing a "capture the flag" exercise in our lab with user volunteers serving as masqueraders to create suitable datasets for experimentation and evaluations, and b) using other approaches for classification other than one-class support vector machines and averaging of Hellinger distance scores within one block of commands. These may include the rate of change of the Hellinger distance or the quantiles of the empirical distribution of the scores, as well as other one-class anomaly detection methods. Moreover, we will conduct an analysis of why the taxonomy achieved better results for certain users and not for others, and will enhance our Hellinger distance-based approach with an incremental update feature in order to adapt to concept drift. Intuitively, this is achieved by adjusting the frequency table by decrementing the frequency entry of the oldest command type, and incrementing the category of the most recently issued command and incrementally updating the average and standard deviation.

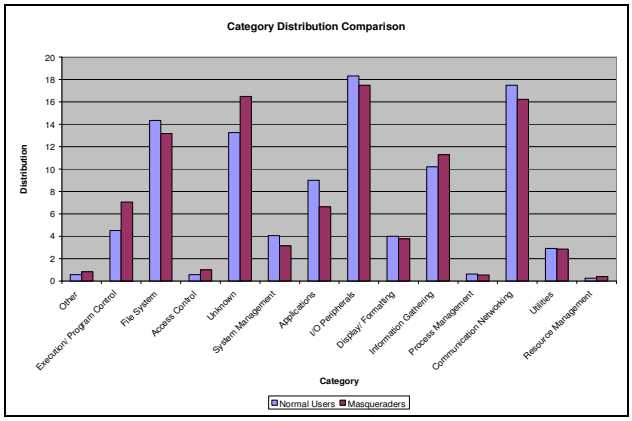


Figure 12: Distribution of command categories for normal user and for masquerader commands

7. Conclusion

Masquerade attacks are a serious computer security problem. In this paper, we have presented a taxonomy of Unix and Linux commands developed to improve upon previous user modeling approaches based upon sequences of commands. The proposed approach aims to capture the intent of a user more accurately. We have used the taxonomy for one-class modeling of user behavior in order to detect masquerades in UNIX environments using a standard benchmark dataset. Even though the data set used is not the most suitable data for the masquerade detection problem, the command taxonomy-based modeling achieved results comparable or slightly better than modeling the behavior with simple commands for this dataset. This establishes some evidence that our conjecture that a taxonomy should help in detecting masqueraders while decreasing the number of false positives is correct. Future larger scale studies in our lab using volunteer masqueraders will be reported upon in future work. We expect to see better results using real data where the masquerader is expected to perform extensive search while exploring the new environment that he/she has access to.

Acknowledgements

This material is based upon work supported by the US Department of Commerce, National Institute of Standards and Technology under Grant Award Number 60NANB1D0127. The I3P is managed by Dartmouth College. The views and conclusions contained in this document are those of the authors and should not be and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland, the I3P, or Dartmouth College.

References

[1] 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, Feb. 2001.

[2] http://www.schonlau.net

[3] R. A. Maxion and T. N. Townsend, "Masquerade Detection Using Truncated Command Lines", International Conference on Dependable Systems & Networks (DSN-02), pp. 219-228, Washington D.C., June 2002.

[4] K. Wang and S. J. Stolfo, "One-Class Training for Masquerade Detection", 3rd IEEE Workshop on Data Mining for Computer Security, Nov. 2003.

[5] B. D. Davison, and H. Hirsh, "Predicting Sequences of User Actions", Working Notes of the Joint Workshop on Predicting the Future: AI Approaches to Time Series Analysis, Fifteenth National Conference on Artificial Intelligence (AAAI98)/Fifteenth International Conference on Machine Learning (ICML98), AAAI Press, 1998.

[6] T. Lane and C. Brodley, "Sequence Matching and Learning in Anomaly Detection for Computer Security", Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management, pp. 43-49, 1997.

[7] R. A. Maxion and T. N. Townsend, "Masquerade Detection Augmented with Error Analysis", IEEE Transactions on Reliability, Vol. 53, No. 1, March 2004

[8] R. A. Maxion, "Masquerade Detection Using Enriched Command Lines", International Conference on Dependable Systems & Networks (DSN-03), pp. 5-14, San Francisco, California, June 2003. IEEE Computer Society Press, 2003.

[9] K. S. Killourhy, and R. A. Maxion, "Investigating a Possible Flaw in a Masquerade Detection System", Technical Report, School of Computing Science, Newcastle University, CS-TR N° 869, Nov 2004.

[10] B.K. Szymanski and Y. Zhang, "Recursive Data Mining for Masquerade Detection and Author Identification", 2004 Information Assurance Workshop, Proceedings of the 5th annual IEEE conference on Systems, Man and Cybernetics, 2004.

[11] S. Coull, J. Branch, B. Szymanski, and E. Breimer, "Intrusion Detection: A Bioinformatics Approach", Proceedings of the 19th Annual Computer Security Applications Conference, 2003.

[12] K. H. Yung, "Using Self-Consistent Naïve-Bayes to Detect Masqueraders", PAKDD 2004, pp 329-340.

[13] M. Oka, Y. Oyama, and K. Kato, "Eigen Co-occurrence Matrix Method for Masquerade Detection", Publications of the Japan Society for Software Science and Technology, 2004.

[14] M. Oka, Y. Oyama, H. Abe, and K. Kato, "Anomaly Detection Using Layered Networks Based on Eigen Co-occurrence Matrix", Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection, 2004.

[15] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Self for Unix Processes", Proceedings of 1996 IEEE Symposium on Computer Security and Privacy, 1996.

[16] B. D. Davison and H. Hirsh, "Toward An Adaptive Command Line Interface", Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI97), Elsevier Science Publishers, 1997.

[17] H. S. Teng, K. Chen, and S. C-Y Lu, "Adaptive real-time anomaly detection using inductively generated sequential patterns", Proceedings of the 1990 IEEE Symposium on Security and Privacy, 1990.

[18] B. Scholkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution",

Technical Report, Microsoft Research, MSR-TR-99-87, 1999.
[19] http://www.csie.ntu.edu.tw/~cjlin/libsvm/