Proc. 5th IEEE System, Man and Cybernetics Information Assurance Workshop, West Point, NY, June, 2004, pp. 424-431

Recursive Data Mining for Masquerade Detection and Author Identification

Boleslaw K. Szymanski, *IEEE Fellow*, and Yongqiang Zhang
Department of Computer Science, RPI, Troy, NY 12180, USA

Abstract- In this paper, a novel recursive data mining method based on the simple but powerful model of cognition called a conceptor is introduced and applied to computer security. The method recursively mines a string of symbols by finding frequent patterns, encoding them with unique symbols and rewriting the string using this new coding. We apply this technique to two related but important problems in computer security: (i) masquerade detection to prevent a security attack in which an intruder impersonates a legitimate user to gain access to the resources, and (ii) author identification, in which anonymous or disputed computer session needs to be attributed to one of a set of potential authors. Many methods based on automata theory, Hidden Markov Models, Bayesian models or even matching algorithms from bioinformatics have been proposed to solve the masquerading detection problem but less work has been done on the author identification. We used recursive data mining to characterize the structure and high-level symbols in user signatures and the monitored sessions. We used one-class SVM to measure the similarity of these two characterizations. We applied weighting prediction scheme to author identification. On the SEA dataset that we used in our experiments, the results were very promising.

Keywords- Masquerade detection, author identification, recursive data mining, one-class SVM, intrusion detection

I. INTRODUCTION

This paper focuses on two related and important topics in system security: masquerade detection and author identification.

Masquerade detection is often considered the most serious and challenging problem in computer security. Masquerader hides his/her identity by impersonating a legitimate user in a computer system or network and may maliciously damage the system. The typical ways in which masquerade attacks succeed include: obtaining a legitimate user's password, accessing an unattended and unlocked workstation, forging email address in messages, overtaking a computer via a network access. Masquerade detection is challenging for the following reasons: (i) masqueraders entering the system as valid users cannot be detected by the existing access control or authentication, (ii) by perfectly mimicking user's behavior, masqueraders are undetectable, and (iii) the legitimate user may be detected as a masquerader if the user's behavior changes.

To enable masquerade detection, a string from a legitimate user is collected and used to generate a signature containing some attributes (features) of this user. This signature is then compared to the attributes generated from the currently monitored string of the potential masqueraders. If normal and intrusion activities are sufficiently distinct, attributes generated from the legitimate user activities will be more similar to the user's signature than those generated from the masquerader's session. Most previous research follows this logic to distinguish the strings from legitimate users and masqueraders.

A related problem to masquerade detection is identifying the potential internal masqueraders, which can be generalized as an *author identification problem*. This problem is relevant in secured environments, in which only a small number of users with known signatures can originate an attack. Other examples of usefulness of the author identification problem include finding equivalences between emails originated from differently named accounts or detecting plagiarizing among papers or programs. For author identification, a string from each potential author is collected to generate a signature (some attributes or features). Each signature will then be compared to the attributes of the currently monitored string. The author is then decided based on the degree of similarity of the current session and the author signature.

In masquerade detection and author identification, the input is a string of objects (commands, packets, system calls, lines of program execution trace, etc.) produced by a source. The task is to assess whether the monitored string confirms to the "usual" behavior of this source, in case of intrusion detection, or which of many possible sources is the most likely producer of the monitored string, in the authorship identification case. The assessment is based on the unique signature of each source collected in the controlled experiment in which the authorship of the signature can be assured.

In this paper, we propose a novel recursive data mining method originated in our simple yet powerful model of cognition, called a *conceptor* [5]. In a conceptor, the first level of abstraction of input is based on repeating patterns, but the subsequent ones are based on repeating patterns of lower level abstractions. In recursive data mining method, the input is encoded into symbols and then mined for dominant patterns. Dominant patterns, defined later, are then assigned new codes that are then used to generate the data representation of a higher level. The input is thus recursively mined until no new dominant pattern exists. During the recursive mining, we generate several features that will be used to form the user's signature profile and will be compared to the features of monitored string. For masquerade detection, one-class SVM is applied to predict the intrusion. For author identification, a weighting prediction scheme is proposed to predict the authorship.

