## ABSTRACT

MCKINNEY, STEVEN J. Insider Threat: User Identification Via Process Profiling. (Under the direction of Professor D. S. Reeves).

The issue of insider threat is one that organizations have dealt with for many years. Insider threat research began in the early 80's, but has yet to provide satisfactory results despite the fact that insiders pose a greater threat to organizations than external attackers. One of the key issues relating to this problem is that the amount of collectable data is enormous and it is currently impossible to analyze all of it, for each insider, in a timely manner. The purpose of this research is to analyze a portion of this collectable data, process usage, and determine if this data is useful in identifying insiders. Identification of the person controlling the workstation is useful in environments where workstations are left unattended, even for a short amount of time. To do this, we developed an insider threat detection system based on the Naïve Bayes method which examines process usage data and creates individual profiles for users. By comparing collected data to these profiles we are able to determine who is controlling the workstation with high accuracy. We are able to achieve true positive rates of 96% while maintaining fewer than 0.5% false positives.

Insider Threat: User Identification Via Process Profiling

by
Steven J. McKinney

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Computer Networking

Raleigh, North Carolina

2008

APPROVED BY:

| | |
|---|---|
| Dr. P. Ning | Dr. J. Doyle |

Dr. D. S. Reeves
Chair of Advisory Committee

# DEDICATION

Soli Deo gloria

("Glory to God alone")

# BIOGRAPHY

Steve McKinney received his Bachelor of Computer Science degree from Appalachian State University in Boone, North Carolina. He is currently pursuing a Master of Science in Computer Networking with concentrations in Network Software and Information Assurance at North Carolina State University in Raleigh, North Carolina.

## ACKNOWLEDGMENTS

I would like to thank my wife, Katie, for her support and commitment in helping me succeed. Also, to my daughter, Anne, who constantly reminds me that there is more to life than school work.

I am grateful for the advice and guidance of my advisor, Dr. Douglas Reeves, and for that of my committee members, Dr. Jon Doyle and Dr. Peng Ning.

Financial support for this work was provided by the Department of Homeland Security, without which I would not be in graduate school.

A special thanks to all of the members of the Cyberspace and Information Infrastructure Research Group at Oak Ridge National Laboratory who gave advice and direction for this project. In particular, Dr. Erik Ferragut, Dude Neergaard, Dr. Frederick Sheldon.

A special thanks as well to the employees of a local, but anonymous, architecture firm for graciously offering to collect data for this project. In particular, to my brother-in-law, who made it all happen.

# TABLE OF CONTENTS

**LIST OF TABLES** ..... **vii**

**LIST OF FIGURES** ..... **viii**

**1 Introduction** ..... **1**
1.1 Motive and Goals ..... 1
1.2 Contribution ..... 2
1.3 Thesis Organization ..... 3

**2 Background and Related Work** ..... **4**
2.1 Intrusion Detection Systems ..... 4
2.2 Insider Threat Detection ..... 5
2.3 Classification of Existing Insider Threat Systems ..... 7
2.3.1 Access Control ..... 7
2.3.2 Semantic Analysis ..... 8
2.3.3 System Calls ..... 9
2.3.4 Resource Usage ..... 10
2.3.5 Hybrid Systems ..... 11
2.3.6 Other Approaches ..... 12
2.4 User Profiling and Identification ..... 15
2.4.1 Process based ..... 15
2.4.2 User Interaction based ..... 16
2.5 Supervised Learning Algorithms ..... 17

**3 Proposed Approach** ..... **18**
3.1 Overview ..... 18
3.2 Assumptions ..... 18
3.3 Technical Approach ..... 19
3.3.1 Data Specification ..... 19
3.3.2 Data Collection ..... 20
3.3.3 Algorithm Specification ..... 21
3.3.4 Analysis ..... 23

**4 Experiment and Results** ..... **25**
4.1 Experimental Setup ..... 25
4.1.1 Data Preparation and Tests ..... 26
4.2 Performance Results ..... 29
4.2.1 Impact on Monitored Machines ..... 29
4.2.2 Log Size ..... 29
4.2.3 Classifier Speed ..... 29

4.3 Correctness Results . . . . . 30
4.3.1 Effects of Different Metrics . . . . . 31
4.3.2 Effects of Different Tests . . . . . 34
4.3.3 Causes of False Positives . . . . . 41
4.4 Limitations and Possible Attacks . . . . . 42

**5 Conclusions and Future Work . . . . . 44**
5.1 Summary of Results . . . . . 44
5.2 Future Work . . . . . 45

**Bibliography . . . . . 47**

**Appendices . . . . . 54**

**Appendix A User Processes . . . . . 55**

**Appendix B Naïve Bayes Example . . . . . 57**

# LIST OF TABLES

Table 4.1 TPR / FPR for User Process Handle Count Test . . . . . 33

Table 4.2 Confusion Matrix for User Process Handle Count Test . . . . . 33

Table 4.3 TPR / FPR for User Process 45s Average Handle Count Test . . . . . 36

Table 4.4 Confusion Matrix for User Process 45s Average Handle Count Test . . . . . 36

Table 4.5 Confusion Matrix for Groups - Common Test . . . . . 38

Table 4.6 Comparison of Naïve Bayes and Updateable TPR / FPR . . . . . 39

Table 4.7 Confusion Matrix for Groups - Updateable Common Test . . . . . 40

Table B.1Naïve Bayes Example Training Set. . . . . 57

# LIST OF FIGURES

Figure 4.1 Metric versus Accuracy comparison ..... 31

Figure 4.2 Average Interval versus Accuracy comparison ..... 35

Figure 4.3 TPR / FPR versus Training Time ..... 41

# Chapter 1

# Introduction

Insiders, those within or closely related to an organization, pose the greatest risk to an organization's information infrastructure. They are purposefully given access to and knowledge of information systems, primarily computer systems and the organization's network. In the past, insiders have used their access privileges to cause harm to organizations by stealing or corrupting data, impersonating other users, commiting fraud, and modifying performance reports [53, 29].

## 1.1 Motive and Goals

Due to the authorized access that insiders are given, detecting threats posed by them is much different than for those external to the organization. Common security applications such as firewalls and Intrusion Detection Systems (IDSs) are in place to prevent external threat, but in most cases insiders are not restricted or monitored by these mechanisms [3]. The purpose behind the majority of IDSs that have been developed is to detect external threat, but comparatively little time has been spent researching insider threat detection systems. Even mechanisms such as access controls and authentication measures, which are widely deployed, can easily be circumvented when employees leave workstations unattended.

According to the 2007 E-Crime Watch survey conducted jointly with the US Secret Service and CERT, 34% of e-crime damage was due to insider threats. That is compared to a close 37% due to external threats. [38]. Given this, and that a single malicious insider can cause an organization significant financial damage (over 7 billion dollars in one case)

[3, 14, 10], it is imperative that the research community begin making progress toward the development of insider threat detection systems.

In this work, we seek to mitigate this threat by creating an insider identification system which uses supervised learning techniques to warn of changes in computer usage and identify the user responsible for the anomaly. To do this, several things had to be accomplished. We did not know of any existing repository of insider threat related data, so we had to collect our own. After collecting the data, we needed to find ways to create models which reflect user behavior, and that also have the ability to dynamically adapt to user behavior as it changes. The models must be generated with as little data as possible because the system cannot function until the models are created. We also wanted to improve upon past work, though this distinction is made difficult due to the lack of public datasets and that the type of data we analyzed has not been used before. In light of this, the obvious goal is a high true positive rate along with a low false positive rate, preferably of less than 1%.

## 1.2 Contribution

To our knowledge, no previous work in insider threat, outside of masquerade detection, has approached anomaly detection using only process-related data and no work in user profiling has used process data for identification in multi-tasking environments. The most closely related previous work [62, 61] utilizes a pre-defined set of processes, along-side many other measures, to detect anomalous insider activity. Their work was able to detect anomalous activity, but did not attempt to determine *who* was using the system.

Research in user profiling has been conducted, but generally stops with masquerade *detection* not *identification*. In the cases where process data was used, it was limited to command-line environments. While this is useful for systems which are used predominately via the command-line, it is not suitable for most organizations' needs. Multi-tasking operating systems have become ubiquitous in the workplace and must be considered when developing detection systems. We have found data from multi-tasking operating systems to be very useful in identifying insiders with true positive rates over 96% and false positive rates of less than 0.5%. While these results are not directly comparable, they exceed the results of previous work in the field.

Researchers have also attempted to use user input such as mouse movements and

keyboard typing rates for masquerade detection. This research has had varied success, and it is limited by the type of data being analyzed. Namely, it can only detect deviations in how the user physically interacts with the computer and not what the user does with those interactions. For example, a person could type the commands to delete a portion of the filesystem or launch their favorite text editor, both while maintaining their normal typing rate. This type of detection is also limited to physical locality: remote connections cannot be monitored without changes to the detection system.

Our work also goes beyond most previous insider threat research in that we analyze application usage data from Microsoft Windows systems deployed in a real organization. Our system runs on Microsoft Windows XP and Vista, thus it is directly applicable to the vast majority of organizations.

