User Identification Via Process Profiling

[Extended Abstract] *

Steve McKinney
Cyber Defense Laboratory
Department of Computer Science
N.C. State University, Raleigh, NC, USA
mjs at terabox.org

Douglas S. Reeves
Cyber Defense Laboratory
Department of Computer Science
N.C. State University, Raleigh, NC, USA
reeves at eos.ncsu.edu

ABSTRACT

Insiders are authorized persons who possess special privileges of access; these privileges in some cases may be abused. One way in which an insider attack may occur is when user X makes use of user Y's unattended (but logged in) computer, and masquerades as user Y.

This paper presents a method of masquerade detection. A light-weight monitor collected information about computer usage by employees of a small organization for a period of three weeks. A profile of each user was developed using a Naïve Bayes classifier that analyzed handle counts of processes as the input. Under conditions specified in the paper, users were correctly identified using this technique approximately 97% of the time, with a misidentification rate of .4%.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Unauthorized Access; H.2.8 [**Database Applications**]: Data mining

General Terms

Security

Keywords

Data mining, Insider threat, Naïve Bayes

1. INTRODUCTION

Insiders (authorized members or participants in an organization) pose the greatest risk to an organization's information infrastructure. They have knowledge of and are given access to protected assets, which gives them the opportunity to engage in misconduct with serious consequences. Their motivations for doing so may include revenge, greed, arrogance, or divided loyalties. Their actions can include stealing or corrupting data, committing fraud, interference with normal organizational operations, or exposure of the organization to unacceptable risk.

Most system defenses, such as intrusion detection systems (IDS) and firewalls, are intended to detect and/or prevent unauthorized access by outsiders. They do not, by and large, protect against misbehavior by insiders. The primary protections against insider attacks are access control and audit logs, but it is often relatively easy for insiders to manipulate or bypass these mechanisms.

As an example, a trusted insider might *masquerade* as another. This may occur because of insufficient protection of passwords and other authentication tokens, or failure to prevent unauthorized computer access (e.g., leaving a computer unattended while still logged in). In such cases, access control protections will make little difference.

The goal of this work is to create an insider identification system which uses supervised learning techniques to warn of changes in computer usage, and identify the masquerader responsible for the anomaly. To accomplish this goal, we monitored (with permission) the computer systems of a small organization for a three week period. Data was collected about processes running on the computers. We hypothesized that such measurements would be successful in identifying users, including masqueraders. This data was used as input to a naïve Bayes classifier to create models of user behavior. The models were then applied to data collected after construction of the models. The quality of the results (successful identifications, and misidentifications) were tabulated and compared to alternative methods. During this process, no attacks or attempts at confounding occurred; only normal employee activities were measured.

This approach has several advantages. The process data was collected from computers being used to run many applications simultaneously in a modern windows-based environment (Microsoft Windows XP). The method not only detects anomalies, but attempts to identify the user responsible. The evaluation results are also encouraging. The rest of this paper describes the method, results, and a comparison to related work.

2. METHOD

This work uses data mining algorithms known as supervised learners (classifiers) to detect anomalies. Supervised learners use a set of labeled training data to create models from which future predictions are based [14]. We primarily

*A full version of this paper is available as *User Identification Via Process Profiling* at `www.lib.ncsu.edu/theses/available/etd-05092008-154325/`

CSIIRW April 13-15, Oak Ridge, Tennessee, USA

used Naïve Bayes as the learning method, but also compared it with Updateable Naïve Bayes.

One of the key issues in model construction is determining which data to use. There are many candidates: file accesses, network traffic, login events, privileged operations, etc. We focused on process resource usage due to its: ease of collection, modest performance impact, and compatibility with user privacy concerns. The only publicly available dataset, the Schonlau dataset [9], only contains command-line data, and was therefore not suitable for our purposes.

We developed our own data collector for the Microsoft Windows operating system. A logon monitor used the Microsoft System Event Notification Service (SENS) to record when a user logged on, logged off, locked the screen, unlocked the screen, started the screensaver, and stopped the screensaver. A separate process monitor collected data on all running processes. This service used Microsoft's .NET framework and collected data approximately every second. The monitor recorded the names of processes, process' handle count, working set size, total user processor time, total privileged processor time, and then timestamped each record. Data was collected on a total of 57 distinct applications, from vendors such as Adobe, Microsoft, Mozilla, Apple, etc.