This paper is structured as follows: first, we discuss the recent works in Section II, and then illustrate in detail our recursive data mining method in Section III. Section IV describes the user identification algorithm. The results of the experiments with masquerade detection and author identification are presented in Section V. Section VI concludes with a discussion of future work.

II. RECENT WORK

A. Data

Schonlau et al. [12] collected UNIX commands and built a truncated command dataset, commonly called *SEA* dataset. In *SEA* dataset, 15,000 sequential commands for each of 70 users are recorded. Among 70 users, 50 are randomly chosen as victims and the remaining 20 as intruders. Each victim has 5,000 clean commands and 10,000 potentially contaminated commands. For each user, the first 5,000 commands are taken as the training data and used for signature generation. The remaining 10,000 commands form the testing data. The data is grouped into blocks of 100 commands. A testing block is either contaminated completely or not at all.

Although widely used for masquerade detection, this dataset has some limitations [12]. First, command arguments, which may contain valuable information for intrusion detection and authorship, were not collected because of privacy concerns. Second, the 20 intruders are "other" users from a general population similar to the legitimate users and not "real" masqueraders who may exhibit unusual patterns distinguished more easily from normal patterns. Third, mistyped commands may violate the signature profile built for a user. Still, *SEA* data is a valuable set and we used it to test our method for both masquerade detection and author identification.

B. Methods

Many methods have been used for masquerade detection in the *SEA* dataset. Schonlau et al [12] summarizes several approaches based on pattern uniqueness, Bayes one-step Markov model, hybrid multistep Markov model, text compression, Incremental Probabilistic Action Modeling (IPAM), and sequence matching (see Table 1, the method names follow Schonlau [12]). Wang [17] used one-class training based on data representative of only one user and demonstrated that it worked as well as multi-class training. Coull [3] applied bioinformatics matching algorithm for a semi-global alignment to this problem. Finally, Lee [9] built a data mining framework for constructing features and model for intrusion detection.

Approaches	False Alarms	Hit Rates
Uniqueness	1.4%	39.4%
Bayes one-step Markov	6.7%	69.3%
Hybrid Multistep Markov	3.2%	49.3%
Compression	5.0%	34.2%
Sequence Matching	3.7%	36.8%
IPAM	2.7%	41.4%
Semi-Global Alignment	7.7%	75.8%

Table 1: Results from previous approaches to masquerade detection

For author identification, Vel [15, 16] researched email communications, Hill [6] built a vector space model for author identification in a double-blinded review process by using citation lists, and Krsul [8] employed programming style characteristics to identify the author of a program. In this paper, we solve the masquerader identification problem.

Marcken [10] developed an unsupervised learning algorithm for speech segmentation that is also based on hierarchical pattern recognition. Unlike our method in which reduction of string length is based on the frequency and length of patterns, his framework used minimum description length theory (MDL) for grammar compression. Another big difference is that we use fixed length of slide window that limits the length of a pattern.

Data mining techniques have been successfully applied to many business and scientific domains, including some that are closely related to intrusion detection. Lee [9] builds data mining framework for constructing features and model for intrusion detection. In this paper, we propose a recursive data mining method to solve the masquerade detection and author identification problems.

III. RECURSIVE DATA MINING METHOD

Each user is prone to type commands which will create some patterns. For example, a user is likely to type *ls* first, then *cd*, then *ls* and so on, when searching for a file. The habit of each user in typing the sequence of commands may reflect

his identity and thus can be used as his signature. In *recursive data mining*, an input is first encoded into input symbols and then recursively mined for frequent patterns. The dominant patterns are the largest and most frequent patterns in the string. The dominant patterns are encoded by new unique symbols and the input is rewritten by replacing each dominant pattern with its symbol. The input data are considered the 1st level data representation, L-level data representation, after dominant pattern substitution, becomes L+1-level data representation. The process stops when no new dominant patterns are present in the transformed input. Since input is recursively mined for patterns, the pattern search can be done in each step in a limited-size sliding window, without changing the final representation of the result. Using a window speeds up the process of pattern recognition.

