Masquerade Detection Augmented With Error Analysis

Roy A. Maxion, *Member, IEEE,* and Tahlia N. Townsend

***Abstract*—A masquerade attack, in which one user impersonates another, may be one of the most serious forms of computer abuse. Automatic discovery of masqueraders is sometimes undertaken by detecting significant departures from normal user behavior, as represented by a user profile formed from system audit data.**

A major obstacle for this type of research is the difficulty in obtaining such system audit data, largely due to privacy concerns. An immense contribution in this regard has been made by Schonlau *et al.*, who have made available UNIX command-line data from 50+ users collected over a number of months. Most of the research in this area has made use of this dataset, so this paper takes as its point of departure the Schonlau *et al.* dataset and a recent series of experiments with this data framed by the same researchers [1].

In extending that work with a new classification algorithm, a 56% improvement in masquerade detection was achieved at a corresponding false-alarm rate of 1.3%. In addition, encouraging results were obtained at a more realistic sequence length of 10 commands (as opposed to sequences of 100 commands used by Schonlau *et al.*). A detailed error analysis, based on an alternative configuration of the same data, reveals a serious flaw in this type of data which hinders masquerade detection and indicates some steps that need to be taken to improve future results. The error analysis also demonstrates the insights that can be gained by inspecting decision errors, instead of concentrating only on decision successes.

***Index Terms*—Anomaly detection, classifiers, fraud detection, masquerade detection, Naive Bayes, user command data.**

ACRONYMS

1v49 = 1	versus 49 data configuration
ROC =	Receiver operating characteristic
SEA =	Schonlau *et al.* data configuration.

I. INTRODUCTION

COLLOQUIALLY, the masquerade problem can be described with the following scenario. A legitimate user takes a coffee break, leaving his/her terminal open and logged in. During the user's brief absence, an interloper assumes control of the keyboard, and enters commands, taking advantage of the legitimate user's privileges and access to programs and data. The interloper's commands may comprise read or write access to private data, acquisition of system privileges, installation of malicious software, etc. Because the interloper is impersonating a legitimate user, s/he is commonly known as a masquerader. The term may also be extended to encompass the case of abuse of legitimate privileges, that is, the case in which a user "masquerades" as him/herself; e.g., an insider.

The assumption underlying the common approach to detection of such illegitimate activity is that the masquerader's behavior (including the illegitimate behavior of a legitimate user) will deviate from the historical behavior of the legitimate user. User profiles can be constructed from monitored system-log or accounting-log data. Examples of the kinds of information that can be derived from these (and other) logs are: time of login, physical location of login, duration of user session, cumulative CPU time, particular programs executed, names of files accessed, and so forth [2]. This paper is focused on the latter attributes, because recent research in masquerade detection has been based largely on the Schonlau *et al.* dataset, which consists exclusively of the commands issued by users running the UNIX operating system, without arguments or other elements of the command line.

Masquerading can be a serious threat to the security of computer systems and the computational infrastructure. A well-known instance of masquerader/insider activity is the case of Robert P. Hanssen, the FBI mole who allegedly used agency computers to ferret out information later sold to his co-conspirators [3]. Hanssen's status as a user was legitimate, but his behavior was certainly not. More than 60% of companies surveyed in 1996 reported such misuse, at enormous cost. Attacks on national security are clearly even more of a concern than economic losses from attacks on business.

There have been several attempts to tackle the problem of detecting masqueraders. A nice collection of such work, in which a number of masquerade-detection techniques were applied to the same data set, is presented by Schonlau and his colleagues [1]. In terms of minimizing false alarms, the best result achieved in that work used a metric based on the uniqueness of commands to a user, obtaining a 39.4% hit rate with a corresponding 1.4% false alarm rate. In terms of detecting masqueraders, the best results reported in the Schonlau et al. work were for a Bayes one-step Markov model, with a 69.3% hit rate and a 6.7% false alarm rate.

This paper takes the work of Schonlau *et al.* as a point of departure. It uses the data provided by Schonlau *et al.*, and demonstrates a new technique for masquerade detection which yields a 56% improvement in correct detection at the lowest false alarm rate reported in the literature so far. Experiments were also conducted using a more realistic sequence length of just 10 commands, as opposed to the 100 used by Schonlau *et al.*; encouraging results were obtained. In addition, a thorough analysis of

This work was supported by The Defense Advanced Research Projects Agency (DARPA) under Contracts F30602–99-2-0537 and F30602-00-2-0528.

R. Maxion is with the Dependable Systems Laboratory, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA15213 USA.

Tahlia Townsend is with the Yale University School of Law, New Haven, CT 06520 USA (e-mail: maxion@cs.cmu.edu).

Digital Object Identifier 10.1109/TR.2004.824828

the errors made by the detector on an alternative configuration of the same data is provided. This error analysis exposes a serious impairment of the Schonlau *et al.* data, providing insight into what allows a masquerader to evade detection.

II. Problem and Approach

Two objectives dominated this study: to determine whether or not a new classifier could improve upon the detection rates reported in previous work; and to provide a detailed examination of whatever classification errors occur in the study. An error analysis facilitates more learning than a simple report of classification results, because the errors and their causes will show what needs to be done to effect improvements in the future.

The problem is to detect illegitimate user activity. The assumption is that illegitimate activity will manifest as a deviation from the historical behavior of the user with respect to some attribute. The most recent research in masquerade detection has been based on the Schonlau *et al.* dataset, which consists exclusively of UNIX user commands [1]. These are the data used in this study. Thus, the attribute under consideration is command usage, and the task is to classify sequences of UNIX commands as "self" (issued by the authorized user), or "nonself" (issued by a masquerader). This task is reminiscent of text-classification, in which a document, seen as a collection of words, must be assigned to the class of politics or sports, for example. A classification algorithm known as Naive Bayes has a history of good performance in text-classification [4], and is employed as the detection algorithm in this work. Details about the detection algorithm are supplied in Section IV.

The following section discusses related work, after which a description is provided for the algorithm employed in the current study. Thereafter, details of data and experiments are given, followed by results, error analysis and discussion. Section XI describes terms and concepts that may aid understanding of general issues in classification and classifiers. Readers unfamiliar with classifiers may wish to read this section now.

III. Related Work

Masquerade data are extremely hard to obtain, due to concerns about user privacy and corporate confidentiality. Recently, Schonlau and his colleagues made a major contribution by publishing a large collection of UNIX user command data. This dataset formed the basis for experimentation by several researchers using a number of masquerade-detection techniques, the results of which were published as a compendium in [1].

These authors used UNIX command data from 50 users, injected with normal data from other users, to model a masquerade attack. Data for each user comprised 15,000 commands. The first 5,000 commands constituted training data (i.e., were free of injected commands); the remaining 10,000 commands were probabilistically injected with commands from other users. The idea was to discriminate sequences of 100 commands typed by the target user from injected sequences of 100 commands taken from other users. Further details regarding the data can be found in Section V.

The review paper by Schonlau *et al.* [1] compared the performance of six masquerade-detection algorithms (some new, others drawn from the computer science literature) on the data described above. Researchers sought to target a false alarm rate of 1%. All methods had relatively low hit rates (39.4%–69.3%) and high false alarm rates (1.4%–6.7%). The results were compared using both cluster analysis and ROC curves, revealing that no single method completely dominated any other. An overview of the various masquerade detectors used by Schonlau *et al.* is provided below.

1) Uniqueness: This approach, due to Schonlau and Theus [5], is based on ideas about command frequency. It postulates that commands not seen in the training data are indicative of a masquerade attempt and that the fewer the users employing a command, the more indicative that command may be of a masquerade. Uniqueness was a poor performer in terms of detecting masqueraders (39.4%), but it was the only method able to approach the target false alarm rate of 1% (1.4%) [1].

2) Bayes 1-Step Markov: This detector is based on single-step transitions between commands, and is due to DuMouchel [6]. The detector determines whether or not observed transition probabilities are consistent with historical probabilities. This technique was the best performer in terms of correct detections (69.3%), but it failed to get close (6.7%) to the desired false alarm rate.

3) Hybrid Multi-Step Markov: This method is based on Markov chains, and is due to Ju and Vardi [7]. The implementation of this model in [1] actually toggled between a Markov model and a simple independence model, depending on the proportion of commands in the test data that had not been observed in the training data. The performance of this method was among the best of the methods tested (49.3% hits, 3.2% false alarms).

4) Compression: The idea behind the compression approach is that new data from the same user should compress at about the same ratio as old data from that user, whilst data from a masquerading user will compress at a different ratio and thereby be distinguished from the legitimate user. (In fact, the experiment described uses a one-sided test, assuming only that masquerader data will be harder to compress than legitimate user data.) This idea is credited to Karr and Schonlau in [1]. Compression was the worst performer of the methods tested, with 34.2% hits and 5.0% false alarms.

5) IPAM: This detector (incremental probabilistic action modeling) is based on single-step command transition probabilities, estimated from training data, and was developed by Davison and Hirsh [8] for their work in predicting sequences of user actions. The performance of IPAM was poor (41.1% hits, 2.7% false alarms).

6) Sequence-Match: This approach is based on the early work of Lane and Brodley, refined in [9]. A similarity match is computed between a user's most recent commands and a corresponding user profile. As implemented on the Schonlau *et al.* data, this method was a poor performer (36.8% hits, 3.7% false alarms).

IV. The Naive Bayes Algorithm

This section provides a rationale for the use of a Naive Bayes classification algorithm in masquerade detection, and describes the details of the algorithm.

A. Why Use Naive Bayes for Masquerade Detection?

Naive Bayes classifiers (also known as simple Bayes classifiers) have a long history of use in pattern recognition and text classification. They have a number of attractive attributes: simplicity; computational speed (learning time is linear in the number of examples); potential for on-line use; and inherent robustness to noise and irrelevant attributes [10]. Noteworthy accuracy has been achieved for text classification with Naive Bayes on many natural domains, and a large body of research has shown Naive Bayes to be competitive with more sophisticated rule-induction and decision-tree algorithms; see, for example, [11].

In text classification the task is to assign a document to a particular class, typically using the so-called "bag of words" approach, which profiles document classes based simply on word frequencies [4]. Deciding whether a newspaper article is about sports, health or politics, based on the counts of words in the article, seems analogous to the task of deciding whether or not a stream of commands issued at a computer terminal belongs to a particular authorized user (who is likely to use various commands idiosyncratically). Despite its simplicity and success in text classification, hitherto the Naive Bayes approach has not been applied to user profiling with command-line data and masquerader detection.

The success of Naive Bayes has often struck researchers as surprising, given the unrealistic assumption of attribute independence which underlies the Naive Bayes approach. However, [12] demonstrates that Naive Bayes can be optimal even when this assumption is violated.

B. Description of Naive Bayes

In the present context, the classifier works as follows. The model assumes that the user is a multinomial machine,[1] generating sequences of commands, one command at a time, where each command has a fixed probability that is independent of the commands preceding it (this independence assumption is the "naive" part of Naive Bayes). The probability for each command c for a given user u is based on the frequency with which that command was observed in the training data, and is given by:

$$\theta_{c,u} = \frac{\text{Training Count}_{c,u} + \alpha}{\text{Training Data Length} + (\alpha \times A)}$$

where $\text{Training Count}_{c,u}$ is the number of times command c was seen in the training data of user u, α is a pseudocount, Training Data Length is the total number of commands issued during the training phase (the same for all users in this dataset) and A is the number of distinct commands (i.e., the alphabet size) in the data. The pseudocount can be any real number larger than zero (0.01 in this study), and is added to ensure that there are no zero counts; the lower the pseudocount, the more sensitive the detector is to previously unseen commands. The pseudocount term in the denominator compensates for the addition of a pseudocount in the numerator.

Assuming independence, the probability of a test sequence of L commands in which command c appears N_c times, drawn with replacement from an alphabet of A possible distinct commands distributed according to the probabilities of occurrence observed in the training data of user u is:

$$L! \prod_{c=1}^{A} \frac{(\theta_{c,u})^{N_c}}{N_c!}$$

This probability is the product of two parts. One part contains the factorials, the other the θ's. The former is a function of the sequence alone; the latter is a function of the user's habits and the counts in the sequence. The first part is largest for high-entropy sequences. This facet of the multinomial model results in an intrinsically low probability for certain sequences, for example a sequence of identical commands, irrespective of the user's habits. For classification, it is desirable for the probabilities used to be a function of the user's habits alone. For this reason, the factorial component of the formula was neglected and sequence probabilities were calculated simply as $\prod_{c=1}^{A}(\theta_{c,u})^{N_c}$. Thus, the *conditional probability of a test sequence of the five commands "a a b b b" given the user* $u1$, is:

$$P(aabbb \mid u1) = \theta_{u1,a} * \theta_{u1,a} * \theta_{u1,b} * \theta_{u1,b} * \theta_{u1,b}$$

or $(\theta_{u1,a})^2 * (\theta_{u1,b})^3$ where $\theta_{u1,a}$ is the probability that User1 typed the command a.

In order to detect masqueraders based on unlabeled test sequences, it is necessary to know the *conditional probability of the user given the sequence*. The probability of the user given the sequence can be obtained from the probability of the sequence given the user by putting a uniform prior on the users and invoking Bayes' rule for conditional probabilities. Assuming that all users are equally likely, by Bayes' rule one obtains:

$$P(\text{User} \mid \text{Sequence}) = \frac{P(\text{Sequence} \mid \text{User})P(\text{User})}{P(\text{Sequence})}.$$

For a given test sequence $P(\text{Sequence})$ is fixed and $P(\text{User})$ is defined to be the same for all users, i.e., the probability of the user given the sequence is the same as the probability of the sequence given the user, up to a constant which can be ignored.

For each User X, a model of Not X can also be built using training data from all other users. The probability of the test sequence having been generated by Not X can then be assessed in the same way as the probability of its having been generated by User X. The larger the ratio of the probability of originating with X to the probability of originating with Not X, the greater the evidence in favor of assigning the test sequence to X. The exact cutoff for classification as X is the ratio of probabilities below which the likelihood that the sequence was generated by X is deemed too low. This cutoff can be determined by a cross-validation experiment, calculating the probability ratios for sequences which are known to have been generated by self. The range of values covered by these legitimate sequences is examined, and a cutoff is chosen.

Further details regarding Naive Bayes can be found in [14] and [15].

[1] The multinomial distribution is an extension of the binomial distribution. A binomial distribution has two discrete outcomes, as in a coin toss; the probability of either outcome does not change, irrespective of how many times the coin is tossed, and each outcome is independent of previous outcomes. The multinomial distribution generalizes this concept to situations in which there are more than two discrete outcomes [13].

V. Data

The data used in the present study are those employed in the Schonlau *et al.* masquerade-detection study; they are available for download at http://www.schonlau.net, or from the authors.

As described by Schonlau *et al.*, the data consist of 15,000 commands, from each of 70 different users, recorded over a period of several months. Some users generated their 15,000 commands in a few days; others took a few months, although details of which users took how long are not provided with the dataset. The commands were extracted from the UNIX `acct` auditing mechanism. Examples of commands are: `sed, eqn, troff, dpost, echo, sh, cat, netstat, tbl, sed, eqn, sh`. Some commands not explicitly typed by a user were included as well (e.g., those generated by shell files or scripts); also included were names of executable, user-defined programs (e.g., a program that a user might have named *foo*).

