*Statistical Science*
2001, Vol. 16, No. 1, 58–74

# Computer Intrusion: Detecting Masquerades

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

*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 six methods, including two approaches from the computer science community.

*Key words and phrases:* Anomaly, Bayes, compression, computer security, high-order Markov, profiling, Unix.

## 1. INTRODUCTION

Intrusion detection in computer science is an important and widespread problem, as evidenced by the report of the President's Commission on Critical Infrastructure Protection (1998). Attacks on popular Web sites (such as occurred in February 2000) inflict significant economic loss. In addition, intrusion threatens the privacy of individuals who use computers and the Internet.

We address in this paper the question "What opportunities and challenges does the problem of intrusion detection offer to statisticians?"

The problem of intrusion detection is inherently statistical because it is data-driven. The data, whose sources range from network hardware to system log files, are of both immense scale (gigabytes per day in some settings) and unusual nature (many data elements are text). New techniques may be necessary to analyze data that can be retained only fleetingly or not at all.

There is still another challenge: data with *real* intrusions are very difficult to obtain. Some intrusions are undetected, and many system operators are reluctant to release data that, in effect, admit that intrusions have occurred.

To provide a partial answer to the question in view of these difficulties, we designed and conducted an experiment, using synthesized intrusions that make the problem hard. Intruders are "other" users from a population similar to the legitimate users. To help assess the opportunities for statistics and to understand the richness of the problem, multiple statistical approaches were employed. Like any experiment, ours involved seemingly "arbitrary" decisions, but their net effect was to stress the methods.

As discussed in more detail in Section 6, we conclude that statistical methods *can* find intrusions, even in difficult circumstances. Moreover, different methods have identifiable (and even quantifiable) strengths and weaknesses. Significant challenges for statistics and statisticians remain.

This article is structured as follows: In the next section we discuss the background of the problem. In Section 3 we describe the data and the experiment that we designed to compare several anomaly detection methods. In Section 4 we describe our methods and also two approaches from the computer science community. Section 5 then analyzes the results of the experiment and Section 6 concludes with a discussion.

*Matthias Schonlau is with RAND, 1700 Main Street, Santa Monica, California 90407-2138. William DuMouchel is with AT&T Labs Research, 180 Park Avenue, Shannon Laboratory, Florham Park, New Jersey 07932. Wen-Hua Ju is with Avaya Labs Research, 2C-280, 700 Mountain Ave., Murray Hill, New Jersey 07974-0636. Alan F. Karr is with the National Institute of Statistical Sciences, 19 Alexander Drive, Research Triangle Park, North Carolina 27709-4006. Martin Theus is with VIAG Interkom, Marsststr. 33, 80335 Muenchen, Germany. Yehuda Vardi is with Rutgers University, Department of Statistics, 110 Frelinghuysen Rd., Piscataway, New Jersey 08854-8019.*

## 2. BACKGROUND

There are many different types of intrusions. Denning (1997) divides attacks into eight 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 e-mail address, or by gaining password access);
- Jamming or flooding (overwhelming a system's resources, e.g., by an e-mail flood or HTTP requests);
- Injecting malicious code such as viruses and Trojan horses (via floppy disks or e-mail attachments);
- Exploiting design or implementation flaws (often buffer overflows, which overwrite other data and can be used to get control over a system);
- Cracking passwords and keys.

We focus here on a common form of spoofing, namely on detecting masquerades—people who hide their identity by impersonating other people on the computer. Masqueraders could be insiders or outsiders. In practice, though, they are mostly insiders because most outside intruders immediately try to gain access to the account of the superuser and therefore are a special case.

The annual computer crime and security survey (Power, 1999, page 26) shows that attacks detected stemming from "insider abuse of net access" and "unauthorized access by insiders" are on the rise. Together, they surpass the number of viruses and system penetrations detected. Financial losses attributed to insiders are of the same order of magnitude as losses attributed to outsiders (Power, 1999, page 32).

Methods for computer intrusion detection fall into two broad categories: pattern recognition (also called misuse detection) and anomaly detection. All commercial intrusion detection systems and comprehensive research efforts of which we are aware use pattern recognition, while some, like IDES (Lunt et al., 1988), NIDES and Emerald (Porras and Neumann, 1997) use both approaches. Comparing several intrusion detection systems based on pattern recognition, Lippmann et al. (2000) concluded that "research should focus on developing techniques to find new attacks instead of extending existing rule-based approaches."

Pattern recognition refers to attempting to recognize the attack signatures of previously observed intrusions (for example, frequent changes of directory or attempts to read a password file). Computer scientists appear to consider pattern recognition as the first line of defense. Clearly, pattern recognition 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 intrusion. Anomaly detection is probably most appropriate for detecting masquerades.

The literature contains a vast array of specific approaches to computer intrusion detection. For a general overview see Denning and Denning (1997) or Amoroso (1998). Work in the area includes ideas inspired by biological immune systems (Forrest, Hofmeyr, Somayaji and Longstaff, 1996), applied work for the statistical anomaly detector(s) at SRI (Javitz and Valdes, 1993), black-box approaches using neural networks (Tan, 1995), graph-based systems for very large networks (Staniford-Chen et al., 1996), characterizing machines by their network activity and flagging low probability events (Marchette, 1999), and stand-alone intrusion detection systems such as Bro (Paxson, 1998). In Section 4 we describe two computer science approaches to anomaly detection that are directly relevant to this article. Scott (2000) develops a Markov modulated nonhomogeneous Poisson process model for telephone fraud detection. Scott uses only event time data, but detectors based on Scott's model may be combined with other intrusion detection algorithms to accumulate evidence of intrusion across continuous time.

## 3. DATA AND EXPERIMENTAL DESIGN

### 3.1 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, write or execute `myfile`.

Our data source is the UNIX `acct` auditing mechanism. Examples of some auditing entries are given

TABLE 1
*Examples of accounting entries generated by the* UNIX `acct` *auditing mechanism*

| **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 |

in Table 1. Our analysis is only based on the first 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 required to collect 15,000 commands differs vastly from user to user. Some users generate this many commands in a few days, others in a few months.