By recursive de-mining, a pattern in the L+1-level can be extended to a longer pattern if we replace its L-level symbols with their corresponding dominant patterns. In this way, we can restore the patterns in any level to their first level representations. We use the features on patterns in each level as a user's signature. So our method actually uses features for both short patterns and long patterns in the first level representation as a user's signature.

In our applications, we use patterns of fixed length and containing several commands and gaps. More formally, the input data contains M records, each with values of l attributes. Each attribute i has a domain D_i of all possible values associated with it (including a null value). Values are either discrete (countable) or non-empty intervals over continuous domains. The pattern of radius r anchored at record m is any subset of a matrix $M_{i,l}$ such that each row consists of values of l attributes in a record $m+i$ where $0 \le i < r$. For *SEA* dataset, we only consider one attribute, a user command, so l is 1. We also only consider patterns of radius of up to 6.

A. Encoding Phase

Each input command is assigned an integer as its code. However, a group of related commands may be assigned an identical integer, making them equivalent in recursive mining. Such encoding granularity impacts the frequency of patterns. In essence, coarse grain encoding "glues" together a range of original input values into a single symbol.

Command Types	Policy 1	Policy 2	Policy 3
Unix Commands	1~1000	1~1000	one symbol for commands with close semantics
Not Unix, Frequently Used	1001~2000	1001	1001
Not Unix, Not Frequently Used	2001~3000	2001	2001

Table 2: Different encoding of commands in *SEA* input

For *SEA* dataset, input contains user-typed commands. Most of them are UNIX commands, but there are of course mistyped commands too. We group the commands into three types: UNIX commands, non-Unix but frequently used commands, non-Unix and not frequently used commands. We experimented with the encoding policies presented in Table 2.

B. Training Phase

In training phase, the training data are recursively mined and the dominant patterns are recorded in the training dictionary. The procedure can be illustrated as follows:

First, we recognize the significant patterns in the input string. We also count the frequency of each pattern p in the input data, A_p. Then, we calculate the expected number of occurrences of pattern p in the input string E_p:

$$E_p = \frac{\frac{(N-k)!}{\prod_{i=1}^{q}(f_i - g_i)!} \times (N-s+1)}{\frac{N!}{\prod_{i=1}^{q} f_i!}} = (N-s+1) \times \frac{(N-k)!}{N!} \times \prod_{i=1}^{h} \frac{f_i!}{(f_i - g_i)!}$$

where f_i denotes command i's frequency in the string; g_i denotes command i's frequency in pattern p; k denotes the number of commands in pattern p; h denotes the number of distinct commands in pattern p; q denotes the total number of distinct commands in the input string; s denotes the length of a slide window; N denotes the length of the string. Let

$$N_p = \frac{(N-k)!}{\prod_{i=1}^{q}(f_i - g_i)!} \times (N-s+1), N_s = \frac{N!}{\prod_{i=1}^{q} f_i!}$$

Then N_p is the total number of strings in which pattern p may appear. N_s is the total number of strings that can be created by using the commands in the input string. $\frac{N_p}{N_s}$ is thus the expected frequency of pattern p.

If $A_p > m \times \max(E_p, 1)$, then pattern p is a significant pattern, where m is a constant whose choice impacts the number of dominant patterns recognized in the input.

Second, we recognize the dominant patterns by finding the longest significant pattern; if there are several candidates, the most compact one is selected. Each dominant pattern is assigned a new code.

Third, we rewrite the current string into a new string replacing each window with the code of the dominant pattern

followed by the commands in the gaps. The next replacement window starts at the first gap of the previous window in the current string. If the window contains no significant patterns, the next one starts at the symbol next to the start of the current window.

