International Conference on Dependable Systems & Networks: Bethesda, Maryland, 23-26 June 2002.

Masquerade Detection Using Truncated Command Lines

Roy A. Maxion and Tahlia N. Townsend
maxion@cs.cmu.edu and tahlia@cs.cmu.edu

Dependable Systems Laboratory
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 / USA

Abstract

A masquerade attack, in which one user impersonates another, can be the most serious form 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. While the success of this approach has been limited, the reasons for its unsatisfying performance are not obvious, possibly because most reports do not elucidate the origins of errors made by the detection mechanisms. This paper takes as its point of departure a recent series of experiments framed by Schonlau et al. [12]. 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%. A detailed error analysis, based on an alternative data configuration, reveals why some users are good masqueraders and others are not.

1. Introduction

Colloquially, the masquerade problem can be described in 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 (or some other computer identity, such as a program), s/he is commonly known as a masquerader. It should be noted that there are many ways for a masquerader to gain access to legitimate user accounts, e.g., through a purloined password or a hacker's break in. The term may also be extended to encompass abuse of legitimate privileges – the case in which a user "masquerades" as himself; such a person is sometimes termed an "insider," especially when the person is an accepted member of the organization sponsoring the target system.

Masquerading can be a serious threat to the security of computer systems and computational infrastructures. A well-known instance of masquerader activity is the recent case of Robert P. Hanssen, the FBI mole who allegedly used agency computers to ferret out information later sold to his co-conspirators [6]. Hanssen was a legitimate user, but his behavior was improper. More than 60% of companies surveyed in 1996 reported some sort of misuse, at enormous cost. Of even greater concern than economic losses, of course, are attacks on national security.

Detecting masqueraders has long been a challenge, dating as far back as 1988 for practical detection systems [8]. The typical approach is based on the idea that masquerader activity is unusual activity that will manifest as significant excursions from normal user profiles. User profiles are constructed from monitored system-log or accounting-log data. Examples of the kinds of information 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, user commands issued, and so forth [7]. When a deviation from normal behavior is observed, a masquerade (or other misuse) attempt is suspected. To facilitate comparison with other work, this study employs truncated user-command lines (no arguments) as data.

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 [12].

In terms of minimizing false alarms, the best result achieved in that work used a uniqueness metric for a 39.4% hit rate at a 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 (69.3% hits and 6.7% false alarms). Although these results may seem disappointing, they are in fact quite good, considering the difficulty of the problem. One of the best aspects of the Schonlau et al. work is the data set itself. User data are extremely hard to obtain, due to concerns about user privacy and corporate confidentiality. That Schonlau and his colleagues made their data available is a major contribution.

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 with a 56% improvement in correct detection at the lowest false alarm rate reported in the literature so far. It also amplifies the results with an analysis of the errors made by the detector as applied to an alternative configuration of the Schonlau et al. data set. The error analysis exposes certain limitations of the Schonlau et al. framework, and suggests what could be done to achieve better results in the future.

2 Problem and approach

This paper addresses the problem of masquerade detection as exemplified in the task of classifying sequences of user-command data into either of two categories: self (the legitimate user) and nonself (the masquerader). The data used are those of Schonlau et al., described in Section 5. These data consist of 15,000 commands from each of 50 different users. The data are configured in two ways: (1) randomly injected with data from users outside the community of 50 (which approximate an incursion by a masquerader); and (2) each user crossed with every other user to compare the effects of every user acting as a "masquerader" against all other users. The first configuration is that employed by Schonlau et al., hereafter referred to as the SEA configuration; the second configuration contrasts with the first in that (1) it provides a consistent set of intrusions against all users, which allows meaningful error analysis, and (2) it samples a greater range of intrusive behavior (2450 sequences from 49 masqueraders per victim, as opposed 0-24 sequences from a maximum of three different masqueraders per victim under the SEA regime).

A new detection algorithm, inspired by Naive Bayes text classification, was used. This algorithm was chosen because the self/nonself classification task is similar to the generalized task of text classification, and the Naive Bayes classifier has a history of good performance in that context [10].

Two objectives dominated the study: first, to determine whether or not a new classifier could improve upon the detection rates reported in previous work; second, to provide a detailed examination of whatever classification errors occur in the study. An error analysis facilitates more understandings 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 next section provides a survey of recent literature concerned with the particular data and problem at hand. Descriptions of the Naive Bayes classifier and the data are given, followed by an overview of issues involved with the classification of user-command data into self and nonself categories. The experimental methodology is presented, afterwhich are the results, error analysis and discussion.

