Detect Masqueraders Using UNIX Command Sequences

Peng Jia PENGJ@CS.CMU.EDU
Center for Automated Learning and Discovery, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Project Advisor: Roy A. Maxion MAXION@CS.CMU.EDU
Center for Automated Learning and Discovery, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

Masqueraders are people who impersonate other people on a computer system and they pose threat to the system security. This paper reported two experiments on masquerader detection using UNIX command sequences. One experiment detects masquerader using normal user activity variation measured by distance between two adjacent sequences' probability distributions. The other method use Naive Bayes classifiers and detect masqueraders by looking at the classifiers misclassification behavior. Both methods use command occurrence probability distribution modeling the UNIX sequence. The Naïve Bayes classifier shows better detection performance measured by missing alarm rate and false alarm rate, this method was analyzed in more detail in user self-recognition analysis. The results show that Naïve Bayes classifier behaves differently for various user command generating patterns.

1. Introduction

1.1 Problem Definition

Timely detection of computer system intrusion is a problem that is receiving increasing attention. While firewalls and access control are being used to keep intruder from breaking into a system, 80% of intrusions or attacks are from within an organization (Cerias website, 2001). Compare with outside intruders, these insiders know the system structure and where to find valuable information, thus they potentially pose bigger threat to the system security. However, no technique can *prevent* legitimate user from abusing their rights in a computer system. The only solution to the problem is to equip a computer system with an effective intrusion detection mechanism.

There are two main approaches to build up the intrusion detection system. One is called misuse detection and the other is called anomaly detection. In misuse detection, one tries to detect intrusion by recognizing well defined attacks patterns in user's data. In anomaly detection, usually a user profile is first built using the user's normal activity data. Any significant deviation of current user data pattern from the profile indicates possible intrusion. Since masquerader behaviors are highly unpredictable, the second path is often used in masquerader detection.

While a variety of information such as CPU usage, I/O usage, command names and options, timestamps, etc can be obtained from a system auditing mechanism, lots researches focused on anomaly detection using only UNIX command sequences, where the only information a detector has command sequences a user generated.

This paper reports work on masquerader detection using UNIX command sequences.

1.2 Prior work and our work

Researchers from both statistical field and computer sciences field have implemented various methods to solve the problem of masquerader detection using UNIX command sequence. Their work is reviewed here:

Schonlau & Dumouchel (working paper), Schonlau & Theus (2000) and have tried method called *uniqueness*. This method based its judgment on the intuition that commands not previously seen in training data is a good indicator of intrusion in the testing data, where the rarer the command has been seen in a community of users, the higher the indicating power of the command. They designed a testing score that increases with the number of unpopular commands a user used in the testing data but rarely or not used in the training data. A higher score trigger the alarm of possible intrusion. Although this method is based on a simple intuition, it obtained good results compare with other more complex detectors on the same set of data. This implies that frequency property of the command sequence is an important indicator of intrusion.

Not as uniqueness method that only considered command frequencies, Bayes First-order Markov model by Dumouchel & Schonlau (working paper) and Ye, Li, chen, Emran and Xu (in press) utilized the information of

1-step command transition probabilities. Transition matrices were built for each user's training data and testing data and the detector trigger the alarm when there is considerable difference between the training data transition matrix and testing data matrix. This method has comparable alarming accuracy as that of uniqueness method. Thus it discloses the importance of the order property of command sequence while only 2-order sequences are considered.

A higher order Markov model (Ju & Vardi, 1999) although has the potential to be a more accurate detector, is being avoided in anomaly detection because of its intimidating computational cost. A more efficient method proposed by (Schonlau, Dumouchel, Ju Karr et al, working paper) is a hybrid model that combines higher-order model with an independence model. The higher-order model is used when user's testing data does not have too much unobserved data. It focuses on a subset of the most used commands and thus only need to deal with a significantly reduced dimensionality. The independence model is used when the testing data contain many commands unobserved in the training data. Although this model is more complex and considered more characteristics of the data, it does not achieve higher detection accuracy. In fact, it is even inferior to uniqueness model and first-order Markov model on the same set of testing data.