Fifty of the seventy users were chosen as subjects. For these users, the first 5000 commands are contiguous and may be used for training a detector. The last 10,000 commands are broken into blocks of 100 commands each (the number 100 was chosen for simplicity's sake, according to Schonlau (pp. 60)). These 100 blocks, to be used for testing, were injected with "masquerader data" drawn from 100-command blocks of the remaining 20 users; that is, data blocks from the other 20 users were employed as if these 20 users were masqueraders. The injection methodology was as follows: with a probability of .01, a user block was replaced with a masquerader block. If such a replacement was made for a given block, then the next user block would be replaced with a masquerader block (from the same masquerader) with a probability of .8. Each block is either pure user or pure masquerader - there are no mixed blocks. Because the injected replacements were done probabilistically, each of the fifty injected users contained different numbers of masquerader blocks; some users had no injections, whereas other users had as many as 24. The data thus configured will be referred to as the SEA (S̲chonlau E̲t A̲l.) configuration.

VI. Improvements on Previous Results

A. Methodology

A separate classifier was built for each of the fifty users, individually called User X when not referring to any user in particular. During the training phase, the classifier built a profile of self and a profile of nonself (i.e., calculated the parameters for a multinomial model of command generation based on the command frequencies observed in the training data), using the first 5000 commands of User X's data for the former and the 49 × 5000 training data commands of the other 49 users for the latter profile.

During the testing phase, the classifier was presented with 10,000 commands split into either 100 unlabeled blocks of 100 contiguous commands, or 1000 unlabeled blocks of 10 contiguous commands.[2] This testing data was predominantly self data, but had been randomly injected by Schonlau *et al.* with between 0 and 2400 commands (in contiguous blocks of a maximum length of 100) generated by another user, as a proxy for a masquerader. Each test block was classified by the detector as self or nonself, according to whether the ratio of the log probabilities assigned to the block under the self and nonself models was greater than a threshold (nonself) or less than or equal to that threshold (self).

An initial experiment was conducted during which the profiles of self and nonself were not updated to reflect the information contained in the test blocks. A second experiment introduced a very straightforward method of updating. The command frequencies for each block of data identified as self were passed back to the detector to update the profile of self, whilst information from each block of data classified as nonself was used to update the model of nonself. No attempt was made to update the threshold, nor was information from blocks identified as masquerader material fed back into the model of nonself.

The threshold value for self, against which a test block could be compared to decide whether it should be classified as self or nonself, was established on the basis of five-fold cross-validation, as detailed here. The training data for User X were split into five different sets of 4000 and 1000 commands, for training and testing respectively. Labeling the blocks of 1000 making up the 5000 as A, B, C, D, E (in chronological order), the five sets of training and testing data were composed in the following way: ABCD and E; ABCE and D; ABDE and C; ACDE and B; BCDE and A. For each user and each set of 4000 and 1000 commands, a detector was trained on the 4000 commands of User X and 49 × 5000 commands of Not X information (i.e., the aggregated training data of the other 49 users). The remaining 1000 commands of User X data were segmented into 10 nonoverlapping blocks of 100 contiguous commands or 100 nonoverlapping blocks of 10 contiguous commands. Each block was presented to the detector, and was assigned a log-probability under both the self and nonself models. The five repetitions thus generated 50 or 500 (5 × 10 or 5 × 100) log-probability ratios for each user, one for each block of self data tested. The ratio of the magnitudes of these log-probabilities was obtained and noted for each test block and each set of 4000 and 1000 commands. The reader should bear in mind that the logarithms of the probabilities were actually used for calculations; thus, perhaps counter-intuitively, the larger the ratio of self to nonself, the greater the likelihood that the block was generated by nonself. To minimize false alarms, it is sensible to take the value corresponding to the least likely block of self data, i.e., the maximum ratio value obtained over the five repetitions, as the threshold for a given user. This was the strategy used for classifying blocks of 100 commands. However, the variance in probabilities for blocks of 10 commands is much greater than for blocks of 100, and taking the maximum therefore leads to a high threshold, thus increasing the chances of accepting a block of nonself as self. For blocks of 10 commands, the threshold for each user was therefore taken to be the mean of the five maximum values obtained over the five repetitions.

[2] The split into blocks of 100 commands each has no intrinsic significance for these experiments. It was made so that the experimental results would be directly comparable with Schonlau's previous results. The alternative block size of 10 was equally arbitrary; it was chosen to illustrate what happens at smaller, more realistic block sizes, since one wishes to detect masqueraders in as few commands as possible.

TABLE I
SUMMARY RESULTS OF NAIVE BAYES DETECTOR ON SEQUENCE LENGTHS 10 AND 100

Sequence length	100 commands		10 commands	
Updating	OFF	ON	OFF	ON
Hits %	66.2	61.5	55.2	47.1
Misses %	33.8	38.5	44.8	52.9
False Alarms %	4.63	1.3	4.7	1.6

TABLE II
SUMMARY RESULTS OF VARIOUS SCHONLAU *et al.* DETECTORS ON SEQUENCE LENGTH 100

Method	Hits %	Missed Alarms %	False Alarms %
Bayes 1-Step Markov	69.3	30.7	6.7
Hybrid Multi-Step Markov	49.3	50.7	3.2
Uniqueness	39.4	60.6	1.4
IPAM	41.1	58.9	2.7
Sequence Match	36.8	63.2	3.7
Compression	34.2	65.8	5.0

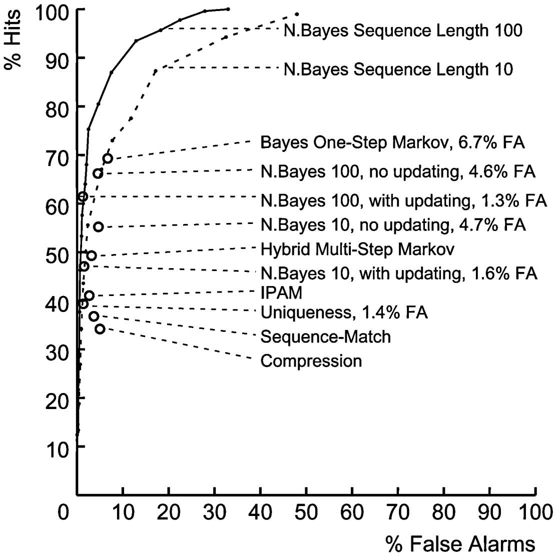

Fig. 1. Receiver operating characteristic (ROC) curves for Naive Bayes classifier, with updating, on sequences of length 10 and 100. Superimposed are best results achieved by the various methods described in the text. Solid curve shows sequence length 100; dotted curve shows sequence length 10.

Preliminary experiments indicated that using a different, individualized threshold for each user often yields worse results for individual users, as well as over the dataset as a whole. This can happen when a user's behavior is more variable in the training phase than in the testing phase. Under these circumstances, cross-validation indicates the need for a high threshold to minimize false alarms. However, the regularity of the user's behavior in the true testing phase (as opposed to the cross-validation testing phase) means that no advantage is gained by the high threshold in terms of false alarms; what accrues is only the disadvantage in terms of missed intrusions due to nonself blocks slipping under the high-threshold bar. Therefore, it was decided to employ a generic threshold for all users. Although there are many ways in which a generic threshold can be obtained, due to the early and exploratory nature of our investigation it was decided to employ the simple approach of taking the mean of the individual thresholds derived from the cross-validation process described above. It is worth noting that the 100 likelihood ratios obtained for each user from cross validation do not conform to any standard distribution, so that establishing a threshold with more advanced statistical methods would have involved a selection of smoothing methods which would have been complex and not necessarily any less ad hoc than the straightforward use of the mean.

B. Results

1) Summary Results: The overall performance of the Naive Bayes classifier on sequences of length 10 as well as sequences of length 100, with and without updating, is summarized in Table I. The percentage of hits, misses and false alarms per user is detailed in Tables VIII, IX, XII, and XIII, located in Section XII. Section XII also contains Tables X and XI, matrices which reveal the exact locations in the data stream of hits, misses and false alarms for each user (note that constructing such matrices was only possible for the experiments in which the data were partitioned into 100 blocks of 100, since it is impossible to print a readable matrix with 1000 columns on a single page).

2) Comparison With Prior Work: Table II shows the results obtained by Schonlau *et al.* with a variety of detectors for sequences of 100 commands (all of their detectors used updating).

In assessing the results of a masquerade detector one is concerned with the trade-off between correct detections (hits, or true positives) and false detections (false alarms, or false positives). These are often depicted on a receiver operating characteristic curve (called an ROC curve) where the percentages of hits and false alarms are shown on the y-axis and the x-axis, respectively.[3] ROC curves for the Naive Bayes classifier (with updating) on sequences of length 10 and 100 with the SEA data are presented in Fig. 1. Superimposed upon the curves are the points showing the best results reported by Schonlau *et al.* [1] as well as those of the present work.

The ROC curves show the results for a Naive Bayes classifier, with updating, applied to the SEA data configuration as the decision threshold was stepped through its range. The solid curve applies to the case in which the data are segmented into blocks of 100 contiguous commands; the dotted curve applies to blocks of just 10 contiguous commands. Lenient decision criteria allow a higher hit rate, but also a higher false-alarm rate; more stringent criteria tend to reduce both rates. Each point on an ROC curve indicates a particular trade-off between hits and false alarms. Points nearer to the upper left corner of the graph are the most desirable, as they indicate high hit rates and correspondingly low false-alarm rates. Recall that the unit of classification employed by both Schonlau et al. and the present study is a block of 100 contiguous commands. For the SEA data configuration as employed by Schonlau *et al.*, the random injection methodology of the latter researchers led to between 76 and 100 blocks of unlabeled self data per user, for a total of 4769 possible false alarms, and between 0 and 24 blocks of masquerader data per user, for a total of 231 possible missed alarms. This study documents an additional experiment in which these same data were segmented into blocks of just 10 consecutive user commands.

[3] For a thorough exposition of ROC curves, see [16].

In this regime, there were between 760 and 1000 blocks of unlabeled self data per user, for a total of 47 690 possible false alarms, and between 0 and 240 blocks of masquerader data per user, for a total of 2310 possible missed alarms.

The goodness of a detector may be judged in terms of its ability to limit either false alarms or missing alarms. In different situations it may be appropriate to attach different weights to missed alarms and false alarms. Therefore, the overall "goodness" of each of several detection methods can be ranked by applying a cost function of the form:

$$\text{Cost} = \alpha(\text{Misses}) + \beta(\text{False Alarms})$$

The cost of a false alarm in terms of misses will vary from one installation to another, so there is no obvious way to set the relative values of α and β to give an optimal cost. A first attempt at ranking the methods therefore uses a cost function in which α and β are both equal to one:

$$\text{Cost} = \text{Misses} + \text{False Alarms}$$

Under these circumstances, or if a missing alarm is deemed to carry more weight than a false alarm, the best-performing method for Schonlau *et al.* was Bayes 1-Step Markov, with 30.7% missing alarms, and corresponding 6.7% false alarms. The performance of the Naive Bayes algorithm on the same sequence length of 100 commands compares favorably with this result, limiting missing alarms to 33.8% and achieving a better false alarm rate of 4.6%, without the need for updating.

In practice, it is often deemed more important to limit false alarms than to limit missing alarms. Thus, the Schonlau *et al.* experiments targeted a false alarm rate of 1%. By this standard, the top-performing algorithm reported by the Schonlau team was Uniqueness, which attained a false-alarm rate of 1.4% whilst achieving a hit rate of 39.4%. For Uniqueness to emerge as the best of the methods described by Schonlau *et al.*, it is necessary to set the cost of a false alarm to 6 times that of a miss, i.e., to use the cost function

$$\text{Cost} = (\text{Misses}) + 6(\text{False Alarms})$$

To effect a proper comparison with the results of Schonlau *et al.* requires ranking the classification outcomes on the same basis as Schonlau *et al.* did. Hence, Table III presents the classification methods (on blocks of 100 commands) ranked according to the latter cost function: false alarms are six times as costly as misses. When false alarms and misses are weighted in this way, Naive Bayes with updating switched on, using a threshold extracted from the training data by cross validation, outperforms Uniqueness by a wide margin, achieving a 61.5% hit (detection) rate versus 39.4% for Uniqueness, an improvement of 56%, accompanied by a 7.1% improvement in false alarms, from 1.4% to 1.3%. On the basis of cost, the same Naive Bayes classifier realized a reduction of 32.9%, from 69.0 to 46.3.

The range of coefficients for the false-alarm term for which the whole ranking will be the same as that observed with a coefficient of 6, is 5.71 to 7.05. That is, if the β term in the expression above varies from 5.71 to 7.05, the rank ordering of the methods shown in Table III will not change.

TABLE III
RANKING OF CLASSIFICATION METHODS, USING SEA DATA CONFIGURATION AND COST = MISSES + 6^* (FALSE ALARMS) AS A RANKING FUNCTION (BASED ON SCHONLAU *et al.* ASSESSMENT OF UNIQUENESS AS THEIR BEST METHOD—SEE TEXT FOR DETAILS).

Method	Hits	Misses	FA	Cost
N. Bayes (Updating)	61.5	38.5	1.3	46.3
N. Bayes (no Upd.)	66.2	33.8	4.6	61.4
Uniqueness	39.4	60.6	1.4	69.0
Hybrid Markov	49.3	50.7	3.2	69.9
Bayes 1-Step Markov	69.3	30.7	6.7	70.9
IPAM	41.1	58.9	2.7	75.1
Sequence Matching	36.8	63.2	3.7	85.4
Compression	34.2	65.8	5.0	95.8

Naive Bayes with updating is superior both to Naive Bayes without updating and to Bayes 1-Step Markov at any weight for false alarms greater than or equal to 1.45 times that of a miss. In other words, given the expression $x * \alpha = \beta$, Naive Bayes with updating will remain at the top of the ranking as long as $x \geq 1.45$ (although the rankings of the rest of the methods may change).

In the real world, having to wait for 100 commands before evaluating the legitimacy of a user is clearly undesirable. When the length of the sequence of commands analyzed by the detector is reduced to just 10 commands, this waiting period decreases by a factor of 10. As can be seen from Table I, even with 10 times less information, Naive Bayes limits missed alarms more successfully than any of the algorithms employed by Schonlau *et al.* with the exception of Bayes 1-Step Markov. Not only does Bayes 1-Step Markov require 10 times as much information to achieve its 14% higher hit rate, it does so at a false alarm rate that is 2% higher than that of un-updated Naive Bayes working on sequences of size 10. In the case that false alarms are judged to have more weight than missing alarms, Naive Bayes results with sequences of just 10 commands perform very favorably again, limiting false alarms to 1.6% with a hit rate of 47.1%, a 19.5% improvement over the Uniqueness method, for only a 0.2% increase in false alarms.

3) Missed Intrusions: What Went Wrong?: Table X (see Appendix, Section XII) provides a detailed chart that breaks down the classification decisions made by Naive Bayes (without updating) on the SEA dataset, with sequences of size one hundred. Each row of the chart shows both the overall hits, misses and false alarms for a given user, and the position in the data stream of each misclassification made by the classifier for that user. The purpose of such a chart is to allow the experimenter to identify quickly the data upon which the classifier failed to do well, so that an investigation into what went wrong may be conducted. Space considerations prohibit an exhaustive analysis here, but a few of the most interesting features of the chart will be discussed.