3 Recent literature

The point of departure for the work described in this paper lies with the efforts of Schonlau and his colleagues, who have provided a nice retrospective of their accomplishments, rendered over the course of several papers, in [12]. These authors used keyboard command data from 50 users, injected with data from users outside the community of 50. Data for each user comprised 15,000 commands. The first 5,000 commands constituted ground truth (i.e., contained no injected data), and the last 10,000 commands were probabilistically injected with commands issued by another user. The idea was to detect blocks of 100 commands typed by the "masquerader," discriminating them from blocks of 100 commands typed by the true user. Details of the data, how they were gathered, and how the original experiments were conducted can be found in Sections 5 and 6.

The review paper by Schonlau et al. [12] describes an experiment in which six masquerade-detection schemes are compared on the same user data seeded with "masqueraders." Some of the detection techniques were original, while others were drawn from the computer science literature. The investigators targeted a false-alarm rate of 1%. All of the methods had relatively low hit rates (ranging from 39.4% to 69.3%) and high false alarm rates (ranging from 1.4% to 6.7%). The results are also compared using cluster analysis and ROC curves, which reveal that no method completely dominates another. This section provides an overview of the various masquerade detectors used by Schonlau et al. Their results are shown graphically in Figure 1, along with the results of the present study.

Bayes 1-Step Markov. This detector is based on single-step transitions from one command to the next, and is due to DuMouchel [3]. 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, but failed to get close to the desired false alarm rate.

Hybrid Multi-Step Markov. This method is based on Markov chains, and is due to Ju and Vardi [4]. The implementation of this model in [12] actually toggled between a Markov model and a simple independence model, depending on the proportion of commands in the test data that were not observed in the training data. The performance of this method, shown in Figure 1 was among the highest of the methods tested.

IPAM. This detector is based on single-step command transition probabilities, estimated from training data. IPAM (incremental probabilistic action modeling) was developed by Davison and Hirsh [1] for their work in predicting sequences of user actions. It is described in [12], from which the result depicted in Figure 1 is drawn. IPAM's performance ranks with those in the lowest-performing group.

Uniqueness. This approach, due to Schonlau and Theus [13], is based on ideas about command frequency: commands not seen in the training data may indicate a masquerade attempt, and the more infrequently a command is used by the user community as a whole, the more indicative that command is of being a masquerade. Uniqueness was a relatively poor performer in terms of detecting masqueraders, but was the only method able to approach the target false alarm rate of 1% [12].

Sequence-Match. This approach is based on the early work of Lane and Brodley, refined in [5]. This method computes a similarity match between the most recent user commands and a user profile. The technique is reviewed in [12], and the result portrayed in Figure 1 is taken from there. On the Schonlau et al. data, it was a poor performer.

Compression. The idea behind the compression approach is that new data from a given user compresses at about the same ratio as old data from that same user, and that data from a masquerading user will compress at a different ratio and thereby be distinguished from the legitimate user. This idea is credited to Karr and Schonlau in [12] whose result is depicted in Figure 1. Compression was the worst performer of the methods tested.

4 The Naive Bayes classification algorithm

Naive Bayes classifiers are simple, probabilistic classifiers known for their inherent robustness to noise and their fast learning time (learning time is linear in the number of examples). They have a history of successful use in text classification, where 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 [10]. Deciding whether a newspaper article is about sports, health or politics, based on the counts of words in the article, is similar to the task of deciding whether or not a stream of commands issued at a computer terminal belongs to a particular authorized user. Despite its simplicity and success in text classification, hitherto the Naive Bayes approach has not been applied to user profiling, with command-line data, for masquerader detection.

In the present context, the classifier works as follows. The model assumes that the user generates a sequence of commands, one command at a time, each with 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 seen in the training data, and is given by:

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

where α is a pseudocount and A is the number of distinct commands (i.e., the alphabet) 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. The probability that a test sequence of the five commands "a a b b b" was generated by a particular user, say User 1, $u1$, is:

$$P_{u1,a} * P_{u1,a} * P_{u1,b} * P_{u1,b} * P_{u1,b}$$

or $(P_{u1,a})^2 * (P_{u1,b})^3$ where $P_{u1,a}$ is the probability that User1 typed the command a. 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 cut-off for classification as X, that is the ratio of probabilities below which the likelihood that the sequence was generated by X is deemed too low, can be determined by a cross-validation experiment during which probability ratios for sequences which are known to have been generated by self are calculated, and the range of values these legitimate sequences cover is examined.