## 1.3 Thesis Organization

The first chapter of this work has served as a brief introduction to the problem of insider threat and provides the motivation for our direction in this research. Chapter 2 presents a brief overview of intrusion detection and how it differs from insider threat detection, a categorized and extensive look at previous work in the field, and a concise introduction to Naïve Bayes which is the primary learning technique used in this work. An overview of the approach used, as well as the technical approach comprises Chapter 3. In Chapter 4 we discuss our implementation of the approach, outline our experiment, and present our results. Chapter 5 draws some concluding remarks, poses open questions, and suggests direction for future work.

# Chapter 2

# Background and Related Work

This chapter provides an overview of intrusion detection as it relates to this work, addresses the difference between intrusion detection and insider threat detection, presents related work in those fields, and gives a brief introduction to the types of knowledge discovery algorithms used in this research.

## 2.1 Intrusion Detection Systems

The idea of intrusion detection systems for computer systems was primarily introduced by Anderson's Computer Security Threat Monitoring and Surveillance paper in 1980 [4]. Dorothy Denning published a seminal paper in the field in 1987 titled An Intrusion Detection Model [22], and the first published systems we are aware of are IDES and HayStack, both in 1988 [36, 63]. Interestingly, the meaning of intrusion detection has shifted since its inception by Anderson. These early papers were the result of government research efforts targeting host-based intrusions and attempted to detect malicious insider activities. In the early 90's, as computer networks became more accessible, the focus of the term shifted to network intrusion detection and even at the present, the terms intrusion detection and network intrusion detection are often used interchangeably.

There are two primary approaches to detecting intrusions, misuse detection and anomaly detection. Misuse detection, also known as signature or rule-based detection, attempts to define malicious behavior through a set of signatures generated from known malicious attacks. Signatures are often used in real-time detection systems because they are generally able to make quicker decisions. One disadvantage of this type of system is

they can only detect known attacks. Even small variations of known attacks are able to foil signature based systems.

Anomaly detection systems use heuristic-based approaches and can be trained to recognize normal or anomalous behaviors. These systems generally use statistical algorithms to detect anomalies and rely on the idea that malicious activity is a subset of anomalous activity. Granted, this is not always the case as is shown by the large numbers of false positives generated by current systems. A regular 9-5 employee who cannot sleep and comes into work at 5am one morning would be an example of such a case.

The reporting thresholds can be raised to reduce false positives, but this has the effect of increasing the amount of false negatives, or undetected intrusions. Thus, a trade-off is always in order.

As alluded to above, most systems, regardless of the detection type, produce large numbers of false positives which can be burdensome to system and network administrators.

## 2.2 Insider Threat Detection

Although intrusion detection began with the purpose of protecting computer systems from insiders, the ubiquity of networks has shifted the focus of government and organizations to defending against external attacks. While network-based attacks have spawned a large amount of research since the early 90's, insider threat detection research papers have been scarce.

There are several reasons for this trend: organizations trust their hires, insider threat detection is difficult, insiders are undeterred by external defenses, the amount of harvestable data is enormous, privacy is a large concern in the workplace, and data for testing is difficult to obtain.

Organizations, including the government, generally like to think highly of their recruitment process and their employees. This can lead to a cyber infrastructure that is highly developed in regards to keeping those outside the organization out, but severely crippled by the lack of checks and balances on internal usage. There are numerous examples of this, including the infamous CIA and Robert Hanson case in which he sold secrets to Russia for over 20 years and made over 1.4 million dollars [54]. In January of 2008, a French bank, Societe Generale, reported losses of over 7 billion dollars due to fraudulent trades by an employee [10].

As is evidenced by the low number of publications, the lack of a satisfactory solution, and the growing financial burden to companies [14], insider threat is a difficult research topic. It was first placed on the INFOSEC Research Council's Hard Problem List in 1999 and remains there as of the last version of the document in 2005. The council is composed of members from many government agencies such as NSF, CIA, NSA and DHS [19].

Many insider activities are not monitored or preventable using traditional external defenses. These defenses are primarily concerned with network data, and while insiders can and have utilized organizational networks, the focus of these tools is on prevention of incoming traffic, not internal or egress traffic. Network perimeter monitors may filter incoming attacks to critical servers, but will not detect internal attacks on those resources or exfiltration of sensitive data to which insiders already have access.

One of the key issues in attempting to detect insider threat is determining which data to analyze. Analysis can be conducted on many types of data such as document semantics, network traffic, network-based social networks, resource utilization, and physical security devices such as access card reader logs. Collecting and analyzing all available data is not computationally feasible, which leads the question: Which sets of data are most important, or have the most value, in detecting threat? Helping answer this question is a goal of this research. We do not have the ability to examine each type of data for its utility, so we focus on one subset, the process usage of users. From this data we attempt to define the types of tests that are useful for user identification.

The ability to indiscriminately monitor cyber-related attributes is a privacy concern to many employees. Employees expect that their employer will respect and trust them. They do not think they should be monitored, especially in cases where they have done nothing worthy of suspicion. Most organizations require that employees sign a form prior to employment that waives all expectations of privacy while at work.

Real data sets are extremely difficult to obtain, mainly due to privacy [27] and security concerns. Organizational IT groups are hesitant to let outsiders install monitors on their machines, and they should be. We encountered this in our research, several organizations would not allow the monitors to be installed for security reasons and of those that allowed the monitors, it took several weeks to convince them that we did not have malicious intent.

Another issue regarding real data is that researchers have no control over the data

being collected or the people it is being collected from. Vacations, employee turnover, and users switching computers all have an effect on the data and the length of time it takes to collect it. Researchers also must have contacts inside the organization that have time to help them understand the data. This work would not have been possible without help from the company as computer security and architecture have little in common. The architecture firm's domain knowledge of the data provided tremendous insight and was critical in understanding and forming the tests we used.

Of the insider threat related work mentioned below, approximately two thirds had no data, and of those that have data, it is more commonly collected from Linux systems. This is obviously a significant issue as an overwhelming majority of organizations utilize Microsoft Windows workstations. The data used by user-profiling techniques has been collected on live systems, but as mentioned previously, it focuses on "one task at a time" usage or on mouse and keyboard input.

## 2.3 Classification of Existing Insider Threat Systems

This section is divided into subsections that identify the type of insider threat research conducted. Much of the work below spans several categories, in those cases the work is placed in the most relevant category or in the "Hybrid Systems" section.

### 2.3.1 Access Control

The Display-Only File Server (DOFS) was created by Yu and Chiueh in 2004 to combat insider information theft [72]. Their system consists of central file servers which are queried when users need a file. The system verifies that access is allowed and then sends the user an image of the file instead of the file itself. This research imposes limitations on what the user is able to do with the file and incurs significant per-user overhead on the central servers.

Pramanik et al. proposed a security policy and framework [51] for insider threat similar to Digital Rights Management (DRM). The approach focused on prevention instead of detection which presented some undesirable effects such as preventing users from opening files which they are normally allowed to access. It required applications to have knowledge about the system and thus required modification to all applications for it to be effective.

Park, et al. extended the current Role-Based Access Control (RBAC) model to better fit the insider threat scenario [48, 47] with Composite Role-Based Monitoring (CRBM). In RBAC (and CRBM), user roles are used to determine access, not their identities. The CRBM model allowed finer-grain roles to be created and compared authorized use against the user's currently assigned task. CRBM provided horizontal and vertical role mappings and attempted to meet the complexity and scalability demands of large organizations. Three role structures are used to define users' access privileges: Organization, Application, and Operating System roles. These translate to three sessions when a user is logged in and each is monitored against expected and unexpected behaviors. An experiment was conducted using coarse-grained synthetic data (the number of searches conducted) which may not be realistic considering the data it is possible to collect. The results presented did not back their claims of providing "stronger and more accurate anomaly-monitoring mechanisms than those of existing approaches" and preventing insider threats.

Seo and Sycara created an "on-the-fly" content-based access control scheme based on the binary classification problem [59]. Costs were associated with access decisions and are relative to the importance of the information being requested. Linear Discriminant Analysis (LDA), Logistic regression (LR), and Support Vector Machines (SVM) were used to classify text. They found that using LR with their costing mechanism produced the lowest false-positive and false-negative rates. They did not compare their approach to other document management schemes such as Access Control Lists (ACLs), but planned to do so. They consider their approach complementary to other schemes and not a replacement for them.

### 2.3.2 Semantic Analysis

In 2004, Symonenko et al. combined social context (insider contacts/relationships), user roles, and semantics to detect insider threat [66, 70, 71]. The published work only discusses the semantic portion of the project. Natural Language Processing (NLP) is used to determine Topics of Interest (TOI) and geo-political Areas of Interest (AOI) contained in documents. This information is compared against insiders' assigned TOI and AOI. The research assumed that insiders are assigned to "relatively long-term" tasks and that they spend most of their time working on the currently assigned task. Support Vector Machines (SVMs) and ontologies are used assign terms within documents to categories in order to reduce the dimensionality of the data. Clustering was used to group documents. The se-

mantic distance between, and size of, the clusters was used to analyze threat. Humans reviewed threat indicators and determined whether an insider's assigned task area should be modified or precautionary action needed to be taken. The data set used in the paper was surprisingly small, only five documents.

