Detecting Masqueraders Using High Frequency Commands as Signatures

Ming Dong Wan, Han-Ching Wu, Ying-Wei Kuo, James Marshall, Shou-Hsuan Stephen Huang
Department of Computer Science, University of Houston, Houston, TX 77204, USA
E-mail: shuang@cs.uh.edu

Abstract

Network intruders commonly use stolen passwords or other means to log into legitimate users' computer accounts. To prevent this from happening, it is important that we are able to distinguish a user as a true user or a masquerader. Uniqueness of user command has been used in the past as signature of users. This project explores the high frequency commands to see if they work well as signatures. Experimental result was provided to show that they work as well as the Uniqueness method. Besides, the comparisons with other methods were also presented.

Keywords: Intrusion Detection, Masqueraders, Profiles, Signatures, Network Security.

1. Introduction

More and more computer systems are subject to attack by network intruders. Intrusion detection system is becoming an increasingly important tool to detect and analyze security attacks for a computer system.

In general, the techniques of the Intrusion Detection System (IDS) can be categorized into two main types: Anomaly Detection Systems and Misuse Detection Systems. For misuse detection, an IDS builds the attack profile and compares it to either attack signatures. However, the major drawback of this type of detection is of little use for unknown attack techniques. The Anomaly Detection System is to monitor a host or a network to compare their states to the normal behaviors and look for all the anomalies [1]. Therefore, the goal is to establish a "normal activity profile" for a system, and detect all the system states varying from the established profile which is the subject of this paper.

In order to train the system to recognize normal system activity, some anomaly detection researches focus on building the profile databases with the system call sequences. Forrest et al. [2] determined normal behavior in terms of short and fixed length sequences of system calls in the training data. By building up a separate database of normal behavior for each process of interest, the constructed database could be used to monitor the process' ongoing behavior. However, processing high dimensional data is computationally costly and it is difficult to detect an intrusion in real time. The main limitation of this method is no rationale for selecting the optimal pattern length. Therefore, the concept of using dynamic window sizes to model the process behavior was introduced. Wespi et al. [3] developed a technique to generate tables of variable-length patterns automatically using the combination of the Teiresias algorithm and pattern-reduction algorithm. Eskin et al. [4] developed the technique which allowed the size of the window depended on the context rather than picked a fixed window size. The main advantage of using the dynamic window sizes is to decrease the size of the normal profile databases.

The other commonly used way to build up the normal system profile is using artificial intelligence type techniques. Debar et al. [5] used a recurrent neural network for the model. Part of the output of the network was fed back as input for the next step and created an internal memory inside the neural network. These networks kept a trace of past events into their internal memory. Since no explicit action could be taken to erase a specific event from the memory of the network, the network could remember all the events from the beginning.

Another type of approach is to define what normal usage of the system comprises using the mathematical model. Oka et al. [6] proposed Eigen co-occurrence matrix (ECM) method which was inspired by the Eigenface technique. This method modeled a sequence by using a co-occurrence matrix to correlate an event with any following events that appear within a certain distance. However, the computational cost of learning behaviors was significantly high. Wang et al. [7] built normal behavior models based on the frequencies of individual system calls or commands embedded in each segment of the data and applied Non-negative Matrix Factorization to extract the features from the blocks of audit data associated with the normal behaviors.

978-0-7695-3096-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WAINA.2008.38

More recently, Schonlau et al. [8, 9] proposed the uniqueness approach that studied the uniquely used and unpopular commands to detect the masqueraders. In contrast to their approach, we studied the other end of the spectrum, i.e., the *high frequency commands* (HFC) to characterize a user's normal behavior. Under Unix system, each user had a distinctive behavior using Unix commands. We built a profile for each user to represent the typical behavior. If the input data, called signature, deviated from the profile significantly, we should be able to identify the masqueraders. To verify the hypothesis that HFC served as a good signature, we devised algorithms to compute the profiles of HFC, signatures, and defined the dissimilarity between them. The larger the difference was, the more probable the signature was a masquerade.