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, [2] demonstrates that Naive Bayes can be optimal even when this assumption is violated. Further details regarding Naive Bayes can be found in [9] and [11].

5 Data

The data used in the present work were the same as the data used in the several Schonlau et al. [12] masquerade-detection studies. As described by Schonlau et al., user commands were captured from the UNIX acct auditing mechanism. Only two fields were used – command name and user. This limitation was imposed for privacy reasons. Examples of commands are: `sed, eqn, troff, dpost, echo, sh, cat, netstat, tbl, sed, eqn, sh` and so forth. The first 15,000 commands for each of about 70 users were recorded over a period of several months. Some users generated 15,000 commands in a few days; others took a few months. Some commands not explicitly typed by the user (e.g., those generated by shell files or scripts) were also included, as were names of executable programs.

The data used in this study were obtained from Schonlau et al., who graciously made their data available for download at http://www.schonlau.net.

6 Experimental method (Schonlau et al.)

This section describes the experimental method used by Schonlau et al. This description is drawn from [12].

"We randomly selected 50 users [out of the 70 from whom data were collected] to serve as intrusion targets. We then used the remaining 20 users as masqueraders and interspersed their data into the data of 50 users. ...

For simplicity, because it facilitates presentation of the results, we decided to decompose each user's data into 150 blocks of 100 commands each. ... The first 50 blocks (5000 commands) of all users are kept aside as training data – as far as we know they are not contaminated by masqueraders. For blocks 51 through 150 we made the simplification that a block is contaminated either completely or not at all; there are no mixed blocks.

Starting with block 51, we insert masquerading data as follows: if no masquerader is present, a new masquerader continues to be present in the following block with a 1% probability. If a masquerader is present, the same masquerader continues to be present in the following block with a probability of 80%. While the exact values of the probability are arbitrary, they reflect the requirements that (1) there are an arbitrary number of masqueraders in the data (including the possibility of none), (2) the length of the masquerading varies and (3) most of the data are not contaminated.

Data that correspond to different masqueraders are always separated by at least one block of uncontaminated data. Inserting masquerading data increases the number of commands observed. We truncate the data to 150 blocks per user in order not to give away the amount of masquerading data inserted.

Masquerading data are drawn from the data of masquerading users as follows: we determine the length of the masquerade and choose a masquerader and a start data block at random. The random choice is repeated if there are not enough contiguous masquerading data left or if the masquerading data were previously used."

The various masquerade-detection techniques employed by Schonlau et al. were applied directly to the data sets that emerged from the procedure just described. For each user, the user's first 5,000 commands were used as training data; the last 10,000 (masquerader-injected) commands were used as testing data for that same user.

7 Experimental method (revised)

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 given masqueraders. Although this may be faithful to real-world conditions, failure to run a consistent set of masqueraders against all users in a test setting makes it difficult to draw constructive inferences from the hit and miss rates.

When an algorithm fails to identify a masquerader block which is only found in one user's data, it is not clear whether the failure is due to characteristics of the user, the masquerader or the algorithm. This makes it hard to improve future performance. Furthermore, the number of masquerade events was different for different users, and in the majority of cases, where there were multiple masquerade events, they were largely drawn from the same masquerader. 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.

Recall that the objectives of the present study were twofold: to effect a comparison between the Naive Bayes classifier and the classifiers used in Schonlau et al.; and to carry out an analysis of classification errors (misses and false alarms). These goals required two different experiments, SEA and 1v49 respectively, as sketched below.

SEA experiment. This experiment followed the same methodology, and used the same data configuration, as did Schonlau et al., hence the moniker "SEA." Its goal was to deploy and test a new classifier (Naive Bayes) that is possibly better suited to what is very reminiscent of a text classification task than the classifiers used by Schonlau et al., and to produce results that could be compared easily with those achieved by previous researchers using the same data but different classifiers.

A separate Naive Bayes classifier was built for each of the 50 users. During the training phase, the classifier built a profile of self and a profile of non-self, using the first 5000 commands of a given user's data for the former profile, and the 49 x 5,000 training data commands of the other 49 users for the latter profile. The test data were identical to those used in the Schonlau et al. original experiments. Two versions of the Naive Bayes classifier were run on this data configuration; one was the simple, unadorned version typically used in the literature, and the other was enhanced with an updating scheme whose purpose was to accommodate drift in the data due to changes over time.