Cathey et al. attempted to detect misuse by clustering information from document and user queries [15, 37]. They used four different clustering methods based on: the documents, the query results, the relevance feedback, and a fusion of the three. In the document method, they clustered an entire collection of documents (2GB) and compared the accessed clusters to those in the user's profile. The query method only clustered documents that the user accesses. The relevance method clustered terms that the user searched for or that were contained in the documents as opposed to the documents themselves. The level of misuse was defined by calculations of dissimilarities in the accessed documents and user profile. The relevance feedback method appeared most promising from their results with "misuse detection" rates of 90-100 percent. Their later paper presented more research on the relevancy feedback method, but the results were less impressive. The detection rate was around 83% and false positive rates were between 12 and 15 percent.

Aleman-Meza, et al. have begun a project based on statistical, NLP, and machine learning techniques to measure the relevance of accessed documents to insiders' tasks [1]. Their research focuses on documents in the National Security and Terrorism domain. Documents are ranked by relevance and given a classification based on semantic associations (one of: Highly Relevant, Closely Related, Ambiguous, Not Relevant, or Undeterminable). Investigators can view the relations which led to a document's ranking. Ranking of parameters and the context of the investigation can be customized via a GUI. The approach was tested with a set of 1000 documents and the results appear promising. This was a short paper and future work included further evaluation of the approach.

### 2.3.3 System Calls

Liu et al. attempted to compare the features of system calls such as n-grams of system call names from traces, frequency counts of system call names, and system call parameters such as the return code [35]. They attempted to find anomalous behavior by analyzing these features using k-nearest-neighbor outlier detection. They found that although this analysis was useful for intrusion detection, it produced large numbers of false

alarms for insider threat.

Battistoni et al. created a kernel driver for the Windows operating system which restricted calls to "dangerous" system calls [7, 6]. They created an Access Control Database (ACD) of permissible parameters for these system calls and intercepted the calls to ensure the parameters were benign. Significant overhead was introduced (274%) when intercepting the NtOpenProcess system call and opening files incurred about 40% overhead. The authors only intercepted and created access rules for system calls and left others for future work. The performance issues and the possibly infinite number of permissible parameters for system calls that must be entered into the ACD seem to make this approach seem infeasible.

Nguyen, et al. designed an experimental system to detect insider misbehavior by monitoring file system and process-related systems calls [46]. The authors chose these types of calls because they are critical and unavoidable in carrying out any activity on computer systems. They used two approaches: user-oriented and process-oriented. The user-oriented approach was broken into normal users and system users (bin, nobody, xfs, etc...). Normal, or human, users displayed a wide range of variability in directory and file accesses over a period of 16 days and were not ideal for creating behavior profiles. System users, on the other hand, displayed very predictable behavior and had relatively small lists of accessed files. Building a profile for these accounts would be useful in detecting system account compromises. The authors recommended creating system user profiles consisting of: normally accessed files, frequency of access to those files, and changes in the correlation of files. They found that the process-oriented model gave better results for modeling user behavior. Of all the processes traced, 92% had a fixed list of files they access, many of them accessed files in patterns, and 92% had a fixed list of possible child processes. The authors used this to detect buffer overflow attacks, and were successful with all overflow attacks except those that utilized processes which do not have a fixed list of child processes (xterm).

### 2.3.4 Resource Usage

Shavlik et al. used data from Windows 2000 workstations to create user profiles and were able to produce less than one false alarm per user per day [62, 61]. System performance and utilization data, event logs, and user state information such as typing rates, select running programs, and system API invocations were all monitored and collected.

Application related data was also collected such as the number of file handles open in Microsoft Excel and the amount of processor time consumed by cmd.exe. The Winnow machine learning algorithm, a type of the weighted majority algorithm, was used (and was found to produce better detection rates than Naïve Bayes networks). They also found that using probabilities relative to the general population produced better results than those that only considered a single user, for example:

$$\frac{P(value \mid user\ X)}{P(value \mid general\ public)}$$

This allowed the system to find actions that were rare for the user in light of those that were rare for other users. Data was collected on 16 users between 9am and 5pm on workdays. Anomalous behavior was conducted by running the logged features of one user through the profile of another user. Detection rates were as high as 97.4 percent when classifications were made every 640 seconds, or 10.7 minutes. At 80 second classification intervals, the true positive rate was 71.2% with less than 0.28% false positives. Note that this research conducted masquerade detection, not user identification.

### 2.3.5 Hybrid Systems

The Detection of Threat Behavior (DTB) project is ongoing research being conducted by George Mason University [17, 18, 30]. The research models user queries in a controlled environment with strict access control policies against human behavior models. Multi-Entity Bayesian Networks (MEBN) and data mining techniques are used to detect anomalous behavior and ontologies are used to map organization-specific policies into the system. Documents in the system are dynamically classified using document relevancy techniques [16]. This allows the system to associate intention with each user and use it in the comparisons to human behavior models.

Calandrino and McKinney approached the insider threat problem in 2006 using a signature based system for real-time detection and data mining to detect longer range anomalies in user behavior and previously unknown indicators of undesirable activity [13]. This research was severely hampered by lack of data and, as a result, its effectiveness could not be properly tested.

Qiao et al. created an event driven architecture based on Subject-Verb-Object (SVO) monitors to detect insider threat [23, 50, 52]. “Subject” monitors included user and process monitors which captured information such as user name, privilege levels, login

times, process ids, and process security levels. A "Verb" monitor logged access types such as read, write, and execute, and the "Object" monitor recorded file attributes like owner, size, path, and type. Each of these monitors independently analyzed threat by consulting a User Metadata repository consisting of user profile data (work hours, processes, I/O activities). Subject, verb, and object alarms are combined together and evaluated for threat based on the STRIDE/DREAD threat model (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Escalation of Privilege; Damage Potential, Reproducibility, Exploitability, Affected Users, Discoverability). They use a weighted sum algorithm to update "normal" user behavior. The only result presented in the three papers pertains to the alert fusion mechanism that groups common alerts into a single alert, which was successful.

Bradford and Hu discuss the idea of proactive forensics and present a three-tiered monitoring approach to insider threat detection [12]. The authors define "Proactive forensics" as "the design, construction, and configuring of computer systems to make them most amenable to digital forensics analyses in the future." A black-box approach is used in that they are only concerned with the inputs to and outputs from processes. They describe their Proactive Process Monitor (PPM) as a three-layered detection and forensics mechanism. The top layer consists of a (growing) list of processes known to be malicious or unauthorized. Once a violation occurs, process specific audit data is logged to a secure site for later analysis. The middle layer contains rules which monitor user role and process interaction. Genetic algorithms (GA) are used to add to the rule set as new instances of misuse are reported. The bottom layer uses statistical analysis to detect "low and slow" shifts in behavior. System calls, CPU/memory usage, and process running times are logged at this level. The authors chose this structure to reflect the amount of system resources taken by the monitoring mechanism and noted that the main goal of proactive forensics is to adaptively increase logging given statistical justification. They use sequential hypothesis testing to justify increases in logging in their 2004 paper [11]. A statistical process analysis prototype was developed, but no results were presented.

### 2.3.6 Other Approaches

In 2003, Spitzner proposed the use of honeypots and honeytokens to detect insider threat [65]. Honeypots are systems that derive value from unauthorized use. They are

non-production systems that have little value and are not used by anyone. When activity occurs on the honeypot, it is assumed to be malicious and is recorded. Honeytokens are digital data, such as a record or file, which should not be accessed in normal operations. For example, a hospital administrator may create a record for George W. Bush and watch the record to determine if it is accessed by staff. The major disadvantage to this research is that the honeypot or honeytoken must be used before it is of any value.

Anderson et al. proposed a document event based architecture for detecting insider threat [5]. There were three primary components to the system: sensors, content-based routing, and EventTrails. Sensors were primarily deployed in applications such as MS Word and Internet Explorer, but were also placed within the OS. The sensors captured events such as document opens, saves, closes, and keyboard input. The events were sent via a content-based routing system (developed by the authors) to a dynamic set of analysis and control programs. The control programs were able to interact with sensors to prevent malicious actions from being taken. The communication system used SSL to encrypt communication and per event signatures to ensure message integrity. EventTrails were created by querying a database containing the events and used in forensic analysis. Trails could be user or document-based.

Hsieh proposed the use of a combination of SELinux, Stenography, Watermarking, and XML access control mechanisms to combat insider threat. They suggested SELinux for its implementation of MAC, but admitted that the policy creation and maintenance could be overwhelming (default policy of 250,000 rules). Stenography and watermarking were mentioned as a way to tag documents with information about their content. Tags would be examined by the OS and policies regarding access to information content would be enforced by the OS. They suggested StegFS, a stenographic file system, may be beneficial as it is able to hide the existence of protected files from users who do not have access key(s) for the files. XML based access control mechanisms such as the eXtensible Access Control Markup Language (XACML) were mentioned as a way to express the access control policy and overcome the course-granularity of SELinux policy. This work is in the planning and design phase.