While an analysis using arguments as well as commands would be desirable, arguments of commands were not collected because of privacy concerns. Some commands recorded by the system are not explicitly typed by the user. For example, a shell file is a file that contains multiple commands, and running a shell file will cause all of its commands to be recorded. Other examples are so-called `.profile` files and `make` files. Names of executable programs are also interpreted as commands because they are recorded in the audit stream.

## 3.2 Experimental Design

We randomly selected 50 users to serve as intrusion targets. We then used the remaining 20 users as masqueraders and interspersed their data into the data of the 50 users. Using users as masqueraders is a worst case scenario for an already difficult problem, for two reasons. First, there is a good chance that a “real” masquerade will exhibit an unknown unusual pattern, and it is easier to distinguish between unusual and normal patterns than it is to distinguish among several normal usage patterns. Second, using normal users instead of data from real masquerades means that we do not make any implicit assumptions about the masquerades. This is desirable since we avoid restricting ourselves to known attacks. Additional assumptions, however, would make detection much easier.

In essence, in detecting masquerades we try to classify data into two groups that one might label “good” and “bad.” However, we have to characterize “good” in the absence of training data for “bad.”

For simplicity, because it facilitates presentation of the results, we decided to decompose each user’s data into 150 blocks of 100 commands each. (We ran this particular experiment only once because we did not want to appear to use the best out of several runs. Previously, the methods were used to investigate block sizes of 100, 500 and 1000. As one would expect, a longer block leads to somewhat better discrimination.) 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%. While the exact values of the probabilities 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.

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. (In some ways, therefore, the experiment resembles “contests” held to evaluate intrusion detection methods; these also typically use inserted intrusions.) The investigators knew the probabilities with which a masquerader would appear and disappear but were not allowed to use this information. The fact that intrusions have lengths that are multiples of 100 commands also could not be used. The only piece of information used was the fact that masquerades only start at the beginning of blocks; thus the individual tests for masqueraders also started at the beginning of blocks. We started at the beginning of a block for convenience. It makes it much easier to keep track of false and true alarms. It is also not clear how to interpret an alarm for a semi-contaminated block for the ROC curves. Any advantage that the detection system had by knowing that intrusion starts at the beginning of a block would have been slight in the overall scheme. If the masquerader started at arbitrary points, only the first block in the masquerade would contain both contaminated and uncontaminated commands, all subsequent blocks for the same masquerader would be contaminated (except for the last block).

The data used in this experiment are available for download from `http://www.schonlau.net` and from `http://www.niss.org`.

## 4. OVERVIEW OF METHODS

In what follows we describe distinct approaches labeled, "Uniqueness," "Bayes one-step Markov," "Hybrid multistep 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 standalone intrusion detection systems. Integration of subsystems is not discussed here and is one of the areas that merit further research.

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."

### 4.1 Uniqueness

The uniqueness approach is based on the idea that commands not previously seen in the training data may indicate an attempted masquerade. Moreover, the fewer users that are known to use that command, the more indicative that command is of a masquerade. This approach is due to Schonlau and Theus (2000).

4.1.1 *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 and many more are unpopular. Moreover, uniquely used commands account for 3.0% of the data, and commands used by five users or fewer account for 8.3% of the data. This is a testimony to the diversity within the user communities of UNIX systems. Note that spotting unusual behavior is generally easier in a homogeneous user community than in a highly diverse one.

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 so that commands from groups with unpopular commands are assigned lower ID's than commands from groups with more popular commands. The