Looking at the rightmost columns corresponding to hits and misses, one sees a large number of blank entries. Where there is no entry for either hits or misses, no masquerader data were injected by Schonlau *et al.* for that user. The reader is reminded that the injection methodology employed by Schonlau *et al.* resulted in different numbers (between 0 and 24) and types of masquerader blocks being injected into different users' data, and is referred to Section V for more details. From the columns in the chart, one learns that the worst affected users in terms of hits relative to misses were 3, 9, 12, 15 and 43. Careful examination of the graphical representation of results provided by Schonlau *et al.* reveals that these users suffer a similar number of misses irrespective of the algorithm employed. Three explanations for this are provided below: one for users 3, 15, and 43; one for user 9; and one for user 12.

Examining the probabilities associated with self and nonself for users 3, 15, and 43 one discovers that on the whole the masquerader blocks are distinct from the user blocks in terms of likelihood. Discrimination fails, however, because the threshold established by cross-validation is too low. Looking at the training data for these users one finds that each has a couple of rather unusual blocks. These blocks are assigned a low probability by the detector during training. In choosing a threshold, it was sought to minimize false alarms, therefore picking the lowest probability assigned to each user during the training period. The uniform threshold was then obtained by taking the mean of these minima. Thus, the presence of a couple of unusual training blocks for a handful of users results in a uniform threshold which is lower than optimal, allowing masqueraders to pass undetected. An alternative method of choosing the threshold should improve the hit rates for these users, although it is likely to have a negative impact on the more critical false alarm rate.

Turn now to the issue of missed intrusions for User 9, the most targeted user, with 24 masquerade events. No detector identified any more than 14 of these blocks. Two masqueraders are involved here (we only know that the two *may* be different, not that they actually are). Without updating, Naive Bayes accepts 11 of the first masquerader's 22 blocks as self, and both of the second masquerader's blocks. An examination of command frequencies for the first masquerader shows that this masquerader has a few different modes of behavior. Nine of the eleven blocks from this attacker, which were erroneously accepted as self, correspond to one of the distinct modes; the remaining 2 accepted blocks correspond to another. Both of these modes of behavior seem to be accepted by the Naive Bayes detector because of the overwhelming dominance of commands `cpp, rm` and `ls`, all three of which fall in the top 25 commands by frequency for User 9. The same blocks of masquerader data also foil the uniqueness and compression algorithms applied by Schonlau *et al.*, but are detected by mechanisms which make use of sequence (e.g., n-grams or tuples) as well as frequency information (as is the case for both of the Markov model detectors examined by Schonlau *et al.*).

Exploiting sequence information would be to no avail in the case of the missed intrusions suffered by User 12, however, due to the shared-tuple phenomenon. User 12 is not one of the top users in terms of shared tuple use (see Section VII-D-2 for details); but even so, the fact that none of the six masquerader blocks gives rise to an alarm is attributable to the presence of sequences of commands popular within the community as a whole in both the victim and the masquerader. Specifically, a certain 33-tuple is employed 8 times by User 12 and occurs 4 times in the masquerader data, affecting 3 of the six blocks. Another tuple, of length 10, crops up 33 times in User 12's training data and 9 times in the masquerader data, affecting 5 of the six blocks. The sequence of length 33 is common to the training data of 10 of the nonmasquerader users in this community, whilst the sequence of 10 commands referred to is employed by 27 users during training. This case is a reminder of the dangers posed by the presence of shared sequences of commands in a user environment, such as those commands generated by system-level scripts that were installed by a system administrator.

Focusing for a moment on successful detections, one sees from the chart (Appendix, Table X) that all the masqueraders embedded in the data of users 10, 24, and 26 were detected by Naive Bayes. The performance of the detectors tested by Schonlau *et al.* was not so consistently robust with respect to these masqueraders, with the exception of the attacks against User 10, which are perfectly detected by all their algorithms except compression. In the following paragraphs it is explored why this is so.

With respect to the masquerader embedded in the data of User 10, the strong performance of every algorithm except compression is hardly surprising—the 13 blocks of masquerader data consist entirely of one long string of 1300 `popper` commands. A compression-based system using a two-sided test (looking for blocks which are too easy to compress as well as those which are too hard) would probably perform just as well. This 5.6% of the total potential hits is really a freebie. It is reasonable to suppose that no real masquerader would issue 1300 identical commands in a row.

The situation with regard to the 21 blocks of data drawn from a single masquerader and injected into the data of User 24 is similar to that described above for the User 10 masquerader. In this case, the 2100 commands are almost exclusively long, single-command strings of `netscape` or `popper` or the trigram `netscape, popper, popper`, with an occasional smattering of `movemail`, `sendmail` and a few other commands. Since User 24 does not use any of *netscape, popper* or `movemail`, this masquerader is trivially easy to detect.

Two masqueraders make 10 and 3 attempts respectively against User 26. These are fairly easy to detect because the masqueraders and the user favor different commands; for example, of the 10 commands employed most frequently by the first masquerader, four are never used by User 26 in the training phase and another four are little used (fewer than 20 times) during training.

Table XI (see Appendix, Section XII) provides a detailed breakdown of the classification decisions made by Naive Bayes, with updating, on the SEA dataset, with sequences of size one hundred. With respect to the effects of updating on the hit and miss rates, the most striking feature is perhaps the loss of performance with respect to identification of nonself for User 42. Updating results in a drastic drop from 18 to only 6 hits for this user. The injection methodology employed by Schonlau *et al.* resulted in 20 blocks of masquerader material (drawn from a single masquerader) for this user, so the response of the detector to this user's data alone can cause the hit rate to fluctuate by up to almost 9%. Of the 20 attacks against this user, 18 are rejected by the detector when updating is switched off, but only 6 are detected when updating is turned on. The particular masquerader involved makes extensive use of bi- and trigrams made up of the command `sendmail`. This command occurs frequently for many of the users, but only infrequently in user 42's training data, so the masquerader is classified as nonself by the simple detector. However, it just so happens that the early part of the testing phase sees a sudden increase in User 42's employment of this command. When the profile built on the training data is updated to take account of this, the masquerader blocks cease to be more typical of the background model than of User 42, and are thus accepted as self. A situation like this sharpens the inappropriateness of using normal user data as masquerade data, as was done by Schonlau *et al.* One alternative might be to use synthetic data instead.

4) False-Alarm Analysis: Most of the observed false alarms are due to concept drift, sometimes called nonstationarity. Concept drift refers to the fact that user behavior may not be perfectly static; as time passes, new behaviors may emerge, making a user's behavior at one point in time, as represented by his profile, appear different from his behavior at a later point in time. If a user's behavior changes after the building of the profile, the classifier may not recognize data generated by the authentic user; hence, false alarms will be raised.

Table X (see Appendix, Section XII) provides a detailed breakdown of the classification decisions made by Naive Bayes, without updating, on the SEA dataset, with sequences of size one hundred. Looking at the column of false alarms, one sees that the majority of alarms are due to just a handful of users, namely 10, 12, 13, 16 and 20. These five users contribute 142 of the 221 false alarms, i.e., 64%. Comparing the positions of the exclamation points (indicating false alarms) in the row for any of these users to the appearance of the scattergram for that user (e.g., User 10 in Fig. 2), reveals the problem: substantial patches of behavior are not represented in the training data. This raises the question of what happens when updating is introduced; this is addressed below, with reference to Table XI.

Looking at the column giving the number of false alarms per user for the Naive Bayes detector with updating switched on (Table XI, Appendix, Section XII), one sees that false alarms drop from 25 to 1 for User 12, from 36 to 4 for User 13, from 32 to 4 for User 16 and from 31 to 3 for User 20. This is a 90% reduction in the false alarms raised for these users!

Decreasing the size of the sequences used for classification from 100 to 10 has only a small negative impact on the false alarm rate. With updating switched on, the false alarm rate for Naive Bayes classifying sequences of just 10 commands is only 0.3% worse than can be obtained with sequences of 100 commands. A large proportion of this 0.3% increase is due to increases in the false alarms due to users 18, 39, and 41. However, for each of these users, the individual false alarm rate remains well below 1%. The real problem is caused by intractable concept drift in the data of users 10 and 36. Concept drift is dramatically illustrated in Fig. 2, showing a scattergram of position in the data stream versus command id for User 10. This was one of the four users to whom 60% of the observed false alarms can be attributed. The drastic changes in command-line behavior illustrated on the scattergram coincide with the blocks which give rise to false alarms.

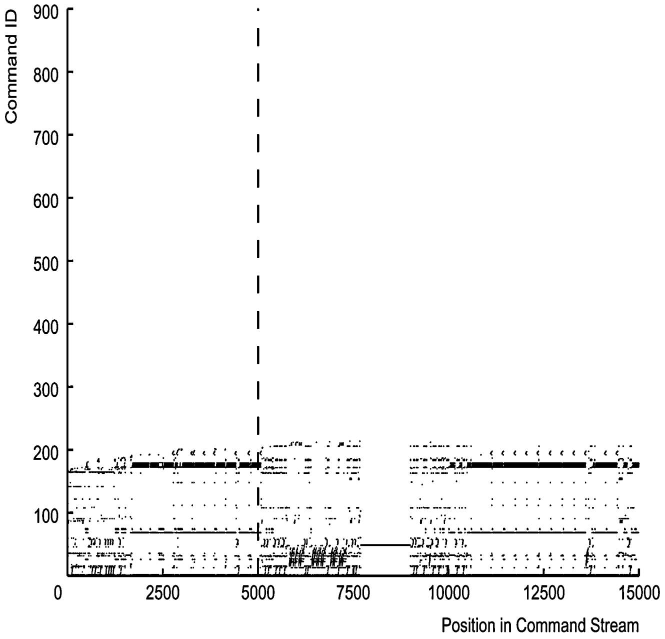

Fig. 2. Concept drift illustrated by User 10: scattergram representation of commands. Vertical dashed line indicates division between training data (left of line) and testing data. Note regions of test data bearing no resemblance to any portion of training data.

A combination of exploiting information about nonself (training the classifier on both self and nonself data) and updating the profiles is quite effective in tackling false alarms. However, the problem persists for User 10. This is partly due to the fact that the new behavior tends to cluster in just such a way that when the data are divided into blocks of 10 or 100 commands, there are no blocks in which the amount of data representing the new behavior is both small enough not to set off an alarm and large enough to bring about a sufficient change in the profile such that future instances of this behavior are accepted as self. The other problem with User 10 is that the new behavior exhibited by this user is actually common in the community as a whole, so that a classifier trained on self and nonself will preferentially assign those blocks to the nonself model. This is another illustration of the dangers posed by the presence of shared tuples (see Section VII-D-2 for details about shared tuples).

Users 10 and 36 contribute nearly one third of the total false alarms, with or without updating, and at either sequence length, putting a lower limit of 0.5% on the false alarm rate. It is interesting to note that the same users contributed 11 and 22 percent of the total false alarms under the Bayes 1-Step Markov and Compression regimes tested by Schonlau *et al.*; by themselves, these users gave rise to false alarm rates of 0.73% and 1.09% under those two methods respectively. Unfortunately, working with truncated command lines alone (i.e., command lines whose arguments have been stripped away), there does not appear to be any way to spot these troublesome users in advance; the training data do not contain any warning of the behavioral discontinuities which will appear in the test data. Perhaps such an indicator could be found in another source of information about the users, for example degree of programming experience or type of work.

With respect to users 10 and 36, all the methods tested on the SEA data configuration have done very well in identifying masquerade attacks. However, due to the random source injection protocol employed by Schonlau *et al.* in creating the test set, it is not possible to know whether this is due to some facet of the behavior of the users in question, or simply a fortuitous user-masquerader (self/nonself) pairing, where the randomly selected user just happens to have behavior that is distinct in every way from that of the victim. It is difficult to draw sensible generalizations of the kind necessary to improve upon results when every victim confronts a different masquerader. To get a better handle on questions like this, the present paper includes experiments with an alternative data configuration, termed 1v49, which is described in Section VII.

VII. Insight Into Masquerader Success

A. Motivation

A major shortcoming in the methodology followed by Schonlau *et al.* is that the design of the test set precludes sensible error analysis. Different masqueraders were injected into different users, and not all users were injected with masqueraders. In the real world, it is likely that only a few users will be subject to masquerader intrusions, possibly by a different masquerader in each case. Schonlau *et al.* sought to reflect these realities in the design of their test set. However, in a test setting, failure to run a consistent set of masqueraders against all users makes it difficult to draw constructive inferences from the hit and miss rates. When an algorithm fails to identify a masquerader block that occurs only in the data of a single victim, one does not know whether the failure is due to characteristics of the victim, the masquerader, or the algorithm. Perhaps data drawn from masquerader Y is particularly hard to detect when embedded in the data of User Z, but rather easily flagged as anomalous in the context of User W's normal behavior. If, in addition, the number of blocks of masquerader data varies from user to user, with particularly heavy concentrations in just a few of the users' streams, there is the potential for large fluctuations in the hit and miss rates simply due to experimental design. Not only was the number of masquerade events different for different users in the Schonlau *et al.* data, but in the majority of cases, where there were multiple masquerade events they were largely drawn from the same masquerader (no user was injected with data from more than three different masqueraders). For example: $22/24$ of User 9's, $21/21$ of User 24's, and $20/20$ of User 42's masquerader blocks were drawn from a single masquerader. Note that these masqueraders constitute 27.3% $(63/231)$ of the total injected intrusions. For each of the methods reported in the Schonlau *et al.* paper, a substantial proportion (15%, 16%, 21% and 14% respectively for Bayes 1-step Markov, Uniqueness, Hybrid Multi-step Markov and Compression) of the hits achieved were achieved by repeatedly identifying blocks taken from the same masquerader. Choosing the masquerader-user pairs differently might have had a significant impact on the success profiles reported.

B. Methodology

To address the methodological defect discussed above, the data were reconfigured for the current study to yield a test set that was consistent in the number and origin of the masquerade events encountered by each detector, i.e., if the detector that was trained on User X encountered masquerade events from 5 different masqueraders, the detector trained on User Y would encounter the same 5 events. In the new configuration, the training data remained the same as in the SEA configuration, i.e., the first 5 ,000 commands of a given user. However, the SEA testing data was altered. In the new configuration, the data for testing nonself recognition were 245,000 commands of testing data for each user, made up of the first 5,000 commands of the other 49 users (as if all 49 of the other users were masqueraders). The new data configuration is therefore referred to as the 1v49 (1 versus 49) configuration. It resulted in 2450 blocks of 100 contiguous masquerader commands $(245\,000/100)$ for each user, compared with between 0 and 24 for Schonlau *et al.* The test of self-recognition remained the same as in the SEA configuration, with between 76 and 100 blocks of self data culled from the second 10,000 commands being presented to the detector.

In the new data configuration, for each User X the training data from the other 49 users becomes testing data for nonself, that is the ability of the detector for User 1 to identify nonself is tested on 50 blocks of data from each of users 2 through 50. This means that the training data of those 49 users cannot be used to allow the detector to build a profile of nonself, as well as of self. Thus, the Naive Bayes algorithm employed in this experiment differs from that employed in the experiments on the SEA data configuration in that it trains only on self—this is referred to as One-Class Training Input, as opposed to Two-Class Training Input. The One-Class Training Input classifier only computes the probability of a test block under a model of self, i.e., the threshold value is a single probability rather than a ratio of two probabilities. The threshold for this experiment was chosen by five-fold cross-validation—the detector for User X was trained and tested on five different divisions of User X's training data into 4000 and 1000 commands, resulting in $5 \times 1000/100$, i.e., 50 probabilities, one for each test block under the model of self built based on the training data. The probability corresponding to the block of self data that was least likely to have come from the profile recorded was taken to be the threshold value. Once again, the reader is reminded that the magnitude of the logarithm of the probability was used, so the value corresponding to the least likely block would actually be the maximum of the fifty values obtained. The mean of the maxima from all the users was then taken as the generic threshold for all the users.