Sequence Matching is a method proposed by Lane & Brodley (1997). A similarity measure is used in comparison of a command sequence with sequences in a user profile. A similarity score lower than a threshold will trigger the alarm. This method, when tested using the same data used in (Schonlau et al .working paper a) and (DuMouchel, working paper), generated accuracy result no better than the above methods.

In Incremental Probabilistic Action Modeling (IPAM) of Davison & Hirsh (1998), a 1-step command transition probabilities matrix is used to predict the new command given the new command's previous command. Too many bad predictions indicate a possible masquerader intrusion. This method has similar accuracy as sequence matching.

Other major methods mentioned in anomaly detection literature include: decision tree (Ye et al In press) neural network (Lunt 1993) expert system (Lunt 1993), Hotelling's T^2 test, Chi-square multivariate test (Ye et al In press) and compression (Welch 1984) method. All the detection methods, except for compression, have the common characteristic that they do not utilize very complex mathematical measures. Instead, simple statistics such as frequencies of the commands and 2-gram probabilities are used.

However, it is hard to decode the information of human behavior from UNIX command sequences. Beside, there are environmental regularities (Maxion & Tan 2000) that also encoded in the sequences. Schonlau et al (working paper a) mentioned that any single method above is not sufficient to do the work. It is necessary to look for other methods that look at the data from different perspectives.

Different from all previous method, the two methods used in this project attempts to look at user's command probability distribution. The first method also attempts to quantify normal human activity variation and utilized this information for anomaly detection.

For the first method, we set a threshold that describes how widely a user's normal behavior varies over time. One threshold was set for each user by getting the average probability density distances between each pair of adjacent command sequences. We detect masquerader by comparing current user variation with this threshold and trigger the alarm when a sequence of commands shows a big deviation from the user normal behavior.

For the second method, we assume each user's UNIX command sequences follows multinomial distribution. A set of multinomial distribution models were built, one for each user in the data set. Given a sequence of command, a Naïve Bayes classifier find the multinomial distribution which gives the testing sequence of commands the highest probability and assign the distribution's user as the testing sequence's user. If the predicted user and the actual user of the testing block are not the same, then the testing block is judged to be a suspicious block

The second method showed much better detecting performance than the previous approach, although it has a false alarm rate that is very high. Following this result, we go further in analyzing how multinomial modeling of user's command sequence and Naïve Bayes classifiers perform in user's self-recognition. We trained the Naïve Bayes model on all the training data and tested the model on smaller chucks of command sequences from the training data to see whether the Naïve Bayes classifier can successfully recognize a "mini-self". Our results show that different users have different levels of self-recognition performance. Users have better performance of self-recognition also have better performance in masquerader detection.

2. Approach

2.1 Data

The data used was collected on a UNIX system by the system's auditing mechanism. It includes 15,000 UNIX command sequences, one sequence for each user being investigated. The data is semi-synthetic: Initially, about 70 users' activity data were collected on a real UNIX system. 50 of these users were selected randomly as the intrusion targets while the rest users were left as the potential masqueraders and these 20 user's data were inserted at random to the target user's command sequence to synthesize masquerader data. The first 5,000 commands of each potential target have not been contaminated and are purely the corresponding actual

user's normal activity data, whereas the left 10,000 has been divided into blocks (each has length of 100 commands) and some of the blocks has been replaced with potential masquerader's command block at random.

The task here is: Use the first 5,000 commands of each potential intrusion target as the training data to build a profile for the user. Test each blocks in the rest 10,000 commands and trigger the alarm when suspicious blocks are found, where a significant deviation of a user's current behavior from his/her profile will trigger the alarm for possible masquerader intrusion.