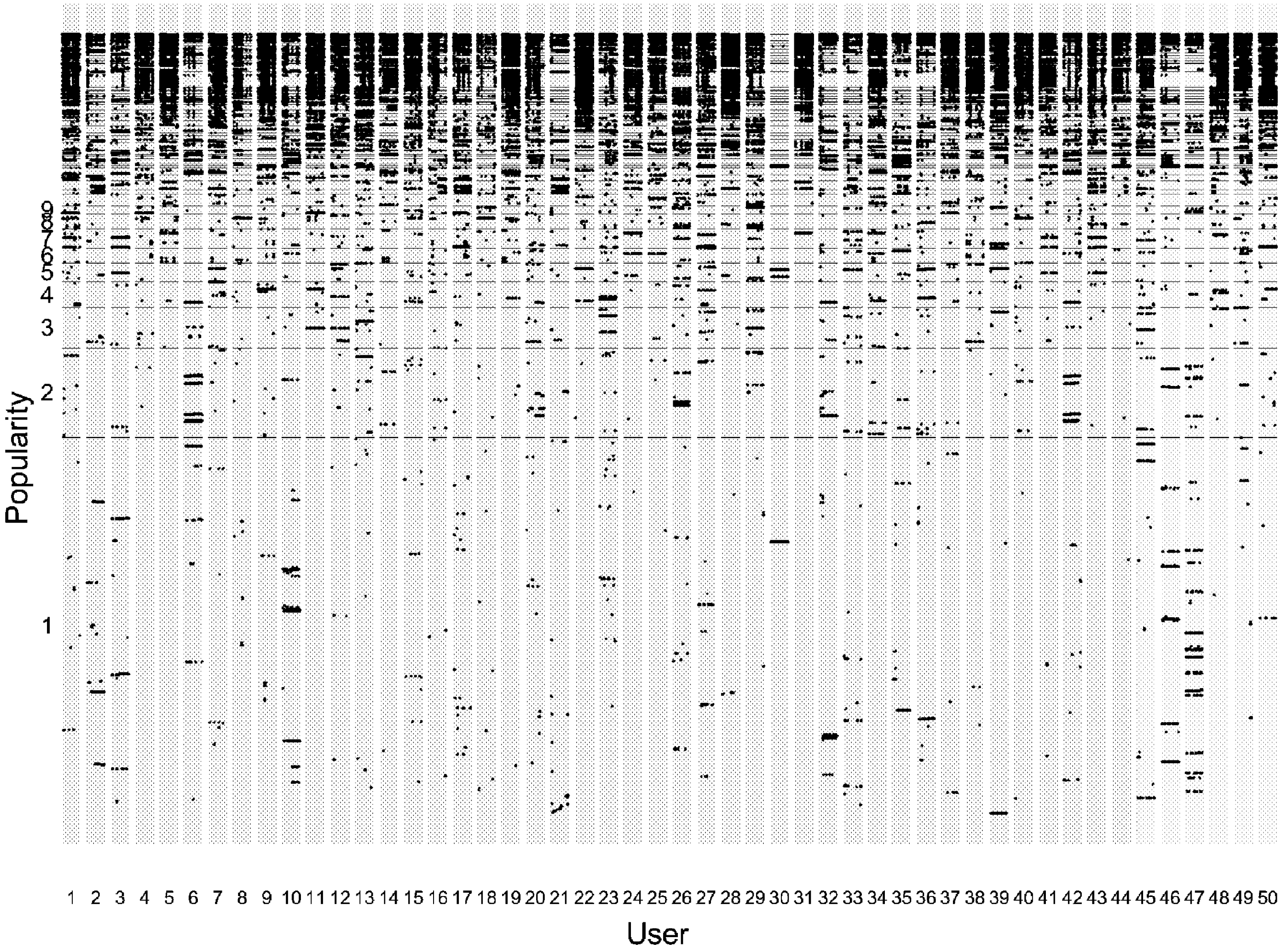

FIG. 1. *Visualization of command popularity. Each panel corresponds to one user. Within each user's panel, each command is represented by a single dot. Each command is assigned a distinct ordinate such that commands with higher popularities have greater ordinates than those with lower popularities. Commands are plotted versus "time" within each panel. That is, for each user we plot* 5,000 *points. The coordinates for any one point are given by its entry "time" (the position within the* 5,000 *points) and the command ordinate.*

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, as we have already noted, that about half of all commands are uniquely used. That many more commands are unpopular can also be seen from Figure 1.

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

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

where the weights $W_{uk}$ are

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

where

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

and

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

The fraction $(1 - U_k/U)$ acts as a uniqueness index. It is 0 if all users have used this command before and 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 who uses commands similar to the ones 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.

4.1.3 *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 five times, each time assigning scores to a distinct set of 1000 commands. We set the threshold to the 99th percentile of the combined scores across all users and all five 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 4.4.2). It turned out that individual thresholds tended to yield a false alarm–missing alarm trade-off 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: for example, a negative score indicates that the test data contains few or no incidences of new commands and therefore are not suspicious.

4.1.4 *Updating.* This method is particularly sensitive to contamination of the training data. Using all alarm-free test data for updating training data 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, 11,000 and 13,000). While nonsimultaneous updating is possible, we chose to update simultaneously for all users, as if all users were submitting commands at the same rate.

## 4.2 Bayes One-Step Markov

The uniqueness approach only considered command frequencies. The Bayes one-step Markov approach goes further: it is based on one-step transitions from one command to the next. The approach, due to DuMouchel (1999), uses a Bayes factor statistic to test the null hypothesis that the observed one-step command transition probabilities are consistent with the historical transition matrix.

4.2.1 *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,

$$\begin{aligned} &H_0\colon P(C_t = k \mid C_{t-1} = j) = p_{ujk},\\ &H_1\colon P(C_t = k \mid C_{t-1} = j) = Q_k, \end{aligned} \tag{2}$$

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

where $p_{ujk}$ is the historical transition probability from command $j$ to command $k$ for user $u$, that is,

$$p_{ujk} = P(\text{next command} = k \mid \text{previous command} = j,\ \text{User} = u).$$

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

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

$$q_{uk} = P(\text{next command} = k \mid \text{User} = u)$$

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,

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

The $\alpha_{uk}$ above and the $\alpha_{0k}$ 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 4.1.

We test the hypothesis $H_0$ versus $H_1$ in (2) by forming the Bayes factor $BF$, the ratio of the probabilities of the data under the two hypotheses,

$$BF = P(C_1, \ldots, C_T|H_1)/P(C_1, \ldots, C_T|H_0).$$

The larger $BF$ is, the more evidence there is against $H_0$ in favor of $H_1$. In fact, $x = \log(BF)$ is 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

$$BF = \frac{\prod_k \big(\alpha_{0k}(\alpha_{0k}+1)\cdots(\alpha_{0k}+n_{u.k}-1)\big)}{\alpha_{0.}(\alpha_{0.}+1)\cdots(\alpha_{0.}+T-1)\prod_{j,k} p_{ujk}^{n_{ujk}}}.$$

4.2.2 *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:

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

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.

4.2.3 *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 $N_{ujk} \leftarrow N_{ujk}2^{-n_u/5000} + n_{ujk}$ so that command relative frequencies have a "half-life" of 5000 commands. The user specific thresholds are also updated by an exponential weighting algorithm.

### 4.3 Hybrid Multistep Markov

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

We overcome the high dimensionality inherent in a multistep Markov chain as follows: (1) restrict attention to a subset of the most used commands (with the remaining commands represented under a single "command" labeled other), (2) use a *mixture transition distribution* (MTD) approach to model the transition probabilities (Raftery, 1985; Raftery and Tavaré, 1994).

When the 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 versus commands may be more appropriate. Our method automatically toggles between the two models as needed.

4.3.1 *The multistep 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 $\mathsf{other}_u$. We then combine the most frequently used $K_u$-1 commands and $\mathsf{other}_u$ 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

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

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

We fix $l = 10$ in our experiments and make the simplifying assumptions that for all users $u$: (1) $r_u(\mathsf{other}_u|i) = \varepsilon$, $\forall\, i \in M_u$, where $\varepsilon$ is small (we take $\varepsilon = 0.00001$), and (2) $r_u(i|\mathsf{other}_u) = \frac{1-\varepsilon}{K_u-1}$, $\forall\, i \in M_u$ 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 value. Note that $\mathsf{other}_u$ contains rarely used commands. The rationale for choosing $\varepsilon = 0.00001$ is that there is a 1% chance to get a rare command and there are (roughly) 1000 commands in $\mathsf{other}_\mathsf{u}$.