1v49 experiment. To address the methodological shortcoming discussed at the beginning of this section, a new test set was employed to facilitate error analysis in the present study. This configuration was consistent in the number and origin of the masquerade events encountered by each detector, i.e., if the detector that was trained on UserX encountered masquerade events from 5 different masqueraders, the detector trained on UserY would encounter the same 5 events. Using this configuration, the Naive Bayes classifier was trained (constructed a user profile) on the first 5,000 commands of a given user, but tested on the first 5,000 commands of each of the other 49 users as if all 49 of those users were masqueraders. The data configuration is therefore referred to as the 1v49 configuration. It resulted in 2450 masquerade blocks for each user compared with between 0 and 24 for Schonlau et al. The test of self-recognition remained the same as in the Schonlau et al. paradigm, with between 76 and 100 blocks of self data culled from the second 10,000 commands being presented to the detector. Note that the detector for this experiment differs from the one used with the SEA data in that it trains on positive examples only, i.e., it does not build a profile of nonself. The training data for making a nonself profile in the SEA experiment is used as testing data in the 1v49 experiment. The 1v49 data configuration was used with the simple Naive Bayes classifier (described in Section 4) to obtain results which were amenable to error analysis.

8 SEA results

In assessing the results of a masquerade detector we are 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.[1] The ROC curve for Naive Bayes, with updating, on the SEA data is shown in Figure 1 along with the superimposed results of the various Schonlau et al. experiments reviewed in Section 3. Note that the unit of classification, employed by both Schonlau et al. and the present study, is a block of 100 contiguous commands.

The curve itself shows results for a Naive Bayes classifier with updating applied to the SEA data configuration as the decision threshold was stepped through its range. 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 the 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 is a block of 100 consecutive user commands. For the SEA data configuration there were 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.

¿From the figure it may appear that the Schonlau et al. method of Bayes One-Step Markov classification produced the best results. This is certainly true when considering hit rate alone (proportion of masqueraders correctly identified as masqueraders). However, the false-alarm rate is also important, because in a real-world situation false alarms must be investigated, and these investigations have a cost. Consequently a common goal in masquerade-detection systems (and intrusion-detection systems) is to reduce the false-alarm rate as far as possible; Schonlau et al. had a target level of 1%. When false alarms are considered, the Bayes One-Step Markov method, with a 6.7% false alarm rate looks less promising; the Naive Bayes result dominates.

Although the quality of a detector may be judged in terms of its ability to maximize hits and to limit either false

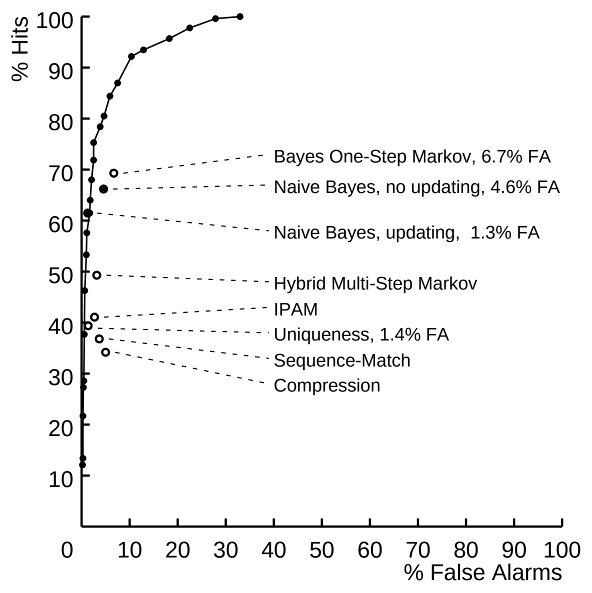

Figure 1: Relative operating characteristic (ROC) curve for the Naive Bayes classifier (with updating) as applied to the SEA-configured data. Also included are best-outcome results achieved by other methods as described in the text.