2.2 Approach:

2.2.1 MASQUERADER DETECTION USING DISTANCE MEASURE

The intuition behind this detection approach is: a user uses a relatively fixed set of commands. Given a command sequence accumulated during a specific time period, the proportion each of those commands occurred is also user specific. Therefore, we can use a vector of attributes representing the proportional occurrences of each command in a sequences and we expect these proportional occurrences vector can describe a user's behavior during a specific time. A user's behavior is not fixed but keeps changing by the user's adaptive behavior. We can measure this user behavior variation by looking at the distance between two proportional occurrence vectors. Thus, we can break a user's command sequence into blocks in some natural way, i.e. by the day the command was generated. We can build proportional occurrence vectors for those blocks and learn the user's behavior adaptation rate by looking at how fast or how widely the distance between those vectors change. When a masquerader intrude a system, he/she has high probability of failing to imitate a normal user' variation rate because he/she has high probability to use set of commands that are unfamiliar to the actual user. This variation in command usage pattern may bring big change to the block's representing vector. Thus a larger distance from a user's normal block vector may occur. Therefore, we can summarized a user's normal behavior variation as a threshold by looking at how widely the average distance change. Whenever a distance larger than the threshold probability distance occurs, we may set off the alarm for suspicious masquerader intrusion. Following is the procedure of the experiment:

1. Divide the training data into blocks. Each block has 100 commands.[1]
2. Build a vector for each of these blocks. The length of each vector is the "vocabulary" size of the data (how many commands appears in the data, in the data set we are currently using, this is 856). The elements of the vector are commands occurrences or the probability each command occurs in the block.
3. The distance between each pair of adjacent blocks was calculated. Dot product, Euclidean distance as well as a distance measure of probability vector borrowed from information retrieval field were tried here for distance computation.

For training data, after all the distances between each adjacent pair of blocks were calculated, a threshold value based on this distances measures was computed to represent user's activity variation. Three kinds of threshold values have been used in the initial experiments: a. the maximum block distance within training data; b. the average block distance within training data; c. exponentially weighted moving average. Each of these three threshold values was combined with each of those three distance measures. Therefore, 9 possible combinations were used in the initial implementation.

4. For each block in the testing data, we compute its distance to the previous *normal* user block. If the distance is above the threshold, the intrusion alarm will be triggered.
5. If there are 3 blocks in a row are judged as normal by the algorithm, the distance between the center block and the third block will be used for updating the threshold. If the fourth block is still normal, the distance between it and the third block will be used for updating the threshold. Intuitively, this is for updating the system's knowledge of user's behavior variation.

The implementation

In the implementation part, we first compute the probability distribution vector or command occurrence vector for each block. Prior research has discovered that detecting based on frequency alone provides rather good performance with relatively low computational cost. Thus, this project plans to use frequency or probability distribution of each command within a block as the component of the signature vector. The signature vector will take the following form

$$VF_b = (f_1, f_2, \ldots f_i \ldots f_n) \text{ or } VP_b = (p_1, p_2, \ldots p_i \ldots p_n)$$

Where

b----the serious number of the block,

n-----the total number of commands of the corresponding UNIX system,

f_i----the number of times command i presents in block b

p_i----the probability that command i occurs in block b.

[1] But clearly, there are more meaningful ways to divide the commands into blocks. e.x. by the day a command was generated or by the session a command was generated. But our experiment data's characteristic make this segmentation is the only way we can choose. Further consideration about data segmentation is discussed in the discussion section of this paper.

VF_{b}---the frequency vector for block *b*

VF_b ----the probability vector for block *b*

Σf_i = *block size*

$\Sigma p_i = 1$

a. Euclidean distance

$$d(VF_a - VF_b) = (\sum_{i=1}^{n} (f_{ai} - f_{bi})^2)^{1/2}$$

b. Cosine of vectors of frequencies (doc product)

$$d(VF_a - VF_b) = \frac{\sum_{i=1}^{n} (f_{ai} f_{bi})}{\sum_{i=1}^{n} (f_{ai})^2 \sum_{i=1}^{n} (f_{bi})^2}$$

This distance is identical to the correlation coefficient of the two sequences of frequencies.

c. Distance between probability vectors

$$d(VP_a, VP_b) = H(\frac{1}{2}VP_a + \frac{1}{2}VP_b) - \frac{1}{2}[H(VP_a) + H(VP_b)]$$

$$= \sum_{i=1}^{n} \frac{p_{ai} + p_{bi}}{2} \log \frac{p_{ai} + p_{bi}}{2} + \frac{1}{2}\sum_{i=1}^{n} p_{ai} \log p_{ai} + \frac{1}{2}\sum_{i=1}^{n} p_{bi} \log p_{bi}$$

where H(.) is the information entropy function.