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

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

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

4.3.2 *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:

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

where $w_k = 1 + 1/U - U_k/U$. 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 $\varepsilon$ if $N'_{uk} = 0$ (to avoid taking the logarithm of zero).

4.3.3 *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 the log-likelihood-ratio statistic for the Markov and independence models, respectively, as follows:

$$X_{1u} = \log \frac{\max_{v \neq u} L(c_1, \ldots, c_T | \widehat{\mathbf{\Lambda}}_v, \widehat{\mathbf{R}}_v)}{L(c_1, \ldots, c_T | \widehat{\mathbf{\Lambda}}_u, \widehat{\mathbf{R}}_u)}$$

and

$$X_{2u} = \log \frac{\max_{v \neq u} \prod_i q_{vc_i}}{\prod_i q_{uc_i}}.$$

To bring the two statistics $X_1$ and $X_2$ into the same scale, we regress $X_1$ on $X_2$ 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 obtain the estimate $\hat{\rho}$ of $\rho$.

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," $x_u$:

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

This results in a hybrid method that switches automatically between the Markov and independence models according to whether the test data are "usual" or "unusual." The latter occurred 8% of the time in our data set.

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

4.3.5 *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 re-estimated based on the augmented training data at each update. The threshold parameters, $\mu$ and $\sigma$, 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.

### 4.4 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.

4.4.1 *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. The compression algorithm builds compression rules from the beginning of the file. It seems more natural to us that the compression algorithm starts the training (rule building) with the training data also. To accomplish that, the training data needs to come before the test data rather than the other way around.

We define the score $x$ to be the number of additional bytes needed to compress test data when appended to training data,

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

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).

4.4.2 *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

$$x_{ub}^{cv} = \text{compress}(C) - \text{compress}\big(C - C_{ub}^{100}\big), \quad b = 1, \dots, 50,\ u = 1, \dots, U,$$

where $C_{ub}^{100}$ 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. Because we cannot determine the 99th 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. That is,

$$\text{threshold}_u = \bar{x}_u^{cv} + \Delta,$$

where $\Delta$ is determined from the pooled data across all users,

$$\Delta = x_{0.99}^{cv} - \bar{x}^{cv}, \tag{5}$$

where $x_{0.99}^{cv}$ is the sample 99th percentile of $x_{ub}^{cv}$. Note that the choice of $x_{0.99}^{cv} = 0$ leads to the thresholds that were proposed in Section 4.2.2 for the Bayes one-step Markov model (except for a shrinkage factor of 1/2).

4.4.3 *Updating.* After every 2000 test commands for a given user (i.e., after commands 7000, 9000, 11,000 and 13,000) 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 $\Delta$). If one or more alarms were raised, the previous training data and threshold remain in place.

### 4.5 Additional Methods

We have not been able to use commercial intrusion detection tools on our data. A problem with the commercial systems is that they usually require data at the system call level and then reconstruct the UNIX audit trail from the system calls. Also, because the commercial systems look at more than one variable, it is difficult to isolate the detector for any one variable.

We did, however, compare our methods to two methods from the computer science community, which we describe very briefly here.

4.5.1 *IPAM.* "Incremental probabilistic action modeling" (IPAM) is the name of an algorithm by Davison and Hirsh (1998). It is based on one-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 $\alpha$, and $1 - \alpha$ is added to the most recent transition. Davison and Hirsh's choice of $\alpha$ is 0.9.

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.

4.5.2 *Sequence-match.* For each new command, Lane and Brodley (1998) compute a similarity measure between the most recent 10 commands and a user'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. Because 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.

## 5. RESULTS

The experimental set-up described in Section 3 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.

### 5.1 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 multistep Markov, Bayes one-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 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 alarm rates range from 30 to 60 per cent. In looking at these large percentages one should keep in mind that the problem is very hard and can be viewed as a worst case scenario, and that these are not stand-alone intrusion systems. 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 four 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.

### 5.2 Selected Individuals

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

The panels in Figure 3 correspond to eight particular users. Each panel 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 test 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 in Figure 3 indicates the presence of a masquerader. Lines crossing above the threshold when no gray shading is present are thus false alarms.

We only comment here on the first panel and let the reader draw other conclusions from the graph in Figure 2. For 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 approximately between commands 8200 and 8900 consist of repeated sequences `sed, ln, sh`. User 9 had used all three commands separately before, but the one-step transitions from `sed` to `ln` and from `ln` to `sh` had never appeared before.

### 5.3 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 multistep Markov with a correlation coefficient of 0.79 clearly are part of the same cluster. This may be because the hybrid elements of the hybrid multistep 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 methods is 0.62. Depending on the clustering algorithm, the Bayes one-step Markov method could be associated with