By way of comparison, in order to gauge whether the defects observed in the SEA data configuration were in fact causing any large swing in the results, the One-Class Training Input detector was also run on the SEA configuration. The results are reported in Table IV, below.

C. Results

The test of self-recognition capacity is the same in both data configurations, hence the false alarm rate is the same. It is reassuring that the performance of the algorithm is reasonably stable

TABLE IV
SUMMARY RESULTS OF ONE-CLASS TRAINING INPUT; NAIVE BAYES DETECTOR ON SEA AND 1v49 DATA CONFIGURATIONS.

Configuration	Hits %	Misses%	False Alarms %
1v49	62.8	37.2	4.63
SEA	61.5	38.5	4.63

over the two different data configurations. The slight improvement in performance on the 1v49 dataset is particularly encouraging since this is the more rigorous test of the ability to identify nonself. The 1v49 data configuration requires the detector for a given User X to identify 2450 separate blocks of intrusive data, drawn from 49 different masqueraders, compared to between 0 and 24 blocks from a maximum of 3 different masqueraders for the SEA configuration. The variety of masquerader behavior which the detector must discriminate from legitimate self behavior is therefore greater with the 1v49 configuration, and the nonself recognition test is thus more rigorous.

The rankings provided in Table III suggest which of the several classifiers/detectors might be the best performer in a given situation, but they do not reveal anything about the sources of classification error. The 1v49 experiment, described in this section, was designed explicitly to investigate the Naive Bayes classification errors, and provide insight into such questions as what makes a successful masquerader, or what makes a user an easy or a hard target.

D. Error Analysis

The base results of the 1v49 experiment are: 62.8% hits, 37.2% misses, and 4.63% false alarms, making the 1v49 outcome roughly equivalent, in terms of classification accuracy, to Naive Bayes (no updating) in the SEA version of the experiment. A detailed breakdown of the results of the 1v49 experiments, on a user by user and block by block basis, is provided in Section XII, Tables XIV, XV, XVI and XVII. The error analysis draws on the information contained in Table XIV and Table XVI. Representations like the one in Table XVI are often called confusion matrices or misclassification matrices, because they show which users were confused with which other users (or which users were misclassified as being which other users). To facilitate explanation, an excerpt of the confusion matrix is depicted in Table V. The number in each cell of the table indicates the number of missed detections when one user (victim, or self) is intruded upon by another (intruder, or nonself). For example the 46 in row 9, column 5 indicates the number of times that User 5 was undetected when masquerading as User 9. The victims (self) are listed down the rows; the intruders (nonself) are listed across the columns. Note that the numbers along the diagonal, as expected, are all zeros.

The far right column of Table V shows the total missed intrusions for each victim, indicating the victim's susceptibility to attack across the population of 50 masqueraders. The bottom row shows the total number of successful intrusions for each intruder, as indicators of attacker success. With such a table, it is easy to determine the most egregious masquerader as well as the most susceptible victim. For example, *within this portion of the table*, User 5 was the most successful masquerader, with 1617 incursions that were undetected, while User 7 showed the greatest susceptibility as a victim, with 1299 missed detections.

Examination of the full confusion matrix (Table XVI, Appendix, Section XII), reveals several interesting observations, addressed in detail below. User 30 suffered no successful attacks as a victim, and was never successful as an attacker; User 38 was the most successful attacker, with 1649 undetected attacks (out of 2450 attempts); and User 12 was most often victimized, with 1474 erroneously-accepted attacks.

1) What Makes A Masquerader Successful?: Table XVI (Appendix, Section XII) shows that the most successful masqueraders were users 38, 05 and 25. These masqueraders were successful across the board; more than half the users falsely accepted 35 or more of these masqueraders' blocks as self, leading to success rates of 67% $(1649/2450)$, 66% $(1617/2450)$ and 64% $(1578/2450)$, respectively.

What could make it so easy for these masqueraders to imitate the other users? As a first step toward answering this question, scattergrams were made of the position in the sequence of 5000 training commands (which can loosely be regarded as the time axis) against the command employed at that position. The scattergrams for users 38 and 5 are shown as Figs. 3 and 4.

The scattergrams for these and a number of other users had a striking feature—patterns of wavy lines which suggested the presence of repeating sequences of commands in the data. Preliminary investigations of the data by hand indicated that the data did in fact contain sequences of commands repeatedly employed by multiple users. These sequences were dubbed "shared tuples", and a thorough investigation of the extent of their occurrence and the effect of their presence on detection accuracy was launched.

2) Shared-Tuple Investigation: A sliding window facilitated identification of all the sequences of between 2 and 100 commands which were present in the training data as a whole. Sequences which did not occur in the data of at least 10 users, and sequences which only occurred as a subset of a longer sequence, were discarded. Seventy-two such nonoverlapping sequences were identified. It was found that these shared sequences, or tuples, accounted for 34.75% of the training data as a whole—an enormous proportion. The longest shared tuple meeting the 10-user criterion contained 49 commands, and was employed by 18 users. Over 60% of the training data for User 38 was accounted for by such shared sequences. By way of example, the following 10-tuple occurred multiple times in the data of 27 users: `hostname, id, nawk, getopt, true, true, grep, date, lp, find.`

Having compiled a list of shared tuples, the proportion of each user's training data which had been generated by employing these shared sequences was calculated; the amount of data attributable to the use of shared tuples is referred to as coverage. Table VI shows the proportion of the training data for each given user, made up of tuples shared by at least 10 users; i.e., the coverage of such tuples. Correlating the coverage with the number of times a user succeeded in passing himself off as one of the other users, the linear relationship between the two had a P-value less than .0001 $(R^2 = .52, r\hat{h}o = .72)$, indicating a correlation not due to chance. In other words, use of sequences of between 2 and 49 commands which are found in the data

TABLE V

CONFUSION MATRIX (EXCERPT): SIMPLE NAIVE BAYES CLASSIFIER WITHOUT UPDATING, 1v49 CONFIGURATION. "SELF" IS LISTED DOWN THE LEFT COLUMN; "NON-SELF" IS LISTED ACROSS THE TOP. NUMBERS IN THE MATRIX INDICATE MISSES—THE NUMBER OF TIMES THE DETECTOR ERRONEOUSLY ACCEPTED A MASQUERADER (NON-SELF) AS SELF, e.g., USER 5 SUCCESSFULLY MASQUERADED 46 TIMES AS USER 9.

VICTIM	INTRUDER											MISSED INTRUSIONS
	1	2	3	4	5	6	7	8	9	10	···	
1	0	2	40	47	41	1	40	3	50	9	···	1296
2	0	0	3	3	4	0	4	6	0	1	···	359
3	26	0	0	15	12	1	16	1	3	9	···	549
4	35	3	2	0	44	0	36	14	48	5	···	1110
5	10	3	2	18	0	0	32	3	34	2	···	1028
6	13	0	0	1	0	0	9	0	9	5	···	212
7	36	3	4	44	50	2	0	3	49	4	···	1299
8	2	1	2	38	36	0	9	0	14	0	···	611
9	37	3	2	45	46	1	40	2	0	14	···	1162
10	9	0	2	25	0	3	14	2	26	0	···	365
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
SUCCESSFUL INTRUSIONS	1124	402	339	1405	1617	74	1379	360	1359	270	···	

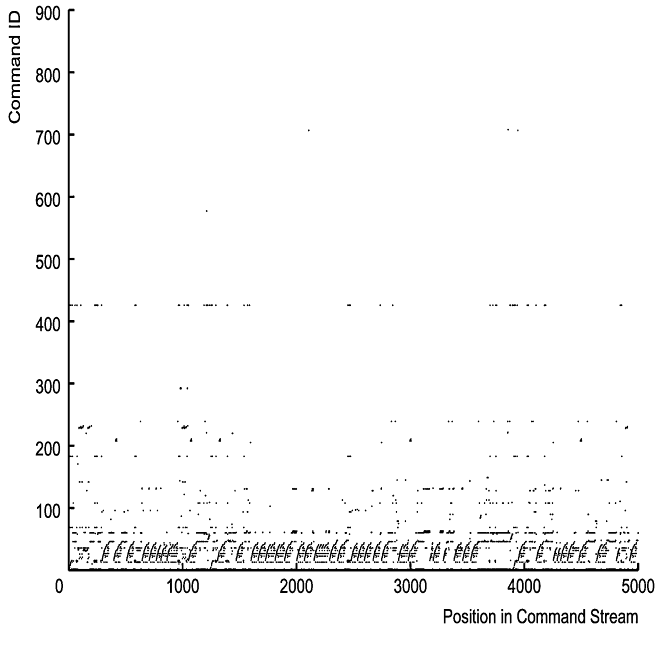

Fig. 3. User 38: scattergram representation of training data.

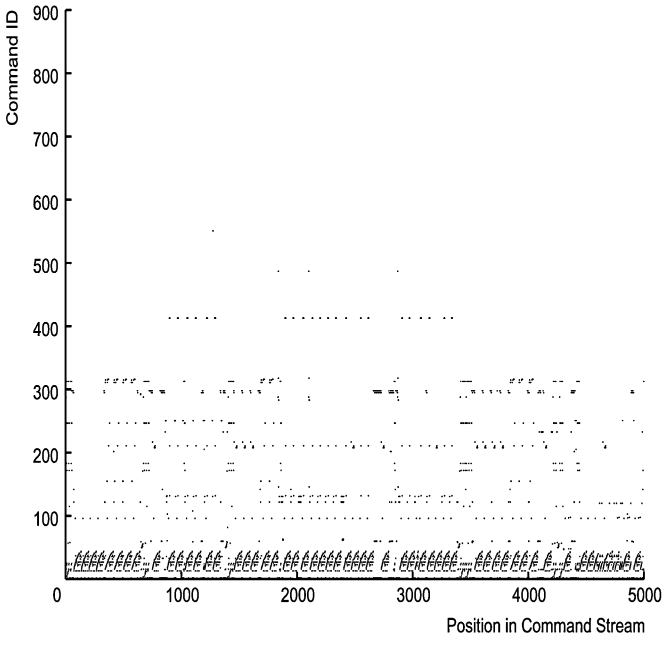

Fig. 4. User 5: scattergram representation of training data.

of 10 or more users explains 52% of the observed variation in the ability of the 50 users to masquerade as one another. Fig. 5 shows the correlation between tuple coverage and the number of blocks of data that each user was able to pass off as other users, i.e., masquerade success.

In terms of coverage of the training data by tuples common to 10 or more users, the top masqueraders are ranked 2nd, 12th, 14th and 4th respectively. The order of success does not exactly follow the order of coverage by shared tuples; that is, the correlation is not perfect, because the popularity of the tuples concerned (i.e., the number of users sharing the tuple) also counts. Why is it that some users with a lot of shared tuples do not do well as masqueraders? Referring to the interesting points on the correlation plot: Why is not User 25 a more successful masquerader than User 5? Why are not users 16 and 21 more successful as masqueraders? Why is User 31 successful?

TABLE VI
RANKED PERCENTAGE OF TRAINING DATA COVERED BY TUPLES COMMON TO AT LEAST 10 USERS.

Rank	User	Coverage %
1	16	70.5
2	38	66.6
3	21	58.1
4	25	57.9
5	41	55.1
6	14	54.1
7	15	52.2
8	22	51.4
9	1	51.2
10	19	50.4
11	40	48.7
12	48	48.6
13	5	48.5
14	37	45.4
15	9	44.2
16	17	42.4
17	4	41.9
18	39	41.7
19	33	41.1
20	13	40.1
21	28	39.7
22	31	39.5
23	7	38.4
24	45	37.8
25	43	37.4
26	47	37.3
27	2	37.2
28	50	36.7
29	44	35.8
30	49	34.2
31	12	33.9
32	11	32.1
33	24	29.5
34	34	29.2
35	20	27.1
36	23	26.6
37	26	25.0
38	3	23.0
39	42	22.3
40	6	20.2
41	29	19.8
42	27	16.7
43	10	14.4
44	35	13.3
45	8	9.3
46	18	3.9
47	46	3.5
48	32	2.1
49	36	1.3
50	30	0.0

User 38 uses more and more-popular (popularity is measured in terms of the number of users employing a tuple) shared tuples than any of the other masqueraders. The three tuples with the most coverage in this user's training data make up 31% of that data and have an average popularity of 22. The shared tuples employed by User 37 are very popular (the top tuples in terms of coverage have an average popularity of 22), but occur more sparsely than for the other top masqueraders (making up only 18% of User 37's training data). Over half of User 25's training data is made up of popular shared tuples. However, User 25's data also contains many blocks largely composed of long strings of command 48, `netscape`. The frequency with which User 25 employs the netscape command is anomalous with respect to the rest of the community. The average frequency with which this command is used by members of the community after removal of User 25 is 185, whereas User 25 employs the command 1266 times! This causes User 25 to enjoy markedly less success than the other top masqueraders in masquerading against users 8, 19, 24, 28, 31, 34 and 44, none of whom employ `netscape` commands to any significant extent (the command is not featured in a list of the top 25 commands by frequency for any of these users).

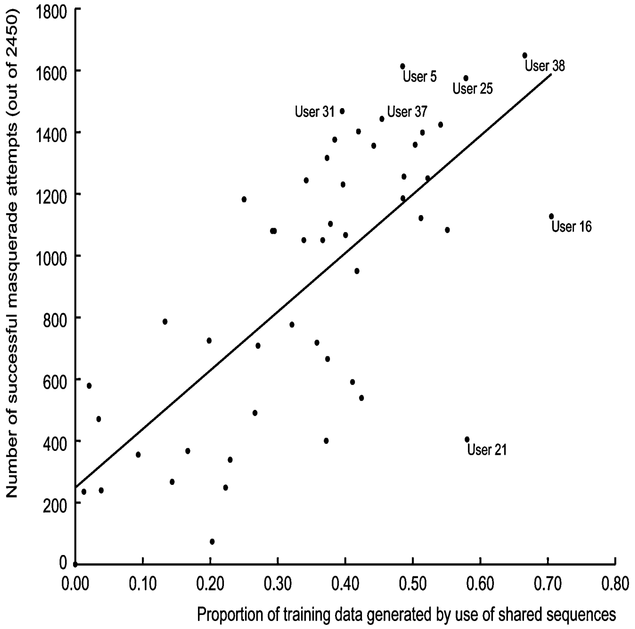

Fig. 5. Correlation between use of shared tuples (common to 10 or more users) and masquerader success. To avoid clutter, only selected points are labeled with a user number.

User 16 has the highest coverage by shared tuples—a whopping 70.5% of this user's training data is accounted for by tuples common to 10 or more users. However, User 16 ranks 20th in terms of masquerader success. User 16's failure as a masquerader can be attributed to this user's unusually frequent use of strings of `netscape` and of the bigram 189, 189 (`gcc, gcc`). User 16 employs `popper` 414 times, compared to an average of 185 over the other users, and uses command `gcc` 532 times, compared to an average of 63 over all users. Non-overlapping tuples made up only of these two commands make up 17% of User 16's training data. A further 23% of User 16's training data is accounted for by the shared 5-tuple `uname, nawk, cpp, cc1, as`. This tuple occurs very frequently in the training data as a whole, but is only shared by 12 users.

A high frequency of the same 5-tuple is behind the poor performance as a masquerader of User 21, despite 58% coverage by shared tuples. In this case, the 5-tuple `uname, nawk, cpp,` `ccl, as` accounts for an enormous 37% of the user's training data.