There are other choices for string rewriting that lead to different information kept in the higher level and, thus, will influence the training dictionary and the feature values that we will get in the following detection phase.

Finally, we append all the dominant patterns in the current level into the training dictionary. If there are no new dominant patterns in the current level, the training stops; otherwise, the rewritten string is used as an input for the next iteration of the data mining.

C. Detection Phase

In this phase, we apply training dictionary to each unit in testing data that are either a 100-command block (for intrusion detection) or the whole string (for author identification) and obtain features for each unit. We will also compute features for training data for the later prediction.

First, we search the training dictionary for each pattern in a unit and record the dominant patterns in this unit. Currently, a perfect match is required between the domain pattern in a dictionary and the substring in a unit. However, a similarity, such as small number of mismatches in corresponding position of the compared strings, or as measured by the Needlman-Wunsch algorithm [13], can also be used.

Second, we record features for each unit. We experimented with the following features: number of distinct patterns in each unit; number of dominant patterns in each unit; number of distinct dominant patterns in each unit; number of distinct input symbols in each unit; number of users sharing a pattern in each unit; length reduction in each step of the input rewriting in each unit; weighted number of the distinct dominant patterns in each unit (considering the frequency of the dominant patterns in the training dictionary).

Finally, we rewrite the current string, as in training phase.

D. Prediction Phase

For masquerade detection, we apply Support Vector Machine (SVM) [1, 4] to detect the intruded blocks. We train on the features of training blocks and predict on the features of testing blocks. Since for each user, we only have non-intruded data in the training blocks, we take it as the negative data and use one-class SVM [2, 14] to detect intrusion. We can also take the data in the training blocks of the current user as the negative data and the data in the training blocks of other users as the positive data and apply more traditional two-class SVM. However, as discussed by Wang [17], the positive training data belongs to different users and the positive testing data is from users other than the users in the training data, so such an approach is not appropriate.

Before applying SVM, we need to normalize the data. We apply two normalization methods. Suppose we have a dataset: $X = \{x_1, x_2,\ldots,x_n\}$, then each element will be normalized by the following two methods:

a. $x_i' = \dfrac{x_i - x_{avg}}{\sqrt{\sum_{i=1}^{n}(x_i - x_{avg})^2 / n}}$

where x_{avg} is the average of x_i, i=1, 2,…, n

b. $x_i' = \dfrac{x_i - x_{\min}}{x_{\max} - x_{\min}}$, where x_{min} and x_{max} are the minimum and maximum elements in the set, respectively.

For author identification, we will describe an algorithm to identify the authorship in section IV.

E. Analysis Phase

For masquerade detection, we compare the true value from *SEA* dataset with the value predicted in Prediction Phase. For any block, false positive rate is defined as the ratio of the number of false positive blocks to the number of clean blocks in the true value data; hit rate is defined as the ratio of the number of true positive blocks to the number of intruded blocks in the true value data.

For author identification, we will calculate the percentage of the number of authors that are correctly predicted.

IV. WEIGHTING PREDICTION SCHEME

After detecting the masquerade, it is important to know the identity of the masquerader. Masquerader identification can be generalized to a much broader problem: author identification. In the author identification problem, we assume to have *N* users with known signatures. Then, a sequence of symbols whose author is unknown is processed with the signature profile of each user to obtain *N* sets of feature values. The feature values are then weighted in each feature set. Our author decision rule is simple and straightforward: the user with the largest weighted feature value is the author. The feature values are then compared to decide the identity of the author of this unknown sequence of symbols.

A. Obtaining Features

Features are obtained by using our recursive data mining method and test it on *SEA* data but with all intrusion blocks removed from testing data. The Training Phase is the same as for masquerade detection. Detection Phase is different, as for each set of testing data we apply the training dictionaries

of all users, obtaining an *N by N* matrix R. Each element *R* (*I*, *J*) is a feature tuple which is obtained by applying user *I*'s training dictionary to user *J*'s testing data. The *N* feature sets for user *J*'s testing data which are obtained by applying each user's training dictionary to user *J*'s testing data form column *J*.