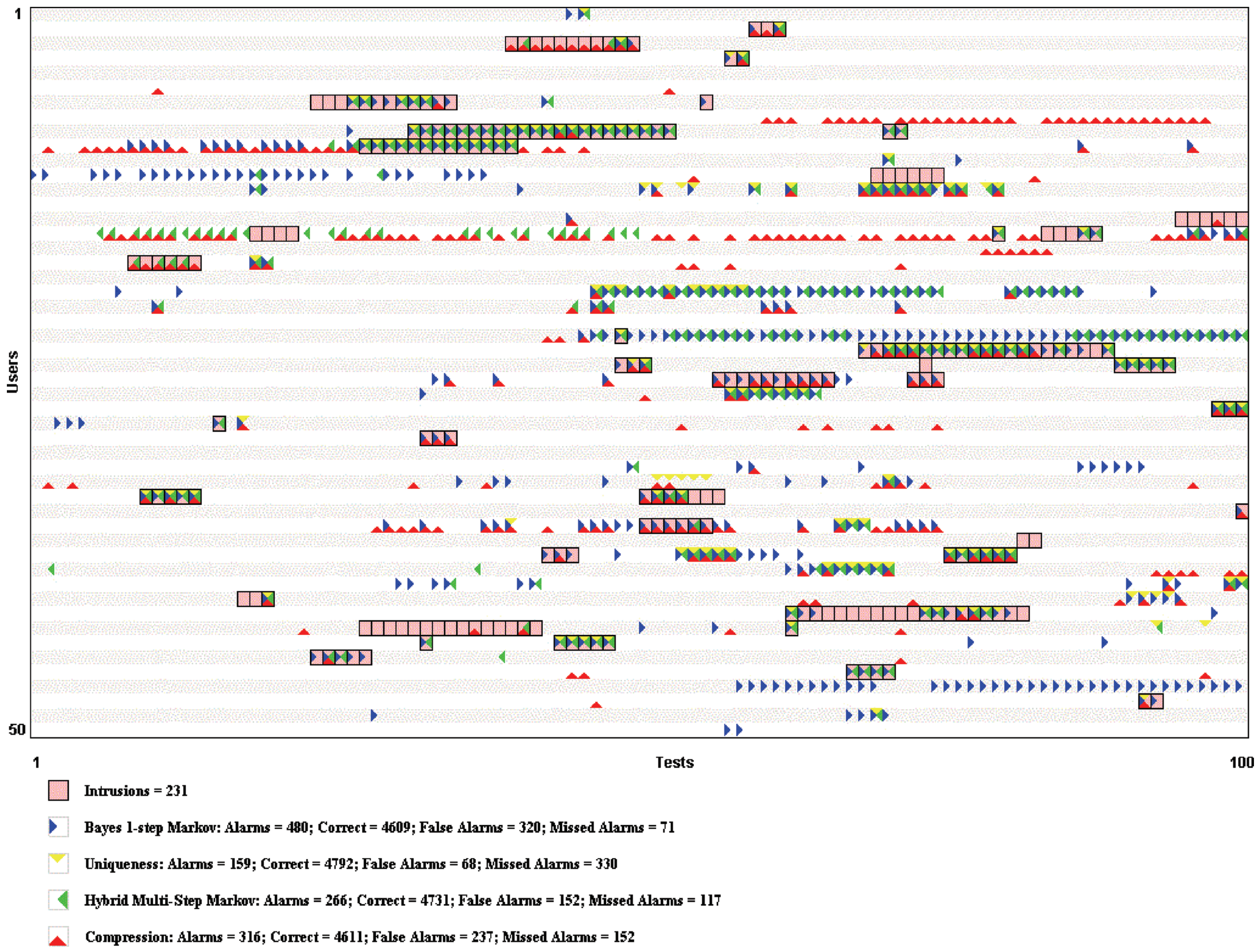

FIG. 2. *Plot of alarms given by various methods for each of* 50 *users. Compromised data have a red background shading.*

TABLE 2
*False alarms and missing alarms for all methods*

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

[a] The investigators corresponding to the first four methods were asked to target a false alarm rate of 1%

either of these two groups. Given that both IPAM and Bayes one-step Markov are based on the one-step command transition matrix, it is surprising that they 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 0.57.

### 5.4 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. Lippmann et al. (2000) used ROC curves in their comparison of several intrusion detection systems based on pattern recognition.

Because 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, 1% false alarms corresponds to about 70% missing alarms, and 5% 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 method clearly dominates the other methods. IPAM and Sequence-Match are surprisingly similar for high false alarms. For a very low false alarm rate (less than 0.5%), they do better than all other methods.

When using ROC curves, the unit of analysis is the block rather than the intrusion. (The unit of analysis for ROC curves is the block rather than the intrusion because if we only look at intrusion data for the ROC curve, and ignore the "clean" data, we would both ignore important data and would not be able to establish false alarm rates.) 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.

### 5.5 ROC Curves without Updating

All analyses were based on the updating algorithms as described in Section 4. 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: because we do not update training data with test data that had high scores (in order to avoid contamination with masquerading data), the average score for the training data tends to decrease. As a consequence, the thresholds, calculated from the updated training data, decrease, too. Eventually normal variation in the 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 multistep Markov methods benefited from updating. Given that the updating may inadvertently contaminate the training data with masquerading data, this outcome was not obvious a priori.

## 6. DISCUSSION

Our first conclusion is that statistical methods can detect intrusions, even in difficult circumstances. All of the methods presented here detect anomalies in command usage, and do so surprisingly well, considering the paucity of information.

In essence, in detecting masquerades we try to classify data into two groups that 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 multistep Markov method explicitly assumes that the masquerader resembles one of the other users. To a lesser extent, the