User 31 is as successful as users 37 and 25, despite a substantially (20%) lower coverage by shared tuples. This may be explained by the fact that although shared tuples make up only 40% of this user's training data, those shared tuples which user 31 does employ are highly popular—the three tuples which occur most frequently in User 31's data have an average popularity of 23.

In summary, a careful analysis of the data reveals that good masqueraders succeed against the Naive Bayes detector by virtue of one or more of the following attributes: heavy use of popular shared tuples; moderate use of very popular shared tuples; absence of anomalously high-frequency use of any particular command; and absence of frequent use of unpopular tuples. Shared tuples are a serious problem. The reader is reminded that the amount of training data accounted for by shared tuples ranges between 0 (for User 30) and 70.5% (for User 38), for a total of 34.8% of the training data as a whole. 52% of the variation in ability to masquerade can be explained by the amount of the user's data that is made up of shared tuples. The more shared tuples, the greater the user's chances of success as a masquerader. What is learned from the scattergram of Fig. 5 is that if an insider wishes to thwart any system which uses command frequencies, command popularities or n-tuple frequencies, and works in an environment with many built-in, widely-used scripts, he need only invoke a few of these scripts to generate a slew of commands with which to camouflage any unusual commands resulting from his nefarious activities. Shared tuples, arising perhaps from scripts built in to the computing environment, are very effective camouflage for the insider masquerader. This observation suggests that monitoring and removal of scripts common to the environment during data collection would be an important element in improving detection accuracy. A brief investigation into the effects of "de-tupling" the data is described next in Section VII-D-3.

3) Experiment With De-Tupled Data: The 1v49 experiment revealed that the presence of shared tuples in the data adversely affects detection accuracy. Thus, it seems reasonable to examine the effect of removing the tuples from the dataset.

Complete removal of all the tuples (sequences of 2 or more commands shared by 10 or more users) leads to a drastic reduction in the amount of training data available for certain users. The worst affected is User 16, for whom just 1473 commands remain after all of the shared tuples have been stripped out. An alternative to entirely stripping out the shared tuples is to map each distinct shared tuple to a new command. When this is done, User 38 turns out to suffer the greatest reduction in training data, with only 2027 of the original 5000 commands remaining.

Clearly it is not possible to conduct an experiment with the de-tupled data which will yield results directly comparable with those of previous experiments on this dataset. Nonetheless, it seems reasonable to run a few experiments with data from which shared tuples have been removed, either by stripping them out entirely or by condensing each tuple to a single, new command, in order to get a feel for how the accuracy of detection may be improved by processing command-line data in this way.

TABLE VII
SUMMARY RESULTS OF DE-TUPLED DATA EXPERIMENTS

	Left intact	Condensed	Removed
Hits %	68.8	75.0	76.2
Misses %	31.2	25.0	23.8
False Alarms %	NA	NA	NA

To this end, the locations of those sequences of commands common to at least 10 users identified by the investigation described in Section VII-D were established for each user. The training data were then processed in two ways: first by removing such sequences entirely; and second by replacing each such sequence with a single, new, unique command. Thus, three classes of training data were created for each user: data with shared tuples, data without shared tuples, and data with condensed shared tuples. Within each class, the data were then further processed by truncation to just 1000 commands.

Five-fold cross-validation was performed on the three classes of training data, using sets of 800 and 200 commands, and a sequence length of just 10 commands, in order to establish thresholds. For each User X and each class of data, a Naive Bayes detector was then trained on 1000 commands of User X data, and subsequently tested on 49×1000 commands drawn from the other 49 users. These test data were presented to the detector in the form of blocks of 10 commands.

The experiment was intended to investigate the effect of different treatments of the shared-tuple problem on the hit rate. False alarms were not investigated. The results are presented in Table VII. The table shows that misses were reduced by 23.7% when the shared tuples were completely removed, going from 31.2% misses in the intact data to 23.8% missed in the de-tupled data. At the same time, the hit rate increased by 10.8%, from 68.8% in the intact data to 76.2% in the de-tupled data. This suggests that at least moderate performance improvements can be obtained simply by ensuring that tuples are not shared among users. In the context of previous work in masquerade detection, improvements of this magnitude could be regarded as quite substantial. Of course the improvements shown here could be higher if the criteria were less strict. The removed or condensed tuples were common to at least ten users; many tuples are common to fewer than ten users, and if these had also been removed, performance improvements would have been even greater.

4) What Makes a User a Hard Target?: The toughest users to imitate were users 30, 47 and 32. These users suffered $0\% \, (0/2450), 3\% \, (74/2450)$ and $7.6\% \, (185/2450)$ misses respectively. Investigation of these users' data reveals that the characteristics which contribute to robustness against attack are: very heavy use of particular commands, regular use of rare commands, and infrequent use of popular commands. For example, more than half of the data for User 47 are made up of a single command—`cat`—occurring a total of 2565 times compared to an average over all users of just 223. In addition, User 47 employs a number of unique or rare commands, for example, `cgiparse`, with relatively high frequency, in a substantial number of blocks. User 30 has an extremely individual command-line profile. This user employs only 5 commands, of which two are barely used at all. Three commands, `tcsh`, `rshd` and `rdistd`, make up 99.6% of this user's training data. `rdistd` alone constitutes 33% of this user's training data, and is unique to this user. The other two commands have popularities of 5 and 15 and are employed with more than 5 times the average frequency.

5) What Makes a User an Easy Victim?: The users with the highest miss rates were users 12, 50 and 15. These users suffered 60.16% (1474/2450), 58.82%(1441/2450) and 57.10% (1399/2450) misses respectively. If one models average user behavior with a vector of average command frequencies, and calculates the Euclidean distance between this average vector and the frequency vector for each user's own training data, one discovers that the profiles of users 12, 50 and 15 are the second, third and fourth closest to the average. The profile closest to the average is that of User 22. This user ranks 6th in terms of miss proneness. In other words, what these users have in common is their "averageness". The correlation between proximity to the average and vulnerability is not perfect because the particular user-masquerader pairwise interactions also contribute significantly to the final tally of misses for a particular user. For example, a glance at the confusion matrix column corresponding to User 6 reveals that this user is in general a very poor masquerader, managing to pass off only one block against User 12, the most victimized user in general. The inability of User 6 to masquerade is the flip-side of this user's relative immunity to imitation by intruders. Both can be explained by regular, well-distributed use of unpopular commands; ten of the 25 commands most frequently employed by User 6 are used by four or fewer other users, and six of these ten unpopular commands occur in more than 40 of this user's 50 training blocks. Nonetheless, User 6 passes as User 42 fifty out of fifty times, and User 42 passes as User 6 forty-three out of fifty times, because these two users happen to share the same pool of otherwise very unpopular commands, and use them with similar frequency.

VIII. Discussion

Although the results achieved by using the Naive Bayes classifier constitute a 56% improvement over the previous result with the lowest false alarm rate, these improved results remain unsuitable for fielding a masquerade-detection system. The hit rate is too low, and the false alarm rate is too high. To be effective in an operational environment, a detector needs to have a false alarm rate of not more than 1%, and probably much lower than that. Acceptable false alarm rates are determined by the number of events per unit time that could raise an alarm. If a detector examines a million events per day, which for some detection environments would be considered a low number, a 1% false alarm rate would give 10,000 false alarms per day. That many false alarms cannot be handled by the limited number of analysts on duty, so the false alarm rate would need to be considerably lower. Concomitantly, hit rates must be higher to avoid missing true masquerade events. Reaching effective performance levels will require improvements in several areas:

1) Data: The poor performance of every method tried so far on these data (i.e., previous studies, as well as the present study) may indicate that command-line data alone, without arguments, are not enough to profile a user. Useful additions might be: arguments to commands, type and length of sessions, or user categories (job-type, department, etc.).

Careful data collection is important. Algorithms looking at the order of commands (e.g., Markov models) may have been adversely affected by the fact that these data are not truly sequential. In `acct`, the package used to collect the data, commands are not logged in the order in which they are typed, but rather when the application ends; moreover, commands issued in different terminal windows are interleaved. Such factors can constitute a considerable confound for Markov-based or other sequence-based detectors.

Careful preliminary examination of the data is important in avoiding things like including shared scripts which are built into the user environment (e.g., system login scripts) and which reduce the individuality of the data upon which the detector is trying to build a user profile.

It should be noted that using data blocks of 100 commands is rather unrealistic, because any serious and sensible masquerader is likely to hit and run in far fewer than 100 commands. Detection within a block of ten commands would seem more realistic. This study used a block size of 100 to allow comparison with the several Schonlau *et al.* results; however, a preliminary investigation using blocks of size ten has shown promising results: a 50.4% hit rate with a 2.1% false alarm rate. This compares quite favorably, all considered, with the block-100 results (61.5% hits and 1.3% false alarms).

Modeling a masquerade event by injecting one ordinary user's data into another ordinary user's data is far from ideal. Obviously, a real masquerader will not simply sit down at a victim's terminal and go about his normal business, but will seek to emulate the victim as accurately as possible. Real masqueraders have objectives and methods that may be more stealthy, and hence harder to detect. However, in the absence of any real masquerader data, the Schonlau at al. approach is a pragmatic starting point for a principled assault on the masquerader problem.

What would constitute better data? Since it is not completely clear what the effect of the Schonlau *et al.* data set is, because there is no data set against which to compare it, one can only speculate about how to effect improvements in the data. Some ideas are:

- gather data over equal lengths of time for all users, as opposed to a few days for some users and a few months for others;
- try to balance data across users by obtaining the same number of user/login sessions for each user;
- get time-stamps for logins and for each user command issued, so that one day can be differentiated from another;
- remove shared scripts, such as system login scripts, because these scripts inject shared tuples into the user data;
- get richer user data in terms of complete user commands complete with all the command-line arguments;
- get a job description for each user (e.g., programmer, researcher, manager, secretary, sys admin, etc.); and
- improve the features in the data by using latent features instead of, or in addition to, the raw features.

2) Methodology: The particular way an experiment is done can have dramatic effects on its outcome; small methodological details, accidentally overlooked, can impede other researchers from replicating or evaluating results. Experimental methods can substantially influence the validity of results. For example, the way that masquerade data are composed from segments of many users may be biased, and that bias affects outcomes. In the Schonlau *et al.* work this is manifested by the hit rates depending on the coincidences of which masquerade blocks were injected into which users. An additional biasing factor is the differential injection rates of masqueraders into different victims; some victims were only sparsely injected, some heavily and/or from all one masquerader, and some not at all. To gain the most in terms of understanding what works, what does not work, and why, a uniform injection methodology is needed.

3) Error Analysis: Error analysis is seldom done on classification or learning results. The present study makes it clear that an examination of errors can reveal important facts about what causes the detector to make a mistake, thus providing a basis for future improvement. Errors often teach more than correct results do.

4) Classifier: It is interesting to ask why Naive Bayes performed so much better than its competitors. Unfortunately, the answer to this question is as yet unclear. The hit rates of Naive Bayes might be explained by Naive Bayes being good at summing weak evidence. Such a situation would ensue when no single factor is particularly indicative of a masquerade intrusion, but many small things, in combination, are. The pseudocount probably helps to explain the lower false alarm rate of Naive Bayes compared with the Markov detector; around two hundred commands occur in testing that do not occur in training, i.e., there are a lot of zeroes in the historical transition probability matrix formed on the basis of the training data. Shared tuples may explain the poor performance of Uniqueness (Schonlau *et al.* note that with Uniqueness an intruder using mostly common commands will slip past the detection system). However, a detailed explanation for what it is that makes Naive Bayes appropriate for these data awaits further research.

IX. Conclusion

This paper has shown several important aspects of studying masquerader behavior as represented in monitored data—two things in particular:

1) that the design of a study (e.g., the 1v49 configuration) can influence substantially the outcome, especially when the goal is to test the efficacy of a detection scheme; and
2) that there is much to be learned from performing even a simple error analysis.

The biggest cause of false alarms was due to changes in user behavior over time (concept drift); however, this can be mitigated to a large extent (90% in this work) by incorporating a concept updating scheme in the classifier. Sequences of commands (tuples) that are shared among users can foster successful masquerades; in fact, the more tuples that a user shares with another, the more successful a masquerade attack against that user is likely to be. This shared-tuple discovery was facilitated by a simple graphing scheme which revealed patterns that appeared similar across the data of several users. Further research is needed to improve and understand classifier performance, but even more important is the gathering of competent data that truly support the undertaking of masquerade detection.

Appendix I
Background on Classifiers

This section provides brief background information about classifiers and issues pertaining to them.

1) Overview of Classifiers: The task of a classifier is to assign objects to classes: for example, to assign oranges to the class of fruits, carrots to the class of vegetables, etc. Classifiers may operate in either of two modes: unsupervised and supervised. In unsupervised mode, the classifier is presented with a collection of objects, and the classifier's task is to determine how many classes are represented by the collection, and which objects belong in which class. By contrast, in supervised classification the class structure is known in advance; the objective is for the classifier to determine the rules by which new objects can be sorted into their corresponding classes. The classifiers used in the present work are of the supervised type. Details of classification algorithms and processes can be found in such references as [17] or [15]. To understand classification as described in this paper, the reader will need to be familiar with the concepts sketched below.

2) Training Phase: A supervised classifier must be shown examples of objects in the classes of interest before it can begin assigning unlabeled objects to classes. In the authorship attribution task, for example, the goal is to attribute a text document or book to a particular author. In order to do this, the classifier must first be trained by showing it examples of works by these authors. The training phase enables the classifier to formulate rules for assigning texts to one author class or another. In this work, the task is to assign a sequence of n commands to a particular Unix user. In the training phase the classifier must be given information about the user's normal use of commands.

A classifier may also be given a notion of the kinds of things that do not constitute a class of interest. In this case, the classifier is presented with both positive and negative examples during the training phase, that is, a classifier designed to identify cancerous skin lesions might be given pictures of cancerous skin lesions upon which to build a model of the class Cancer, and pictures of noncancerous lesions upon which to build a model of Not Cancer.

3) Testing Phase: In the testing or operational phase, the trained classifier is shown examples from unknown classes, and is asked to assign each such example to one of the classes for which it has been trained.

4) One-Class versus Two-Class Classification: One-class classification (or one-class training) involves building a user profile or model on the basis of only the user's data (self), and not the data of any other user (nonself). In contrast, more typical (two-class) classification problems try to distinguish between two (or more) classes of objects, whereas one-class classification describes a single class of objects, and distinguishes it from all other possible objects.

5) Features: Classification rules are based on measured aspects of the objects to be classified, called features. In the authorship attribution example the features may be things like word frequency (different authors use different words in different proportions), sentence length, average number of syllables per word, etc. In this study, the features of interest are the frequencies with which each command is used.

6) Detection Thresholds: The rules formed during a classifier's training phase may be thought of as a metric for judging how closely an unknown object matches a known object. An object is rarely a perfect or exact match for a class, even if the object's assignment to that class would be regarded as correct. If the metric is probabilistic, the probability of an object's being a member of a class will almost never be 1 (or 0). Thus it is necessary to establish a threshold to determine how close to 1 the probability must be in order for the object to be assigned to the class. Typically, establishing thresholds is regarded as a matter of statistical estimation. The threshold must be estimated from the training data.