[1]For a thorough exposition of ROC curves, see [14].

alarms or missing alarms, this can sometimes be too simplistic. In reality there is often a bias favoring either false alarms or misses. Therefore, the overall “goodness” of each of several detection methods can be ranked by applying a cost function of the form:

$$Cost = \alpha(Misses) + \beta(FalseAlarms)$$

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:

$$Cost = Misses + FalseAlarms$$

By this measure, the best methods are: Bayes One-Step Markov, followed by Naive Bayes (without and with updating, respectively), Hybrid Markov, IPAM, Uniqueness, Sequence Matching and Compression. The consensus in the literature, however, seems to be that a false alarm should be more expensive than a miss; what is not clear is how much more expensive. In the Schonlau et al. work, researchers were asked to target a false alarm rate of 1%; the only successful method by this criterion was Uniqueness. In order for Uniqueness to emerge as the top of those 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

$$Cost = (Misses) + 6(FalseAlarms)$$

Table 1 presents the detection methods ranked according to the latter cost function: false alarms are six times as costly as misses. When false alarms and misses weighted in this way, the best methods are Naive Bayes, Uniqueness, Hybrid Markov, Bayes One-Step Markov, IPAM, Sequence Matching and Compression. The Naive Bayes method with updating, using a threshold extracted from the training data by cross validation, obtained a false alarm rate of 1.3% with a concomitant hit rate of 61.5%. This constitutes an improvement of 56% over Uniqueness, the best result achieved by Schonlau et al. (hit rate 39.4%, false alarm rate 1.4%).

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

Table 1: Ranking of classification methods, using SEA data configuration and Cost = Misses + 6 * (False Alarms) as a ranking function.

9 1v49 results and error analysis

The rankings provided in Table 1 suggest which of the several classifiers/detectors might be the best performer in a given situation, but they don’t reveal anything about the sources of classification error. The 1v49 experiment, described in Section 7 was designed explicitly to investigate the Naive Bayes classification errors, and provide insight into the questions: what makes a successful masquerader; what makes a user an easy or a hard target; what causes false alarms?

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. The primary tool emerging from the 1v49 experiment is a confusion matrix, an excerpt of which is depicted in Table 2 (the full table is not shown due to space limitations). The number in each cell of the table indicates the number of missed detections when one user (victim/self) is intruded upon by another (intruder/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, while the intruders (nonself) are listed across the columns. Note that the numbers along the diagonal, as expected, are all zeros.

The far right column 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 1 showed the greatest susceptibility as a victim, with 1295 missed detections.

Examination of the full table reveals several interesting observations, some of which will be addressed in detail here. 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.

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	···	

Table 2: Confusion matrix (excerpt): simple Naive Bayes classifier with updating, 1vs49 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.

9.1 What makes a successful masquerader?

The top masqueraders were users 38, 05 and 25 with 67% (1649/2450), 66% (1617/2450) and 64% (1578/2450) success rates respectively. All were successful across the board, that is, more than half the users falsely accepted 35 or more of these masqueraders' blocks as self. A careful analysis of these users' data reveals that they succeed by virtue of one or more of the following attributes: high use of popular shared tuples (see Section 9.1.1); moderate use of very popular shared tuples; absence of unusually heavy use of any particular command or unpopular tuple.

It is instructive to examine a scattergram of the data for the most successful masquerader, User 38, shown in Figure 2. For purposes of graphing the data this way, each distinct command in the data set was converted to a unique integer. The scattergram shows all 15,000 commands for User 38, with a vertical dashed line dividing the 5,000 training-data commands on the left from the 10,000 testing-data commands on the right. Notice the dense patterns of wavy lines that hug the x-axis; these represent repeating sequences of commands. These repeating sequences occur not just for User 38, but are common to, or shared by, many other users as well. This observation led to an exploration of the extent to which such repeating sequences were shared among all the users, and the extent to which such sharing might affect classification outcomes.

9.1.1 Shared tuples

Investigation of the training data as a whole reveals that sequences of at least two commands (2-tuples) occurring in the data of at least ten users account for 34.8% of the training data (after removal of overlaps). The amount of training data accounted for by these shared tuples ranges between 0 for User 30, the least successful masquerader, and 70.5% for User 38, the most successful masquerader. The longest such tuple contained 49 commands, and was shared by 18 of the 50 users. An examination of the relationship between shared tuples and masquerade success discloses that the more shared tuples, the greater the user's chance of success as a masquerader. Plotting the number of shared tuples occurring in the data of a user against that user's success as a masquerader, reveals that 52% of the variation (r^2 = .52, ρ = .72) in ability to masquerade can be explained by the amount of the user's data that is made up of shared tuples. The correlation is not perfect, because not only the number of shared tuples, but also the popularity of the tuples affects masquerader success. For example, although a large proportion of the training data for User 21 is comprised of shared tuples, this user does rather badly as a masquerader, because 37% of its training data is made up of a single 5-tuple which is only used by 12 other users. Shared tuples, arising perhaps from scripts built in to the computing environment, are effective camouflage for the masquerader.