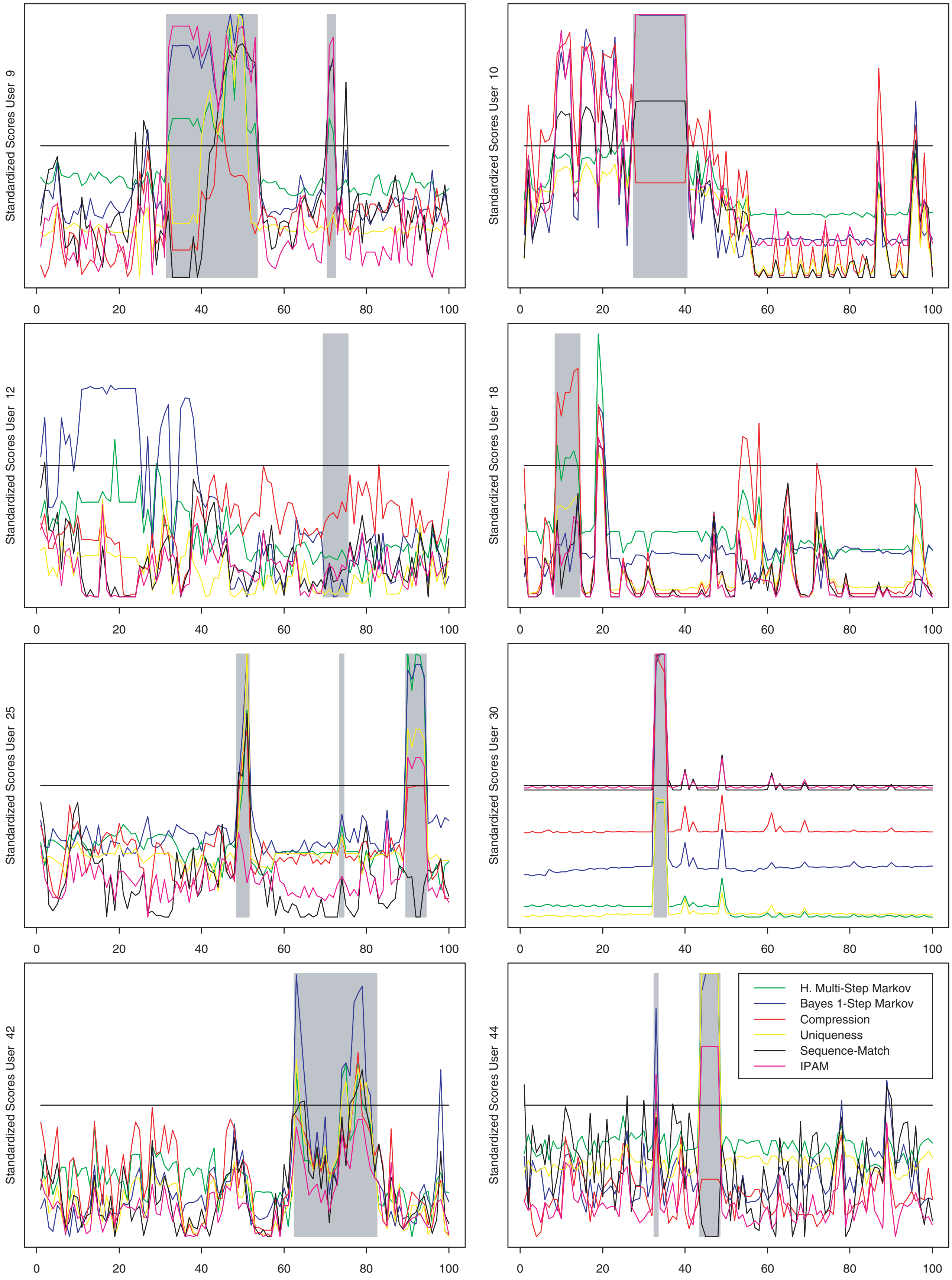

FIG. 3. *Plot of standardized scores over "time" for eight different users.*

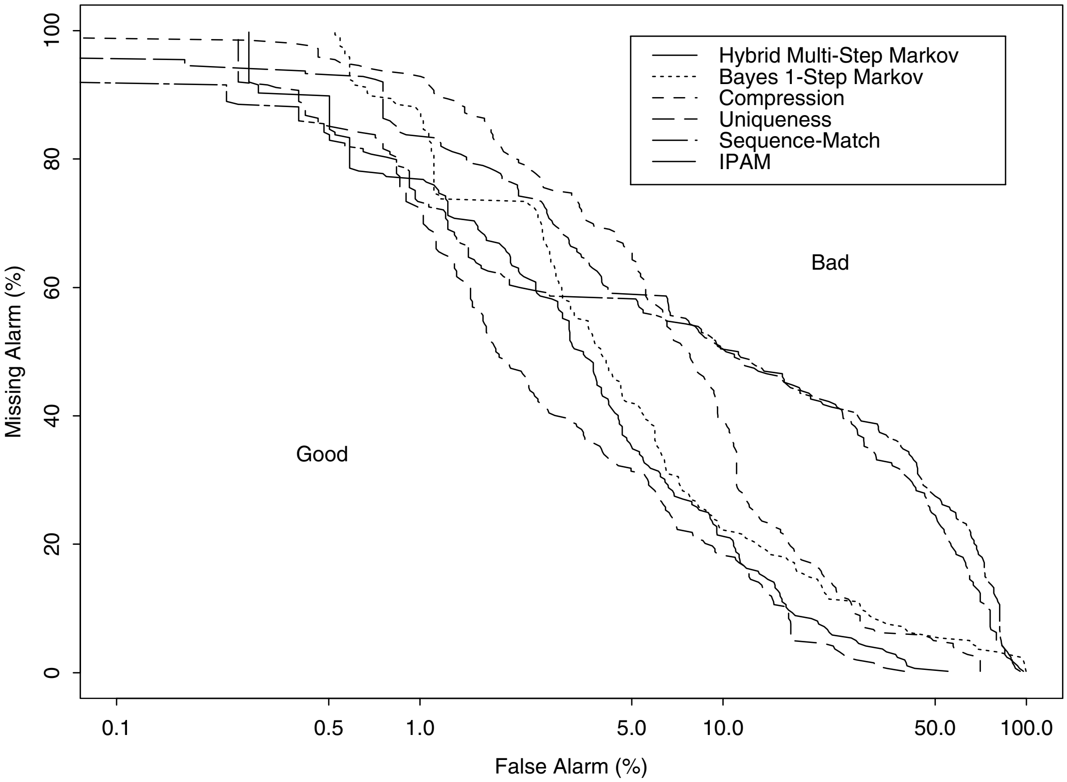

FIG. 4. *ROC curves for all methods. Methods use updating.*

uniqueness method and the Bayes one-step Markov method also make similar assumptions through the concepts of command popularity and parameter estimation of the Dirichlet distribution, respectively. The compression method has an inherent notion of what “bad” constitutes.

The second principal conclusion is that our experiment demonstrates that the methods have rather different characteristics. First, 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, and also has difficulties with updating algorithms. The multistep Markov method, while very successful, is computationally demanding. Even though they were only used 8% of the time, the hybrid elements of the hybrid multistep Markov method did improve its ROC curve.

Finally, the Bayes one-step Markov method did slightly worse than the Markov and the uniqueness methods, but it is less sensitive to contamination in the training data 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, 1999; DuMouchel and Schonlau, 1999). This approach was abandoned because it required very voluminous user profiles and was not easily extended to make use of the information in unpopular–unique commands.

To some extent, the performances of the various methods are 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.