Magklaras and Furnell presented a human-centric taxonomy for classifying insiders and proposed the Insider Threat Prediction Tool (ITPT) [39]. Their taxonomy incorporated system role (similar to user roles), reason of misuse (intentional or accidental), and system consequences (which categorize the source of misuse evidence like network data or

hardware reconfiguration). They presented a high-level architecture for the ITPT which consisted of multiple monitors that interacted with OS API's based on monitoring criteria. Collected data was analyzed by the ITPT Analyzer which reported to the ITPT Manager. System administrators interacted with the manager to access individual profiles. The analyzer used signature and statistical processing to identify threat, but the algorithms were not mentioned. They emphasized that monitors should produce OS independent data so that cross-platform issues would not affect the rest of the system. Several types of file-monitoring were suggested, such as: looking for the presence of certain files/directories, detecting suspicious strings/patterns in files, and file integrity checks. They also mentioned that content may be compressed or encrypted which can result in a failure to detect threat indicators.

They also presented the Insider Threat Prediction Model (ITPM) [40]. The model "is a three-level hierarchy of mathematical functions." Several insider and system attributes were used in a weighted sum to derive Evaluated Potential Threat (EPT). These attributes included: the user's role, access to sensitive data, ability to alter hardware, level of knowledge, information stored on their computer, network traffic, and records produced by other intrusion detection systems. The three-level hierarchy was formed by combining and abstracting these attributes. Future work included a more precise definition of the architecture, development of algorithms, prototype development, and integration with the authors' Intrusion Monitoring System.

Schultz presented a framework which built upon past insider threat models [57]. He mentioned that there is no single indicator that can generally provide conclusive indication of insider attacks. Several "potential indicators" were given such as: deliberate markers (markers left behind to make a statement), meaningful errors (analyzing error logs to determine what insider may be trying to do), preparatory behavior (actions taken to plan an attack like the use of finger, ping, etc...), correlated usage patterns (finding similar patterns among multiple systems, such as using grep on several systems to find files that contain certain information), verbal behavior (monitoring emails and recording requests for elevated privileges), and personality traits (traits like introversion which would be recorded by managers). He also suggested the use of multiple regression equations to combine these (quantified) indicators. Future work included proving the framework, and validation testing of the model.

## 2.4 User Profiling and Identification

### 2.4.1 Process based

In 1997, Ryan, et al. used neural networks on command-line data [55]. They looked at commands used over the period of each day of data. The data was generated by 10 users for 12 days on Linux workstations. The profiles they generated were based on which commands were used and how often the commands were used. The technique does not offer real-time detection.

In 1998, Davison, et al. attempted to identify patterns in user behavior so that user interfaces could adapt to their usage [21]. Their primary goal was predicting the next command the user would issue. This research used UNIX commands and collected data from 77 users over a period of 2-6 months. They used their own algorithm based on the Markov assumption that the next process depends only on the previous one. It is interesting that even though their algorithm was the best of the ones they tested, that Bayesian Networks were very close in terms of prediction performance.

Maxion, et al. published three papers, one in 2002, in 2003, and one in 2004 [41, 42, 43]. Their work pointed out some issues with previous research, notably [56], and offered some insightful ways to improve reporting of results for this area of work. One of the key suggestions was to show the "1v49" results. The name came from a previous paper where the authors had a set of 50 users and only chose to show the results for a select few. Maxion's point was that to show the true usefulness of the approach, all of the results must be shown. They also contributed to the field by proposing an approach based on Naïve Bayes which improved upon the results of previous work. In the 2003 paper they used the commands as well as their arguments to improve upon the 2002 results. The authors admitted their approach was not sufficient to deploy as it produced 60% true positives at a 1% false positive rate (the false positive rate deemed by the authors to be acceptable). Both papers also made use of the Updateable Naïve Bayes algorithm which proved to be effective in keeping up with "concept drift" or changes in user behavior over time. The 2004 paper added to their earlier work by defining the reasons behind the false positives found in their system.

In 2004, Li, et al. used network data to identify users [33]. They used 35 "sessions" of Windows data generated by eight users and used one and two class SVMs for detection. Their results were comparable to past work analyzing the same type of data in a Unix

environment.

There have been several other papers [68, 20, 28, 31, 58, 60] published based on the Schlonau data set which was used by Maxion, but the improvements have been marginal. Some of the work uses different experiments thus making it hard to make direct comparisons. 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 [60], and from Seo and Cha who obtained 97.4% true positives with 23.77% false positives[58]. Sharma used a combination of Naïve Bayes and a weighted radial basis function and Seo used an SVM-based approach.

### 2.4.2 User Interaction based

The first research we are aware of that monitors user input to identify individual users is the work of Bleha, et al. in 1990 [9]. They used Bayes classifiers to analyze the keystroke data and obtained good results from their method. A variety of systems have been proposed that monitor keystrokes, mouse activity, or a combination of the two. These systems have been shown to be effective, but they do have limitations. Mainly, they can only monitor *how* the user provides input to the machine and not *what* they do with that input. This is not the approach of this thesis so we only include two of the most recent papers from this research area.

In 2006, Garg, et al. created a GUI based profiling system for Microsoft Windows [24]. They concentrated on mouse and keyboard activity generated during 86 sessions by 3 users. They used SVMs for analysis and the results were not sufficient for a deployable system.

Bhukya, et al. developed a GUI user profiling system for the Linux operating system in 2007 [8]. The data they collected is almost identical to that of [24] which used Microsoft Windows. One-class SVMs were used and they focused on mouse and keyboard activity. Their data came from 8 users each with 8 "sessions" of input. Their results were less than desirable.

[49] provides a survey of techniques used in identifying users based on keyboard typing rates. Results included in the survey were varied, but one paper was able to attain a false alarm rate of 1.11% with a 0% false acceptance rate [34]. Much of the work relating to profiles of typing rates focused on user authentication so the data consisted of short sequences of input as opposed to hours of keyboard input.

## 2.5 Supervised Learning Algorithms

This work uses data mining algorithms known as supervised learners to detect anomalies in the data. Supervised learning algorithms use a set of labeled training data to create models from which future predictions are based. In contrast, unsupervised learners attempt to classify data without the help of a training data set [69]. We primarily focus on Naïve Bayes as our learner, but also use Updateable Naïve Bayes. Both of these techniques are supervised learners and are discussed in more depth in section 3.3.3.

# Chapter 3

# Proposed Approach

## 3.1 Overview

The primary objective of this research is to create profiles of user-process activity and, given a set of data, determine which user generated it. This approach goes beyond that of previous work in the field of process-based masquerade detection by:

- Analyzing process data in Microsoft Windows as opposed to the Linux command line
- Detecting and *identifying* masqueraders
- Yielding high true positive rates while maintaining low false positive rates

The data included in these profiles is derived from data collected by two monitors which were developed for computers running the Microsoft Windows operating system. These monitors record user logon events and resource usage measurements for *every* running process. All of the data was collected from an architecture firm over a three week period. There were nine employees monitored, each of whom used Windows XP Professional.

## 3.2 Assumptions

We assume that all data collected is data from the user logged in to the workstation at the time of collection and that each user is carrying out their normal duties. This is a strong assumption and it is a common one in anomaly detection research. In a

real environment, employees could be monitored more closely while *training data* is being collected so that the integrity of the log data is better maintained.

It is also assumed that we can use data from a user's co-workers to represent masquerade activity on that user's computer. We realize that this is a strong assumption and masqueraders will act differently during attacks, but we have not been able to obtain data from recorded attacks. This is also a common problem in this type of research [43]. We have attempted to create system-independent tests to prevent skew in the results derived from differences in the systems on which data was collected. This is covered in more detail in section 4.1.1.

We also note that this is not meant to be an exhaustive solution to the insider threat problem, but a portion of it. As was previously noted, there are large amounts of data that can be analyzed to detect and identify insider threat and each has strengths and weaknesses. A robust system may also have to monitor document semantics, network usage, and possibly social networks to detect and attribute threat and provide sufficient evidence for criminal prosecution. Even with these monitors, there will be attacks that cannot be detected by computer systems; writing sensitive information down on paper or memorizing it are two prime examples.

The current monitors are susceptible to attack by a moderately technical user. Our primary focus in this research is to identify users and we assume that the users did not tamper with the monitors or log data. We provide ideas for hardening the monitors and data in our future work.

## 3.3 Technical Approach

### 3.3.1 Data Specification

As mentioned before, one of the key issues in detecting insider threat is determining which data to collect. The volume of collectable data makes it infeasible to collect everything, so some subset must be chosen.

We decided to base our system on process resource usage and monitor usage for all running processes. Several past efforts in masquerade detection have been based on process data, but those systems targeted command line processes which are generally executed in succession. Several user processes are often running in tandem on the average workstation

and this type of usage has not been covered by previous work to our knowledge. A complete list of the processes used in this research can be found in Appendix A.

We also had to be mindful of the performance impact of the monitors on the systems. The collected data needed to be useful for detection, but also not interfere with the user's ability to complete their work. We have analyzed the impact on the end user in terms of processing and storage in section 4.2.1.

### 3.3.2 Data Collection

To our knowledge, there is no existing repository of insider threat data so we had to collect the data. We contacted the authors of some of the related work, but they were unable to release data due to privacy concerns. The Schonlau dataset [56] is publicly available, but it only contains command-line processes.

