NISS

Computer Intrusion: Detecting Masquerades

Matthias Schonlau, William DuMouchel,
Wen-Hua Ju, Alan F. Karr, Martin Theus,
and Yehuda Vardi

Technical Report Number 95
March, 1999

National Institute of Statistical Sciences
19 T. W. Alexander Drive
PO Box 14006
Research Triangle Park, NC 27709-4006
www.niss.org

Computer Intrusion: Detecting Masquerades

Matthias Schonlau[1], William DuMouchel[2], Wen-Hua Ju[3], Alan F. Karr[1], Martin Theus[4], Yehuda Vardi[3]

[1] National Institute of Statistical Sciences, 19 Alexander Drive, Research Triangle Park, NC 27709-4006

[2] AT&T Labs Research, 180 Park Avenue, Shannon Laboratory, Florham Park, NJ 07932

[3] Rutgers University, Dept. of Statistics, 110 Frelinghuysen Rd., Piscataway, NJ 08854-8019

[4] VIAG Interkom, Marsststr. 33, 80335 Muenchen, Germany

Abstract

Masqueraders in computer intrusion detection are people who use somebody else's computer account. We investigate a number of statistical approaches for detecting masqueraders. To evaluate them, we collected UNIX command data from 50 users and then contaminated the data with masqueraders. The experiment was blinded. We show results from our methods and two approaches from the computer science community.

Keywords: Anomaly, Bayes, Compression, Computer Security, High Order Markov, Profiling, Unix

1 Introduction

Intrusion detection in computer science is an appealing problem area because of its importance and the widespread interest in the subject, as evidenced by the report of the President's Commission on Critical Infrastructure Protection (1998).

There are many different types of intrusions. Denning (1997) divides attacks into 8 basic categories:

- eavesdropping and packet sniffing (passive interception of network traffic)
- snooping and downloading
- tampering or data diddling (unauthorized changes to data or records)
- spoofing (impersonating other users, e.g. by forging the originating email address, or by gaining password access)
- jamming or flooding (overwhelming a system's resources, e.g. by an email flood)
- injecting malicious code (via floppy disks, email attachments, etc.)
- exploiting design or implementation flaws (often buffer overflows; overflows overwrite other data and can be used to get control over a system)
- cracking passwords and keys

Due to a lack of actual intrusions (or at least due to our belief that we have no intruders inside our firewall) we focus here on a common form of spoofing, namely on detecting masquerades. Masqueraders are people who impersonate other people on the computer. They could be the offspring of users who use their parents' company account inspite of company policy, they could be users that play jokes on other users, or they could be malicious

intruders intentionally trying to hide their identity by impersonating other users. They could also be intruders from outside - although in practice most outside intruders immediately try to gain access to the account of the superuser and therefore are a special case. A computer crime and security survey (Computer Security Institute, 1998) ranking computer security problems in terms of their estimated financial damage found that unauthorized access by insiders was most damaging, accounting for about one third of the total loss.

Methods for computer intrusion detection fall into two broad categories: pattern recognition and anomaly detection. Pattern recognition refers to attempting to recognize the attack signatures of previously observed intrusions. It is our impression that computer scientists consider pattern recognition as the first line of defense. Clearly, it can be very powerful when the intrusion method is known. Unfortunately, like researchers, hackers come up with new ideas but unlike researchers they do not publish their work, at least not before an attack. Anomaly detection can defend against novel attacks and it is here that statistics seems most useful.

In anomaly detection, usually a historical profile is built for each user, and sufficiently large deviations from the profile indicate a possible intruder. Anomaly detection is not useful for most of the categories mentioned above and probably most appropriate for detecting masquerades. All commercial intrusion detection systems that we are aware of use pattern recognition, while some, like IDES (Lunt et al. 1992), NIDES and Emerald (Porras and Neumann 1997) use both approaches.

The literature focuses on a vast array of specific approaches to computer intrusion detection. For a general overview see Denning and Denning (1997) or Amoroso (1998). We describe two of the computer science approaches to anomaly detection that are directly relevant to this article in Section 3.

This article is structured as follows: In the next section we discuss the data and the experiment that we designed to compare several anomaly detection methods. In Section 3 we describe our methods and also two approaches from the computer science community. Section 4 then analyzes the results

of the experiment and Section 5 concludes with a discussion.

2 Data and Experimental Design

2.1 Data

Under the UNIX operating system users give commands. For example, a user might type `more myfile` in order to read `myfile` one screen at a time. In this example `more` is the command and `myfile` is an argument to that command. As a second example, typing `chmod +777 myfile` allows all users to read, write and execute `myfile`. Here both `+777` and `myfile` are considered arguments, `+777` specifies who exactly can read and/or write and/or execute `myfile`.

Our data source is the UNIX `acct` auditing mechanism. Examples of some auditing entries are given in Table 1. Our analysis is only based on the first

Command Name	User	Terminal	Start Time	End Time	Real (sec)	CPU (sec)	Memory Usage(K)
`chmod`	matt	pts/93	13:26:29	13:26:29	0.01	0.01	8.00
`more`	karr	pts/31	13:27:36	13:27:39	3.01	0.01	20.00
`cat`	vardi	pts/96	13:27:58	13:27:58	0.01	0.01	8.00
`whoami`	theus	pts/99	13:28:07	13:28:07	0.02	0.01	16.00
`sendmail`	karr	pts/91	13:28:17	13:28:17	0.02	0.01	124.00

Table 1: Examples of accounting entries generated by the UNIX `acct` auditing mechanism

two fields, "Command Name" and "User".

The first 15,000 commands for each of about 70 users were recorded over a time period of several months. The time span it took to collect 15,000 commands differs vastly from user to user. Some generate this many commands in a few days, others in a few months.

While the availability of arguments would be desirable, they were not collected because of privacy concerns. Some commands recorded by the system are implicitly and not explicitly typed by the user. A shell file is a file that contains multiple commands. Therefore running a shell file will cause all of its commands to be recorded. This also includes so called `.profile` files, and `make` files. Names of executables (i.e., programs) are also interpreted as commands since they are recorded in the audit stream.

2.2 Experimental Design

We randomly selected 50 users to serve as intrusion targets. We then used the remaining users as masqueraders and interspersed their data into the data of the 50 users.

For simplicity, 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 appears 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%. 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. 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 was repeated if there were not enough contiguous masquerading data left or if the masquerading

data were previously used.

We conducted the study in a blind fashion: none of the investigators knew the locations or number of the masqueraders at the time they were analyzing the data. The investigators knew the probabilities with which a masquerader would appear and disappear but were not allowed to use this information. The only piece of information used was the fact that masquerades only start at the beginning of blocks and so did the individual tests for masqueraders.

The data used in this experiment are available for download from http://www.research.att.com/~schonlau/ .

3 Overview of Methods

In what follows we describe distinct approaches labeled, "Uniqueness", "Bayes 1-Step Markov", "Hybrid Multi-Step Markov", "Compression", and two additional methods from the computer science literature labeled "IPAM" and "Sequence-Match". All methods attempt to detect anomalies and should be thought of as subsystems rather than as stand-alone intrusion detection systems.

The methods all operate in essentially the same way. First, the 5000 commands of training data are used to construct user profiles. Then, for each block of 100 commands, a score is computed and if the score exceeds a threshold, an alarm (indicating a potential masquerade) is triggered. When data are deemed to be free of masquerades they may be used to update the profiles. For each method we will describe how to generate the score as part of the model, how to set thresholds, and how to update the profile.