2. High Frequency Command Method

Profile Algorithm. Assume that we have a command array Cmd[1..N] of N commands and the corresponding frequencies Freq[1..N]. To simplify the description, we further assume that they are already sorted in decreasing order of the frequencies. The task is to find the top *n* commands where *n* is a small integer number (n = 40 in our examples). The top n commands will be stored in PCmd[*1..n*] and their frequencies in PFreq[*1..n*]. In case some of the frequencies are zero, we define CmdSize to be between 1 and n as the size of the commands in the profile with non-zero frequencies.

Algorithm 1: Profile Algorithm

```
CmdSize = max{i | Freq[i] ≠ 0} ;
if (CmdSize>n) CmdSize = n;
for (i=1; i<=n; i++) {
   if (i<=CmdSize) {
      PCmd[i] = Cmd[i];
      PFreq[i] = Freq[i];
   else {
      PCmd[i] = "*"; // dummy
      PFreq[i] = 0;
```

Algorithm 2: Signature Algorithm

```
{
  for (i=1; i<=CmdSize; i++)
    for (j=1; j<=m; j++)
      if(PCmd[i]==Cmd[j])
         SFreq[i]=Freq[j];
  for (i=CmdSize; i<=n; i++)
    SFreq[i]=Freq[j]  where j is the smallest
            index of an unused command in Cmd;
}
```

Signature Algorithm. Once a user profile is built, it is used to build signatures. Notice that the signature of a user is dependent on its profile, in particular, the top *n* commands. The algorithm selects the frequencies of the top n commands of the signature out of *m* commands. However, in case some of the profile command frequencies are zero, the algorithm continues to select the next frequencies from the signature.

An example of a user profile and two signatures constructed using the above algorithms are given in Table 1. Signature 22 is a true signature of the user while Signature 48 is a masquerader.

Dissimilarity Algorithm. Once we computed the profile and signature of a user, we compared them directly or we could smooth them to see how different they were. Figure 1 shows a profile and two signatures of the same user. The profile (dark solid line) is sorted in decreasing order. However the signatures were of zig-zap shape. We applied a trendline to capture the trend of the original signature. For our experiments, we tested the first and the second order trendlines. We shall treat the original signature a 0-th order trendline. If the dissimilarity of the profile and the trendline was large, it meant that the signature behaved differently from the profile and potentially contained masquerader.

Trendlines of Order 1 and 2 were the smoothening method that used the first and second order polynomial trend-lines to compute the area differences between a signature and a profile. In the definition of the dissimilarity below, f_p is the frequency of the profile, f_s is the frequency of the signature, and n is the number of top frequency commands:

$$d_i = \frac{1}{t_n} \int_{x=0}^{n} | f_p(x) - f_s(x) | \, dx \qquad (1)$$

where $i \in \{1, 2\}$ d_1 and d_2 are the dissimilarities of Order 1 and Order 2, respectively and t_n is the accumulated frequency of top n commands. For Order 0 trendline, the definition of the dissimilarity of a signature and its profile becomes discrete:

$$d_0 = \frac{1}{t_n} \sum_{x=1}^{n} | f_p(x) - f_s(x) | \qquad (2)$$

Figures 1(a), (b) and (c) show a user profile and two signatures of the user using trendlines of orders 0, 1 and 2 respectively. One of the signatures was a true signature of the same user (Signature 22 in Table 1). The other signature (of a masquerader, Signature 48 in Table 1) was a contaminated signature by an intruder. This example illustrated the dissimilarity measure, showing that the behaviors of user's profile and its self-signature are similar, so that both lines are closer. On the other hand, the signature of the masquerader behaved very differently from the profile. It is probably easier to see the difference between the signatures with the trendlines in Figure 1(b) and 1(c) than that of 1(a).

Table 1: A sample profile and two signatures (only the top 10 commands are shown)

	Commands	Profile	Sig.22	Sig.48
1	egrep	17.44	16	0
2	expr	15.40	24	0
3	java	8.72	8	0
4	dirname	6.16	8	0
5	basename	4.42	4	0
6	.java_wr	4.36	4	0
7	make	3.66	6	0
8	diff	3.54	0	8
9	ls	3.50	8	2
10	uname	2.80	0	0

The example shown in Figure 1 is an ideal one. In our other experiments, the true signature might differ from the profile, or the masquerader signature looked similar to the profile. These situations would raise the false alarms (false rates). It makes sense that a user occasionally uses some unusual commands for special work requirements. These unusual commands will increase the dissimilarity between the profile and the signature. The problem is how high the dissimilarity value can be tolerated. A threshold shall be selected to separate a true user from a masquerader.