B. Weighting Prediction Scheme

For the decision of authorship, different features make different contributions. Assigning weight to each feature is necessary to quantify the amount of its contribution in deciding the author. Comparisons of features are performed by the following weighting prediction scheme.

First, we assign weight to each feature. Since we know the original author of each set of training data, we can use it to obtain the weight for each feature. We first split each user's training data into two parts: *t* for training set and *v* for validation set and then apply our recursive data mining method. As a result, we obtain an *N* by *N* matrix *T* of feature tuples.

To find the weight of each feature, for each column in *T*, we count the number of users whose value for this feature is bigger than or equal to that of the original user. According to author decision rule, the user with the largest weighted feature value is the author. Since we only consider one feature each time, so the weighted feature value is just the feature value itself. The users, who have the feature values bigger than or equal to that of the original user, are called *candidate authors*. We record two parameters: C_i, average number of candidate authors among the validation sets, and A_i, number of *wrong authors* (also the number of columns having more than one candidate authors among the validation sets). So if we attain a smaller number of wrong authors or candidate authors for one feature than the other, then this feature should be considered more relevant and is given larger weight in the combination of multiple features than the other one. We assign a weight W_i^c or W_i^w based on these two parameters, respectively, to each feature F_i, i=1,…,t according to the following formula [11]:

$$W_i^w = \frac{1}{A_i} \Big/ \sum_{j=1}^{t} \frac{1}{A_j}, \qquad i = 1,...,t$$

or $$W_i^c = \frac{1}{C_i - 1} \Big/ \sum_{j=1}^{t} \frac{1}{C_j - 1}, \qquad i = 1,...,t$$

Hence, the larger the number of wrong authors or the candidate authors, the smaller the weight. We use C_i -1 in the second weight calculation to remove the original author from consideration.

Second, for each feature F_i, i=1,…,t, if the element *R* (*I*, *J*) has the biggest value in its column, we set the value of predictor P_i to 1; otherwise, we set it to 0.

Third, we calculate the prediction value by linear combination of the weights and features. Let P (*I*, *J*) denote the weighted predictor value for each element *R* (*I*, *J*). Then weighted predictor value can be calculated by the formula:

$$P(I,J) = \sum_{i=1}^{t} (W_i \times P_i), \qquad I = 1,...,N; J = 1,...,N;$$

W_i is either W_i^c or W_i^w

Finally, for each column, we find the element with the highest weighted value. The corresponding row index in matrix *R* is the author of the testing string. That is, for all P(*I*, *J*) in column *J*, we find the maximum value of *P*(*I*, *J*) and the corresponding *I* is the author number.

V. EXPERIMENT RESULTS

A. Masquerade Detection

In masquerade detection, recursive data mining generates features for each block (including the training block and testing block) that is used by a one-class SVM to classify the blocks and predict the intrusion. We used LIBSVM2.4 [1] for this purpose. For the first variant of our method, called Individual Intrusion Detection, we selected four features for each block: number of dominant patterns in level 1, number of *distinct* dominant patterns in level 1, number of dominant patterns in level 2, and number of *distinct* dominant patterns in level 2. Changing the threshold in one-class SVM (a variable in LIBSVM2.4 that defines the outliers) yields different false positive and hit rates. Likewise, changing parameter *m* in recursive data mining impacts these rates. The best results are obtained by setting *m* to 6 for the first level of processing and 5 for the subsequent levels.

We also introduce an additional variant of our method, called Communal Intrusion Detection. Besides the features used in Individual Intrusion Detection, it also uses the number of users sharing each dominant pattern. That is, for each dominant pattern in a block, we count the users who share it.