2.2.2 MASQUERADER DETECTION USING NAÏVE BAYES CLASSIFIER

Naïve Bayes model has been used widely in text classification, medical diagnosis, and computer performance management. A naïve Bayes classifier in essence builds density functions for each class that are marginally independent, and then classifies a data point based on which density function has the maximum value at the point. Since text classification using naïve Bayes classifier is familiar to many people, we list some mappings in table 1. for better understanding of our approach:

The command occurrences of a block of command are assumed to follow a multinomial distribution. Using *u* to represent user; *b* to represent block; *n* represent the total number of different commands all users used in the training data and N_{it} as the number of command *t* appears in block *i*, the probability of a block of data given its user is:

$$p(b_i \mid u_j; \theta) = p(|b_i|) \mid b_i \mid ! \prod_{t=1}^{|n|} \frac{p(c_j; \theta)^{N_{it}}}{N_{it}!}$$

We therefore need to use the training data to estimate the parameters for the multinomial distribution in order to build density functions for each class. The parameters are

Table. 1 mapping of masquerader detection to text classification using Naïve bayes classifiers

Term	**Text classification**	**Masquerader detection**
Training data	Set of labeled documents	The command sequences generated by each user's normal activities
Testing data	Set of un-labeled document wait to be classified	Blocks of command sequences in the last 10000 commands that have been contaminated.
Prediction	for each documents in the testing set, predict the probabilities that this document comes from every possible class, and chose the class with the highest probability as the document's class	For each block in the testing data, compute the probabilities that a block was generated by every possible user and choose the user with the highest probability as the generator of the block. If this prediction is different from where the user comes from , then the alarm will be set off.

estimated in the following way (McCallum & Nigam, 1998):

$$\hat{\theta}_{c_t|u_j} = P(c_t \mid u_j, \hat{\theta}_j) = \frac{1 + \sum_{i=1}^{|B|} N_{it} P(u_j \mid b_i)}{n + \sum_{s=1}^{|n|} \sum_{i=1}^{|B|} N_{is} P(u_j \mid b_i)}$$

Therefore, after training we had a set of multinomial density functions, each represents one user.

Given a new block b_{new}, we use Bayes rule to first predict the probabilities that each possible user being the generator of the block.

$$p(u_j \mid b_{new;} \hat{\theta}) = \frac{p(u_j \mid \hat{\theta}) p(b_i \mid u_j; \hat{\theta}_j)}{p(b_i \mid \hat{\theta})}$$

we then chose the u_j with the highest $p(u_j \mid b_{new;} \hat{\theta})$, as the predicted generator of $b_{new.}$ If the actual user of the block is the predicted user, then the block is normal. If actual user of the block is not the predicted block, then the block will be flagged as a suspicious block.

3. Results

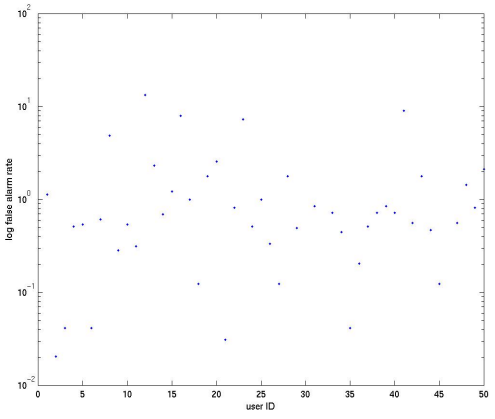
The results are summarized into two parts. The first part reports the experimental results on masquerader detection. We find that Naïve bayes model has different performance among users and the difference is large. So we conducted self-recognition test to try to find the reason of the difference. The result is reported in section 3.2.