3. Experimental Results

In order to test our algorithms, we also applied the database provided by Schonlau et al [10]. The database contains 50 users with 15,000 commands for each user. For each user, the first 5,000 commands contain the *clean data*. The *masquerade data* is put into the remaining 10,000 commands for testing. We used the first 50 clean blocks to build the profile for each user, and used each masquerade block (from blocks 51 to 150) to build a signature. Hence, each user has one profile and 100 signatures. We conducted our experiment in three methods (orders 0, 1, and 2).

For each user, when we computed the area difference between its profile and each of its 100 signatures, we obtained 100 dissimilarity measures. Thus for the 50 users, there were a total of 5,000 dissimilarity measures. We compared the computed results with the true results provided from [10]. For a masquerade block, if the dissimilarity score was higher than the threshold, it was positively identified as true positive. On the other hand, if the score was lower than the threshold, a *false negative* (FN) happened. Similarly, for a clean block, if the dissimilarity score was lower than the threshold, then our algorithm made a correct prediction (true negative). If the score is higher than the threshold, then we had a *false positive* (FP) case. We conducted experiments to see what a reasonable threshold should be.

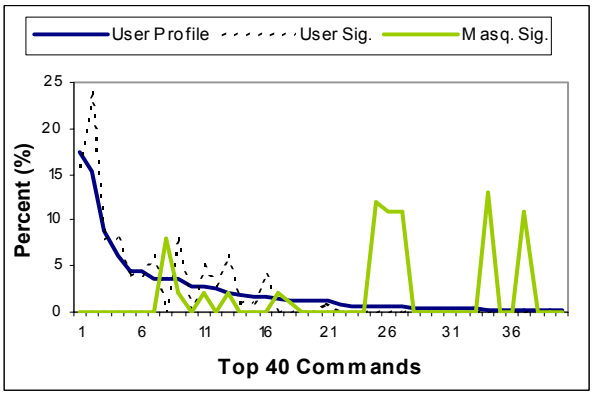

Figure 1(a). A profile and two signatures of order 0

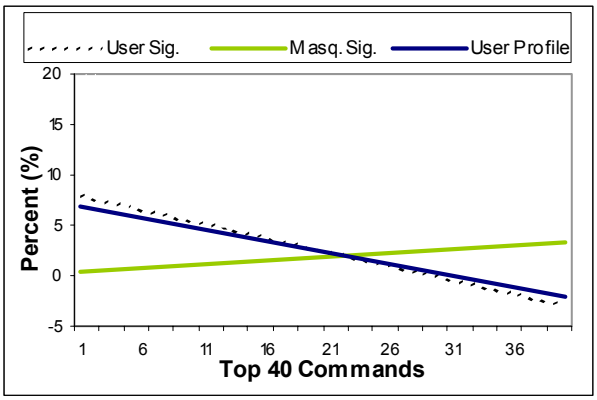

Figure 1(b). A profile and two signatures of order 1

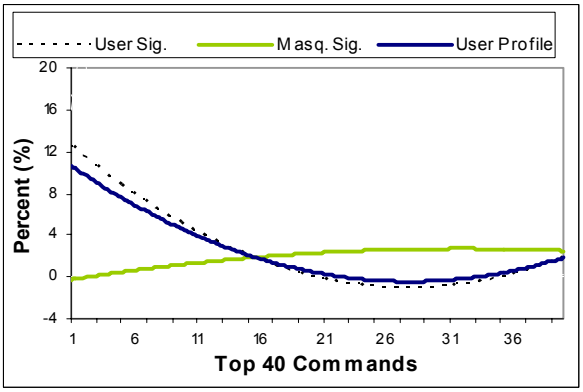

Figure 1(c). A profile and two signatures of order 2

Figure 2 shows the relationship between thresholds and false rates for the three trendlines. Order 0 had much higher FP rate than those of Orders 1 and 2. Contrary to our original expectation, Order 1 had slightly better false rates than Order 2. The thresholds were selected between 0.1 and 1.5. For example, by properly selecting the parameters, one could limit both FP and FN rate to about 20%. In addition, Figure 2 also

shows the performance of the HFC method that was competitive with the Uniqueness method.