Utilities had to be developed to collect the data as we were not aware of existing ones. We developed a logon monitor service in C♯ which 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. We also developed a process monitor service in C♯ which collected data on all running processes. This service used Microsoft's .NET framework and collected data approximately every second. In addition to the names of the processes, we also collected each process' handle count, working set, total amount of user processor time, total amount of privileged processor time, and a timestamp for each record. This data was chosen for three reasons: it was easily collectable using Microsoft's libraries, we thought it would give insight into how processes are used, and we needed to collect small amounts of data in order to increase the chances of an organization agreeing to collect data for us.

To re-iterate the privacy problems mentioned previously, we encountered organizational privacy concerns when submitting requests to install the monitors, throughout the monitoring process, and when the data was taken outside the organization. The first attempt at collecting data was so severely limited by communication and privacy issues that the data was not able to be used in time for this thesis to be published. We do plan to utilize this data in the future.

We also encountered several issues when collecting the second set of data which is used in this work. The collection process was prolonged due to such factors as employee

vacations, computers being re-assigned to different users, privacy issues, and security concerns relating to installing the monitors. The company also asked to remain anonymous and that we not release the data which was collected.

### 3.3.3 Algorithm Specification

We focused on the Naïve Bayes algorithm and a variant of it, Updateable Naïve Bayes. We originally considered other algorithms such as Support Vector Machines (SVM) and linear regression, but these algorithms were prohibitive in the amount of computation time they required. We had several criteria which the algorithm needed to meet, it should:

- Generate models of user behavior quickly and efficiently
- Create models of user behavior that consume little storage space
- Derive useful models of behavior with little training data
- Classify incoming data quickly and efficiently
- Adapt dynamically as user behavior changes over time

These criteria are based on several perceived needs of organizations. They want a system which:

- Shows a quick return on investment
- Requires little to no investment in more computing power and disk space to run the system
- Scales well with a growing work force
- Maintains itself

#### Naïve Bayes

The Naïve Bayes algorithm is a supervised learner which is based on Bayes' Theorem. The difference between Naïve Bayes and Bayes is that Naïve Bayes assumes all attributes being considered are conditionally independent. This is obviously not the case in our situation, but quite often in the past, Naïve Bayes has performed exceptionally well even on data sets with dependencies [73].

Another key reason for choosing Naïve Bayes is that it generally requires little training data to create a useful model. The purpose of this research is detection and identification, which cannot be done while training data is being collected. Furthermore, managers need to supervise employees more closely during the training phase to ensure they are using their own computer. By keeping this phase as short as possible, the managers can return to their normal duties sooner.

Naïve Bayes is also extremely fast in comparison to many classification schemes as learning time is linear in the size of the training set [69]. This is essential to insider threat systems as they must scale to meet the demands of all organizational sizes. It is also important because administrators need to be made aware of attacks as soon as possible. Analyzing data collected several hours in the past may not allow administrators the ability to stop an attack or apprehend the masquerader.

Intrusion detection research has found Naïve Bayes to be effective against the popular KDD'99 intrusion data sets [2]. Maxion [41, 42] found Naïve Bayes and its updateable variant to be effective in detecting masquerade attacks compared to other methods over the same data set. Our preliminary testing showed it to be effective with the insider data we collected, so we chose it as our primary algorithm.

The Naïve Bayes probability can be obtained by the following [45]:

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

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. In classification, Naïve Bayes computes the probability of each class occurring and selects the class with the largest probability:

$$classify(a_1, ..., a_n) = argmax_{c_k} p(C = c_k)\prod_i p(A_i|C = c_k)$$

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. An example of computing probabilities with Naïve Bayes can be found in Appendix B.

#### Updateable Naïve Bayes

Updateable Naïve Bayes is essentially the same algorithm as Naïve Bayes, but it allows the model to change, even after the "training" phase has taken place. During the testing phase this algorithm updates the model for a particular class when a test instance is identified as belonging to that class. This allows the model to be dynamic, and in our situation, allows the model to adapt to the user as they change behavior over time. This initially seems to be an incredibly nice feature, but it does introduce some amount of risk as there is the possibility of the user attempting to train the model.

This is generally more of a concern if the algorithm is being used to model "normal" behavior and detect behavior that is "malicious." In this case, a user can gradually modify their behavior and ultimately carry out "malicious" actions which appear "normal". In our case, a user may try to adapt their behavior to mimic that of another user. This could cause higher false positives and negatives between the two users' and possibly cause an administrator to ignore the alerts after an initial investigation. We discuss this attack and the reasons why we believe it is not a feasible attack in section 4.4.

### 3.3.4 Analysis

We used the WEKA machine-learning software [69] to analyze the data. It provides classifiers for both of the analysis algorithms we chose and is widely used in the knowledge discovery community. The software is open-source and is maintained by the University of Waikato in New Zealand. The open-source nature of the project has been extremely beneficial to this research as we have been able to modify the program to allow more statistics to be reported. This modification was posted to the WEKA mailing list and one of the WEKA maintainers has added it to the source code for future releases [25].

Since every process was recorded along with the four attributes (working set, handle count, privileged processor usage, and user processor usage), dimensionality reduction was critical in forming a subset of data that is computationally feasible to analyze. Fortunately, we were able to do most of the feature selection simply from a knowledge of the

data. For instance, since we are not interested in system processes, we could remove them and focus on user processes. The set of user processes used in the final results were chosen by hand. Since we did not have a solid background in architecture-related programs, we utilized on-line resources [64, 67], as well as the employees at the firm, to determine which processes constituted user processes. This is different from the previous work of Shavlik [62, 61], as they decided which processes to monitor before they began collecting data. It is likely that not all user processes used during the collection period were monitored. This should not be considered a flaw in their work as they had a different approach.

Research has been conducted in automating the feature selection process [32]. The authors of that work used data mining techniques to extract patterns from the data, then used those patterns to guide the feature selection process. The size of our data set allowed us to conduct this process manually, but we believe that automating this process may be a useful area to explore, especially with larger scale experiments.

We were also able to remove data generated when the user's computer was on but they were not logged in by use of the logon monitor. In some cases, this provided a substantial reduction in data as some users left their computer on all the time. This also allowed us to locate the data we were interested in as a masquerading attack is not effective if the target user is not logged in.

After reducing the data set, we performed several types of tests using WEKA's classifiers. These tests are described in more detail in section 4.1.1. WEKA reported the information needed to determine the accuracy of the classifiers. Once we obtained the results, we used them to derive some generalizations for all of the users as well as explain false positives among individual cases.

# Chapter 4

# Experiment and Results

## 4.1 Experimental Setup

A group of employees at an architecture firm in Raleigh, North Carolina agreed to be monitored for the experiment. There were originally twelve Windows XP users and one Windows Vista user. The firm later decided not to release data for the three secretaries due to security concerns. A Windows XP version of the monitor had been placed on the Vista machine by accident so data was not collected from that machine. The nine remaining users constituted three groups: two engineers, five documentation and plan architects (2-D drawings), and two elevation architects (3-D rendering).

The monitors were installed on their machines and ran any time that Windows was running, including times when users were logged off or had locked their screens. We assumed that the users would not tamper with the monitors or logs as they were also interested in the success of the project.

We asked the users to work as usual throughout the experiment, and all data was stored locally on their computer. Although we began collecting data two months in advance, due to the problems we encountered, we were only able to obtain three weeks of data per user. This data set is admittedly small, but in comparison to other work in this field, it is comparable. The Schonlau dataset, which is the most widely used, consists of 50 users, but it only contains 15,000 commands per user. Their data was collected over varying time periods from a few days for some users to a few months for others [43]. Our dataset has a minimum of 300,000 records per user.

All tests were conducted on a system with an Intel Core Duo processor operating

at 2.66GHz and 4GB of system memory. It should be noted that WEKA does not make use of multiple cores for its calculations.

### 4.1.1 Data Preparation and Tests

The data was collected in CSV files which had to be merged before they could be analyzed. We chose to import the data into a MySQL database so we would be able to easily extract the data in which we were interested. The database also allowed us to modify the data to run several different types of tests such as computing averages over various intervals of time.

We only used user processes in the tests so the results would not be tainted by system processes such as Windows services. Each of the user processes were identified and verified as such by hand to ensure the data wasn't biased to non-user activity as described in section 3.3.4. We found 57 user processes and all 57 are used in each of the tests unless otherwise noted.

After collecting the data, we found instances where the logon monitor was not able to function correctly due to it being installed with insufficient permissions. Fortunately, by studying the data generated by other users who had the logon monitor working properly, we were able to determine periods during which a user was logged in without the logon records. There were two processes that each system had installed, namely *acrotray.exe* and *logon.scr*, that allowed us to make these distinctions. The Adobe Acrobat system tray utility, acrotray.exe, only ran when the user was explicitly logged in. It continued to run even if the system automatically locked their screen due to inactivity. The other process, logon.scr, is the Windows logon screen that runs when the user's screen has been locked. By removing records where logon.scr was running and those where acrotray.exe was not running, we were able to obtain the records from when the user was actually logged on.

In order to better simulate a real working environment, the records from each user in the training and test data sets were merged with one another instead of having all of one user's records first, then all of another's. This mimics an environment where records are being received from multiple users each second. It is especially important when using Updateable Naïve Bayes because the model is updated as new records are classified. Placing all of a user's records together may skew the model, and thus the results.