3.1 Result of Masquerader Detection

The performance of masquerader detection was measured by missing alarm rate (incorrect negative/number of positive) and false alarm rate (incorrect positive/sum of correct positive and correct negative).

For the first method, the average missing rate is 59%. The average false alarm rate is 194%, which are not good. Compare with previous methods' performance[2], the result of Naïve Bayes classifier is not quite good, but it is much better than the first method's result. Naïve Bayes detected most of the masquerader blocks but the false alarm rate (67%) is still too high.

Graph. 1 Masquerader detection performance of Naïve Bayes classifier is different among users



When we checked the performance for each individual user based on their false alarm rate[3], we found the performance differs a lot among users. For some user, Naïve bayes classifier performs 100% correct (i.e. user 30, 32, 46), for some other users, the classifier fails to do the work for almost every block (i.e. 12, 41) being tested. Among 50 users, 8 users have false alarm rate lower than 5%. Performances of 36 users are better than random guessing. Graph 1 shows the log false alarm rate vs. user ID.

[2] Their false alarm rate ranged from 3.7% to 6.7%, and missing alarm rate ranges from 30.7% to 65.8%. Naïve Bayes has false alarm rate of 67% and missing alarm rate of 0.86%.

[3] The missing alarm rate is almost the same for all users—since most of the masquerader blocks have been detected.

3.2 Result of user self-recognition using Naïve Bayes Classifiers

Although the results on masquerader detection are not good, the difference of performance among users nevertheless gives us hint that we should check each user's patterns separately to find the reason of why Naïve Bayes classifier performs differently. Naïve Bayes classifier outputs a set of relative probabilities of how likely a block was generated by each of those 50 users being studied. The first ranked user is the predicted user of the block. It is intuitive to us that there must be some similarity between the predicted user's training data and the testing data block. For any user *k* in the data set, if every testing data block of *k* (except those randomly inserted masquerader blocks) is most similar to *k*'s training data, then *k* must have a very low false alarm rate.

To verify this, we did a self-recognition test within training data to see whether users who yield better performance in self-recognition also do better then user who yield worse performance in self-recognition.

We use the training results from last section. They are: 50 multinomial distribution functions, each describes one user.

To test user self-recognition, a sliding window on the *training* data is used for getting blocks of user normal data. Each of these blocks was fed into the Naïve Bayes classifier to get prediction on the generator of the block. If the classifier predicts a block's generator as j and the chuck was actually taken from user j's data, then we say the classifier successfully recognized itself for this block. We had tried block size of 100, 125, 150, 200, 250 and 500. We ranked the user in descending order according to how successful each user can recognize itself using the Naïve Bayes classifier. We also ranked the user in ascending order by each user's false alarm rate in masquerader detection experiment. The results showing that there are positive relations between self-recognition and masquerader detection: about 70% percent of users who ranked high in self-recognition test also ranked high in ma

squerader detection test.

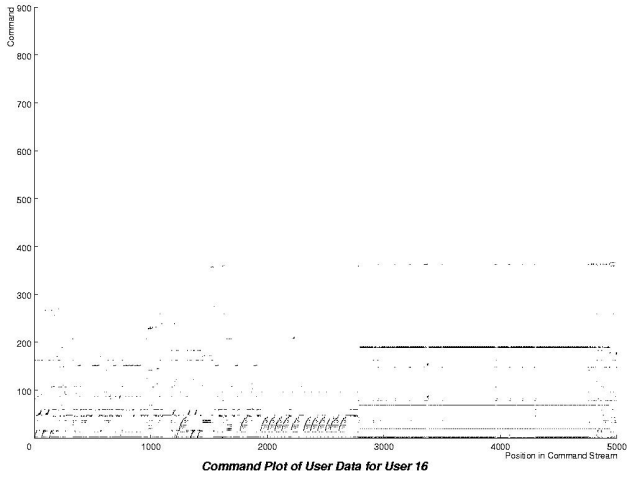
By increasing the block size, the average success rate[4] increases from 76% to 92%. The result also shows some similarity among users: Many users were being classified as user 5, user21, user18 and user38 even when the window size increase to 500. User 5, 21, 18 and 38 can recognize itself most of the time, but they seems created difficulty to other user's self-recognition.