A cross validation method was used first to select the threshold. For each user, the 5,000 clean commands were decomposed into two data sets: 4,000 commands for profile and the remaining 1,000 commands for signatures. This process was repeated five times. Then, the threshold was set to a certain percentage among the 250 dissimilarity values from the training data across all users and all five cross validations. For our training data, the thresholds for 98, 96, and 94 percentiles are 0.9839, 0.7614, and 0.6263 respectively.

In order to compare with other methods, we have to select one fix value, say FN rate, and compare the other value, say FP rate. A second method was used to derive a threshold that can give us a fairly good target FN rate. Using the 231 blocks of contaminated data, we can compute their dissimilarities. Then we select a threshold at 30 percentile of these 231 values (see Table 2). This method allows us to compare our false rate with other methods.

Table 2: False rate of our algorithm

Order	Threshold	FP (%)	FN (%)
0th	1.000	37.534	35.498
1st	0.567	16.922	29.004
2nd	0.653	22.835	29.004

A second way to compare our method with the methods in [10] was to use masquerader sequences. There are 40 masquerader sequences in the test data. Each masquerader sequence contained either one or several consecutive contaminated blocks. We assumed that a masquerader sequence was detected once an alarm was triggered for any contaminated block of this masquerader sequence. For example, our algorithm detected 32 out of 40 sequences while the Uniqueness method discovered only 21. Table 3 shows the comparison between HFC Method and the other six methods mentioned before. By using Order 1 trendline, our algorithm provides compatible results with the Bayes one-step Markov method; and Order 2 trendline also results in better results than the Uniqueness method and the other four methods.

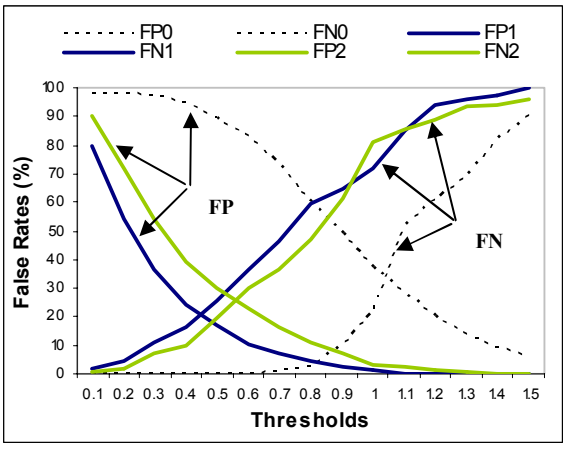

Figure 2. FP and FN rates under different thresholds

Table 3. Masquerader sequence comparison

Methods	*FP*	*FN*	*#of Seq*
High Frequency Comm.	13.9	30.3	32
Uniqueness	1.4	60.6	21
Bayes one-step Markov	6.7	30.7	32
Hybrid multistep Markov	3.2	50.7	27
Compression	5.0	65.8	22
Sequence-Match	3.7	63.2	-
IPAM	2.7	58.9	-

4. Variations and Improvements

By using the first and second order trendline, our experimental results have indicated that our algorithm was better than most of the methods but not as good as the best of the six methods in [10]. In order to improve the performance and make it more competitive with other methods, we shall explore several variations of the original algorithm.

4.1. Voting Method

We shall investigate whether the majority of the methods work better than an individual method. We have tried three variations of voting methods: (i) voting between 2: if any two methods returned normal, then we would treat the user as the true user, otherwise, the user would be treated as a masquerader; (ii) voting among 3: if all three methods returned normal, then treated the user as the true user, otherwise, if any one of the returns masqueraders, the user would be treated as a masquerader; (iii) voting among 3: if the majority of three methods returned true user, then treated the user as the true user, otherwise, the user would be treated as a masquerader. To our disappointment, the improvement is limited. In addition, most of the results made the FP rate even worse.

4.2 Tail Zeros Method

The original algorithm ignored all the zero frequencies in the profile. If there are only 10 commands in the profile with nonzero frequencies, the other 30 commands are left empty and filled with a wildcard command "*". In case the signature has more commands than the profile, we can use the commands from the signature to fill in as the profile commands even though their frequencies remain zero. This gives us longer profiles to compare with. However, using the

new TailZeros modification to compute the dissimilarities for 0th, 1st and 2nd order algorithms, The FP rate and FN rate of all three orders improves only slightly. In the following discussion, we include this modification into the algorithms.