Before we describe the various methods, we first introduce some notation that is common to several methods:

C	training data (command names)
c	test data (command names)
C_{ut}	t^{th} command of user u of the training data
N_{ujk}	number of times user u used the command sequence (j,k) in the training data
N_{uk}	number of times user u used command k in the training data
N_u	length of user u's training data sequence
n_{ujk}, n_{uk}, n_u	as above for test data in a block being evaluated
x_{ub}	Score for user u at block b of method presented
U	total number of users (here 50)
U_k	number of users who have used command k in the training data
K	total number of distinct commands
T	number of commands in a test data block (here 100)

Note that the subscripts u, t and k index users, command order and commands, respectively. When a second subscript is needed to index commands, we use the subscript j.

The command stream for a given user is ordered. To avoid cumbersome sentences we will occasionally refer to that order as "time".

3.1 Uniqueness

The uniqueness approach is based on the idea that commands not previously seen in the training data may indicate a masquerading attempt. Moreover, the commands are more indicative the fewer users are known to use that command. This approach is due to Theus and Schonlau (1998).

3.1.1 Motivation

Uniquely used and unpopular commands are very important for this method. By "uniquely used command" we mean that in a pool of users only one user is using that command. An unpopular command is used only by few users.

It turns out that almost half of the UNIX commands appearing in our training data are unique to a single user, and many more are unpopular. Moreover, uniquely used commands account for 3.0% of the data, and commands used by 5 users or less account for 8.3% of the data.

A command has Popularity i if exactly i users use that command. We group the commands such that each group contains only commands with the same popularity. We assign an ID to each command such that commands from groups with unpopular commands are assigned lower ID's than commands from groups with more popular commands. The order within a group is arbitrary. When plotting the command ID over "time" the usage pattern of uniquely used/unpopular commands emerges. Such a plot is shown in Figure 1 for the first 5000 commands of each of 50 users. Groups are separated by a horizontal line. The fact that the Popularity= 1 group takes up approximately the bottom half of Figure 1 shows that about half of all commands are uniquely used, and many more are unpopular.

3.1.2 Model

We define a test statistic that builds on the notion of unpopular and uniquely used commands:

$$x_u = \frac{1}{n_u} \sum_{k=1}^{K} W_{uk} \ (1 - U_k/U) \ n_{uk} \qquad , \tag{1}$$

where the weights W_{uk} are

$$W_{uk} = \begin{cases} -v_{uk}/v_k & \text{if user } u\text{'s training data contains} \\ & \text{command } k \\ 1 & \text{otherwise} \end{cases}$$

where

$$v_{uk} = N_{uk}/N_u$$

and

$$v_k = \sum_u v_{uk} \qquad .$$

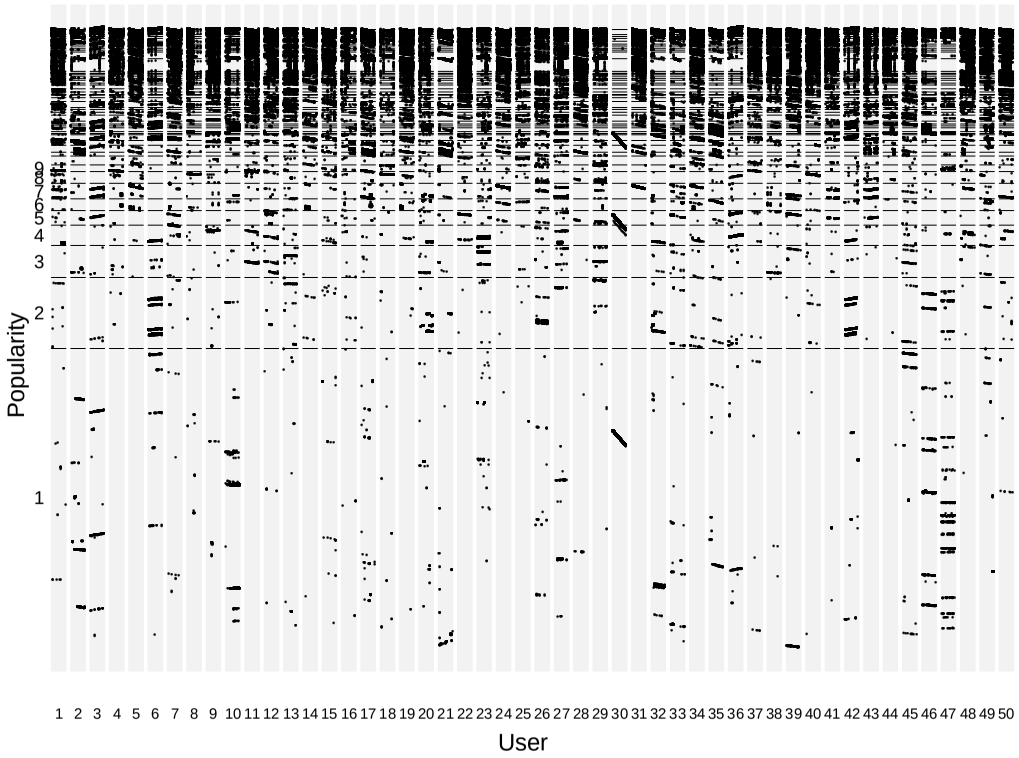

Figure 1: Visualization of Command Popularity. Each panel corresponds to one user. Within each user's panel, each of the user's commands is represented by a single dot. The popularity corresponding to each command is plotted versus "time" (different commands with the same popularity have different ordinates within their popularity group).

The fraction $(1 - U_k/U)$ acts as a uniqueness index: it is zero if all users have used this command before, it is 1 if none of the users has used it before. The weights W_{uk} control whether the uniqueness index should be added or subtracted, depending on whether the command was seen before or not. Hence a user will tend to score low if he/she uses commands similar to the ones he/she used in the training data. The order in which the commands appear does not matter. The quantity v_{uk}/v_k represents the command usage relative to other users. It reduces the score contribution of commands that

other users often use and this user rarely.

3.1.3 Thresholds

We assign the same threshold to all users. This threshold is estimated via cross validation: we split the original training data into two data sets of 4000 and 1000 commands. Using the larger data set as training data, we assign scores for the smaller one. This is repeated 5 times, each time assigning scores to a distinct set of 1000 commands. We set the threshold to the 99^{th} percentile of the combined scores across all users and all 5 cross validations. For our data the resulting threshold is 0.2319.

We investigated assigning different thresholds to different users, in the way to be described for the compression method (Section 3.4.2). It turned out that individual thresholds tended to yield a false alarm/missing alarm tradeoff inferior to the one based on a common threshold for all users. This may be due to the fact that scores are interpretable for all users in the same way: e.g. a negative score indicates that the test data contains few or no incidences of new commands and therefore are not suspicious.

3.1.4 Updating

This method is particularly sensitive to contamination of the training data. Using all alarm-free test data for updating therefore does not seem sensible. Instead we introduce a more conservative second threshold for updating: if the test score remains below the updating threshold then the corresponding data are added to the training data.

In choosing the updating threshold, we note that a masquerader who uses mostly common commands can get a score of slightly below zero, and at the same time contaminate the pool of commands by using a few of his/her own uniquely used commands. To avoid that possibility, we set the updating threshold somewhat arbitrarily to -0.05.

Updating the training data set means updating the vector of distinct commands and the matrix that contains a count of how often each user used a command. Updated scores can be recomputed from that matrix. Here we recompute the scores after every 2000 test commands (i.e. after commands 7000, 9000, 11000, and 13000). While non-simultaneous updating is possible, we chose to update simultaneously for all users, as if all users were submitting commands at the same rate.

3.2 Bayes 1 Step Markov

The Uniqueness approach only considered command frequencies. The Bayes 1 Step Markov approach goes further: it is based on 1-step transitions of commands. The approach uses a Bayes factor statistic to test the null hypothesis that the observed 1-step command transition probabilities are consistent with the historical transition matrix. This approach is due to DuMouchel (1998).