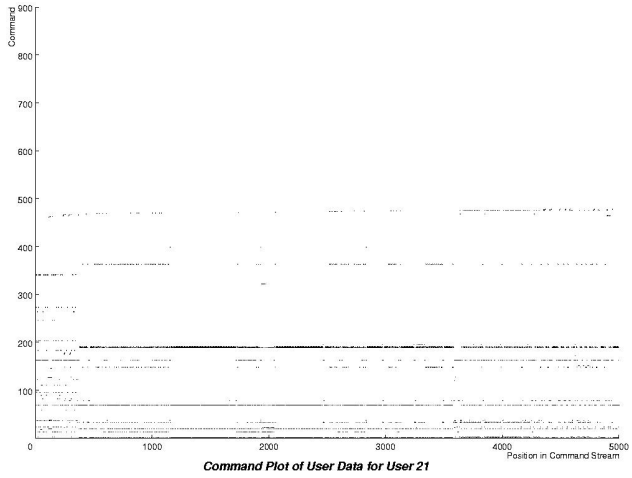
[4] Number of blocks being successfully recognized divided by number of blocks being tested.

Therefore, these users were taken out and the masquerader detection experiment was conducted again on the rest users. This time, the false alarm rate decreases from 67% to 59%, while the missing alarm rate's keeps almost the same. (8.6% to 9.2%). This suggests that in order to increase the masquerader performance, we may first select users according to their self-recognition performance and apply Naïve Bayes classifier to chosen users only.

4. Conclusion and Discussion

Graph. 2 scatter plots of user 21 and user 16—two extreme cases.





We applied two methods to masquerader detection and the result is not very good in both case. However, for the second method, the difference of the classifier's performance among users leads us to analyze why Naïve Bayes works better on some user than on other user. We found out that, in general, users who are more successful in Naïve Bayes's self-recognition experiment can achieve better results in masquerader detection experiments. This is consistent with our intuition. We than take out those users who create difficulty to other user's self-recognition and we observe that the masquerader detection on the rest user has improved performance.

In self-recognition experiment, we also found some users (i.e. user 16) yield bad performance in self-recognition no matter how large is the window size. There are also some users (i.e. 21) who perform well no matter how small is the window size. When we compare these user's scatter plots (see graph 2) , it shows that the former kind of user usually shows abrupt change in their command usage pattern, where the second kind of users' pattern doesn't change much over time. How to summarize these characteristic numerically so that we may set up a threshold for applying Naïve Bayes classifier. This question has been investigated, but no encouraging results has been yielded so far. This is the topic we plan to put more effort in the future.

Rish, Hellerstein & Thathachar (2001) had pointed out that Naïve Bayes classifier works best in two situations: 1. data point 's attributes are independent (that's Naïve Bayes classifier's assumption). 2. data point's attributes are functionally dependent. They also found out that as the entropy of class-conditional feature distributions decreases, the performance of Naïve Bayes model gets improved. Their work is based on simulated data and they suggest using practical application data for doing the analysis again. Given the fact that some of our users' data yield almost perfect performance of Naïve bayes and some user just yield the opposite consequence, it will be interesting to see whether their findings work on our data. We also want to do a further investigation of data characteristics that yield a good performance of Naïve Bayes classifier.

Acknowledgements

I thank Sebastian Thrun for pointing out generative modeling method to me; I Thank Rayid Ghahi for introducing Rainbow (Rainbow Link, 2001) to me. Also want to thank Marcus Louie in visualizing each user's command sequence so that better understanding of the data is possible.