The Naïve Bayes learning algorithm was used for constructing models of user behavior. The reasons for preferring this algorithm included:

- fast execution
- low storage requirements
- modest demands with respect to amount of training data required
- ability to adapt dynamically as user behavior changes

The Naïve Bayes algorithm differs from Bayesian analysis in that it assumes all attributes being considered are conditionally independent. Because of this assumption, computations are considerably simplified (learning time, for instance, has complexity linear to the size of the training dataset). This assumption is clearly questionable for the dataset described above, but it has been shown that in practice Naïve Bayes performs well even on datasets containing dependencies [15].

The Naïve Bayes probability can be obtained by the following [7]:

$$p(C = c_k|A_1, ..., A_n) = \frac{p(C = c_k) \prod_i p(A_i|C = c_k)}{\sum_j p(C = c_j) \prod_i p(A_i|C = c_j)}$$

This states that the probability of class c_k occurring given the set of attributes A_1 through A_n, is equivalent to the prior probability of class c_k occurring times the product of the probabilities of each of the attributes occurring given class c_k, divided by the probability of the attributes occurring. When used as a classifier, the denominator on the right hand side of the equation can be omitted as it is not dependent on the class, c_k, and can be considered a constant. For classification, Naïve Bayes computes the probability of each class occurring and selects the class with the largest probability:

$$\mathrm{classify(A_1, ..., A_n) = argmax_{c_k} p(C = c_k) \prod_i p(A_i|C = c_k)}$$

In order to ensure that the prior probability of the classes did not bias the classifier towards users with more records, we used the same number of training examples for each user.

The Updateable Naïve Bayes algorithm allows the model to change after the training phase. During the testing phase this algorithm updates the model for a particular class when a test instance is identified as belonging to that class. While this offers the benefit of evolving the model as user behavior changes, it risks the possibility of a sophisticated attacker training the model to confound it.

We used the WEKA machine-learning software [14] to analyze the data. It provides classifiers for both of the analysis algorithms we chose and is widely used in the knowledge discovery community. Since every process was recorded along with four attributes (working set, handle count, privileged processor usage, and user processor usage), dimensionality reduction was critical in forming a subset of data that was computationally feasible to analyze. We were not interested in system processes for user identification purposes, so they were removed from consideration. In addition, only records collected when the user was logged on and actively using the system (as indicated by screen locking and screen saver activity) were used for identification purposes.

3. EXPERIMENTAL FINDINGS

A group of employees from an architectural firm in North Carolina agreed to have their workstations monitored for purposes of this research. The nine users whose machines were monitoried consisted of two engineers (labelled in the following E1 and E2), five documentation and plan architects (labelled D1-D5), and two elevation architects (labelled L1 and L2). All used PCs running Windows XP Professional, and all systems had the same hardware configuration. Monitoring ran any time the machines were turned on. Monitoring was unobtrusive and required no action on the part of the users. Three weeks of execution data was collected for each user. The user with the least amount of data had 300,000 records, which corresponded to a total active monitoring time of about 33 hours per week. We used the first 300,000 records of each user for the experiment. Half of the records, per user, (i.e., 150,000) were used for training the Naïve Bayes models, while the second half (the other 150,000) were used for detection / identification.

For purposes of evaluating the algorithm, the true positive rate (TPR) and false positive rate (FPR) are defined in the following way:

$$\mathrm{TPR} = \frac{\text{Number of correct classifications for the user}}{\text{Total number of test records for the user}}$$

$$\mathrm{FPR} = \frac{\text{\# of other users' records misclassified as the user}}{\text{Total \# of test records for other users}}$$

In the results described below, *accuracy* is defined as the true positive rate averaged over all users.

Initial experiments showed that handle count gave the most accurate results. The handle count represents the number of system objects or resources that a process currently has open. It is more a function of the application and how it is being used, than of the system configuration or load. In the data collected, the largest measured handle count was 6,784 (for Internet Explorer); for most processes, the handle count was less than 400. Only results using the handle count metric are indicated below.