3.2.1 Model

We form two hypotheses. The null hypothesis assumes that the observed transition probabilities stem from the historical transition matrix; the alternative hypothesis is that they were generated from a Dirichlet distribution:

$$H_0: \quad P(C_t = k | C_{t-1} = j) = p_{ujk} \tag{2}$$

$$H_1: \quad P(C_t = k | C_{t-1} = j) = Q_k$$

$$(Q_1, \ldots, Q_K) \sim Dirichlet(\alpha_{01}, \ldots, \alpha_{0K}) \tag{3}$$

where p_{ujk} is the historical transition probability from command j to command k for user u, i.e.,

$$p_{ujk} = P(\text{Next Command} = k | \text{Previous Command} = j, \text{User} = u) \quad .$$

We now explain how we estimate the historical command transitions p_{ujk} and the parameters $\alpha_{01}, \ldots, \alpha_{0K}$.

Choosing a Bayesian framework, we estimate the p_{ujk} by shrinking the observed conditional probabilities toward the marginal probabilities:

$$p_{ujk} = (N_{ujk} + v_{uj} q_{uk})/(N_{uj.} + v_{uj})$$

where q_{uk} are the marginal probabilities $q_{uk} = P(\text{Next Command} = k | User = u)$ and v_{uj} are Bayesian hyperparameters controlling the shrinkage. The v_{uj} in turn are estimated by fitting conditional frequencies N_{ujk} to a Dirichlet distribution with means q_{uk}.

The marginal frequencies q_{uk} are estimated by shrinking marginal frequencies observed in the training data toward the average command frequencies for all users:

$$q_{uk} = (N_{u.k} + \alpha_{uk})/(N_{u..} + \alpha_{u.}) \qquad .$$

The α_{uk} above and the α_{0k} used in (3) to specify the alternative hypothesis are estimated by fitting the marginal frequencies $N_{u.k}$ to a Dirichlet-multinomial distribution with a modification of the usual Dirichlet model to take into account the fact that many commands are unique to particular users. See DuMouchel (1999) for details of this modification, which was inspired by the success of the Uniqueness method of Section 3.1.

We test the above hypothesis by forming the Bayes factor. The Bayes factor BF is the ratio of the probabilities of the data under the two hypotheses:

$$BF = Prob(C_1, ..., C_T | H_1) / Prob(C_1, ..., C_T | H_0) \qquad .$$

The larger BF is, the more evidence there is against H_0 in favor of H_1. In fact, $x = log(BF)$ if often called the weight of evidence. On the log scale there is the nice property that evidence from two independent data sets is the sum of their individual evidence. We therefore use $x = log(BF)$ as the test statistic.

It turns out that the Bayes factor BF can be calculated as

$BF =$

$$\prod_k (\alpha_{0k}(\alpha_{0k}+1)\cdots(\alpha_{0k}+n_{u.k}-1)) \,/\, \left(\alpha_{0.}(\alpha_{0.}+1)\cdots(\alpha_{0.}+T-1)\prod_{j,k} p_{ujk}^{n_{ujk}}\right) \quad .$$

3.2.2 Thresholds

We split the training data set into two parts. The model was estimated from one part and based on those estimates. Scores were calculated for the other part. Let $\bar{x}_u$ and $\bar{x}$ be the average such score for user u and across all users, respectively.

We calculate individual thresholds for each user as follows:

$$Threshold_u = (\bar{x}_u - \bar{x})/2$$

The average threshold for all users is intentionally set to 0. The value 0 is the log Bayes factor when the hypothesis and the alternative are equally likely. It thus emerges as a natural choice.

3.2.3 Updating

When any three blocks of testing data (300 commands) in a row are alarm-free then the center block is added to the training data. Should a fourth block be alarm free, then the third block is added, and so forth. Not adding the first and last blocks minimizes the contamination of the training data with masquerading data.

Adding a block to the training data in practice can be done by updating the values of q_{uk} and p_{ujk}. The update algorithm is straightforward, involving a modification of the nonzero counts N_{ujk}. The values of p_{ujk} are never stored, but are only computed as needed, based on q_{uk}, v_{uj} and the nonzero values of N_{ujk}.

An exponential averaging method is used to update the N_{ujk}, to "age out" old data. It is of the form $N_{ujk} \leftarrow N_{ujk}2^{-n_u/500} + n_{ujk}$ so that command relative frequencies have a "half-life" of 5000 commands. The user - specific thresholds are also updated by an exponential weighting algorithm.

3.3 Hybrid Multi-Step Markov

A hybrid method based mostly on a multi-step (also called "high-order") Markov chain and occasionally on an independence model is proposed. This approach is due to Ju and Vardi (1999).

We overcome the high-dimensionality inherent in a multi-step Markov chain as follows: (a) restrict attention to a subset of the most used commands (with the remaining commands represented under a single "command" labeled other), (b) use a *Mixture Transition Distribution* (MTD) approach to model the transition probabilities (Raftery (1985) and Raftery and Tavaré (1994)).

When test data contain many commands unobserved in the training data, a Markov model is not usable. In such instances a simple independence model with probabilities estimated from a contingency table of users vs. commands may be more appropriate. Our method automatically toggles between the two models as needed.

3.3.1 The Multi-Step Markov Model

Let K_u be the smallest number such that the most frequently used K_u-1 commands of user u account for at least 99% of that user's training data. All other commands, including those not appearing in user u's training data, form a category labeled other_u. We then combine the most frequently used K_u-1 commands and other_u to constitute the Markov chain's state space M_u.

Let $\{C_{ut}; t = 1, 2, \ldots\}$ be the sequence of commands of user u. Assuming the MTD model, the transition probabilities of an l-step Markov chain is:

$$\begin{aligned} P\left(C_{ut}\right. &= c_0 | C_{u,t-1} = c_1, C_{u,t-2} = c_2, \ldots, C_{u,t-l} = c_l) \\ &= \sum_{i=1}^{l} \lambda_{ui}\, r_u\left(c_0 | c_i\right) \quad , \quad t = l+1, l+2, \ldots \end{aligned}$$

where $\mathbf{R} = \{r(i|j);\, i, j \in M\}$ and $\mathbf{\Lambda} = \{\lambda_i;\, i = 1, 2, \ldots, l\}$ satisfy certain positivity constraints.

We fix $l = 10$ in our experiments and make the simplifying assumptions that for all users u: (1) $r_u(\mathsf{other_u}|i) = \epsilon$, $\forall i \in M_u$, where ϵ is small (we take $\epsilon = .00001$), and (2) $r_u(i|\mathsf{other_u}) = \frac{1-\epsilon}{K_u - 1}$, $\forall i \in M_u$ except for $i = \mathsf{other_u}$.

Assumption (1) forces transition probabilities to infrequently used commands to be small. Assumption (2) copes with the scarcity of the data by setting all transition probabilities from infrequently used commands to the same quantity. Note that $\mathsf{other_u}$ contains rarely used commands. The rationale for choosing $\epsilon = .00001$ is that there is a 1% chance to get a rare command and there are (roughly) 1000 commands in $\mathsf{other_u}$.

The log-likelihood of a command sequence for user u is

$$\log L = \sum_{i_0 \in M_u} \cdots \sum_{i_l \in M_u} n(i_0, i_1, \ldots, i_l) \log \left(\sum_{j=1}^{l} \lambda_{uj} r_u(i_0|i_j) \right) \tag{4}$$

where $n(i_0, i_1, \ldots, i_l)$ is the number of times that the pattern $i_l \mapsto i_{l-1} \mapsto \ldots \mapsto i_0$ is observed in the command sequence. A maximum likelihood estimate (MLE) maximizes (4) with respect to $\mathbf{\Lambda}$ and $\mathbf{R}$, subject to positivity constraints. See Ju and Vardi (1999) for computational details.