There are many ways to determine such thresholds [17]. In the present work, thresholds are determined through a process called cross validation, in which the training data are partitioned into groups. The classifier is trained on one portion of training data and "tested" on another portion of training data. If the classification metric is probabilistic, it will produce a score for each test sample, giving the probability that the test sample belongs to a particular class. Since the "test" data in this case are actually taken from the training sample, the true class of the test samples is known. The range of probabilities for the true class, which the classifier assigns to the test samples, indicates the extent to which members of the same class are similar to one another; a value in this range can then be chosen as the threshold value, i.e., the value below which an object is deemed too unlikely to belong to the class of interest. Which value in the range to choose depends upon the relative importance of misclassifications versus false rejection. For example, taking the lowest probability for the true class that was assigned to a member of that class as the threshold should minimize the risk of future members of the true class being rejected by the classifier. With a little refinement, this was the strategy adopted in the present work.

The particular kind of cross validation used in the present work is called N-fold cross validation. In this method, the training data are divided into N groups of training and testing data, each of which is used once to train and test the classifier The classifier is trained on one of the N groups, and tested on the rest. The ranges of probabilities for the true class assigned to members of the true class during each of the N repetitions are combined, and a threshold somewhere within the resulting range is chosen, with the exact value depending upon the appropriate bias toward minimizing false acceptances or false rejections.

7) Hits, Misses and False Alarms: The classifier described in the present study may attribute sequences of commands to one of two classes—"self" or "nonself". Correct attribution of a sequence of nonself commands to the class nonself is a "hit". Incorrect attribution of a sequence of nonself commands to the class self is a "miss". Incorrect attribution of a sequence of self commands to the class nonself is a "false alarm".

8) Updating: User behavior may not be perfectly static. As time passes new behavior may emerge; this is known as concept drift. If a user's behavior changes after the building of the profile, the classifier may not recognize data generated by the authentic user, and false alarms will be raised. To deal with this sort of error, the profile used by the classifier must be updated to reflect changes in the user's behavior. If the changes are slow enough, new data generated by the user will vary slightly from the old patterns, but not sufficiently to confuse the classifier. Such data, containing information about old and new patterns of behavior can be passed back to the classifier to update the profile. Information about the new behavior is then contained in the updated profile so that later examples of the new behavior will be recognized by the classifier as consistent with the user profile without giving rise to alarms.

In the current study, experiments were conducted both with and without updating. For the case with updating, a block by block updating function was implemented. Each block of test data that was classified as self was used to update both the profile of self and the profiles of nonself for the other 49 users. In order to do this, the command frequency information from each such block was added to the training profile of self and to the profiles of nonself for the other 49 users, and the parameters were recalculated accordingly. After updating the profiles, the next test block was evaluated.

TABLE VIII
Two-Class Training Input; Naive Bayes Algorithm Without Updating; Sea Data COnfiguration; Sequence Size 100; Threshold 1.263

User	Hits	Misses	False Alarms
1	0/0	0/0	1/100
2	3/3	0/3	0/97
3	6/11	5/11	0/89
4	2/2	0/2	0/98
5	0/0	0/0	0/100
6	0/0	0/0	0/100
7	9/13	4/13	1/87
8	0/0	0/0	0/100
9	11/24	13/24	0/76
10	13/13	0/13	18/87
11	0/0	0/0	1/100
12	0/6	6/6	25/94
13	0/0	0/0	36/100
14	0/0	0/0	1/100
15	0/6	6/6	0/94
16	4/10	6/10	32/90
17	0/0	0/0	0/100
18	6/6	0/6	3/94
19	0/0	0/0	0/100
20	0/0	0/0	31/100
21	0/0	0/0	0/100
22	0/0	0/0	0/100
23	1/1	0/1	0/99
24	21/21	0/21	0/79
25	2/9	7/9	0/91
26	13/13	0/13	4/87
27	0/0	0/0	0/100
28	3/3	0/3	2/97
29	1/1	0/1	10/99
30	3/3	0/3	0/97
31	0/0	0/0	0/100
32	0/0	0/0	0/100
33	0/0	0/0	5/100
34	3/12	9/12	0/88
35	1/1	0/1	0/99
36	6/6	0/6	10/94
37	0/2	2/2	0/98
38	6/9	3/9	5/91
39	0/0	0/0	7/100
40	0/0	0/0	5/100
41	2/3	1/3	7/97
42	18/20	2/20	6/80
43	5/16	11/16	5/84
44	6/6	0/6	2/94
45	2/5	3/5	1/95
46	4/4	0/4	0/96
47	0/0	0/0	0/100
48	2/2	0/2	2/98
49	0/0	0/0	1/100
50	0/0	0/0	0/100
Totals:	153/231	78/231	221/4769

TABLE IX
Two-Class Training Input; Naive Bayes Algorithm Without Updating; Sea Data COnfiguration; Sequence Size 100; Threshold 1.227

User	Hits	Misses	False Alarms
1	0/0	0/0	1/100
2	3/3	0/3	0/97
3	8/11	3/11	0/89
4	2/2	0/2	0/98
5	0/0	0/0	0/100
6	0/0	0/0	1/100
7	9/13	4/13	1/87
8	0/0	0/0	0/100
9	10/24	14/24	0/76
10	13/13	0/13	14/87
11	0/0	0/0	0/100
12	0/6	6/6	1/94
13	0/0	0/0	4/100
14	0/0	0/0	0/100
15	0/6	6/6	0/94
16	3/10	7/10	4/90
17	0/0	0/0	1/100
18	6/6	0/6	3/94
19	0/0	0/0	0/100
20	0/0	0/0	3/100
21	0/0	0/0	0/100
22	0/0	0/0	0/100
23	1/1	0/1	0/99
24	21/21	0/21	0/79
25	8/9	1/9	0/91
26	13/13	0/13	4/87
27	0/0	0/0	0/100
28	3/3	0/3	0/97
29	1/1	0/1	1/99
30	3/3	0/3	0/97
31	0/0	0/0	0/100
32	0/0	0/0	0/100
33	0/0	0/0	1/100
34	0/12	12/12	0/88
35	1/1	0/1	0/99
36	6/6	0/6	9/94
37	0/2	2/2	0/98
38	7/9	2/9	5/91
39	0/0	0/0	0/100
40	0/0	0/0	5/100
41	2/3	1/3	1/97
42	6/20	14/20	0/80
43	2/16	14/16	1/84
44	6/6	0/6	1/94
45	2/5	3/5	0/95
46	4/4	0/4	0/96
47	0/0	0/0	0/100
48	2/2	0/2	0/98
49	0/0	0/0	0/100
50	0/0	0/0	0/100
Totals:	142/231	89/231	61/4769

Appendix II
Details of the Two-Class Training Experiments

The information in this Appendix is discussed in Sections VI-B-3 (Missed intrusions: what went wrong?), VI-B–4 (False-alarm analysis), VII-D (Error analysis) and VII-D-1 (What makes a masquerader successful?).

The first several tables address the two-class training condition for the masquerade detector. Tables VIII and IX compare

TABLE X
Two-Class Training Input; Naive Bayes Algorithm With Updating, Sea Data Configuration, Sequence Size 100, Threshold 1.263.
Key: False Alarm: !/Miss:-/Hit:*

User|Test Block|																																																																																																											|FA|Miss|Hits
	0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2	3	4	5	6	7	8	9	40	1	2	3	4	5	6	7	8	9	50	1	2	3	4	5	6	7	8	9	60	1	2	3	4	5	6	7	8	9	70	1	2	3	4	5	6	7	8	9	80	1	2	3	4	5	6	7	8	9	90	1	2	3	4	5	6	7	8	9			
1																																																																	!																																								1		
2																																																												*	*	*																																														3
3																																								*	*	-	*	*	*	-	-	-	*	-																																																						5	6
4																																																									*	*																																																2
5																																																																																																							
6																																																																																																						
7|																							|-|-|-|*|*|*|*|*|*|*|*|*|									|!|											|-|																																																								|1|4|9
8																																																																																																						
9																																*	-	-	-	-	-	-	-	*	*	*	*	-	-	*	*	*	*	*	*	-	-																					-	-																																13	11
10								!	!	!	!	!			!	!	!	!		!	!	!	!		!		!	*	*	*	*	*	*	*	*	*	*	*	*	*			!																																														!										!					18		13
11																																																																						!																															1		
12						!					!	!	!	!	!	!	!	!	!	!	!	!	!	!	!		!		!	!	!	!			!	!	!																																					-	-	-	-	-	-																										!					25	6	
13										!		!							!								!														!				!					!	!	!	!	!	!			!		!			!					!	!	!	!	!	!	!	!	!	!			!		!	!		!	!		!	!								!	!		!	!	36		
14																			!																																																																																		1		
15																																																																																														-	-	-	-	-	-		6	
16							!		!	!	!	!	!	!	!	!	!	!	-	-	-	*			!			!	!							!	!	!					!			!		!	!							!		!			!	!		!			!			!			!						!	*			!	-	-	-	*	*									!					32	6	4
17																																																																																																					
18									*	*	*	*	*	*																																							!	!			!																																																3		6
19																																																																																																					
20																																														!	!	!		!	!	!	!	!	!	!	!	!		!	!	!	!	!	!	!	!	!	!	!	!	!	!							!	!	!		!	!																31		
21																																																																																																					
22																																																																																																					
23																																																	*																																																									1
24																																																																					*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*														21
25																																																	-	*	*																							-																-	-	-	-	-								7	2
26																																				!				!								!	!										*	*	*	*	*	*	*	*	*	*								*	*	*																										4		13
27																																																																																																					
28																																						!																		!																																																	*	*	*	2		3
29																	*		!																																		!										!	!	!		!		!	!	!			!																														10		1
30																																		*	*	*																																																																				3
31																																																																																																						
32																																																																																																					
33																																							!																!	!		!																!																																5		
34										-	-	-	-	-																																					-	*	*	*	-	-	-																																																			9	3
35																																																																																																					*			1
36																															!								!	!	!											*	*	*	*	*	*												!	!	!			!		!	!																												10		6
37																																																																																-	-																			2	
38																																											-	-	-										!	!	!	!	!																						*	*	*	*	*	*																				5	3	6
39																			!																			!																																																											!	!	!	!				!	7		
40																																!										!	!																																																						!	!						5		
41																			-	*	*																																																						!																	!	!	!	!	!	!						7	1	2
42																										!				!					!									!																						*	*	*	*	*	*	*	*	-	*	*	*	*	*	*	-	*	*	*	*																!	!			6	2	18
43																													-	-	-	*	-	-	-	-	-	-	*	-	*	*	-									!						!							*																																		!			!	!				5	11	5
44																										!									*											*	*	*	*	*																															!																								2		6
45																									*	*	-	-	-																																																													!															1	3	2
46																																																																						*	*	*	*																																	4
47																																																																																																				
48																			!																											!																																																			*	*								2		2
49																														!																																																																										1		
50																																																																																																					
																																																																																																						221	78	153

TABLE XI
TWO-CLASS TRAINING INPUT; NAIVE BAYES ALGORITHM WITH UPDATING, SEA DATA CONFIGURATION, SEQUENCE SIZE 100, THRESHOLD 1.227. KEY: FALSE ALARM: !/MISS:-/HIT:*.

User	Test Block																																																																																																				FA	Miss	Hits
	0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2	3	4	5	6	7	8	9	40	1	2	3	4	5	6	7	8	9	50	1	2	3	4	5	6	7	8	9	60	1	2	3	4	5	6	7	8	9	70	1	2	3	4	5	6	7	8	9	80	1	2	3	4	5	6	7	8	9	90	1	2	3	4	5	6	7	8	9			
1																																																																	!																																								1		
2|																																																													|*|*|*|																																										|3
3																																									*	*	-	*	*	*	*	-	*	*	-																																																							3	8
4																																																									*	*																																																2
5																																																																																																								
6																																																					!																																																		1		
7																									-	-	-	*	*	*	*	*	*	*	*	*										!													-																																																			1	4	9
8																																																																																																							
9																																*	-	-	-	-	-	-	-	*	*	*	*	-	-	*	*	*	*	*	-	-	-																				-	-																															14	10
10									!	!	!	!			!	!	!	!		!	!	!	!		!		!	*	*	*	*	*	*	*	*	*	*	*	*	*																																																													14		13
11																																																																																																						
12																																														!																											-	-	-	-	-	-																													1	6	
13																			!																															!	!	!																																																	4		
14																																																																																																						
15																																																																																									-	-	-	-	-	-		6	
16																			*	-	-	-																																			!																								*				-	-	*	-	-		!	!					!					4	7	3
17																																																																															!																							1		
18									*	*	*	*	*	*																																								!	!			!																																															3		6
19																																																																																																						
20																																															!	!	!																																																										3		
21																																																																																																							
22																																																																																																						
23|																																														|*|																																																											|1
24																																																																					*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*														21
25																																																	*	*	*																								-																*	*	*	*	*								1	8
26																																			!				!									!	!									*	*	*	*	*	*	*	*	*	*								*	*	*																													4		13
27																																																																																																						
28																																																																																																				*	*	*			3
29																*		!																																																																																						1		1
30																																		*	*	*																																																																							3
31																																																																																																						
32																																																																																																						
33|																																					|!|																																																																					|1|	
34											-	-	-	-	-																																					-	-	-	-	-	-	-																																																			12	
35|																																																																																																						|*|	|1
36																															!								!	!	!											*	*	*	*	*	*																!	!	!						!	!																													9		6
37																																																																																					-	-																					2	
38																																												-	*	-										!	!	!	!	!																							*	*	*	*	*	*																				5	2	7
39																																																																																																						
40|																														|!|								|!|!|																																																											|!|!|					|5|	
41																			-	*	*																																																			!																														1	1	2
42																																																																		*	*	*	*	-	-	-	-	-	-	-	-	-	-	*	-	*	-	-	-																					14	6
43																													-	-	-	*	-	-	-	-	-	-	-	-	-	-	-										!												*																																															1	14	2
44																									!								*											*	*	*	*	*																																																													1		6
45																									*	*	-	-	-																																																																															3	2
46																																																																							*	*	*	*																																			4
47																																																																																																							
48|																																																																																														|*|*|									|2
49																																																																																																							
50																																																																																																							
																																																																																																								61	89	142

TABLE XII
TWO-CLASS TRAINING INPUT; NAIVE BAYES ALGORITHM; SEA DATA CONFIGURATION; NO UPDATING; SEQUENCE SIZE 10; THRESHOLD 1.352.

User	Hits	Misses	False Alarms
1	0/0	0/0	10/1000
2	29/30	1/30	2/970
3	51/110	59/110	1/890
4	14/20	6/20	5/980
5	0/0	0/0	0/1000
6	0/0	0/0	2/1000
7	62/130	68/130	6/870
8	0/0	0/0	32/1000
9	100/240	140/240	4/760
10	130/130	0/130	164/870
11	0/0	0/0	14/1000
12	0/60	60/60	259/940
13	0/0	0/0	281/1000
14	0/0	0/0	8/1000
15	0/60	60/60	6/940
16	22/100	78/100	313/900
17	0/0	0/0	3/1000
18	58/60	2/60	66/940
19	0/0	0/0	3/1000
20	0/0	0/0	206/1000
21	0/0	0/0	0/1000
22	0/0	0/0	4/1000
23	10/10	0/10	1/990
24	210/210	0/210	6/790
25	22/90	68/90	0/910
26	99/130	31/130	32/870
27	0/0	0/0	7/1000
28	15/30	15/30	86/970
29	7/10	3/10	107/990
30	30/30	0/30	4/970
31	0/0	0/0	2/1000
32	0/0	0/0	1/1000
33	0/0	0/0	40/1000
34	10/120	110/120	7/880
35	5/10	5/10	3/990
36	45/60	15/60	95/940
37	0/20	20/20	5/980
38	60/90	30/90	54/910
39	0/0	0/0	88/1000
40	0/0	0/0	42/1000
41	14/30	16/30	57/970
42	122/200	78/200	10/800
43	28/160	132/160	45/840
44	59/60	1/60	13/940
45	15/50	35/50	71/950
46	40/40	0/40	2/960
47	0/0	0/0	7/1000
48	18/20	2/20	42/980
49	0/0	0/0	10/1000
50	0/0	0/0	31/1000
Totals:	1275/2310	1035/2310	2257/47690

TABLE XIII
TWO-CLASS TRAINING INPUT; NAIVE BAYES ALGORITHM; SEA DATA CONFIGURATION; WITH UPDATING; SEQUENCE SIZE 10; THRESHOLD 1.342.