4.3. Secondary Profile

We want to investigate the existing relationships between consecutive commands. We cannot afford to study all combination of commands (40x40, for example). So we propose the following method which looks ahead one command from the profile.

First, we build the profile just like what was described in Section 2. After that we compute a secondary profile for the top 40 commands. The *i*-th secondary command which is selected based on the i-th command of the profile.

`PCmd2[i]` = {c | c is the most common command following `PCmd[i]` in the training data set}.

So, for each profile, we have a secondary profile with up to 40 commands. In most cases, the secondary profile looks very different from the first one. Note that the secondary profile depends on the first one. The secondary command profile can be computed in reasonable amount of time proportion to the size of the data set if a small hash table is used.

Once the secondary profile is computed, we can compute the differences between the profile and a signature, each with two arrays. By combining the TailZero and the Secondary Profile methods together, we can demonstrate that the result is better than our original false rate for all three orders, especially for the FP rate. (See Table 4)

Table 4. Original Algorithm vs Secondary Profile Algorithm

	Method	Thre.	FP (%)	FN (%)
0th	Original	1.000	37.534	34.199
	Sec. Prof.	0.933	**15.454**	**29.004**
1st	Original	0.567	16.188	29.004
	Sec. Prof.	0.884	**15.370**	29.004
2nd	Original	0.653	22.835	29.004
	Sec. Prof.	0.884	**15.349**	29.004

The first problem we have faced is to find a way to combine these two algorithms. Although second profile would be used as our testing method, the original algorithm would still be considered. It is because by using two methods simultaneously could reduce the false reading feedback. Consequently, we used the following method to determine the combined results. The combined dissimilarity should be $(d_i*w_1)+(d_i*w_2)$, where d_i is the dissimilarity of order *i* and $w_1+w_2=1$.

Because our previous experiments were based on the fixed FN rate 29.004%, the weights can only depend on each algorithm's FP rate. Whereas the higher the FP rate, the less importance it should be and the less importance it is, the lower the weight it should be. Hence, we created a formula to compute the weights.

w_1: $(100 - FP_1) / [(100 - FP_1)+(100 - FP_2)]$

w_2: $(100 - FP_2) / [(100 - FP_1)+(100 - FP_2)]$

where FP_1 and FP_2 are the FP rate for the original algorithm and the Secondary profile algorithm, respectively (See Table 4).

Table 5 provided the results of combined methods with the calculated weight. For example, in the first order, w_1 is 49.757% and w_2 is 50.243%. By comparing the FP rate between Table 4 and Table 5, we found the rate of both first order and second order dropped. First order FP rate dropped quite a lot from 15.370% to 10.715% and second order dropped only slightly. However, the rate for 0-th order increased from 15.454% to 22.898%. This algorithm helped to reduce FP rate most of the time but not always.

Table 5. Combined results of the two profiles

Order	Thre	FP (%)	FN (%)	W1 (%)	W2 (%)
0th	0.983	22.9	29.0	42.5	57.5
1st	0.743	**10.7**	29.0	49.8	50.2
2nd	0.791	**15.3**	29.0	47.7	52.3

We further investigate the performance with different combination of weights (from 10% to 90%) and the results of different combination of weights are presented in Table 6. From the experiment, we discovered that the best performance we can compute is FP rate 9.205% and FN rate 29.004% with $weight_1$ 20% and $weight_2$ 80%. The result in Tables 4 and 6 both suggested that the secondary profile works better than the primary one which is unexpected.

Table 6. First order trendline with various weight combinations

Thre.	FP(%)	FN(%)	W1	W2
0.835	**9.205**	**29.004**	**20**	**80**
0.772	9.939	29.004	40	60
0.704	12.141	29.004	60	40
0.637	13.923	29.004	80	20

Among all the improvements, using the combination of method TailZero and Secondary Profile gave us the best performance (FP rate 9.20% and FN rate 29.00%) which is better than most of other methods and competitive with the best method: Bayes one-step Markov in [9] (FP rate 6.7% and FN rate 30.7%).