There are two problems that need to be addressed for Communal Intrusion Detection. First, we train the training data by each user, so we may assign different codes to the same dominant pattern for different users. Thus, to compare the dominant patterns containing the newly assigned codes among the users, we need to transform the dominant patterns into their first level representations containing only the input symbols. Second, in one class SVM, we use fixed number of features. However, each block contains different number of dominant patterns and, thus, different number of potential features. To fix the number of features, we group the dominant patterns by the number of users who share them.

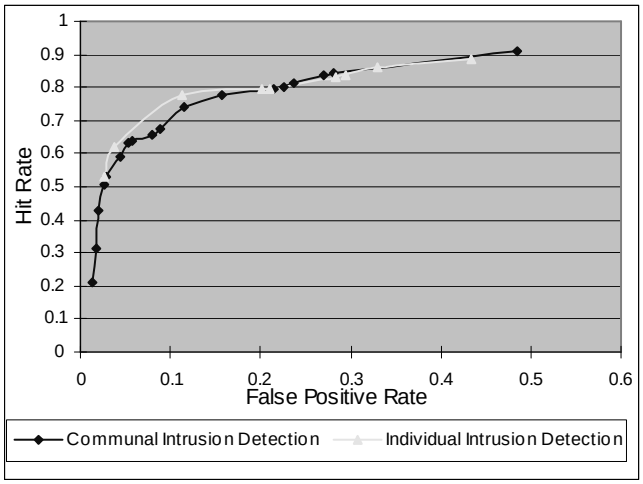

Figure 1: Comparison of Individual and Communal Intrusion Detection

For each dominant pattern *p* in a block, let u_p be the number of users who share it. Then if $2^{i-1} \le u_p < 2^i$, we put *p* into group *i*. Since we only have 50 users in *SEA* dataset, we have at most $\log_2 50 + 1 < 7$ groups. Then, the number of dominant patterns in each group is a feature. We use these features in the first two levels and thus have 18 features in Communal Intrusion Detection.

Figure 1 gives a comparison of Individual Intrusion Detection and Communal Intrusion Detection. The plot indicates that Communal Intrusion Detection works better than Individual one when large hit rate is desired. Individual Intrusion Detection uses fewer parameters, so it appears more stable.

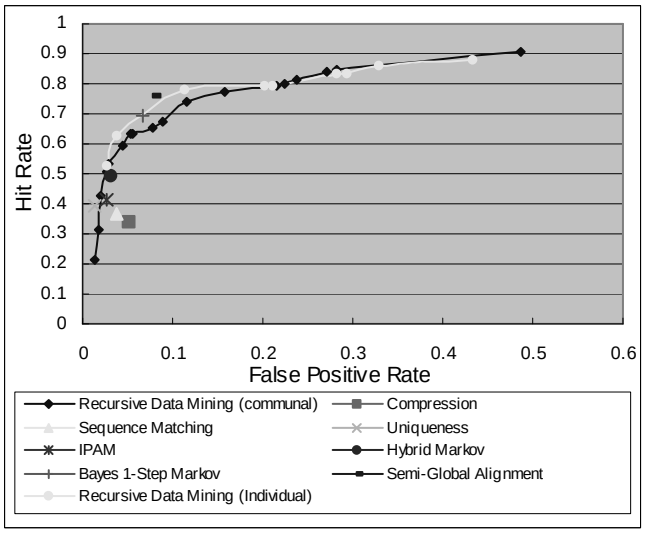

Figure 2: Comparison of Masquerade Detection Methods

Figure 2 compares different masquerade detection methods. Recursive Data Mining plots show both the Individual and Communal Intrusion Detection results.

The plot indicates that the recursive data mining works better than or matches Compression, Sequence Matching, IPAM, Hybrid Markov, Semi-Global Alignment, Bayes 1-Step Markov and Uniqueness methods (for details on those methods, see [12])..