User	Hits	Misses	False Alarms
1	0/0	0/0	10/1000
2	30/30	0/30	1/970
3	53/110	57/110	1/890
4	12/20	8/20	4/980
5	0/0	0/0	0/1000
6	0/0	0/0	4/1000
7	27/130	103/130	6/870
8	0/0	0/0	4/1000
9	57/240	183/240	5/760
10	130/130	0/130	137/870
11	0/0	0/0	2/1000
12	0/60	60/60	8/940
13	0/0	0/0	51/1000
14	0/0	0/0	7/1000
15	0/60	60/60	3/940
16	10/100	90/100	7/900
17	0/0	0/0	4/1000
18	59/60	1/60	75/940
19	0/0	0/0	2/1000
20	0/0	0/0	31/1000
21	0/0	0/0	0/1000
22	0/0	0/0	2/1000
23	10/10	0/10	2/990
24	210/210	0/210	4/790
25	70/90	20/90	0/910
26	80/130	50/130	27/870
27	0/0	0/0	4/1000
28	3/30	27/30	9/970
29	7/10	3/10	13/990
30	30/30	0/30	3/970
31	0/0	0/0	2/1000
32	0/0	0/0	1/1000
33	0/0	0/0	18/1000
34	6/120	114/120	7/880
35	5/10	5/10	2/990
36	38/60	22/60	72/940
37	0/20	20/20	2/980
38	61/90	29/90	50/910
39	0/0	0/0	30/1000
40	0/0	0/0	32/1000
41	14/30	16/30	55/970
42	40/200	160/200	5/800
43	17/160	143/160	9/840
44	54/60	6/60	13/940
45	8/50	42/50	13/950
46	40/40	0/40	5/960
47	0/0	0/0	4/1000
48	18/20	2/20	9/980
49	0/0	0/0	10/1000
50	0/0	0/0	7/1000
Totals:	1089/2310	1221/2310	772/47690

the Naive Bayes classifier results, using a data sequence length of 100, with updating (compensation for concept drift) turned off and on, respectively. Tables X and XI contain charts that depict the exact placement of hits, misses and false alarms for each injection across all 50 users, showing the effects of having updating turned off and on, respectively. Tables XII and XIII compare the Naive Bayes classifier results, using a data sequence length of 10, with updating turned off and on, respectively.

TABLE XIV
ONE-CLASS TRAINING INPUT; NAIVE BAYES ALGORITHM WITHOUT UPDATING; 1v49 DATA CONFIGURATION; SEQUENCE SIZE 100; THRESHOLD 6.479.

User	Hits	Misses	False Alarms
1	1154/2450	1296/2450	1/100
2	2091/2450	359/2450	0/97
3	1901/2450	549/2450	0/89
4	1340/2450	1110/2450	0/98
5	1422/2450	1028/2450	0/100
6	2238/2450	212/2450	0/100
7	1151/2450	1299/2450	0/87
8	1839/2450	611/2450	0/100
9	1288/2450	1162/2450	1/76
10	2085/2450	365/2450	19/87
11	1216/2450	1234/2450	1/100
12	976/2450	1474/2450	1/94
13	1270/2450	1180/2450	32/100
14	1437/2450	1013/2450	0/100
15	1051/2450	1399/2450	0/94
16	1092/2450	1358/2450	59/90
17	1186/2450	1264/2450	0/100
18	2060/2450	390/2450	2/94
19	1426/2450	1024/2450	0/100
20	1317/2450	1133/2450	22/100
21	2219/2450	231/2450	6/100
22	1107/2450	1343/2450	0/100
23	1307/2450	1143/2450	0/99
24	1487/2450	963/2450	0/79
25	1464/2450	986/2450	0/91
26	1838/2450	612/2450	3/87
27	1283/2450	1167/2450	8/100
28	1541/2450	909/2450	0/97
29	1415/2450	1035/2450	10/99
30	2450/2450	0/2450	0/97
31	1416/2450	1034/2450	0/100
32	2265/2450	185/2450	0/100
33	2065/2450	385/2450	7/100
34	1407/2450	1043/2450	0/88
35	2159/2450	291/2450	0/99
36	2218/2450	232/2450	10/94
37	1331/2450	1119/2450	0/98
38	1292/2450	1158/2450	5/91
39	1231/2450	1219/2450	13/100
40	1389/2450	1061/2450	6/100
41	1304/2450	1146/2450	5/97
42	1399/2450	1051/2450	0/80
43	1162/2450	1288/2450	5/84
44	1369/2450	1081/2450	0/94
45	1310/2450	1140/2450	1/95
46	2208/2450	242/2450	0/96
47	2376/2450	74/2450	1/100
48	1061/2450	1389/2450	0/98
49	1340/2450	1110/2450	3/100
50	1009/2450	1441/2450	0/100
Totals:	76962/122500	45538/122500	221/4769

TABLE XV
ONE-CLASS TRAINING INPUT; NAIVE BAYES ALGORITHM WITHOUT UPDATING; SEA DATA CONFIGURATION; SEQUENCE SIZE 100; THRESHOLD 6.479.

User	Hits	Misses	False Alarms
1	0/0	0/0	1/100
2	3/3	0/3	0/97
3	3/11	8/11	0/89
4	2/2	0/2	0/98
5	0/0	0/0	0/100
6	0/0	0/0	0/100
7	9/13	4/13	0/87
8	0/0	0/0	0/100
9	12/24	12/24	1/76
10	13/13	0/13	19/87
11	0/0	0/0	1/100
12	0/6	6/6	1/94
13	0/0	0/0	32/100
14	0/0	0/0	0/100
15	0/6	6/6	0/94
16	1/10	9/10	59/90
17	0/0	0/0	0/100
18	6/6	0/6	2/94
19	0/0	0/0	0/100
20	0/0	0/0	22/100
21	0/0	0/0	6/100
22	0/0	0/0	0/100
23	1/1	0/1	0/99
24	21/21	0/21	0/79
25	7/9	2/9	0/91
26	11/13	2/13	3/87
27	0/0	0/0	8/100
28	3/3	0/3	0/97
29	1/1	0/1	10/99
30	3/3	0/3	0/97
31	0/0	0/0	0/100
32	0/0	0/0	0/100
33	0/0	0/0	7/100
34	6/12	6/12	0/88
35	1/1	0/1	0/99
36	6/6	0/6	10/94
37	0/2	2/2	0/98
38	7/9	2/9	5/91
39	0/0	0/0	13/100
40	0/0	0/0	6/100
41	1/3	2/3	5/97
42	10/20	10/20	0/80
43	2/16	14/16	5/84
44	6/6	0/6	0/94
45	1/5	4/5	1/95
46	4/4	0/4	0/96
47	0/0	0/0	1/100
48	2/2	0/2	0/98
49	0/0	0/0	3/100
50	0/0	0/0	0/100
Totals:	142/231	89/231	221/4769

The rest of the tables address the one-class training condition. Tables XIV and XV compare the Naive Bayes classifier results, using a data sequence length of 100, without updating (compensation for concept drift), on the two data configurations used in the study: 1v49 and SEA, respectively. Table XVI shows the confusion matrix (or misclassification matrix) for the 1v49 study. This matrix provided the information for the error analysis, including tallies of the most successful masqueraders, the hardest targets, and the easiest victims. Table XVII charts the exact placement of hits, misses and false alarms for each injection across all 50 users for the one-class training condition, using the SEA data configuration, sequence size 100.

TABLE XVI

ONE-CLASS TRAINING INPUT; NAIVE BAYES ALGORITHM WITHOUT UPDATING, 1v49 DATA CONFIGURATION, SEQUENCE SIZE 100, THRESHOLD 6.479.

SELF	NON-SELF																																																		MISSES
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	
1	0	2	40	47	41	1	40	3	50	9	40	30	40	37	39	28	16	38	33	24	0	37	14	20	48	38	11	34	16	0	37	11	7	22	20	2	46	48	25	41	49	5	44	16	17	2	24	42	31	31	1296
2	0	0	3	3	4	0	4	6	0	1	0	7	4	2	4	23	21	0	1	2	46	5	1	2	2	2	5	8	4	0	21	11	28	0	28	0	19	8	2	4	4	0	5	7	5	11	32	7	7	0	359
3	26	0	0	15	12	1	16	1	3	9	6	9	12	20	27	14	4	0	7	8	1	11	15	4	30	11	1	6	10	0	4	12	10	5	16	0	14	18	1	18	39	13	40	4	6	5	13	18	24	10	549
4	35	3	2	0	44	0	36	14	48	5	32	29	32	42	40	25	16	3	38	16	0	37	10	25	49	29	6	30	14	0	40	12	6	26	4	2	50	50	27	31	28	3	16	22	26	1	29	25	30	22	1110
5	10	3	2	18	0	0	32	3	34	2	17	27	9	39	35	22	6	2	50	16	0	41	6	50	32	16	7	46	11	0	50	12	10	44	5	6	43	46	27	27	29	5	12	13	47	3	30	20	38	25	1028
6	13	0	0	1	0	0	9	0	9	5	0	0	13	5	0	12	0	0	2	3	0	7	1	0	20	7	0	1	4	0	0	0	6	2	9	0	3	3	2	5	1	43	3	0	0	1	12	4	0	6	212
7	36	3	4	44	50	2	0	3	49	4	22	38	37	50	40	27	7	1	50	20	0	50	11	38	47	35	9	50	19	0	48	12	10	44	9	14	44	48	29	35	29	8	16	18	44	2	25	41	41	36	1299
8	2	1	2	38	36	0	9	0	14	0	3	10	8	22	28	11	4	45	18	4	0	23	3	19	6	10	2	22	11	0	28	10	4	9	10	0	16	26	12	29	17	2	9	7	10	0	18	14	22	17	611
9	37	3	2	45	46	1	40	2	0	14	24	29	35	37	36	27	5	1	38	18	0	41	10	27	44	24	7	33	13	0	47	12	6	25	18	0	41	50	28	33	26	5	16	26	23	32	26	38	32	39	1162
10	9	0	2	25	0	3	14	2	26	0	7	0	15	8	0	26	1	0	0	10	10	14	2	1	14	21	3	8	4	0	0	0	11	0	12	0	5	10	6	9	2	4	6	16	1	11	16	12	0	19	365
11	44	3	4	47	47	1	40	3	49	6	0	37	41	35	34	28	17	1	36	22	0	44	13	26	48	34	15	49	15	0	46	13	6	26	4	0	49	48	30	37	29	7	16	23	21	2	25	44	33	36	1234
12	33	13	5	43	48	1	41	14	48	10	36	0	47	48	32	27	18	4	38	22	2	48	21	33	46	47	33	30	49	0	44	14	18	35	27	28	43	46	32	42	32	6	17	39	28	34	36	36	38	42	1474
13	41	2	2	47	41	1	41	14	46	11	35	26	0	30	35	28	11	3	34	25	1	39	14	18	46	42	7	28	18	0	31	14	7	15	6	0	35	43	29	44	25	3	15	34	22	31	25	44	30	41	1180
14	18	3	3	39	48	0	45	4	26	2	15	23	9	0	35	21	7	1	49	14	0	34	6	41	42	22	7	31	20	0	44	12	10	41	4	6	36	45	20	22	25	6	12	15	46	1	29	19	37	18	1013
15	35	3	4	45	50	1	50	45	42	4	24	24	44	39	0	28	10	45	50	21	0	49	11	35	47	42	8	50	16	0	44	13	10	40	28	45	36	48	27	42	28	10	16	12	45	4	33	33	34	29	1399
16	37	38	5	48	44	1	40	11	44	4	28	25	45	31	34	0	43	2	37	26	47	43	17	26	47	41	6	34	22	0	39	14	28	23	26	0	38	48	33	43	25	6	19	19	25	3	28	48	34	33	1358
17	34	43	5	46	43	0	30	4	17	3	26	38	26	32	30	46	0	1	31	20	48	35	14	20	47	33	18	25	20	0	31	14	28	27	23	0	46	48	26	38	24	3	14	42	37	3	29	36	30	30	1264
18	18	10	2	2	6	0	9	42	1	1	5	11	22	3	29	9	3	0	4	12	0	4	0	1	19	4	3	6	6	0	3	11	0	1	2	0	9	12	0	33	11	1	4	3	5	0	18	24	13	8	390
19	8	3	2	28	50	0	30	4	39	3	16	23	3	40	34	18	7	2	0	9	0	45	6	48	20	15	6	50	11	0	49	12	10	44	17	5	43	49	25	23	28	8	12	20	45	2	30	16	40	26	1024
20	39	12	2	39	46	1	32	11	26	3	27	37	41	31	27	24	14	2	30	0	1	36	9	19	39	33	12	29	20	0	37	23	15	23	38	0	38	43	29	43	27	4	14	14	13	6	23	41	32	28	1133
21	0	36	4	0	0	0	0	0	0	1	0	0	0	2	1	23	20	0	0	0	0	0	0	0	0	1	0	0	6	0	0	12	32	0	35	7	1	0	0	6	0	0	1	5	0	0	34	4	0	0	231
22	38	3	5	41	50	1	45	7	50	8	22	37	37	47	39	28	10	3	45	22	1	0	18	33	49	38	13	36	31	0	45	15	7	39	12	1	44	48	44	46	29	6	19	20	32	24	34	41	34	46	1343
23	33	1	50	31	38	0	31	8	20	8	16	36	28	31	33	24	12	2	25	18	2	28	0	21	49	40	10	21	33	0	26	13	10	23	27	1	39	46	20	34	49	7	41	11	33	3	33	26	30	22	1143
24	6	3	2	35	50	0	25	2	32	2	15	23	3	35	33	19	7	1	49	6	0	40	8	0	21	12	5	50	16	0	50	12	11	41	9	6	37	47	25	19	27	7	12	16	44	0	24	15	39	22	963
25	32	4	4	41	42	0	35	8	28	2	19	29	29	39	39	25	5	4	32	18	0	25	14	28	0	29	6	25	14	0	39	12	6	25	15	1	37	46	18	30	28	4	12	10	32	2	23	26	30	14	986
26	31	0	5	31	0	1	28	10	27	4	7	11	45	30	7	18	9	3	1	10	1	20	23	1	37	0	3	5	27	0	1	15	7	0	11	5	11	11	7	20	3	1	8	6	15	12	33	30	2	19	612
27	29	12	5	17	41	0	22	5	40	11	19	39	35	32	32	24	13	2	30	17	0	44	16	23	44	37	0	32	20	0	39	12	6	27	18	1	39	46	30	38	26	4	10	30	19	34	35	37	31	44	1167
28	6	3	4	21	50	1	16	4	31	4	19	17	3	29	30	16	6	1	50	6	0	36	6	46	12	13	7	0	16	0	47	12	8	38	6	6	30	44	26	23	27	6	13	19	38	0	30	18	37	28	909
29	25	6	6	31	38	0	35	9	22	2	23	41	22	50	31	24	5	5	29	15	2	23	22	20	49	42	12	23	0	0	39	14	8	24	24	4	37	45	20	29	26	2	10	10	27	0	32	22	37	13	1035
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	19	3	2	31	50	0	32	4	43	2	17	28	12	44	32	24	6	1	42	16	0	36	6	49	37	16	6	49	11	0	0	12	7	33	6	4	43	44	27	28	28	4	12	14	36	2	30	25	38	23	1034
32	3	0	4	7	0	0	9	1	1	2	0	0	1	8	0	8	3	0	0	23	1	2	1	0	6	21	0	0	9	0	0	0	15	0	2	2	3	0	1	6	0	1	0	0	4	13	24	3	0	1	185
33	4	37	3	0	0	0	5	0	1	15	0	1	0	7	0	31	18	0	2	9	47	0	6	0	9	12	0	2	13	0	0	12	0	0	50	16	2	0	1	0	0	0	1	5	13	19	30	5	9	0	385
34	11	3	2	17	50	0	41	2	27	2	17	27	10	41	33	18	6	1	50	16	0	35	7	50	20	21	6	42	11	0	50	12	10	0	5	48	43	42	27	28	27	8	12	12	40	1	28	21	39	24	1043
35	0	39	3	1	0	0	0	0	0	16	0	0	0	4	1	23	16	0	0	8	43	0	1	0	0	8	1	0	8	0	0	11	36	0	0	0	1	0	2	0	0	0	0	6	0	34	26	2	0	1	291
36	3	1	1	10	0	0	12	0	2	0	0	0	2	13	1	7	1	0	0	0	0	0	9	3	23	14	0	2	9	0	0	10	22	0	21	0	1	1	0	0	2	0	2	0	22	2	30	0	6	0	232
37	35	3	2	45	48	0	47	4	43	2	28	33	23	48	34	24	16	1	43	20	0	39	10	31	45	25	12	27	8	0	44	11	6	40	4	0	0	47	26	26	29	6	12	25	40	2	25	26	32	22	1119
38	36	7	4	43	46	1	48	3	49	5	22	31	33	42	35	27	9	2	45	21	0	44	12	29	48	28	13	32	14	0	42	12	6	39	7	1	44	0	29	33	29	5	13	13	28	2	29	39	33	25	1158
39	33	3	4	40	46	1	41	8	43	4	22	33	40	29	34	42	15	4	38	18	17	48	16	27	44	31	10	33	19	0	43	14	6	30	18	1	42	48	0	39	28	6	17	10	18	6	29	45	37	39	1219
40	33	3	5	25	42	1	35	47	19	2	20	24	39	39	35	16	7	45	29	23	1	41	4	19	32	42	9	29	18	0	40	14	6	26	4	1	29	29	29	0	26	4	14	11	13	0	28	42	29	32	1061
41	36	3	44	37	44	0	36	3	47	14	19	31	23	33	31	24	6	1	31	18	0	32	15	25	45	21	8	26	12	0	41	12	6	28	14	2	42	46	24	27	0	4	45	22	16	31	34	26	32	29	1146
42	28	2	15	38	43	50	40	9	29	7	11	20	23	34	30	24	8	1	37	20	1	29	15	28	41	24	4	26	14	0	33	10	8	31	11	0	30	43	18	23	29	0	34	13	26	2	16	23	31	19	1051
43	45	20	48	39	45	1	33	8	37	8	14	21	38	28	31	47	29	1	32	17	47	43	16	22	42	40	7	35	16	0	39	12	25	24	16	0	31	44	26	38	49	8	0	15	18	2	26	41	31	33	1288
44	5	28	4	38	45	0	15	3	38	11	16	26	1	44	27	35	34	0	36	9	38	29	6	28	18	9	14	26	7	0	40	11	25	27	38	0	41	46	21	19	27	4	12	0	34	41	30	17	33	25	1081
45	30	3	4	35	49	0	49	3	25	3	17	25	24	39	38	24	8	3	49	17	0	36	14	50	46	30	7	33	19	0	46	13	18	40	21	13	35	44	18	27	28	4	11	17	0	8	35	24	40	18	1140
46	18	0	1	7	0	0	11	0	12	8	0	0	13	5	4	8	1	1	0	1	0	3	0	0	24	17	0	0	3	0	0	12	14	0	26	0	4	5	0	2	2	0	0	4	1	0	20	6	1	8	242
47	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	4	0	0	10	4	0	13	1	1	0	0	2	1	0	0	0	0	31	0	0	0	0	74
48	39	28	5	42	46	1	42	10	50	12	28	33	41	39	33	49	34	3	34	27	48	46	24	23	48	42	8	32	25	0	42	15	25	27	20	1	42	47	37	48	29	5	20	20	15	4	32	0	34	34	1389
49	33	3	6	36	48	0	46	3	25	2	16	28	18	39	32	24	8	1	43	20	0	33	8	44	49	18	7	26	17	0	44	12	23	40	32	7	43	46	24	26	29	5	13	12	38	10	31	23	0	19	1110
50	38	2	5	46	50	2	42	13	47	16	29	34	44	45	38	28	18	4	42	22	1	49	33	33	47	48	34	49	24	0	41	15	6	27	18	2	37	47	43	45	28	7	18	42	38	34	36	41	33	0	1441
Total	1124	402	339	1405	1617	74	1379	360	1359	270	779	1051	1070	1427	1254	1128	541	242	1360	709	406	1404	494	1086	1578	1186	368	1231	727	0	1474	579	593	1081	789	239	1443	1649	953	1261	1084	250	668	718	1106	473	1318	1190	1244	1056	