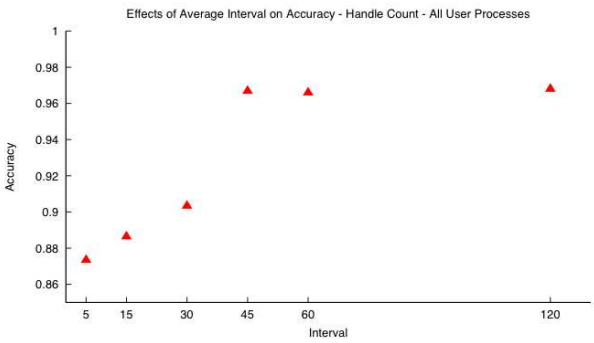


Figure 1: Window Size versus Accuracy

Due to space limitations, we only show results from one method of processing the recorded data. In this approach, the handle counts of each of the users' processes were averaged over time windows of varying sizes, before being classified. The effect on accuracy of these varying sizes is shown in Figure 1. We believe this apporach works well because it represents process usage over time. Shorter windows yield faster classifications, but (as the figure shows) at the expense of a sacrifice in accuracy. Extending the window size beyond a certain point does not improve accuracy, and reduces the speed with which identification can occur.

Table 1: 45s Window/Avg Handle Count

	E1	E2	D1	D2	D3	D4	D5	L1	L2	AVG
TPR	0.929	0.998	0.939	0.94	0.992	0.965	0.99	0.997	0.95	0.967
FPR	0.003	0	0	0.003	0.008	0.009	0.001	0.013	0.001	0.004

The TPR and FPR per user for a window size of 45 seconds are shown in Table 1. The average true positive rate is close to 97%, while the average false positive rate is around .4%.

More detailed results are shown in Table 2. In this "Confusion Matrix", the row label represents the actual user who generated a record, while the column label represents the the classifier's prediction. The Naïve Bayes algorithm, in some cases, mis-classified E1 as D4, L2 as D3, and both D1 and D2 as L1. No other major misclassification occurred. In the worst case, E1 was using only Internet Explorer and was misclassified as D4. The two cases that led to L1's high false positive rate of 1.3% involved two separate, but small, sets of processes in use by D1 and D2. In general, misclassifications continued over relatively long intervals (30 minutes or more) during which the same set of processes were in use. The number of distinct intervals of misclassification was therefore much smaller than the numbers in this table.

Table 2: Confusion Matrix: 45s Avg Handle Count

	E1*	E2*	D1*	D2*	D3*	D4*	D5*	L1*	L2*
E1	3096	0	0	0	4	205	0	28	0
E2	0	3328	0	0	3	0	0	2	0
D1	5	0	3129	42	16	3	0	138	0
D2	0	0	0	3133	18	0	0	182	0
D3	0	0	0	25	3306	0	0	1	1
D4	83	0	0	6	21	3218	0	0	5
D5	0	0	0	0	21	0	3301	0	11
L1	0	4	0	0	3	3	0	3323	0
L2	3	0	0	0	126	19	17	2	3166

Other methods of processing data before classification considered the ratio of handle counts among different processes, both time averaged and non-averaged. Experimental results (not shown) indicated this gave worse results than consideration of the handle counts alone.

A further test investigated the predictive power that can be achieved when only the same set of processes are considered. This could reflect the actions of an insider who was careful to run only the same applications as the user they were attempting to mimic. These tests were run only for pairs of users in the same job category, since their behaviors were likely to be the most similar. Handle counts averaged over 45-second windows were used for classification; there were 3,333 records in the test data set for each user. The confusion matrix for this experiment is shown in Table 3; for clarity, only the false positives are shown. Clearly, the FPR is much higher under these conditions (with an average of 3.8%), which we consider to be a stringent test of the method.

Table 3: Confusion Matrix: Common Test

	E1*	E2*	D1*	D2*	D3*	D4*	D5*	L1*	L2*
E1		142							
E2	3								
D1				73	246	26	796		
D2			1		44	0	0		
D3			0	1066		127	0		
D4			15	21	57		0		
D5			13	144	88	16			
L1									14
L2								232	

The same tests were conducted using the Updateable Naïve Bayes algorithm. For the standard case (considering all processes), it performed no better or slightly worse than the simple Naïve Bayes algorithm. For the restricted case in which only common processes between similar employees were considered, the Updateable Naïve Bayes algorithm improved the FPR to 2.1%.