4.4 Periodical Updating

Periodical Updating: A typical way to improve the false rates is to periodically change the profile. When the signature blocks were compared with the threshold and did not raise the alarm, then they were added into the profile data. After every 20 blocks, the profile and the dissimilarity were recomputed. The results were mixed. When the threshold decreases, FP decreases and FN increases. To ensure that we do not contaminate the profile with masquerader commands, we use a more conservative 2-threshold method. The higher threshold is used to discriminate a masquerader and a lower second threshold is used to admit commands into the profile. Table 7 shows that the improvement was mixed for 0-th and 2nd order trendlines, but for the first order, the updating method reduced both of the false rates.

Table 7. Updating results

Methods		Thresholds	FP (%)	FN (%)
0th	Original	1.01	36.5	37.7
	Update	1.01/0.5	36.7	37.2
1st	Original	0.55	13.9	30.3
	Update	0.55/0.35	**13.4**	**29.0**
2nd	Original	0.6	22.9	30.3
	Update	0.6/0.4	24.7	27.7

5. Conclusion

In this paper, we proposed the High Frequency Command method to characterize a user's normal behavior in order to detect the masqueraders. Our hypothesis is true that the HFC method can be used as a signature to identify a user. The test results also indicate that our method performs as well as the Uniqueness method [9] and sometimes even better.

We applied three trendline methods (Orders 0, 1 and 2) to compute the values of the dissimilarity percentages. The advantage of our approach was that only the few high frequency commands (e.g. forty commands in our study) were needed for analyzing masquerader behavior when compared with Uniqueness method using all commands (more than hundreds). In general the trendline approach works better than without the trendline (i. e., 0-th order). However, we discovered that sometimes the first order trendline worked better than the second order one. This might be attributed to the shape of a second order polynomial which went up at the right hand side. It may be worth the effort for further study to see if we can use non-polynomial lines instead.

6. Acknowledgement

James Marshall, a student at St. Mary's College of Maryland, participated in the summer REU program at the University of Houston. The REU program is funded by DoD's ASSURE program and managed by NSF. Partial support of this research under a grant from Texas Learning and Computation Center (TLC^2) is also acknowledged.

7. References

[1] Denning, D. and Neumann, P., "Requirements and model for IDES-A real-time intrusion detection system", *Comput. Sci. Lab, SRI International, Menlo Park,* CA, Tech. Rep., 1985.

[2] Forrest, S., Hofmeyr, S., Somayaji, A. and Longstaff, T., "A Sense of Self for UNIX Processes", *IEEE Symposium on Computer Security and Privacy, 1996,* pp. 120-128.

[3] Wespi, W., Dacier, D. and Debar, H., "Intrusion Detection Using Variable-Length Audit Trail Patterns", *(RAID)*, 2000, pp. 110-129.

[4] Eskin, E., Lee, W. and Stolfo, S., "Modeling System Calls for Intrusion Detection with Dynamic Window Sizes", in: *Proceedings of DARPA Information Survivability Convergence and Exposition II,* 2001, pp. 165-175

[5] Debar, H., Becker, M. and Siboni, D., "A Neural Network Component for an Intrusion Detection System," *IEEE Computer Society Symposium on Research in Security and Privacy***,** 1992, pp. 240-250.

[6] M. Oka, Y. Oyama, and K. Kato, "Anomaly detection using layered networks based on eigen cooccurrence matrix", *Proceedings of 7th International Symposium on Recent Advances in Intrusion Detection, (RAID),* Sophia Antipolis, France, LNCS 3224, Springer, September 2004, pp. 223-237.

[7] Wang, W., Guan, X. and Zhang, X., "Profiling Program and User Behaviors for Anomaly Intrusion Detection Based on Non-negative Matrix Factorization," *Proceedings of 43rd IEEE Conference on Decision and Control (CDC'04),* Atlantis, Paradise Island, Bahamas, 2004, pp. 99-104,

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

[9] Schonlau, M., Theus, M. , "Detecting Masquerades in Intrusion Detection Based on Unpopular Commands," *Information Processing Letters 76*, 2000, pp. 33-38.

[10] M. Schonlau, W. DuMouchel, Wen-Hua Ju, A.F. Karr, M, Theus, and Y. Vardi, "Computer Intrusion: Detecting Masquerades", *Statistical Science*, 2001;