Intrusion detection is a complex problem requiring systems and strategies (authentication, attack signatures, anomaly detection), rather than iso-

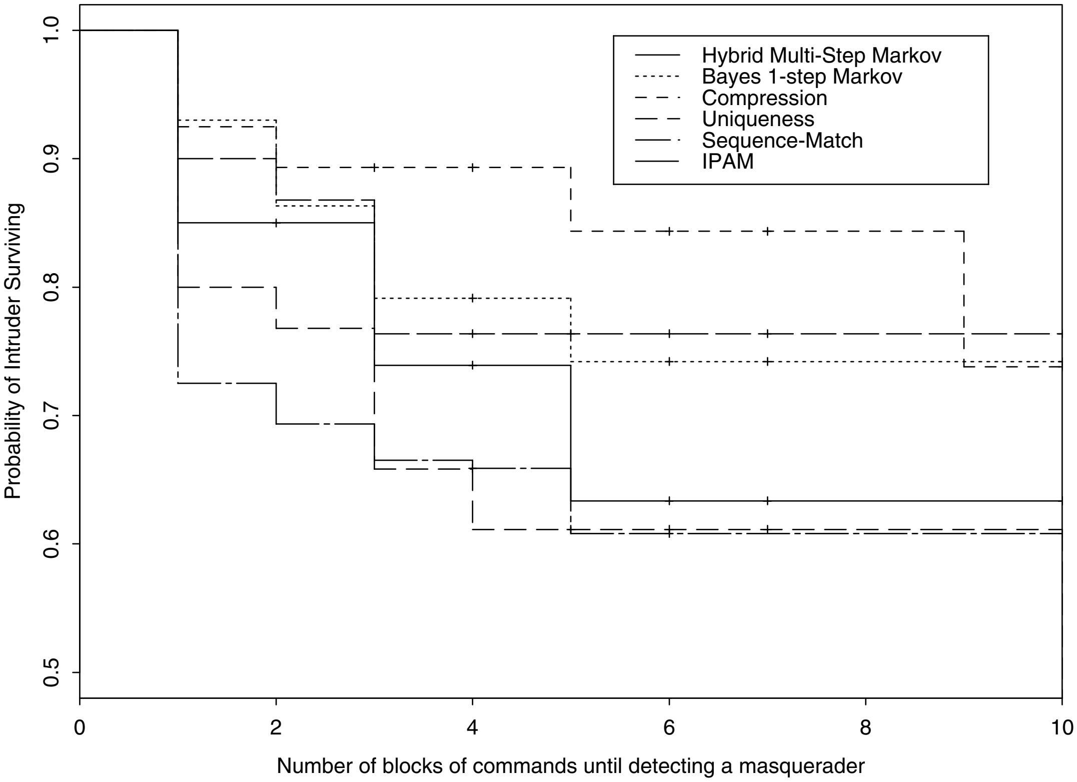

FIG. 5. *Survival plot of intrusions for all methods given a fixed false alarm rate of* 1%. *Ideally there would be no detection delay, that is, instant "death" of the intrusion.*

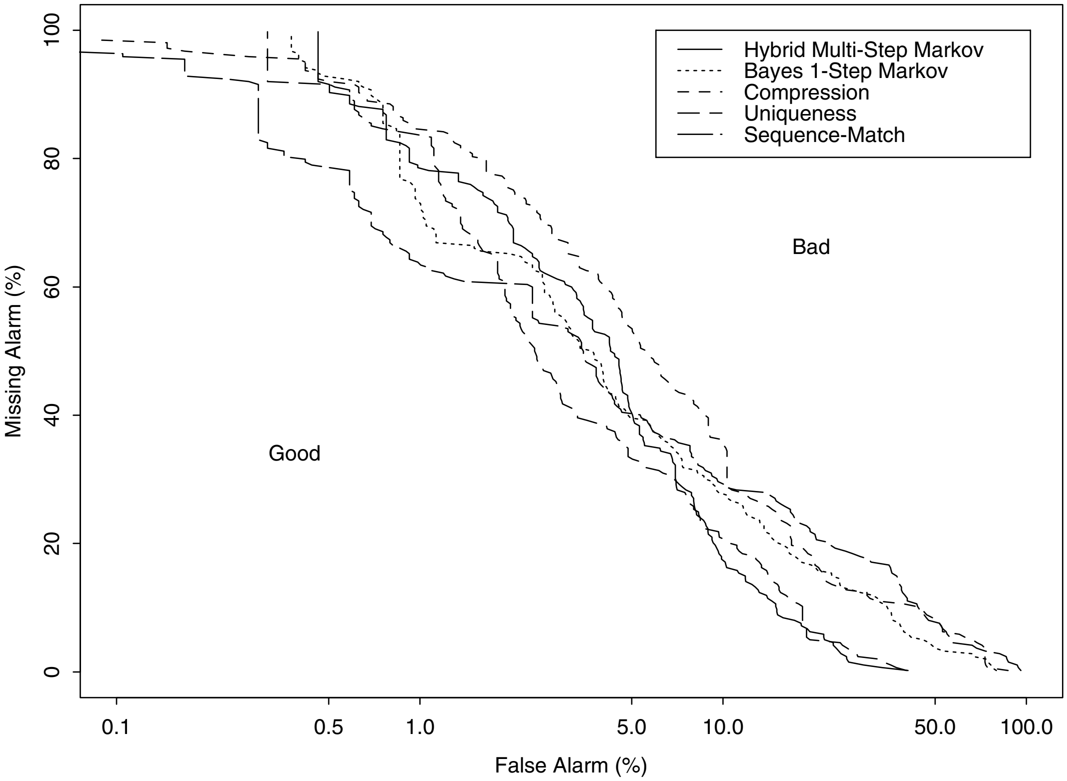

FIG. 6. *ROC curves for all methods except IPAM for which no data was available. Methods do not use updating.*

lated techniques. None of the methods described here could sensibly serve as the sole means of detecting computer intrusions—the rates of successful masquerades preclude this. These tools could, however, be used in conjunction with pattern recognition (or misuse detection) approaches, or other techniques such as profiling based on file–directories accessed and/or biometrics such as keystroke timings. Such combined strategies could be used to reduce false alarms.