3.3.2 The Independence Model

This model assumes that user u's commands are independently generated from a multinomial random distribution:

$$P(C_{u1} = c_1, \ldots, C_{uT} = c_T | \text{user } u) = \prod_{t=1}^{T} P(C_{ut} = c_t | \text{user } u) = \prod_{t=1}^{T} q_{uc_t}$$

As with the previous methods we use the fact that commands used by few users have a greater power to distinguish different users than those that are used by many users. To that end, this method also uses weights that depend on the popularity score, and to achieve this, we transform N_{uk} as follows:

$$N'_{uk} = w_k N_{uk} + (1 - w_k)(N_{\cdot k} \frac{N_{u\cdot}}{N_{\cdot\cdot}}),$$

where $w_k = 1 + \frac{1}{U} - \frac{U_K}{U}$. Frequencies of the more popular commands are shrunk toward their marginal proportions ($N_{\cdot k}\frac{N_{u\cdot}}{N_{\cdot\cdot}}$) to a greater extent. Subsequently, the N'_{uk} are renormalized to achieve $N'_{u\cdot} = N_{u\cdot}$ and q_{uk} is estimated by $N'_{uk}/N'_{u\cdot}$ if $N'_{uk} > 0$ or ϵ if $N'_{uk} = 0$ (to avoid taking log of zero).

3.3.3 Combining the two models

Based on the test data sequence $\{c_{u1}, \ldots, c_{uT}\}$ we test the hypothesis

$$H_0 : \text{commands are generated by user } u$$
$$H_1 : \text{commands are generated by one of the other users}$$

We define log of likelihood-ratio statistic for the Markov and independence models, respectively, as follows:

$$X_{1u} = \log\frac{\max_{v\neq u} L(c_1, \ldots, c_T|\hat{\mathbf{\Lambda}}_v, \hat{\mathbf{R}}_v)}{L(c_1, \ldots, c_T|\hat{\mathbf{\Lambda}}_u, \hat{\mathbf{R}}_u)} \quad \text{and} \quad X_{2u} = \log\frac{\max_{v\neq u}\prod_i q_{vc_i}}{\prod_i q_{uc_i}}.$$

To bring the two statistics X_1 and X_2 into the same scale, we regress X_2 over X_1 with no intercept, $X_1 = \rho X_2$, based on the training data. We use a robust regression procedure based on least median of squares to estimate ρ.

Let s_u be the number of commands that are categorized as $\mathsf{other_u}$ in $\{c_{u1}, \ldots, c_{uT}\}$. We combine X_{1u} and X_{2u}into a single "score", x_u:

$$X_u = \begin{cases} x_{1u} & \text{if } s_u/T \leq \xi, \quad \text{(we choose } \xi = 0.2 \text{ in our experiment)} \\ \hat{\rho}X_{2u} & \text{if } s_u/T > \xi. \end{cases}$$

This results in a hybrid method which automatically alternates between the Markov and independence models according to whether the test data are "usual" or "unusual". The latter occurred 8.4% of the time in our data set.

3.3.4 Thresholds

We reject H_0 if $x_u > \mu + 3\sigma$ where μ and σ are the mean and standard deviation of the scores x_u. We estimate μ and σ from the pooled training data from all users. The same threshold is used for all users.

3.3.5 Updating

When $x_u \leq 0$ for a given set of test data, the corresponding test data are added to the training data set. The updating-threshold (zero) is thus more conservative than the one for raising an alarm. The parameters $\mathbf{\Lambda_u}$ and $\mathbf{R_u}$, which profile an individual user, are reestimated based on the augmented training data at each update. The threshold parameters, μ and σ, are reestimated from all training data scores x_u and all previous test data with a score lower than the old threshold.

In this experiment the estimates of the parameters $\mathbf{\Lambda_u}$ and $\mathbf{R_u}$, were updated once exactly half way through the test data.

3.4 Compression

By compression we mean a reversible mapping of data to a different representation that uses fewer bytes. This approach was implemented by Karr and Schonlau.

3.4.1 Model

The premise of the compression approach is that test data appended to historical training data compress more readily when the test data stems indeed from the same user rather than from a masquerader. We define the score x to be the number of additional bytes needed to compress test data when appended to training data :

$$x = compress(\{C, c\}) - compress(C)$$

where C is the training data, c the test data, $\{C, c\}$ is the test data appended to the training data, and compress() is a function that gives the number of bytes of the compressed data.

There are several compression methods, many of which are based on the Lempel-Ziv algorithm. We use the UNIX tool "compress" which implements a modified Lempel-Ziv algorithm popularized in Welch (1984).

3.4.2 Threshold

The threshold is determined from the training data (the first 5000 commands for each user) by cross validation. We assign an individual threshold for each user. For each user and each of the 50 blocks of 100 commands, we compute

$$x_{ub}^{cv} = compress(C) - compress(C - C_{ub}^{100}), \quad b = 1, \ldots, B = 50 \; u = 1, \ldots, U$$

where and C_{ub}^{100} is the b^{th} block of 100 commands for user u. The superscript cv indicates that the scores are cross - validated.

This yields 50 cross - validated scores for each user. Since we cannot determine the 99^{th} percentile of only 50 empirical scores we make the assumption that each user's threshold is a constant offset from each user's score average:

$$Threshold_u = \bar{x}_u^{cv} + \Delta \qquad ,$$

where Δ is determined from the pooled data across all users:

$$\Delta = x_{.99}^{cv} - \bar{x}^{cv} \quad . \tag{5}$$

where $x_{.99}^{cv}$ is the sample 99^{th} percentile of x_{ub}^{cv}. Note that the choice of $x_{.99}^{cv} = 0$ leads to the thresholds that were proposed in Section 3.2.2 for the Bayes 1-Step Markov model.

3.4.3 Updating

After every 2000 test commands for a given user (i.e., after commands 7000, 9000, 11000 and 13000) we consider updating the training data. When no alarm was raised in the most recent 5000 commands for a given user, then the training data set is replaced by the most recent 5000 commands and the threshold for this user is recomputed (more precisely, we recompute $\bar{x}_u^{cv}$ only, not Δ). If one or more alarms were raised, the previous training data and threshold remain in place.

3.5 Additional Methods

We have not been able to use commercial intrusion detection tools on our data. A problem with the commercial systems that have an anomaly detection component is that they expect the data in a specific format. For example, CMDS (Computer Misuse Detection System, 1999) expects events at the system call level and then reconstructs the UNIX audit trail from the system calls.

The authors' methods are compared to two methods from the computer science community. We very briefly describe these methods here.

3.5.1 IPAM

"Incremental Probabilistic Action Modeling" (IPAM) is the name of an algorithm by Davison and Hirsh (1998). It is based on 1-step command transition probabilities estimated from the training data. The estimated probabilities are updated continually, using an exponential updating scheme. That is, upon arrival of a new command all transition probabilities from the penultimate command to another command are aged by multiplying them with α, and $1-\alpha$ is added to the most recent transition. Davison and Hirsh's choice of α is .9.

Given a command it is then possible to predict the next command by choosing the one corresponding to the highest transition probability. A prediction is labeled good, if the next command turns out to be among the top four predicted commands. The fraction of good predictions of the test data forms the score. If the fraction falls below a threshold, an alarm is raised. The thresholds are estimated from the training data in an *ad hoc* fashion.

3.5.2 Sequence-Match

For each new command, Lane and Brodley (1998) compute a similarity measure between the most recent 10 commands and a users's profile. A user's

profile consists of command sequences of length 10 that the user has used in the past. The similarity measure is a count of the number of matches in a command-by-command comparison of two command sequences, with a greater weight assigned to adjacent matches. This similarity measure is computed for the test data sequence paired with each command sequence in the profile. The maximum of all similarity values computed forms the score for the test command sequence. Since these scores are very noisy, the most recent 100 scores are averaged. If the average score is below a threshold an alarm is raised.