The performance impact of our monitoring utilities was found to be less than 1% of both CPU time and system memory. No user indicated they felt impacted by the monitoring process. Log sizes grew at a rate of approximately 3.7MB per week of monitoring (compressed). Training the Naïve Bayes classifier took 182 seconds of CPU time for all nine users; testing required 601 seconds of CPU time. Testing and training times for the Updateable Naïve Bayes classifier were similar. We also investigated the performance of the WEKA software, and found that on a standard PC it could classify more than 140 users, in real time, for each of whom one record per second was being collected.

4. RELATED WORK

Intrusion detection systems were pioneered by Anderson [1]. Interestingly, the focus in this first report was on detection of malicious insider activities. Since that time, intrusion detection has become almost synonymous with detection of external attacks.

The single greatest challenge to the field is the lack of verifiable data. Due to concerns of privacy and security, few organizations will release data that could be used for experimental validation. We found in our investigation that less than half of the papers in this field validated their ideas ex-

perimentally. Of those that have processed data, it usually has focused on Linux systems and/or user input (command line input, user keystrokes, mouse movements, etc.), not process data on Windows systems, as described above.

Liu et al. [3] investigated the use of system calls to identify insider attacks, but found large numbers of false alarms. Shavlik et al. [13, 12] created user profiles from Windows 2000 workstations and were able to produce less than one false alarm per day per user. System performance and utilization data, event logs, typing rates, running programs, API invocations, and open file handles were all used. A weighted majority machine learning algorithm was used (and found in their experiments to produce better results than Naïve Bayes). Detection rates ranged from 71% to 97.4%, with false positive rates of around .3%. Note this work conducted masquerade detection, not user identification.

User profiling has been done using command line data [8, 2] and user process data. Maxion has published several papers critiquing methodology in this field [4, 5, 6], and investigating the use of the Updateable Naïve Bayes algorithm for this purpose. Some other works have attempted to use the same dataset as Maxion. The best results seen so far have come from Sharma and Paliwal in 2007 who were able to achieve 92.2% true positives with 14.5% false positives [11], and from Seo and Cha who obtained 97.4% true positives with 23.77% false positives[10].

Other proposals for detecting insider attacks were not evaluated experimentally, or have goals or metrics that are orthogonal to ours (e.g., document analysis).

5. CONCLUSIONS

We have shown that is possible to accurately identify users by the creation and use of process profiles. Naïve Bayes and an updateable variant of it were used to achieve a true positive rate of 96.7% with a false positive rate of 0.4% from data collected over a three week period. Out of several tests that were run on the data, we found the process handle count, averaged over 45 seconds intervals, to be the most effective in distinguishing users from one another.

To our knowledge, this work is the first to consider profiling user processes in an environment where it is common for many user processes to be running simultaneously. Though we cannot directly compare to related work in other environments, our work has yielded higher true positives and lower false positives than any of the previous work. We also considered a simulated masquerade attack where the processes used were limited to those common among the users. This work goes beyond previous work in masquerade detection as it not only detects masqueraders, but also *identifies* them.

More extensive experiments for longer periods of time and more users are needed to validate this initial work. Performance under attack by an intelligent, informed adversary, and protection of the monitor and log data, perhaps through the use of virtualization, also remain to be investigated.

6. ACKNOWLEDGMENTS

This research was performed under an appointment to the Department of Homeland Security (DHS) Scholarship and Fellowship Program under DOE contract number DE-AC05-06OR23100. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORAU/ORISE.

The authors would like to thank Oak Ridge National Laboratory's Cyber Security and Information Intelligence Research Group who gave advice and direction for this work. In particular, Dr. Erik Ferragut, Dude Neergaard, and Dr. Frederick Sheldon.

7. REFERENCES

[1] J. Anderson. Computer security threat monitoring and surveillance, 1980.

[2] B. Davison and H. Hirsh. Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time-Series Problems*, pages 5–12, Madison, WI, July 1998. AAAI Press. Proceedings of AAAI-98/ICML-98 Workshop, published as Technical Report WS-98-07.