To summarize, we have presented a framework to detect masquerades by formulating hypotheses and applying statistical theory to test them. We used ROC curves to make comparisons between different intrusion detection methods possible. We introduced the popularity score as a valuable component of a command profile for the uniqueness method. 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 multistep Markov and the independence model. We found ways to reduce the dimensionality of the multistep Markov model.

Our final conclusion is that many challenges and opportunities for statistics and statisticians remain. These include:

- Dealing with the complexity of the problem. Because intruder behavior is both complex and diverse, intrusion detection does not easily lend itself to the usual divide-and-conquer paradigm, meaning that the problem cannot be broken into obvious smaller pieces that can then be solved separately.
- Dealing with the scale of the problem. The potential amount of data to consider is enormous and it is desirable to analyze the data in real time, not only because rapid decisions may be essential but also because the sheer volume of the data may preclude its being stored for later analysis.
- Making use of more detailed data (not only user and commands).
- Understanding the strengths and weaknesses of statistical methods in comparison to other ("computer science") techniques such as pattern recognition. For example, which techniques are good for sounding alarms, as compared to tracking someone who seems suspicious but has not yet triggered an alarm?
- Integration of statistical strategies into intrusion detection systems, including the opportunity for meta-analysis.
- Constructing graphical tools for computer intrusion detection.
- Statistical analysis of automatically generated alarms to distinguish between important and less important alarms (often the number of alarms generated is so large that they are partially or fully ignored).
- Transferring methods and tools to similar settings characterized by intrusions with anomalous behavior, such as credit card and long distance telephone fraud.

## ACKNOWLEDGMENTS

We are very grateful to Brian Davison (IPAM) and Terran Lane (Sequence-Match) 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-97-00867. Schonlau's work is also funded in part by NSF Grant DMS-92-08758. Ju's and Vardi's work is also funded in part by NSF Grant DMS-97-04983 and NSA Grant MDA 904-98-1-0027.

Schonlau was affiliated with the National Institute of Statistical Sciences, Ju with Rutgers University and Theus with AT&T Labs Research when the research was conducted.

## REFERENCES

Amoroso, E. (1999). *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*. Intrusion.Net Books, Sparta, NJ.

Davison, B. D. and Hirsh, H. (1998). Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time Series Problems*. Technical report WS-98-07 (Proceedings of AAAI-98/ICML-98 Workshop) 5–12. AAAI Press, Madison, WI.

Denning, D. E. (1997). Cyberspace attacks and countermeasures. In *Internet Besieged* (D. E. Denning and P. J. Denning, eds.) 29–55. ACM Press, New York.

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

DuMouchel, W. (1999). Computer intrusion detection based on Bayes Factors for comparing command transition probabilities. Technical Report 91, National Institute of Statistical Sciences. Available at www.niss.org/downloadable-techreports.html.

DuMouchel, W. and Schonlau, M. (1998). A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities. In *Proceedings of The Fourth International Conference of Knowledge Discovery and Data Mining* 189–193. New York.

DuMouchel, W. and Schonlau, M. (1999). A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities. In *Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics* **30** 404–413.

Forrest, S., Hofmeyr, S., Somayaji, A. and Longstaff, T. A. (1996). A sense of self for Unix processes. *IEEE Symposium on Security and Privacy*, Oakland, California.

Javitz, H. S. and Valdes, A. (1993). The NIDES statistical component: description and justification. Technical report, SRI International, Menlo Park, CA.

Ju, W. and Vardi, Y. (1999). A hybrid high-order Markov chain model for computer intrusion detection. Technical Report 92, National Institute Statistical Sciences. Available at www.niss.org/downloadabletechreports.html.

Lane, T. and Brodley, C. E. (1998). Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the Fourth International Conference of Knowledge Discovery and Data Mining* 259–263. AAAI Press, Menlo Park, CA.

Lippmann, R., Fried, D., Graf, I., Haines, J., Kendall, K., McClung, D., Weber, D., Webster, S., Wyschogrod, D., Cunningham, R. and Zissman, M. (2000). Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. MIT Lincoln Laboratory. Unpublished manuscript.

Lunt, T. F., Jagannathan, R., Lee, R., Listgarten, S., Edwards, D. L., Neumann, P. G., Javitz, H. S. and Valdes, A. (1988). Development and application of IDES: A real-time intrusion-detection expert system. Technical report, Computer Science Laboratory SRI International, Menlo Park, CA.

Marchette, D. (1999). A statistical method for profiling network traffic. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring* 119–128.

Paxson, V. (1998). Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*.

Porras, P. and Neumann, P. (1997). Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the National Information Systems Security Conference.*

Power, R. (1999). *Current and Future Danger: A CSI Primer on Computer Crime and Information Warfare*, 3rd ed. Computer Security Institute, San Francisco.

President's Commission on Critical Infrastructure Protection (1998). Critical Foundations. United States Government Printing Office, GPO 040-000-00699-1. Washington, DC.

Raftery, A. E. (1985). A model for high-order Markov chains. *J. Roy. Statist. Soc. Ser. B* **47** 528–539.

Raftery, A. E. and Tavare, S. (1994). Estimation and modeling of repeated patterns in high-order Markov chains with the mixture transition distribution model. *Appl. Statist.* **43** 179–199.

Scott, S. (2001). Detecting network intrusion using a Markov modulated nonhomogeneous Poisson process. Available at www-rcf.usc.edu/~sls/research.html.

Schonlau, M. and Theus, M. (2000). Detecting masquerades in intrusion detection based on unpopular commands. *Inform. Process. Lett.* **76** 33–38.

Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R. and Zerkle, D. (1996). GRIDS—A graph-based intrusion detection system for large networks. In *Proceedings The Nineteenth National Information Systems Security Conference.*

Tan, K. (1995). An application of neural networks to Unix computer security. In *IEEE International Conference on Neural Networks*. IEEE, New York.

Welch, T. A. (1984). A technique for high performance data compression. *IEEE Computer* **17** 8–19.