The threshold is determined based on the quantiles of the empirical distribution of average scores. The initial profiles based on the training data contain 4991 command sequences for each user.

4 Results

The experimental setup described in Section 2 yielded a total of 40 masquerader incidents for the 50 users. These incidents account for 4.74% of the data. There were at most three incidents for any one user, and 21 of the users had no masquerader at all. Half the masqueraders were present for four or fewer blocks, while three masqueraders were present for $20 - 22$ blocks.

4.1 Overall Results

We will first visually inspect when various methods gave alarms and when a masquerader was present. This can be seen in Figure 2 for Hybrid Multi-Step Markov, Bayes 1-Step Markov, Uniqueness, and Compression methods. Each row corresponds to one user and each column to one block of data. The presence of a color indicates that the corresponding method gave an alarm. The red background shading indicates the presence of masqueraders. Whether an alarm was raised or not was based on thresholds supplied by the respective methods. All methods except IPAM and Sequence-Match were

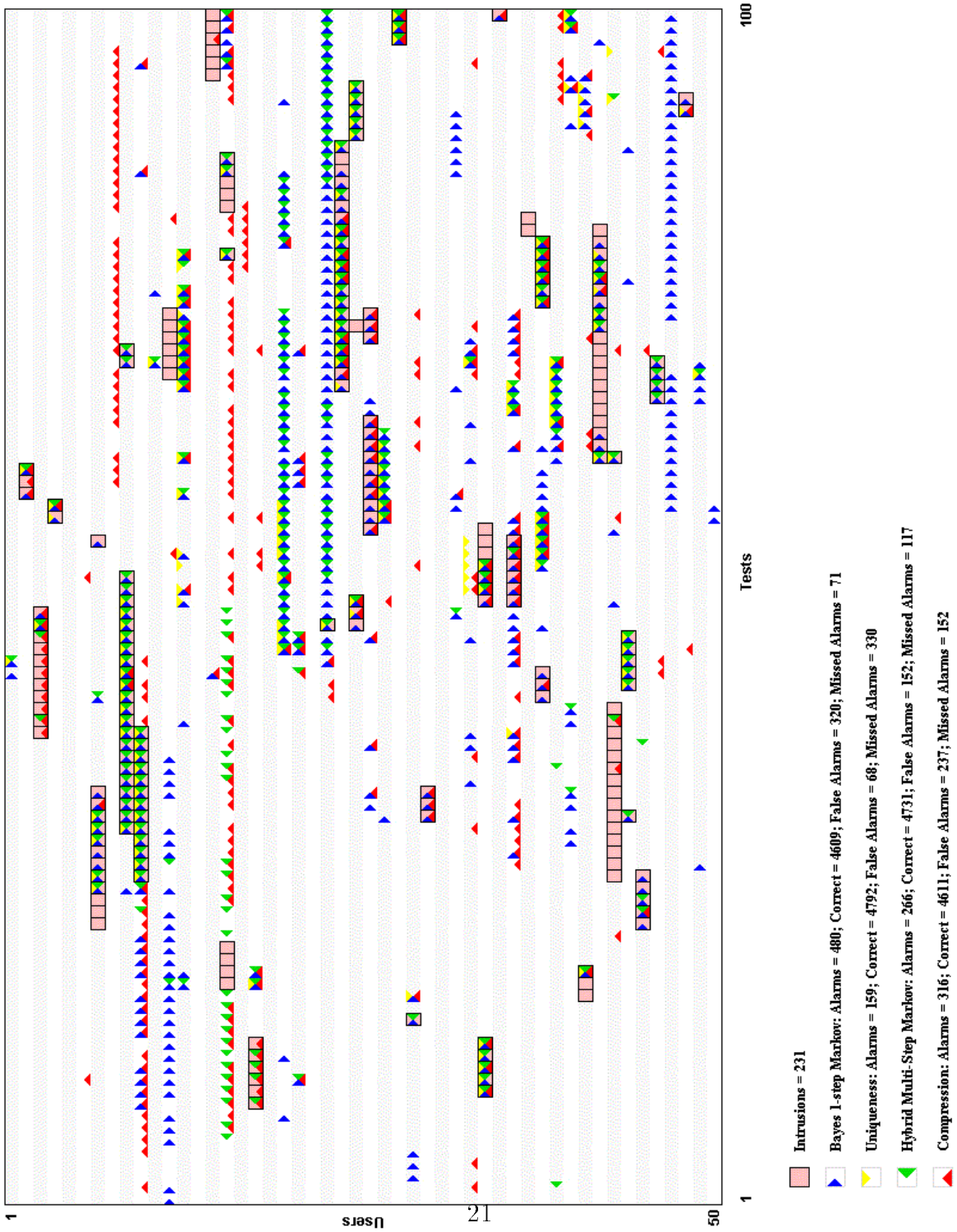

Figure 2: Plot of alarms given by various methods for each of 50 users. Compromised data has a red background shading.

Method	False Alarms (%)	Missing Alarms(%)
Uniqueness	1.4	60.6
Bayes 1-Step Markov	6.7	30.7
Hybrid Multi-Step Markov	3.2	50.7
Compression	5.0	65.8
Sequence-Match	3.7	63.2
IPAM	2.7	58.9

Table 2: False alarms and missing alarms for all methods. The investigators corresponding to the first four methods were asked to target a false alarm rate of 1%.

asked to target a false alarm rate of 1%. The false alarm rates achieved for this data set are given in Table 2. None of the methods accomplished this goal, suggesting that it is difficult to control the false alarm rate. The missing alarms range from 30 to 60 per cent. Because of different false alarm/missing alarm rates, some colors show up more often in Figure 2.

When a masquerader is present, often either all three or none of the methods raise an alarm. This may suggest that masqueraders fall into two groups: easy to detect and very difficult to detect. While the methods sometimes give false alarms jointly (e.g. User 13, User 27) different methods seem to be prone to false alarms for different users (e.g. yellow - user 16, green - user 12, blue - user 33). Particularly interesting is that false alarms often appear in long sequences. It also appears that while methods raise false alarms for different users, they discover the same masqueraders at about the same time.

4.2 Selected Individuals

Figure 2 showed alarms for all users. Now we will look at the actual scores of all methods for selected users.