Unless otherwise stated, all tests utilize 300,000 records per user; which was chosen

based on the minimum amount of data collected by a user when that user was *logged in*. Each record is equivalent to approximately 1.2 seconds of work completed by the employee. 300,000 records is approximately 33 hours of work per week, which seems reasonable considering this employee was an engineer and was often in meetings. We chose to use the same number of records per user to prevent the results from being skewed to the user(s) with more records.

We created several tests which we thought to be useful. Some of the tests were derived from ideas in previous research, such as averaging the measurements over time [61]. The following tests were conducted:

- Unmodified - This test looked at the raw data produced by the monitors.
- Process Set - This test considered only the processes being used for each data point and did not include any measures of how those processes were being used.
- Averages - This test looked at averages of data values over time. For example, the average working set for each process over thirty second time periods. In this test, each of the time periods were distinct and did not overlap.
- Relative - This test examined the usage of processes relative to one another. For instance, how much did a user use Internet Explorer relative to the other processes they use.
- Relative Averages - This test is a combination of the above Relative and Averages tests. The averages were computed first and then were converted to relative data.
- Common Only - This test removed all processes that were not common to all users being tested. We chose to use this test as we do not have data where users attempt to masquerade as one another. This test attempts to create this situation.

In testing the data we wanted to be sure that the tests were not skewed by system-dependent metrics. None of the users in the test shared a common computer and none of the computers had same hardware configuration. These differences in computer hardware had the potential to skew the data and the results.

For instance, using the same processes, in the same manner, on two identical computers, the monitor logs produced by each should be highly similar. But if one of those

systems has 256MB of RAM and the other has 2GB of RAM, the monitor logs would not be similar. There would be differences in the working set of the processes as the working set is dependent on how much physical memory the system contains. Thus, the results of tests based on the working set would show differences in process usage, when in reality it was the system configuration that lead to the difference, not process usage.

We believe a similar dependency exists for the two processor metrics we have collected. However, we do not think the handle count metric is system-dependent. The handle count depicts the number of system objects or resources a process has open. While this count could be limited by the hardware or operating system, we believe that this would be a rare occasion. We checked the data we had collected and the largest handle count for a user process was 6,784 for Internet Explorer. The largest handle count for most processes was less than 400. Microsoft's web site notes that the maximum handle count per process in Windows XP is 10,000, and after that limit is reached programs may stop working [44]. Regardless, this limit is enforced for all Windows XP machines thus it is uniform for all users in this study and not system-dependent. We were unable to find the limit for Windows Vista.

To overcome this limitation for the other three attributes, we created the Process Set, Relative, and Relative Averaging tests. By eliminating the measurements and only considering the set of processes being used, the Process Set test removes the system dependence of the data. Similarly, the Relative tests remove this dependence by examining how processes are used in relation to one another. Due to a lack of data, we were not able to prove that the dependencies are removed by these tests, but think that it is intuitively reasonable and leave proof to future work.

The users were assigned to "groups" by the architecture firm according to the type of work they do. We labeled the users according to these groups: the two engineering employees as E1 and E2, the five documentation and plan employees as D1-D5, and the two elevation architects as L1 and L2.

## 4.2 Performance Results

### 4.2.1 Impact on Monitored Machines

The data collected by the monitors included information about the monitoring processes themselves. We examined this data and found that the monitors required very little processing time: around 150ms of user processor time and 50ms of privileged processor time per minute of logging. The monitors consistently used about 12MB of system memory. Needless to say, we did not receive any complaints from the users being monitored at any point during the experiment.

### 4.2.2 Log Size

After examining all of the data, we found that the logs typically grew at a rate of 27KB/s. Each time the user restarted their machine, the logs were compressed which resulted in an average of 1.7KB/s. Given this, one employee, over a 40 hour period, would generate about 3.7MB of compressed data per week, and 192MB per year. This does not take into account: the possibility of removing some of the attributes collected which were not found to be useful in identifying the user, removing records generated when the user was logged off, or removal of non-user processes. These reductions would make the log sizes considerably smaller.

There were a total of 317 distinct processes and of those, 57 were classified as user processes. We found that each user had, on average, 105 different processes, 31 of which were user processes.

### 4.2.3 Classifier Speed

Training the Naïve Bayes classifier took 182 seconds for all nine users, each with 150,000 records. This translates to 7417 records per second or one record every .0081 seconds. The updateable variant took approximately the same amount of time to train a model. Training the models can be done offline and thus is not as critical as the time required for classifying new records.

The testing phase was also conducted with 150,000 records, each consisting of 57 attributes (processes), for each of nine users. On average, Naïve Bayes required 601 seconds for testing, or 2,246 records per second. Surprisingly, Updateable Naïve Bayes only

took 608 seconds, or 2,220 records per second. Before running the test, we assumed the updateable variant would require considerably more time as it had to update one user's model each time a classification was made. Given the results, we believe the time it takes to classify a record far outweighs that required to update a model. Updating the model consists of modifying the mean and standard deviation for $n$ attributes, whereas testing a record requires computing probabilities for $kn$ attributes (where $k$ is the number of users).

A more interesting measure is the number of users such a system will support. Since Naïve Bayes computes the probability that a record belongs to each user before making a classification decision, the number of records classified per second is based on the number of users. Note that it is also based on the number of attributes, but we will assume this number stays constant. With 1,350,000 records among 9 users, our system computes 20,216 such probabilities per second. By taking the square root of this number, we found that WEKA's implementation of Naïve Bayes can handle approximately 143 users and still be able to classify one record per user per second on the machine used for the computations. This is due to the fact that 143 records would be received each second and each of those records must have probabilities computed for all 143 users. We believe a much larger number of users would be supported given a highly efficient implementation of Naïve Bayes. WEKA is coded in Java, and by no means was written to optimize Naïve Bayes calculations. The system would also support more users if classifications are computed once for every 45 seconds of user data instead of every second as would be the case with the Averages test.

## 4.3 Correctness Results

There are various ways of computing true and false positive rates. In this paper, we use the following method to compute these rates for each user:

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

$$FPR = \frac{Number\ of\ other\ users'\ records\ misclassified\ as\ the\ user}{Total\ number\ of\ test\ records\ for\ other\ users}$$

For example, using the data from table 4.2, E1 is correctly classified 125,362 times. There are 150,000 test records for each user, so E1's TPR is 0.836. Other users are falsely

classified as user E1 59,564 times. There are 1,200,000 test records for the other users, so E1's FPR is 0.05.

### 4.3.1 Effects of Different Metrics

We chose to find the most useful metrics as a starting point for the tests. This allowed us to quickly identify which metrics provided the most differentiating power between the users and helped us focus the other tests around these metrics. We only show comparisons using the unmodified data, but we found similar results among tests using different transformations on the data, such as averaging. Figure 4.1 clearly shows that the handle count was the leading metric in terms of accuracy. For the purposes of this paper, we define accuracy as the average true positive rate for all users.

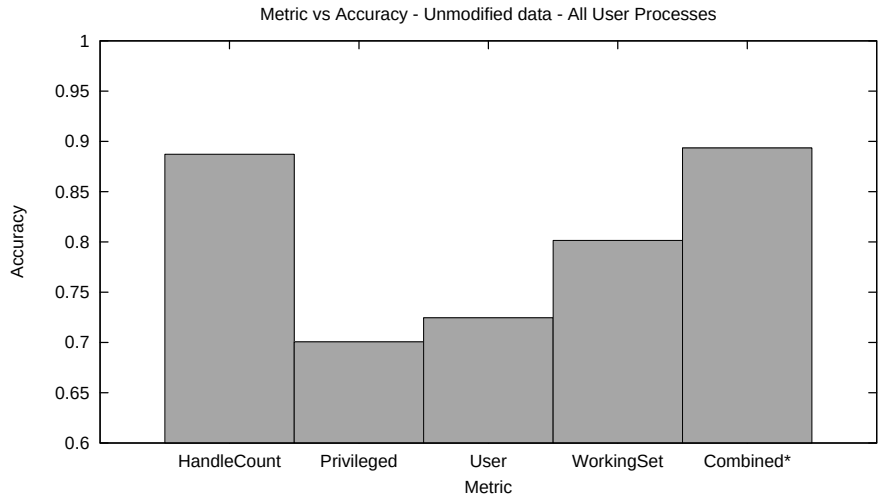


Figure 4.1: Metric versus Accuracy comparison

The privileged processor time provided the least amount of accuracy, around 70%, and we think this reflects the fact that the user has very little direct control over the privileged processor usage. While some user actions require the processor to operate in privileged mode, it is the operating system that invokes this mode, not the user. We see little improvement with the user processor usage, 72%, and believe both processor metrics suffer from what they represent. Both are a measure of the total processor time spent on an application since it was invoked. This means that for all shared processes, there is some range of processor usage that is also common among those processes. As long as programs

are running within these shared ranges, it will be hard to differentiate users with these metrics.

The working set metric raises the bar to 80% accuracy, but is still far from the 88% obtained by the handle count alone. We believe this is due to the fact that the working set is not necessarily representative of the programs the user has open. Once physical memory requirements begin to grow beyond the system's capacity, the working set of processes are paged out to the disk. Since the working set is the measure of physical memory used by a process, this metric may not represent the actual amount of resources being used for all processes. Recall that each of these metrics is per process, so if some or all of Internet Explorer's working set is paged out to disk, this metric will not be representative of the amount of resources Internet Explorer is using.