TABLE XVII
One-Class Training Input; Naive Bayes Algorithm Without Updating, SEA Data Configuration, Sequence Size 100, Threshold 6.479.
Key: False Alarm:!/Miss:-/Hit:*.

User	Test Block																																																																																																			FA	Miss	Hits
	0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2	3	4	5	6	7	8	9	40	1	2	3	4	5	6	7	8	9	50	1	2	3	4	5	6	7	8	9	60	1	2	3	4	5	6	7	8	9	70	1	2	3	4	5	6	7	8	9	80	1	2	3	4	5	6	7	8	9	90	1	2	3	4	5	6	7	8	9			
1																																													!																																																							1		
2																																																											*	*	*																																									3
3																																							-	*	-	-	-	-	-	-	*	*	-																																																			8	3
4																																																									*	*																																											2
5																																																																																																					
6																																																																																																					
7																								-	-	-	*	*	*	*	*	*	*	*	*																				-																																														4	9
8																																																																																																					
9																											!				*	-	-	-	-	-	-	-	*	*	*	*	*	*	*	*	*	*	*	-	-	-																			-	-																																	1	12	12
10									!	!	!	!			!	!	!	!		!	!	!	!	!	!		!	*	*	*	*	*	*	*	*	*	*	*	*	*			!			!																																															!															!				19		13
11																																																																									!																													1		
12																															!																																												-	-	-	-	-	-																												1	6	
13										!		!							!																																					!	!	!		!	!			!		!			!					!	!	!	!	!	!	!	!	!	!		!	!		!	!	!	!			!	!							!	!		!		32		
14																																																																																																					
15																																																																																											-	-	-	-	-	-		6	
16						!	!	!	!	!	!	!	!	!	!	!	!	!	-	-	-	-			!	!	!	!	!	!				!	!	!	!	!		!	!		!	!	!	!		!	!	!	!	!			!			!		!	!	!		!	!	!	!		!	!		!	!	!	!	!		!	!	-			!	-	-	-	*	-					!			!		!	!	!	59	9	1
17																																																																																																					
18									*	*	*	*	*	*					!	!																																																																															2		6
19																																																																																																					
20								!			!		!																																						!	!	!				!	!	!	!	!	!	!					!	!	!				!	!	!										!	!	!																					22		
21																																															!	!													!	!	!										!																																6		
22																																																																																																					
23																																																	*																																																					1
24																																																																					*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*													21
25																																																	-	*	*																													-																	*	*	*	*	*							2	7
26																																			!				!									!											*	-	*	*	*	-	*	*	*	*								*	*	*																													3	2	11
27																																																										!	!	!	!	!	!	!	!																																												8		
28																																																																																																		*	*	*			3
29																*		!																																									!										!	!	!		!		!	!	!			!																															10		1
30																																	*	*	*																																																																							3
31																																																																																																					
32																																																																																																					
33																																																				!	!	!	!	!																!	!																																7		
34										*	*	*	*	*																																									*	-	-	-	-	-	-																																													6	6
35																																																																																																				*			1
36																														!								!	!	!											*	*	*	*	*	*														!	!	!				!	!	!																														10		6
37																																																																																			-	-																				2	
38																																											-	*	-										!	!	!	!	!																									*	*	*	*	*	*																						5	2	7
39		!																																																														!	!	!	!	!	!	!																											!	!	!	!				!	13		
40																															!										!																																																							!	!				!	!	6		
41																		-	-	*																																																																												!	!	!	!	!						5	2	1
42																																																															*	*	-	-	-	-	-	-	-	-	-	*	*	*	*	*	*	*	*	-																					10	10
43																												-	-	-	-	-	-	-	-	-	-	-	-	-	*	-									!						!							*																																		!			!	!				5	14	2
44																																	*										*	*	*	*	*																																																							6
45																									-	*	-	-	-																																																																								1	4	1
46																																																																			*	*	*	*																																			4
47																																													!																																																										1		
48																																																																																													*	*												2
49																													!																																										!	!																															3		
50																																																																																																					
																																																																																																						221	89	142

ACKNOWLEDGMENT

The Defense Advanced Research Projects Agency (DARPA) supported this work under Contracts F30602-99-2-0537 and F30602-00-2-0528. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force, or the U.S. Government.

P. Jia was involved in the early stages of the project, contributing initial ideas and analyses. R. I. Kondor (now at Columbia University) helped with a visualization system for interpreting Naive Bayes classification results. M. Louie helped to work out the shared-tuple analysis, and K. Killourhy helped with data management. D. Banks (Duke University) provided helpful comments over the course of many discussions. The authors are grateful for all of these contributions, as well as for the comments of anonymous reviewers.

REFERENCES

[1] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," *Statistical Science*, vol. 16, no. 1, pp. 58–74, Feb. 2001.

[2] T. F. Lunt, "A survey of intrusion-detection techniques," *Computers & Security*, vol. 12, no. 4, pp. 405–418, June 1993.

[3] V. Loeb, *Spy Case Prompts Computer Search* Washington Post, DC, Mar., 5 2001, pp. A01–00.

[4] A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," *Learning for Text Categorization*, pp. 41–48, 1998. Papers from the 1998 AAAI Workshop, 27 July 1998, Madison, Wisconsin: published as AAAI Technical Report WS-98-05.

[5] M. Schonlau and M. Theus, "Detecting masquerades in intrusion detection based on unpopular commands," *Information Processing Letters*, vol. 76, no. 1–2, pp. 33–38, Nov. 2000.

[6] W. DuMouchel, "Computer Intrusion Detection Based on Bayes Factors for Comparing Command Transition Probabilities," National Institute of Statistical Sciences, Research Triangle Park, Technical Report 91, 1999. NC 27 709-4006.

[7] W.-H. Ju and Y. Vardi, "A Hybrid High-Order Markov Chain Model for Computer Intrusion Detection," National Institute for Statistical Sciences, Research Triangle Park, Technical Report 92, 1999. NC 27 709-4006.

[8] B. D. Davison and H. Hirsh, "Predicting sequences of user actions," *Predicting the Future: AI Approaches to Time-Series Problems*, pp. 5–12, 1998. Papers from the 1998 AAAI Workshop, 27 July 1998, : published as AAAI.

[9] T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 295–331, Aug. 1999.

[10] P. Langley, W. Iba, and K. Thompson, "An analysis of bayesian classifiers," in *Proceedings of 10th National Conference on Artificial Intelligence*. Los Alamitos, San Jose, California, 1992, pp. 223–228.

[11] B. Cestnik, I. Kononenko, and I. Bratko, "Assistant-86: A knowledge-elicitation tool for sophisticated users," in *Progress in Machine Learning*, I. Bratko and N. Lavrac, Eds, Wilmslow, UK: Sigma Press, 1987, pp. 31–45. Proceedings of EWSL 87: 2nd European Working Session on Learning. Bled, Yugoslavia May 1987.

[12] P. Domingos and M. Pazzani, "Beyond independence: Conditions for the optimality of the simple Bayesian classifier," in *13th International Conference on Machine Learning (ICML-96)*, L. Saitta, Ed.. San Francisco, Bari, California, Italy, 1996, pp. 105–112. 03-06 July 1996.

[13] S. Kotz, N. L. Johnson, and C. B. Read, Eds., *Encyclopedia of Statistical Sciences*. New York, NY: Wiley, 1985, vol. 5.

[14] C. D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT Press, 2000. 1999, Second printing, with corrections.

[15] T. M. Mitchell, *Machine Learning*. Boston, Massachusetts: McGraw-Hill, 1997.

[16] J. A. Swets and R. M. Pickett, *Evaluation of Diagnostic Systems: Methods from Signal Detection Theory*. New York, NY: Academic Press, 1992.

[17] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Wiley, New York, 2001.

Roy A. Maxion received the Ph.D. degree from the University of Colorado, Boulder, in 1985. He is a Principal Systems Scientist on the faculty of the Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, and Director of the Dependable Systems Laboratory, Carnegie Mellon University.

He is also on the faculty of several Carnegie Mellon University institutes and centers: Center for Computer and Communications Security (C3S); Institute for Complex Engineered Systems; Human-Computer Interaction Institute; and Center for Automated Learning and Discovery. His research interests include all aspects of dependable systems: fault tolerance and reliability, tolerance of malicious faults (intrusion detection), survivable systems (information warfare defense), dependable software, the interface between cognitive science and dependability (e.g., dependable user interfaces for mission-critical applications), and dependable documentation. His central interest is in automated diagnosis and anomaly detection for fault localization in various domains, including semiconductor fabrication, computer security (intrusion, masquerader/insider and fraud detection), computer hardware and software, and self-healing, autonomic systems. He has authored over three dozen papers and book chapters, and has consulted for both industry and government, including the Pentagon and the US Department of State.

Dr. Maxion was Program Co-Chair of the 2001 International Conference on Dependable Systems and Networks (DSN), Program Committee Member for the 2000 Information Survivability Workshop, and Program Committee Member for RAID-2001, 2002 and 2003. He is a member of the IEEE Computer Society, and is active in the International Federation of Information Processing (IFIP) 10.4 Working Group on Dependability. He served on the US Defense Science Board, Task Force on Technology for Information Warfare Defense, as well as on the 2003 Defense Science Board on Roles and Missions of the DoD in Homeland Security, focusing on information sharing and analysis. He is on the DSN steering committee, and on the executive board of the IEEE Technical Committee on Fault Tolerance. He is Vice-President of Professionals for Cyber Defense.

Tahlia N. Townsend holds an Honors B.Sc. in physics and chemistry from The Open University, Milton Keynes, United Kingdom. At the time this paper was done, she was a Visiting Scientist in the Dependable Systems Laboratory of the Computer Science Department at Carnegie Mellon University. She is currently a JD student at Yale Law School, New Haven, Connecticut.