The 8 panels in Figure 3 correspond to 8 particular users. Each panel

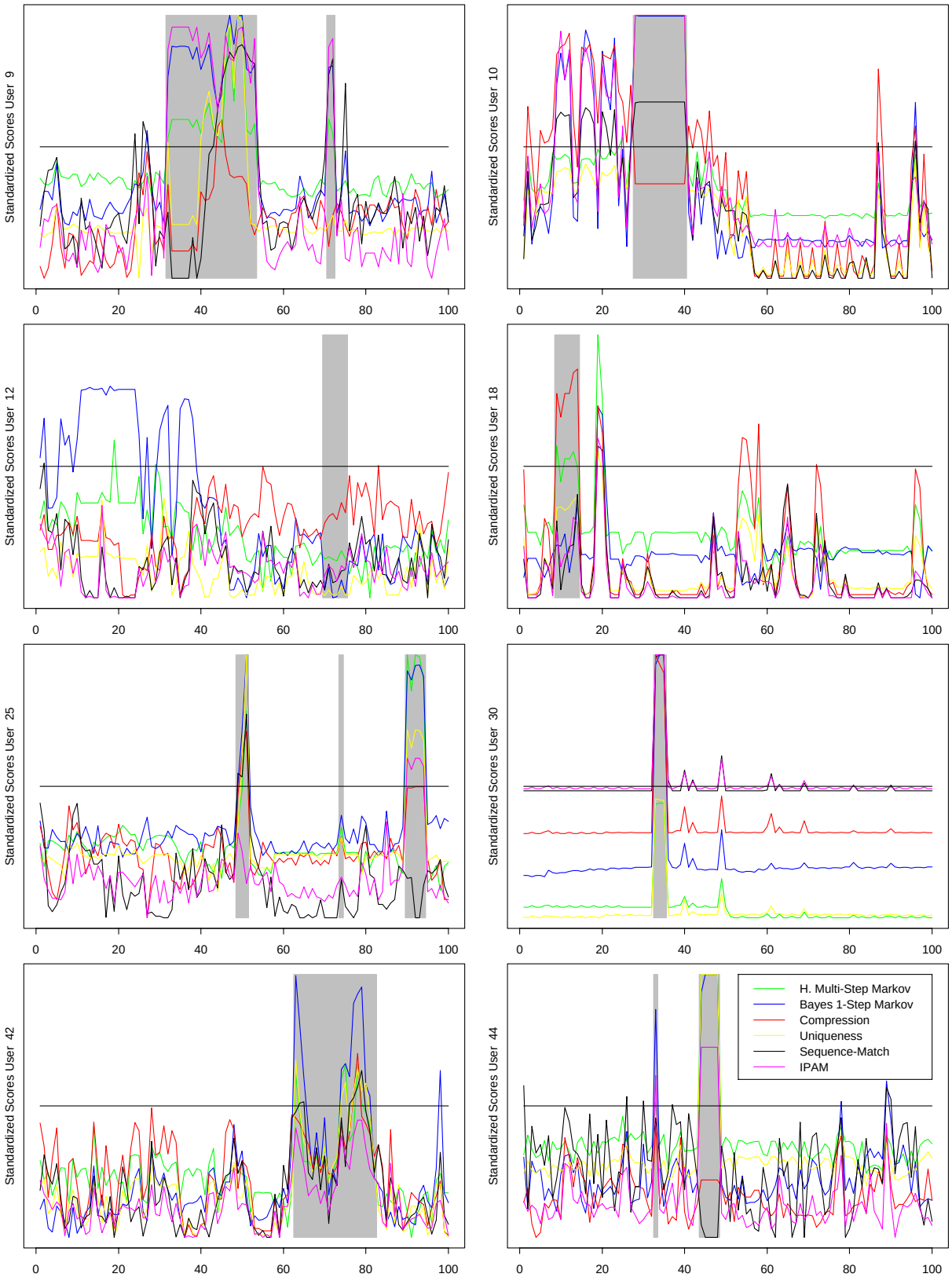

Figure 3: Plot of standardized scores over "time" for 8 different users.

displays standardized scores over "time" for all methods. Each color corresponds to a different method. The horizontal axis - labeled 1 through 100 - corresponds to the 100 blocks of testing data. The scores are standardized such that the horizontal line in the center represents the threshold value for all 100 blocks and for all methods. Thus, if any line crosses above the center line, that method raises an alarm at that point. The thresholds used here are the ones supplied by each method, meaning that all methods have different false alarm rates as given in Table 2. Standardized scores are not comparable across plots since the standardizations are different for each plot.

Gray shading indicates the presence of a masquerader. Lines crossing above the threshold when no gray shading is present are thus false alarms.

In what follows we comment on individual panels:

- User 9: During the first 8 blocks of the first masquerader the scores of half the methods jump up, indicating an alarm, while the scores for the other methods jump down. The features of the scores of the two groups are exactly reversed. We hypothesize that during these blocks this masquerader did not use unusual new commands (uniqueness score is low) and the commands were somewhat repetitive (compression score is low) but the transition between familiar commands were unusual (all transition methods have high scores). It turns out that the data between about commands 8200 and 8900 consist of repeated sequences `sed, ln, sh`. User 9 had used all three commands separately before, but the 1-step transitions `sed, ln` and `ln, sh` never appeared before.

  The first and the second masquerader happen to be the same person. This is the only time this happened in our experiment.

- User 10: During the presence of the masquerader the score of all methods remains constant. The only method not detecting the masquerader is the compression method. We hypothesize that the masquerader did something unusual that was very easy to compress. It turns out that

the masquerading data consists of a single repeated command: `popper`. User 10 never used this command before. Clearly, the masquerading data are highly repetitive and thus can be compressed easily.

- User 12: There is a masquerader present for 6 blocks that not a single method discovers. This raises the question whether the intrusion is discoverable at all. Nothing unusual was found in the data.

- User 18: There is one masquerader that some methods capture precisely. Around block 20 every single method raises an alarm in the absence of a masquerader. There seems to be a true anomaly that does not happen to be a masquerader. User 18 uses the commands `get_acc_line.pro` and `acc.prof`, which did not appear in the training data. There are also some unusual command transitions.

- User 25: Masqueraders 1 and 3 are detected well by most methods, but no method detects the second masquerader.

  There is a delayed reaction by all methods to the first masquerader (except IPAM, which fails to raise an alarm). It turns out that at the onset of the masquerading attempt there are a lot of common commands like `ls`, `more` and `sendmail`. Later, the more unusual sequence of commands `driver`, `edgcpfe`, `uopt`, `ugen`, `as1` appears a number of times.

  During the presence of the third masquerader, all scores take a small dip at the same time, indicating that they are highly correlated. It turns out that the data of the masquerader is dominated by sequences of the form `mars.sh, grep, ping, grep`. The dip is caused when the sequences temporarily occur less frequently than before and after.

- User 30: All methods are highly correlated over the entire 100 blocks. While not all methods raise an alarm (remember that methods are not standardized to the same false alarm rate), the score clearly peaks

during the presence of the masquerader. It turns out that the data of User 30 is highly repetitive: the command sequence `rdistd, tcsh, rshd` repeats itself over and over for the most part.

- User 42: Surprisingly, the distribution of the scores during the presence of the masquerader is essentially bimodal for all methods.

- User 44: The two masqueraders are perfectly captured by most methods. It turns out that the second masquerading attempt consists several hundred repetitions of the command `popper`. Therefore the compression method fails to detect the masquerader. The reason that Sequence-Match misses this masquerader is likely due to an updating problem: while User 44 never uses `popper`, the first masquerader does. Since Sequence-Match did not detect the first masquerader, sequences containing this command may have been added to User 44's profile, leading to a failure to detect the second masquerader.

4.3 Correlation of Methods

Apparently scores are correlated very highly. We ran several cluster algorithms using the correlation matrix as a measure of similarity. The Methods Uniqueness and Hybrid Multi-Step Markov with a correlation coefficient of .79 clearly are part of the same cluster. This may be due to the fact that the hybrid elements of the Hybrid Multi-Step Markov method focus on rare and unique commands similar to the way the Uniqueness method does.

A second group is formed by the two methods contributed from the computer science community, namely IPAM and Sequence-Match. The correlation coefficient between these two groups is .62. Depending on the clustering algorithm the Bayes 1-Step Markov method could be associated with either of these two groups. Given that both IPAM and Bayes 1-Step Markov are based on the 1-step command transition matrix, it is surprising that both methods are more highly correlated with other methods than with each other.

The compression method stands by itself. Its highest correlation coefficient - with the uniqueness method - is .57.

4.4 ROC curves and survival analysis

Up to now it was difficult to compare methods because the visualizations were based on different false alarm rates. By varying the thresholds we obtain different tradeoffs between false alarms and missing alarms. The curve that shows the functional relationship between false alarms and missing alarms is called a ROC curve.

Since some methods have different thresholds for different users there is a question of how to vary the thresholds. We add a constant to all individual thresholds and then compute the corresponding alarm rates. We ignore the possibility that different thresholds might have some effect on the updating algorithm.