[3] A. Liu, C. Martin, T. Hetherington, and S. Matzner. A comparison of system call feature representations for insider threat detection. In *Proceedings from the Sixth Annual IEEE SMC*, pages 340–347, 2005.

[4] R. Maxion and T. Townsend. Masquerade detection using truncated command lines. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 219–228, Washington, DC, USA, 2002. IEEE Computer Society.

[5] R. A. Maxion. Masquerade detection using enriched command lines. In *DSN*, pages 5–14. IEEE Computer Society, 2003.

[6] R. A. Maxion and T. N. Townsend. Masquerade detection augmented with error analysis. *IEEE Transactions on Reliability*, 53(1):124–147, 2004.

[7] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[8] J. Ryan, M. Lin, and R. Miikkulainen. Intrusion detection with neural networks. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 943–949, Cambridge, MA, USA, 1998. MIT Press.

[9] M. Schonlau, W. DuMouchel, W. Ju, A. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades, 2001. Statistical Science (submitted).

[10] J. Seo and S. Cha. Masquerade detection based on SVM and sequence-based user commands profile. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 398–400, New York, NY, USA, 2007. ACM.

[11] A. Sharma and K. Paliwal. Detecting masquerades using a combination of Naïve Bayes and weighted RBF approach. *Journal in Computer Virology*, 3(3):237–245, 2007.

[12] J. Shavlik and M. Shavlik. Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In *KDD*, pages 276–285, 2004.

[13] J. Shavlik, M. Shavlik, and M. Fahland. Evaluating software sensors for actively profiling Windows 2000 computer users. In *Fourth International Symposium on Recent Advances in Intrusion Detection*, 2001.

[14] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition, 2005.

[15] H. Zhang. The optimality of Naive Bayes. In *FLAIRS Conference*, 2004.

User Identification Via Process Profiling

Steve McKinney
Dr. Douglas Reeves

Computer Science
NC STATE UNIVERSITY

Agenda

- Motivation & Related Work
- Contributions
- Approach
- Experiment & Results
- False Positives
- Future Work & Conclusions

NC STATE UNIVERSITY

Motivation

- Insider Threat
  - 2007 E-Crime Watch Survey
    - Responsible for most costly/damaging E-Crimes (34%)
    - Leading cause of: IP Theft, Exposure of sensitive information
  - Impacts revenue ($7B), customer trust
  - Compromised accounts (39%)
  - Shared accounts (31%)
- Masquerade Detection
- User Identification

Related Work

- Process-based profiling:
  - Ryan `97
  - Maxion `02,`03, `04
- User input-based profiling:
  - Peacock `04 survey
  - Garg `06, Bhukya `07
- Hybrid
  - Shavlik `01,`04

Contributions

- A system:
  - For multi-tasking environments
  - For Microsoft Windows
  - Based on actual data
- Improved results
- User identification

Approach

- Collect data about each running process
  - Remove non-user processes
- Use *supervised* learners for classification
- Analyze the results so we can:
  - Understand why false positives occur
  - Improve the tests

Assumptions

- A clean data set
- Co-worker data is representative of masquerader data
- Assume it is possible to securely log the data
- Not an exhaustive solution

NC STATE UNIVERSITY

Experiment

- Developed two C# Windows services
  - Process monitor
    - Handle count
    - Working set
    - Privileged processor usage
    - User processor usage
  - User session monitor
- Installed services at local architecture firm
  - 9 Windows XP users (originally 13)
  - 3 groups of users (originally 4)

NC STATE UNIVERSITY

Experiment Issues

- Insider data is hard to obtain
  - Unable to collect original data set
  - Employee / Computer turnover
  - Travel / Leave
  - Communication issues
  - Privacy concerns
  - “How do we know your monitors aren’t malicious?”