References

M.Schonlau, W.DuMouchel, W-H. Ju, A.F. Karr. M.Theus and Y. Vardi. Computer Intrusion: Detecting Masquerades. *Working paper*[a.]

W. DuMouchel and M. Schonlau. A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities. *Working paper.*

B.D. Davison and H. Hirsh. Predicting sequences of user actions. Presented at the AAAI-98/ICML'98 Workshop

on Predicting the Future: AI Approaches to Time Series Analysis, Madison, WI, July 27, 1998 and published in Predicting the Future: *AI Approaches to Time Series Problems, Technical Report WS-98-07, pp. 5-12, AAAI Press*

Rish, I., Hellerstein J. & Thathachar Jayram. (2001) An Analysis of Data Characteristics that Affect Naïve Bayes Performance. *IBM T.J. Watson Research Center. ICML01 submission.*

W. DuMouchel. Computer intrusion detection based on Bayes factors for comparing command transition probabilities. ***Working paper.***

T. Lane and C.E. Brodley. Sequence matching and learning in anomaly detection for computer security. *In Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management (1997), pp. 43--49.*

M. Schonlau and M. Theus. Detecting Masquerades in intrusion detection based on unpopular commands. *Preprint submitted to information processing letters. 21 June. 2000*

N. Ye, X.Y. Li, Q. Chen, S.M. Emran and M.M. Xu. Probabilistic techniques for intrusion detection based on computer audit data. *In press, IEEE Transactions on system, man and cybernetics.*

R.A. Maxion and K.M.C. Tan. Benchmarking anomaly-based detection systems. *1st international conference on dependable systems & networks: New York, NY 25-28 June 2000.*

S.D.M Wong and Y.Y. Yao. A statistical similarity measure. Proceedings of the tenth annual international ACM SIGIR conference on Research and development in information retrieval June 3 - 5, 1987, New Orleans, LA USA

Dorothy E. Denning. An intrusion-detection model. IEEE Transactions on Software Engineering, *13(2):222-232, February 1987*

T. F. Lunt. A Survey of Intrusion Detection Techniques. Computers & Security, 12 (1993) 405-418

Ju, W., Vardi Y.(1999), A Hybrid High-order Markov Chain Model for Computer Intrusion Detection. National institute of statistical Sciences, Technical Report No. 92. Available at: http://www.niss.org/downloadabletechreports.html

Cerias Website http://www.cerias.purdue.edu/coast/intrusion-detection/introduction.html

Eskin E. (2000). Anomaly Detection over Noisy Data Using Learned Probability Distribution. In *Proceedings of the 17th International Conference on Machine Learning(ICML-2000).*

Javitz H.S. & Valdes A. (1994). The NIDES Statistical Component Description and Justification. *Annual Report A010. SRI project.*

Maloof, M. A., Langley, P., Binford, T. O., & Sage, S. (1998). *Improving rooftop detection in aerial images through machine learning* (Technical Report 98-1). Institute for the Study of Learning and Expertise, Palo Alto, CA.

Shrager, J., & Langley, P. (Eds.). (1990). *Computational models of scientific discovery and theory formation.* San Francisco: Morgan Kaufmann.

Veloso, M. M., & Carbonell, J. G. (1993). Toward scaling up machine learning: A case study with derivational analogy in PRODIGY. In S. Minton (Ed.), *Machine learning methods for planning. San Francisco:* Morgan Kaufmann.

M. Theus and M. Schonlau. Intrusion Detection based on Structural Zeroes. Working paper

A. McCallum and K. Nigam. (1998) A Comparison of Event Models for naïve Bayes Test Classification. In AAAI-98 Workshop on Learning for Text Categorization.

Rainbow Link: (2001). http://www.cs.cmu.edu/~mccallum/bow/rainbow/

APPENDIX A.

Misclassification matrix for window length equals to 100. The row is actual user and the column is predicted user. Each cell tells us the probability that a actual user was classified as the predicted user among all blocks being tested.