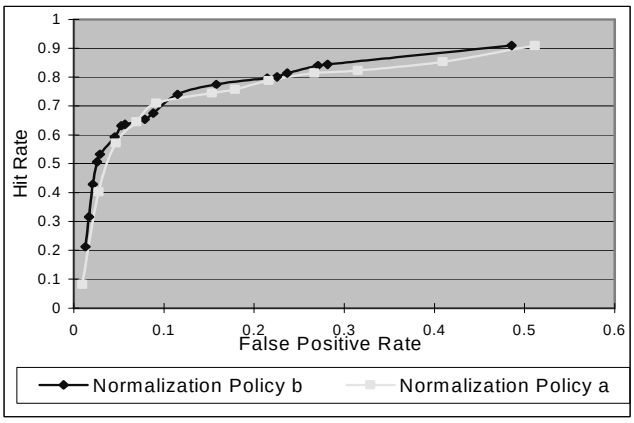

Figure 3: Comparison of two Normalization Policies

2. Comparison of Normalization Policies

Normalization of input for SVM is important for the SVM performance [4], so in Figure 3, we compare the two polices for normalizing the features for intrusion detection. The plots shows that normalization policy *b* works slightly better than policy *a* in most cases. However, the differences are very small.

3. Comparison of Encoding Policies

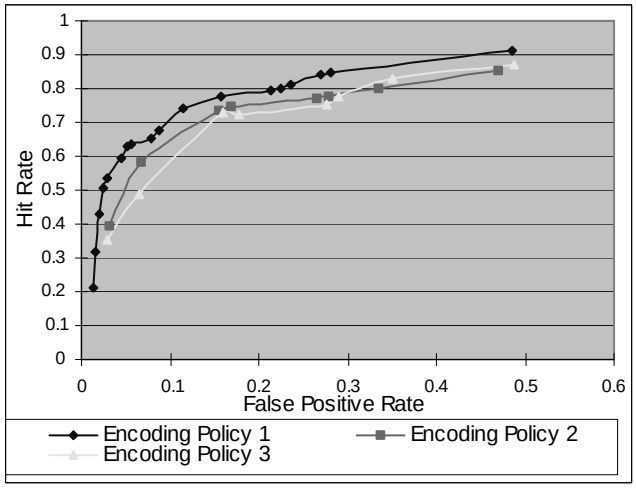

Figure 4: Comparison of two Encoding Policies

Figure 4 demonstrates that Policy 1 outperforms Policies 2 and 3. We hypothesize that a policy in which UNIX semantically similar commands are grouped together should work even better. So far, we have not found such a scheme for command grouping.

B. Author Identification

Individual Intrusion Detection was used to obtain features. Thus, we used the following four features: F_1: number of dominant patterns in level 1; F_2: number of distinct dominant patterns in level 1; F_3: number of dominant patterns in level 2; F_4: number of distinct dominant patterns in level 2. In recursive data mining processing, we choose m=3 for the first level and m=2 for the second level.

We used *SEA* dataset and split training data of each user into two sets: 2500 commands for training set and other 2500 commands for validation set. Then, we calculated the weights W_i^c and W_i^w as shown in Table 3.

	F_1	F_2	F_3	F_4
Average number of candidate authors(C_i)	1.18	1.04	4.1	1.28
Weight (W_i^c) for candidate authors	0.161	0.726	0.009	0.104
Wrong authors (A_i)	3	2	31	7
Weight (W_i^w) for wrong authors	0.330	0.496	0.032	0.142

Table 3: Weights for different features

Based on the W_i^c, we can achieve 88% accuracy rate. That is, for 50 test strings, we correctly find their 44 authors. Based on W_i^w, we can achieve 94% accuracy rate. That is, for 50 test strings, we correctly find their 47 authors.

By using the same dataset, roughly implemented bioinformatics method by Coull [3] gets only 40% accuracy rate, which means only 20 authors are correctly found among 50 users.

In the similar work, yet applied to the totally different domains, so direct comparison of the results is difficult, Krsul [8] reports 98% accuracy rate by using Multi-layer Perceptron and 100% accuracy rate by using Gaussian Classifier in deciding computer program authorship. Vel [15] achieves 84% accuracy rate in email authorship mining.