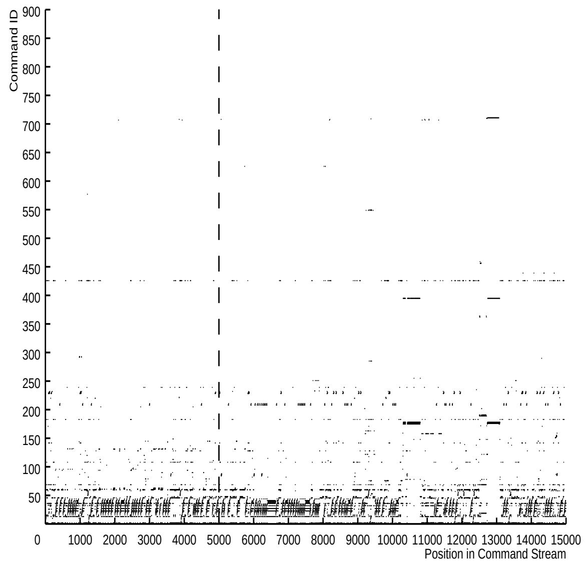

Figure 2: User 38: scattergram representation of commands. Vertical dashed line indicates division between training data (left of line) and testing data. Note repetitious patterns of command sequences along the x-axis.

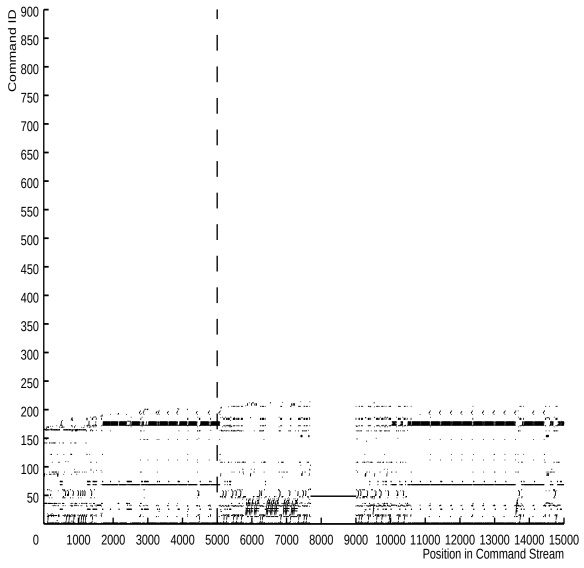

Figure 3: Concept drift illustrated by User 10: scattergram representation of commands. Vertical dashed line indicates division between training data (left of line) and testing data. Note regions of test data bearing no resemblance to any portion of training data.

9.2 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 high frequency 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 is 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, 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.

9.3 What makes an easy victim?

The users with the highest miss rates were users 12, 50 and 15. These users suffered 60.1% (1474/2450), 58.7% (1441/2450) and 57.1% (1399/2450) misses respectively. If we model average user behavior with a vector of average command frequencies, and calculate the Euclidean distance between this average vector and the frequency vector for each user's own training data, we discover 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 sixth in terms of miss proneness. In other words, what these users have in common is their "averageness".

9.4 What causes false alarms?

Concept drift, sometimes called nonstationarity, lies behind the majority of the false alarms observed in the 1vs49 experiment. 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 and false alarms will be raised.

Concept drift is dramatically illustrated in Figure 3, showing a scatterplot of position in the datastream vs. 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 scatterplot coincide with the blocks which give rise to false alarms.

A combination of exploiting information about non-self (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, partly due to the exact location of block boundaries relative to the onset of new behavior patterns, and partly due to the fact 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 non-self model. This is another illustration of the dangers posed by the presence of shared tuples.

10 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:

Data. The poor performance of every method tried so far on this data may indicate that command line data alone, without arguments, is not enough to profile a user. Useful additions might be: arguments to commands; type and length of sessions; user categories (job-type, department).

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; and, commands issued in different terminal windows are interleaved.

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 with the block-100 results (61.5% hits and 1.3% false alarms).