We believe the combined metric does so well because of the differentiating power given to it by the handle count. It should be noted that we had to lower the number of records in the combined test to 200,000 per user due to memory limitations in the WEKA software. We ran the same 200,000 records per user test using only the handle count and received a score of 94% accuracy, so the combination of all metrics does not bring any improvement. We also tried other combinations, but none performed as well as the handle count alone.

The handle count metric stands out from the others and we believe we know why this happens. With the exception of the 10,000 handle limit in Windows XP mentioned in section 4.1.1, this metric is unhindered by the system and directly measures the number of resources the user is requesting by the process. This metric is not reduced at the operating system's discretion as is the case with the working set. It also does not have the shared range of usage that the processor metrics have, at least not to the same extent. While there may be some common number of resources invoked by a process during startup, the handle count quickly begins to reflect the number and rate at which users request resources from the process. Thus, it tends not only to reflect the user's interactions with a process, but also maintain the result of those interactions even if they minimize the process and begin using others.

Table 4.1 displays the true positive rates (TPR) and false positive rates (FPR) for the handle count test used in figure 4.1. Overall, the true positive rate was almost 89% and the false positive rate, 1.4%. In looking at the individual user rates, they are quite acceptable with the exception of two cases, namely, D4 and L1. To obtain a better

understanding of what is happening, we also include the confusion matrix for this test in table 4.2. The rows of the matrix correspond to the actual owner of the record and the columns correspond to the classifier's prediction. The matrix shows D4 being mis-classified as user E1 over 30,000 times and user L1 being mis-classified for the same user almost as often. This is interesting because users E1, D4, and L1 are each from different "groups," so they have different job descriptions. Upon inspecting the records that were mis-classified, we found that L1 was classified as E1 when they only had Revit, or Internet Explorer and Revit, running. Similarly, when D4 only had Revit, or Revit and Outlook, open, they were classified as E1. These results suggest that when users only have one or two common programs open, it may not be sufficient for differentiating between users.

Table 4.1: TPR / FPR for User Process Handle Count Test

| | E1 | E2 | D1 | D2 | D3 | D4 | D5 | L1 | L2 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|
| TPR | 0.836 | 0.958 | 0.853 | 0.945 | 0.999 | 0.672 | 0.985 | 0.739 | 0.998 | 0.887 |
| FPR | 0.05 | 0 | 0.002 | 0.032 | 0.005 | 0.005 | 0.003 | 0.028 | 0.001 | 0.014 |

Table 4.2: Confusion Matrix for User Process Handle Count Test

| | E1* | E2* | D1* | D2* | D3* | D4* | D5* | L1* | L2* |
|---|---|---|---|---|---|---|---|---|---|
| E1 | 125362 | 244 | 0 | 9298 | 71 | 5211 | 0 | 9814 | 0 |
| E2 | 0 | 143648 | 0 | 125 | 265 | 0 | 0 | 5819 | 143 |
| D1 | 9 | 3 | 127947 | 8589 | 1384 | 339 | 333 | 11391 | 5 |
| D2 | 115 | 1 | 0 | 141779 | 1898 | 42 | 0 | 6165 | 0 |
| D3 | 0 | 0 | 0 | 6 | 149844 | 0 | 0 | 12 | 138 |
| D4 | 30004 | 0 | 0 | 14559 | 714 | 100739 | 2758 | 931 | 295 |
| D5 | 0 | 0 | 0 | 0 | 1176 | 0 | 147795 | 0 | 1029 |
| L1 | 29435 | 263 | 2989 | 5923 | 200 | 395 | 0 | 110795 | 0 |
| L2 | 1 | 0 | 0 | 0 | 185 | 0 | 53 | 0 | 149761 |

Each of the following tests use the handle count metric exclusively as it was found to offer the greatest degree of accuracy.

### 4.3.2 Effects of Different Tests

Once we settled on the handle count as a suitable metric, we began running the tests mentioned in section 4.1.1.

#### Process Set

We obtained data for the Process Set test by representing each user process with a one when the handle count was greater than zero, and a zero otherwise. This test attempts to classify a record based on the set of processes running without any information pertaining to how those processes are being used. The average true positive rate was 80.3% and the average false positive rate, 2.5%. Most users had much higher true positive rates, but three were extremely low: L1 at 46.7%, E1 at 53.1%, and D4 at 68.8%. The highest false positive rate was 5.6% and was for D4.

Use of common sets of processes led to mis-classifications for this test. The most common sets we found in the data were:

- Acrobat Reader
- Outlook
- Outlook, Revit
- Internet Explorer, Outlook
- Internet Explorer, Outlook, Excel
- Internet Explorer, Outlook, Revit, Word

#### Averages

The Averages tests performed better than any of the others and we believe this is due to the fact that records in this test represent process usage over time. The first figure, 4.2, shows the change in accuracy as we modified the interval over which the averages were computed. The five second average is similar to the unmodified result seen in section 4.3.1. From there we see slight gains in accuracy continuing through the 15 and 30 second averages. The accuracy increases dramatically between 30 and 45 second averages, then levels off. It makes sense that the accuracy increases over time as more and more data is combined to

form a record, and that at some point a ceiling on the accuracy is reached. We do not have an explanation as to why there is a sharp difference between 30 and 45 seconds. We also tested with a 40 second average and only saw a 0.3% difference in the accuracy from the 30 second test.

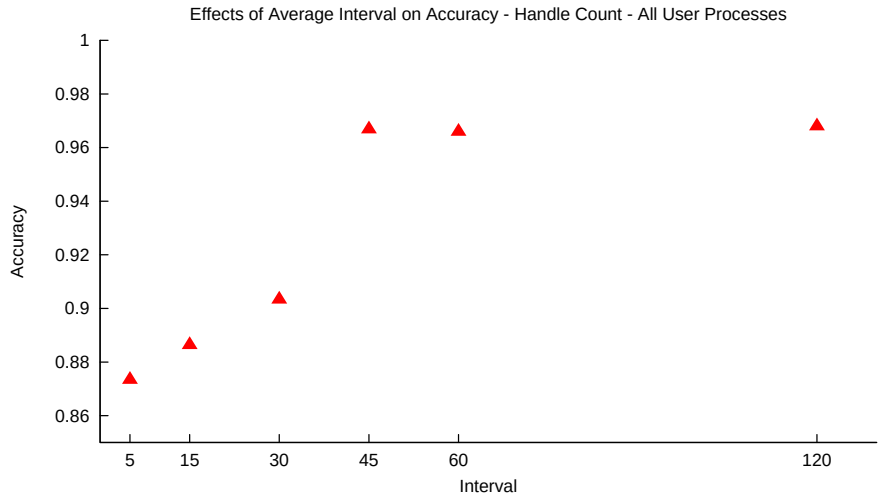


Figure 4.2: Average Interval versus Accuracy comparison

The true and false positive results from the 45 second test are shown in table 4.3. The test had an average true positive rate of 96.7% and a false positive rate of 0.4%. While we cannot directly compare these results to the previous work in user profiling and masquerade detection mentioned in section 2.4, our method seems to have a greater ability to differentiate users. We tested longer intervals, up to five minutes, but only saw marginal gains. At five minute intervals, the average true positive rate was 97% and the average false positive rate was .39%.

Since our data is based on 33 hour work weeks, there are 6.6 hours in each "day." There are 528 45-second periods within 6.6 hours, so a false alarm rate of .4% results in 2.1 false positives per day per user. If we look at 60-second averages, of which there are 396 in a 6.6 hour day, a .4% FPR results in 1.58 false positives per day per user.

It is important to note the trade-offs that take place when varying the averaging interval. When shorter intervals are used, we decrease the amount of time between classifications so that administrators are notified more quickly, but there are increases in the: amount of data which is stored, amount of computation required for the classifier, and in

the number of false positives generated. Longer intervals tend to decrease the amount of false positives, the storage requirements, and the computation time, but they increase the amount of time an attacker can go unnoticed. As can be seen from the figure, the averaging interval reaches a point of diminishing returns where it is unable to attain significantly fewer false positives.

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 |

We see from the confusion matrix in table 4.4 that the Naïve Bayes mis-classified E1 as D4, L2 as D3, and both D1 and D2 as L1. In the worst case, E1 was using only Internet Explorer and was classified as D4. The two cases that led to L1's false positive rate of 1.3% involved two separate, but small, sets of processes in use by D1 and D2.

With the exception of the L2 and D3 case, the mis-classifications continued over long periods (30 minutes or more) during which the same set of processes were in use. The L2/D3 case has much smaller periods (less than 15 minutes) of consecutive mis-classification and the set of processes used during each those periods varied. Among those sets of processes, two continued to appear, although not always together, Photoshop and the Adobe

Acrobat Distiller. On closer inspection of the *training* data, we found that Distiller occurred more than twice as often, and Photoshop more than seven times as much in D3's data when compared to L2's. Thus, when L2 used these processes during the test phase, they increased L2's chances of being labeled as D3.