Figure 4 displays ROC curves for all methods. The lower and the further left a curve is, the better it is. For the best methods, one per cent false alarms corresponds to about 70% missing alarms; and five per cent false alarms corresponds to about 30% missing alarms. The compression method seems to be uniformly inferior to other methods. Between 1% and 5% false alarms the uniqueness methods clearly dominates the other methods. IPAM and Sequence-Match are surprisingly similar for high false alarms. For a very low false alarm rate ($< .5\%$), they do better than all other methods.

When using ROC curves, the unit of analysis is the block rather than the intrusion. It is also of interest to see how long an intrusion "survives" before it is detected. Figure 5 gives the probability of an intrusion surviving as a function of the number of data blocks when the false alarm rate for all methods is fixed at 1%. After the first block between 15% and 30% of the intrusions are caught, after about 10 blocks about $40\% - 50\%$ are caught. The compression method does worse than that.

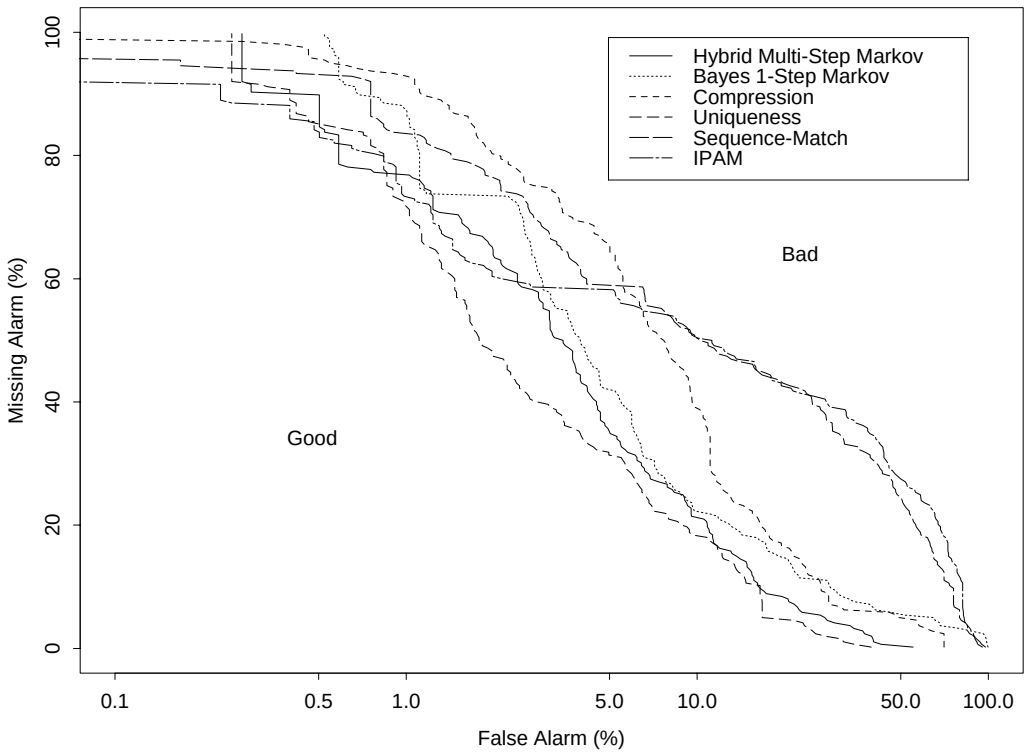

Figure 4: ROC curves for all methods. Methods use updating.

4.5 ROC curves without updating

All analyses were based on the updating algorithms as described in Section 3. Figure 6 gives the ROC curves of algorithms based solely on the initial training data. Methods Sequence-Match and Compression perform better without updating. For compression this is directly attributable to the following difficulties associated with the updating algorithm: Since we do not update training data with test data that had high scores to avoid contamination with masquerading data, the average score for the training data tends to decrease. As the consequence the thresholds - calculated from the updated training data - decrease too. Eventually normal variation in the

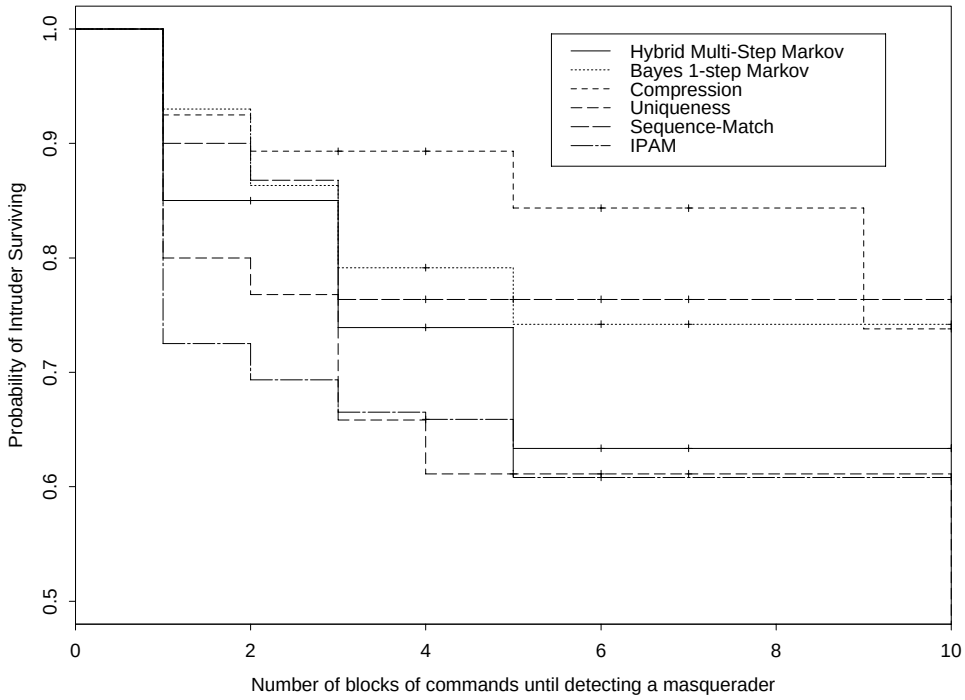

Figure 5: Survival plot of intrusions for all methods given a fixed false alarm rate of 1 per cent. Ideally there would be no detection delay, that is, instant "death" of the intrusion.

scores starts looking like a masquerading attempt. While this is a problem for all methods, it turned out to be especially severe for the compression method.

Both the Uniqueness and Hybrid Multi-Step Markov methods benefited from updating. This was not obvious given that methods may be vulnerable from updating the training data with masquerading data (especially the Uniqueness method).

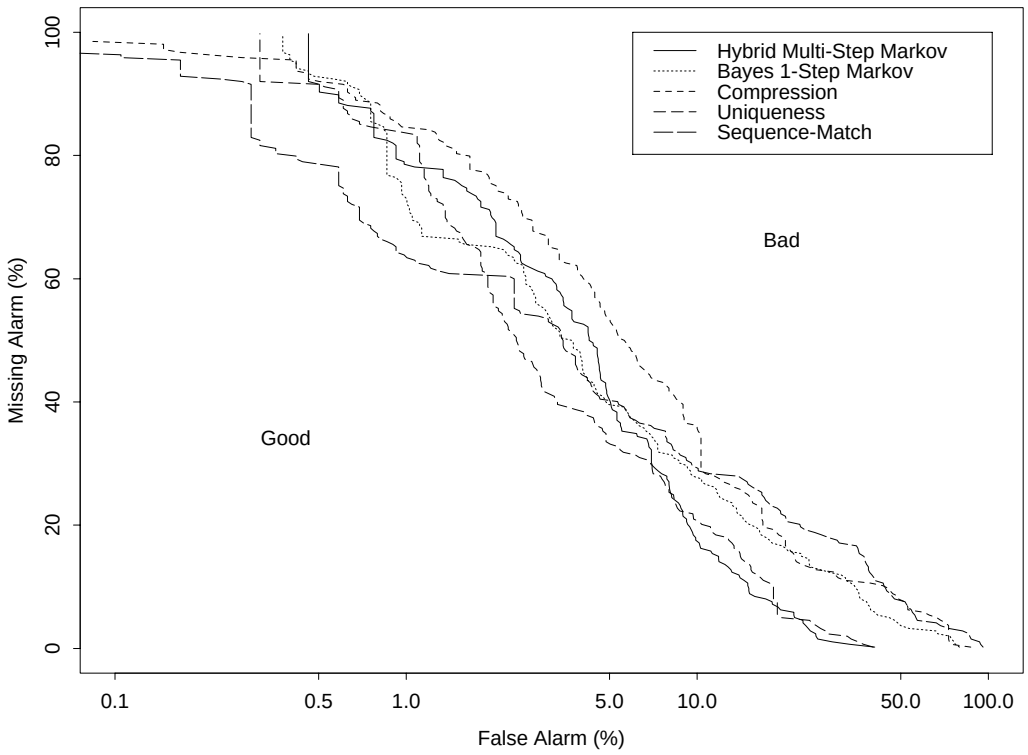

Figure 6: ROC curves for all methods except IPAM for which no data was available. Methods do not use updating.