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 control 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 timestamps for logins and for each user command issued, so that one day can be differentiated from another; shared scripts, such as system login scripts, should be removed, because they 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 of each user (e.g., programmer, researcher, manager, secretary, sysadm, etc.). Finally, the features in the data might be improved, using latent features instead of, or in addition to, the raw features.

Methodology. The particular way an experiment is done can have dramatic effects on its outcome; small methodological details, accidentally overlooked, can impede others 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.

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.

Classifier. It is interesting to ask why Naive Bayes performed so much better than its competitors. Unfortunately, the answer to this question is not at all clear. 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 one thing 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, whereas 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.

11 Conclusion

The present study achieved a 56% improvement over previous results. It has emphasized careful data collection, attention to experimental methodology, and error analysis, the latter of which has led to insights about why some masqueraders are harder to detect than others. Nevertheless, masquerade detection remains among the most challenging of classification problems.

Data are difficult to obtain, and typically provide only positive training examples (for the case of determining whether or not a command sequence is from a given user, as opposed to determining which of many pre-identified users most closely matches a sample). No one really knows what a masquerader looks like, because real-world examples of masquerade attacks are generally not available for analysis. A detector can be given an idea of what constitutes self, and even of nonself, but not of masquerader. Absent any genuine masquerader data, researchers must approximate a masquerade event with injections of data taken from some nonself user going about his or her daily business. This is certainly not a good or faithful way to model real masquerader behavior. The problem of nonstationarity in the data is considerable. Techniques can be developed to handle this in most cases, but some users will continue to be problematic in this regard. It would be helpful to identify these troublemakers in advance, but at present it is not clear how to do so. Collecting more information about these users may help. Real progress lies in being able to understand what makes a good or poor masquerader; for example, once it is known that scripts raise the probability of a masquerader's success, one is empowered to filter shared sequences out of the data, or to treat the data in some other compensatory way. Error analysis has often been neglected in studies of masquerade detection, despite its obvious usefulness in providing insight into what works, what doesn't work, and why.

12 Acknowledgements

We thank the Defense Advanced Research Projects Agency (DARPA), who supported this work under contracts F30602-99-2-0537 and F30602-00-2-0528. David Banks, of the US FDA, provided helpful comments in many discussions. Imre Kondor kindly provided a visualization system for Naive Bayes classification. Marcus Louie helped to work out the shared-tuple analysis, Kevin Killourhy helped with data management, and Patricia Loring helped in formatting the final Latex document. We are grateful for all of these contributions.

References

[1] B. D. Davison and H. Hirsh. Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time-Series Problems*, pages 5–12, Menlo Park, California, 1998. AAAI Press. Papers from the 1998 AAAI Workshop, 27 July 1998, Madison, Wisconsin: published as AAAI Technical Report WS-98-07.

[2] P. Domingos and M. Pazzani. Beyond independence: conditions for the optimality of the simple Bayesian classifier. In L. Saitta, editor, *13th International Conference on Machine Learning (ICML-96)*, pages 105–112, San Francisco, California, 1996. Morgan Kaufmann. Bari, Italy, 03-06 July 1996.

[3] W. DuMouchel. Computer intrusion detection based on Bayes factors for comparing command transition probabilities. Technical Report 91, National Institute of Statistical Sciences, Research Triangle Park, North Carolina 27709-4006, 1999.

[4] W. Ju and Y. Vardi. A hybrid high-order markov chain model for computer intrusion detection. Technical Report 92, National Institute for Statistical Sciences, Research Triangle Park, North Carolina 27709-4006, 1999.

[5] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, August 1999.

[6] V. Loeb. Spy case prompts computer search. *Washington Post*, 05 March 2001, page A01.

[7] T. F. Lunt. A survey of intrusion-detection techniques. *Computers & Security*, 12(4):405–418, June 1993.

[8] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *IEEE Symposium on Security and Privacy*, pages 59–66, Washington, DC, 1988. IEEE Computer Society Press. 18-21 April, Oakland, California.

[9] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, 1999. Fourth printing, 2001.

[10] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Learning for Text Categorization*, pages 41–48, Menlo Park, California, 1998. AAAI Press. Papers from the 1998 AAAI Workshop, 27 July 1998, Madison, Wisconsin: published as AAAI Technical Report WS-98-05.

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

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

[13] M. Schonlau and M. Theus. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters*, 76(1–2):33–38, November 2000.

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