VI. CONCLUSION

Recursive data mining is a new technique for solving computer security problems. In this paper, we show how to apply recursive data mining to solve the masquerade intrusion detection and author identification problems. Compared to the results from other researchers, our results are very promising. In the future, we will attempt to extend our method to other authorship problems, such as email, papers or programs, or to problems in other domains, such as *DNA* identification and text mining problems.

What is different from previous work is that we use information about recursive patterns, instead of symbols or direct patterns, as features of input. The inputs are recursively mined and lower-level patterns are coded and then used for higher-level pattern recognition. The higher-level patterns seem to better capture the subconscious user's behavior than the symbols of the direct pattern do.

In this paper, we did not discuss feature selection. In our experiments, we manually select some features and use those that can lead to the best results. Sensitivity analysis and cross validation technique [7] will be our next step to improve those selections.

Finally, the experiments described in this paper all use perfect match when comparing two dominant patterns. We will compare the results with the processing in which matching will allow for small differences in the compared strings.

REFERENCES

[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] Yunqiang Chen, Xiang Zhou and Thomas Huang, "One-class SVM for Learning in Image Retrieval", *Proc. IEEE Int'l Conf. on Image Processing (ICIP'01 Oral)*, Thessaloniki, Greece, October, 2001.

[3] Scott Coull, Joel Branch and Boleslaw Szymanski et al., "Intrusion Detection: A Bioinformatics Approach", 19th Annual Computer Security Applications Conference, Las Vegas, Nevada, December 08-12, 2003.

[4] Nello Cristianini and John Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[5] Konrad R. Fiakowski & Boleslaw K Szymanski, "A CONCEPTOR: the Network of Neurally Connected Concept Nests for Knowledge Representation", Technical Report, RPI 1996.

[6] Shawndra Hill and Foster Provost, "The Myth of the Double-Blind Review? Author Identification Using Only Citations", *SIGKDD Explorations*, Volume 5, Issue 2 P179~184.

[7] Robert H. Kewley, Mark J. Embrechts, et al. "Data Strip Mining for the Virtual Design of Pharmaceuticals with Neural Networks", *IEEE Transactions on Neural Networks*, Volume 11, Issue 3, May 2000.

[8] Ivan Krsul, Eugene H. Spafford, "Authorship Analysis: Identifying the Author of a Program", Submitted to Computers and Security.

[9] Wenke Lee, “A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems”, PhD Thesis, Columbia University, 1999.

[10] Carl de Marcken, “The Unsupervised Acquisition of a Lexicon from Continuous Speech”, Technical Report A.I. Memo No. 1558, AI Lab., MIT. Cambridge, Massachusetts. 1995.

[11] Fang Qian, Mingjing Li and Hong-Jiang Zhang et al., “SVM Based Feature Weighting Method for Image Retrieval”, Fourth International Workshop On Multimedia Information Retrieval, Juan-les-Pins on the French Riviera, December, 2002.

[12] Matthias Schonlau, W. Dumouchel, Wen-Hua Ju et al. “Computer Intrusion: Detecting Masquerades”, *Statistical Science*, 16(1):58-74, February 2001.

[13] Needleman, S. and Wunsch, C. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, *Journal of Molecular Biology*, 48: 444-453, 1970.

[14] Runar Unnborsson, "Model Selection in One-Class ν-SVMs Using RBF Kernels", 2003, paper available at http://www.hi.is/~runson/svm/paper.pdf.

[15] Oliver De Vel, “Mining E-mail Authorship”, KDD-2000 Workshop on Text Mining, Boston, August, 2000.

[16] Oliver De Vel, A. Anderson and M. Corney, et al. “Mining Email Content for Author Identification Forensics”, SIGMOD: Special Section on Data Mining for Intrusion Detection and Threat Analysis, December 2001.

[17] Ke Wang and Salvatore J. Stolfo, “One-Class Training for Masquerade Detection”, 3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security, Florida, November 19, 2003.