5 Discussion

All methods detect anomalies in command usage. We have shown that all methods can detect anomalies surprisingly well considering the paucity of information.

In essence, we try to classify data into two groups which one might label "good" and "bad". However, we have to characterize "good" in the absence of training data for "bad". It is interesting to reflect on how our methods cope with this problem: The Hybrid Multi-Step Markov method explicitly assumes

that the masquerader resembles one of the other users. To a lesser extent, the Uniqueness method and the Bayes 1-Step markov method also assume that through the concept of command popularity and parameter estimation of the Dirichlet distribution, respectively. The compression method has an inherent notion of what "bad" constitutes.

The experiment has led to more insight into the different methods:

The uniqueness method can be severely affected by accidental contamination of the training data with rare commands from the masquerading data. Further, a single threshold for all users works better than individual thresholds for each user. The uniqueness method is the easiest to compute yet it was the most powerful at anomaly detection in our experiment. It is an open question whether this success can be repeated in other contexts.

The compression method is vulnerable to updating the training data with homogeneous data. It has also difficulties with updating algorithms.

The Markov method - while very successful - is computationally demanding. While only used 5% of the time, the hybrid elements of the Hybrid Multi-Step Markov method did improve its ROC curve.

The Bayes 1-Step Markov method did slightly worse than the Markov and the Uniqueness methods, but it is more robust than Uniqueness and is not as computationally intensive as the Markov method.

Previously, we investigated another approach based on principal components analysis of one step command transition frequencies (DuMouchel, Schonlau, 1998a and 1998b). This approach was abandoned because it required very voluminous user profiles and was not easily extensible to make use of the information in unpopular/unique commands.

To some extent, the performance of various methods is substantially the same. For example, the ROC curves in Figures 4 and 6 do not differ dramatically. Nevertheless, the curves *do* differ, and we believe these differences to be meaningful. One interpretation is that any sensible approach (compression is a good example) works pretty well, because some intrusions are easy to detect. But going from working pretty well to working well requires

insight and effort.

None of the methods described here could sensibly serve as the sole means of detecting computer intrusions. The relatively high missing alarm rates preclude this. We have presented a collection of tools. They would be part of an overall strategy that might include both attack signatures and analysis of more detailed data, including time stamps and command arguments.

In our opinion our contribution to the field of intrusion detection is as follows: We introduced a theoretical framework to detect masquerades by formulating hypotheses and applying statistical theory to test them. We introduced ROC curves to make comparisons between different intrusion detection methods possible. We made a substantive discovery with the Uniqueness method introducing the popularity score as a valuable component of a command profile. We were able to combine the concepts of Dirichlet priors, shrinkage estimation, Bayes factors, and sparse matrix computation for the detection of masquerades. We introduced a way to toggle between different methods, in our case between the Multi-Step Markov and the Independence Model. We found ways to reduce the dimensionality of the Multi-Step Markov model.

One of our goals was to bring this exciting area to the attention of the statistical community. We believe that there is a lot of room for further investigation.

6 Acknowledgements

We are very grateful to Brian Davison (IPAM) and Terran Lane (SequenceMatch) who agreed to run their intrusion tools on our data. We thank Daryl Pregibon for generating the blinded experimental data and Allan Wilks for helping us to collect the command data in the first place. The work of Ju, Karr, Schonlau, and Vardi is funded in part by NSF grant DMS-9700867. M. Schonlau's work is also funded in part by NSF grant DMS-9208758. W. Ju's

and Y. Vardi's work is also funded in part by NSF grant DMS-97 04983 and NSA grant MDA 904-98-1-0027.

References

[1] Amoroso, E. (1999), *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*, Intrusion.Net Books, Sparta, New Jersey, (ISBN 0-9666700-7-8).

[2] Computer Misuse Detection System (CMDS) Website (1999), http://www.cmds.net/profiles.htm .

[3] Computer Security Institute (1998), "CSI/FBI computer crime and security survey results quantify financial losses", *Computer Security Alert*, no. 181, April.

[4] Davison, B. D. and Hirsh, H. (1998), "Predicting Sequences of User Actions" in : *Predicting the Future: AI Approaches to Time Series Problems*, published as Technical Report WS-98-07, (Proceedings of AAAI-98/ICML-98 Workshop) , pp.5-12, AAAI Press, Madison, Wisconsin .

[5] Denning, D. E. (1997), "Cyberspace Attacks and Countermeasures" in *Internet Besieged* Denning, D.E. and Denning P.J. (eds), ACM Press, New York, pp 29-55.

[6] Denning, D. E. and Denning P.J. (eds), (1997), *Internet Besieged*, ACM Press, New York.

[7] DuMouchel, W. (1999), "Computer intrusion detection based on Bayes Factors for comparing command transition probabilities", *National Institute of Statistical Sciences Technical Report No. 91* available at http://www.niss.org/downloadabletechreports.html

[8] DuMouchel, W., Schonlau, M., (1998a), "A Comparison of Test Statistics for Computer Intrusion Detection Based on Principal Components Regression of Transition Probabilities". *Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics*, (to appear).

[9] DuMouchel, W., Schonlau, M. (1998b), "A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities". in Proceedings: *The Fourth International Conference of Knowledge Discovery and Data Mining*, August 27-31, New York, pp. 189-193.

[10] Ju, W., Vardi Y. (1999), "A Hybrid High-order Markov Chain Model for Computer Intrusion Detection", *National Institute of Statistical Sciences. Technical Report No. 92*, available at http://www.niss.org/downloadabletechreports.html

[11] Lane, T. and Brodley, C.E. (1998), *Approaches to online learning and concept drift for user identification in computer security*, in Proceedings: *The Fourth International Conference of Knowledge Discovery and Data Mining*, August 27-31, New York, pp. 259-263.

[12] President's Commission on Critical Infrastructure Protection, (1998), "Critical Foundations", United States Government Printing Office, GPO 040-000-00699-1. Also available online at http://www.pccip.gov/report_index.html

[13] Raftery, A. E. (1985), "A Model for High-order Markov Chains", *Journal of the Royal Statistical Society*, Ser. B, 47, 528-539.

[14] Raftery, A. E. and Tavaré, S. (1994), "Estimation and Modeling Repeated Patterns in High Order Markov Chains with the Mixture Transition Distribution Model", *Applied Statistics*, 43, 179-199.

[15] Theus, M., Schonlau, M., (1998), "Intrusion Detection Based on Structural Zeroes". *Statistical Computing & Graphics Newsletter*. Vol. 9, No 1, 12 - 17.

[16] Welch, T. A. (1984), "A Technique for High Performance Data Compression", *IEEE Computer*, vol 17, no. 6,pp 8-19.