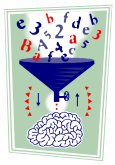
NC STATE UNIVERSITY

Data Preparation

- 15 GB of data collected over 3 weeks
- Logs –> MySQL
- Removed
  - System Processes
  - “Logged out” records
- Transforms
  - Unmodified
  - Discretized
  - Averages
  - Relative
  - Relative Averages
  - Common

NC STATE UNIVERSITY

Data Mining



- WEKA
  - University of Waikato, New Zealand
  - Open source
- Naïve Bayes

$$\text{classify}(A_1, \ldots, A_n) = \text{argmax}_{c_k}\, p(C = c_k) \prod_i p(A_i | C = c_k)$$

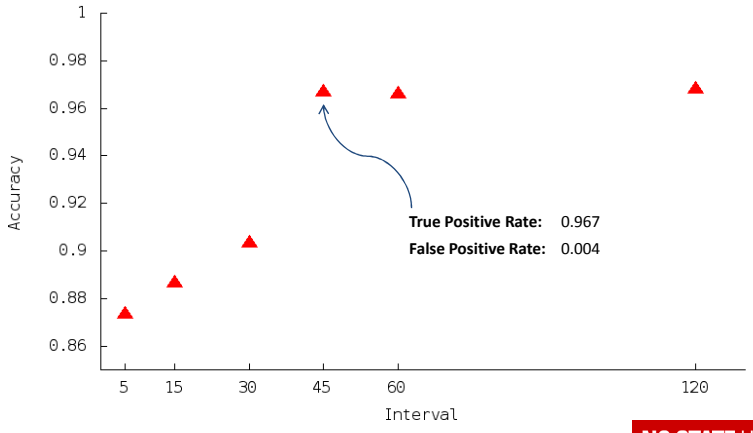
  - Assumes all attributes are independent
  - Extremely fast
  - Updateable variant

Testing

- Previously mentioned transforms
- All versus All
- Uniform training/testing instances for all users
- Varied amount of training data

Accuracy vs. Averaging Interval

- Handle count
- Training - 1.5 weeks; Testing – 1.5 weeks



NC STATE UNIVERSITY

Table 4.3: TPR / FPR for User Process 45s Average Handle Count Test.

	E1	E2	D1	D2	D3	D4	D5	L1	L2	AVG
TPR	0.929	0.998	0.939	0.94	0.992	0.965	0.99	0.997	0.95	0.967
FPR	0.003	0	0	0.003	0.008	0.009	0.001	0.013	0.001	0.004

Table 4.4: Confusion Matrix for User Process 45s Average Handle Count Test.

	E1*	E2*	D1*	D2*	D3*	D4*	D5*	L1*	L2*
E1	3096	0	0	0	4	205	0	28	0
E2	0	3328	0	0	3	0	0	2	0
D1	5	0	3129	42	16	3	0	138	0
D2	0	0	0	3133	18	0	0	182	0
D3	0	0	0	25	3306	0	0	1	1
D4	83	0	0	6	21	3218	0	0	5
D5	0	0	0	0	21	0	3301	0	11
L1	0	4	0	0	3	3	0	3323	0
L2	3	0	0	0	126	19	17	2	3166

NC STATE UNIVERSITY

Common Test

- 45 second average handle count
- Training - 1.5 weeks; Testing - 1.5 weeks

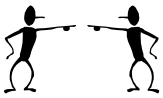
Naïve Bayes

	D1*	D2*	D3*	D4*	D5*
D1		73	246	26	796
D2	1		44	0	0
D3	0	1066		127	0
D4	15	21	57		0
D5	13	144	88	16	

Updateable

	D1*	D2*	D3*	D4*	D5*
D1		69	211	27	522
D2	6		45	9	0
D3	0	124		120	0
D4	15	19	48		0
D5	14	148	39	12	

False Positives



- Small sets of common processes
- Dramatic shifts in behavior
- Slight shifts in behavior

Limitations

- Data
  - Collection issues
  - No public repository
  - No real masquerade attacks in data

Future Work

- Gather / Analyze more data
- Implement secure logging mechanism
- Investigate other metrics
- Consider a combination of Naïve and Updateable Naïve Bayes

Conclusions

- Improved on detection rates of past work
  - 96.7% TP, 0.4% FP
- Able to find causes of false positives
- Could not only detect, but also *identify* masqueraders

NC STATE UNIVERSITY

Questions?

http://www.lib.ncsu.edu/ETD-db/ETD-search/search

http://www.lib.ncsu.edu/theses/available/etd-05092008-154325/

NC STATE UNIVERSITY