### Relative

The Relative test was an early idea in response to the need for system-independent tests and was considered before finding that the handle count performed so well. The test was similar to the Process Set test in terms of accuracy, but with a slightly different goal. Instead of only considering the set of processes being used, the Relative tests consider how those processes are being used relative to one another. We found it to yield an average of 79.1% true positives and 2.5% false positives across all users.

We saw low true positives for several users: D4, L2, L1, and D1; with rates of 43.6%, 69.9%, 73.1%, and 73.8%, respectively. We believe these rates can be attributed to the fact that the handle count ratio for each process is dependent on the other running processes. For example, if two processes are running: one with a handle count of 20 and the other with a handle count of 80, their respective relative handle counts are 0.2 and 0.8. If one of those processes is closed, it leaves the other with a relative handle count of 1.0. This is a drastic change if the process with a handle count of 20 is the one left running. By inspecting the predictions made by Naïve Bayes, we were able to verify that the opening and closing of processes was highly related to mis-classifications in the data.

### Relative Averages

We had hoped that adding the element of time to the Relative test would improve it's effectiveness, but such was not the case. The accuracy of this test at 45 second averaging intervals was lower than that of the Relative test. It averaged 78% true positives and 2.7% false positives. We also attempted different averaging intervals of 30 and 60 seconds. The 30 second test gave 71.7% true positives, and the 60 second test gave an increase of 0.2% over the 45 second case.

**Common**

The Common test was an attempt to mimic a masquerading attack by only utilizing processes which the users have in common. To our knowledge, we are the first to use this type of test in user identification research. We conducted the tests between pairs of users chosen from the groups established by the architecture firm. Since users in the same groups have the most similar job roles, they seem to be good candidates for this type of test.

Table 4.5 displays the results of those tests. The table shows slightly modified confusion matrix where only false positives are displayed. The data being considered in this test are averages of process handle count over 45 second intervals. There are 3,333 records in the test data set for each user.

Table 4.5: Confusion Matrix for Groups - 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 most notable cells in the table correspond to the Documentation and Plans group where D3 was falsely classified as user D2 1,066 times, and D1 classified as D5 796 times. We originally thought the number of shared processes between two users would have an impact on the false positive rate, but this does not seem to be the case. The number of shared processes in the Documentation and Plans group ranged from 23 to 31. The cases with large numbers of false positives shared 28 and 24 processes, respectively.

There are two cases which lead to D3 being mis-labeled as D2: slight changes in one process, and closing one process to leave an even smaller set of processes from which

a classification decision must be made. It was interesting that a change in average handle count for one process, from 77 to 115, was the only difference between a correctly and incorrectly labeled record. We believe this happened as a result of the regular Naïve Bayes model not changing as the user changes behavior. This hypothesis was confirmed when we used the updateable algorithm with the same set of data; the false positive rate dropped from 0.32 to 0.037.

D5's high false positive rate resulted from from the same problems. Running updateable Naïve Bayes lowered the false positive rate to 0.157 from 0.239. A more thorough comparison of the two classifiers is given in the following section.

**Algorithm Comparison**

Table 4.6: Comparison of Naïve Bayes and Updateable TPR / FPR

| | Naïve Bayes | | Updateable | |
|---|---|---|---|---|
| | TPR | FPR | TPR | FPR |
| Unmodifed | 88.7 | 1.4 | 84.8 | 1.9 |
| Process Set | 80.3 | 2.48 | 77.9 | 2.76 |
| Averages | 96.7 | 0.42 | 94.5 | 0.7 |
| Relative | 79.1 | 2.62 | 81.1 | 2.34 |
| Relative Averages | 78 | 2.73 | 88.3 | 1.47 |

We also used the updateable variant of Naïve Bayes for each of the tests. The results from those tests are in table 4.6. Surprisingly, the updateable variant only out-performs the standard algorithm in two of the five tests. We originally thought the updateable version would have classification performance close to or exceeding that of Naïve Bayes. We think this may have something to do with how quickly the updateable variant adapts its model to the user, but are not able to provide proof of this hypothesis. The most interesting result is the performance difference in the Relative Averages test which sees a gain of 10.3 in its true positive rate. We were also unable to determine why this test did so much better with the updateable algorithm. One of the key problems we had in investigating this was that the model for the updateable algorithm is always changing. This made it incredibly difficult to determine what the model looked like when a given record was being tested without running a separate simulation that stopped just before classifying the record.

We also looked at the per user true and false positive outcomes across all the tests shown in table 4.6. We found that certain users were consistently identified better by the updateable model and some better identified by the basic one. This leads us believe that it may be useful to maintain and use both models for classification and combine the results. Wang and Stolfo [68] also found this in their research.

We also provide the false positive table for the Common test using the updateable algorithm in table 4.7. The updateable algorithm considerably outperforms the standard one in the Common test. The average false positive rate over all users is 0.038 with Naïve Bayes and 0.021 with the updateable variant.

Table 4.7: Confusion Matrix for Groups - Updateable Common Test

| | E1* | E2* | D1* | D2* | D3* | D4* | D5* | L1* | L2* |
|---|---|---|---|---|---|---|---|---|---|
| E1 | | 130 | | | | | | | |
| E2 | 34 | | | | | | | | |
| 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 | | | |
| L1 | | | | | | | | | 6 |
| L2 | | | | | | | | 94 | |

One possible explanation for the limited success of the updateable variant is that it is able to adapt quickly to small, incremental changes such as a change in the average handle count of a commonly used process; but is not able to adapt to large changes such as the use of a new program, at least not within the limited frame of data within which we are testing.

**Learning Time**

One of the algorithm characteristics we were interested in was the amount of training data required to generate user profiles. Figure 4.3 plots the false positive rate against the true positive rate for the Naïve Bayes classifier over training periods of one day,

three days, one week, one and one half weeks, and two weeks. This graph shows results for all user processes in the 45 second Average handle count test.

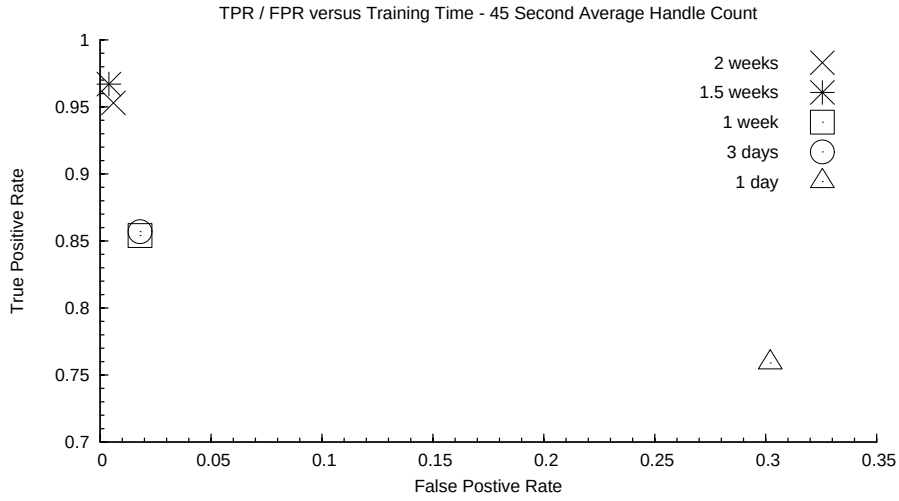


Figure 4.3: TPR / FPR versus Training Time

Only one day of training data yields a rate of 75.8% which is impressive considering that of the following 14 days of data, over 75% can be correctly identified. The three day and one week training periods provide about 85% true positives and 1.8% false positives. The longest periods of one and one half and two weeks provide the best results with greater than 95% true positives and less than 1% false positives.

### 4.3.3 Causes of False Positives

We have mentioned the leading causes for large false positive rates for each of the tests in their respective sections, but will briefly summarize those here. The largest source of false positives throughout the tests were periods where a user had only a few commonly-used programs open. We also found false positives when users dramatically shifted their behavior. For instance, when a user began using a new program which was not in their training data or more heavily used a program that was used very little in the training data. In the Common test, we found that slight changes in the use of a program, coupled with a small commonly-used set of processes, can lead to false positives.

Given this information, we can also postulate as to why some users rarely see a large number of false positives. The use of uncommon processes, using large numbers of

processes simultaneously, unique usage of common processes, and consistent usage of the same programs are all characteristic of this type of user. Two of the users, D1 and E2, in this experiment had consistently low false positive rates of 1% or less in all of the tests. Both had processes that were unique to their profiles, E2 was rarely found to have less than three processes open, and D1's average handle count for Internet Explorer was almost twice the average of any other user.

## 4.4 Limitations and Possible Attacks

Our work is limited by the amount of data we were able to collect. It is essential that more data be collected to run more intensive tests to validate our results. It is also important that the data be able to be made public. A major drawback in this work and in other insider threat related research is that the data is not widely available to other researchers. This prohibits peer validation of research efforts and prevents new research from making direct comparisons with previous work.

The most obvious attack is for one user to masquerade as another. It is not the intent of this thesis to provide a provable defense against this attack, but we show here why we think this attack to be non-trivial given the type of data were are monitoring. For this attack to be successful, the attacker would need access to the targeted user's log data.

We believe that access to the log data can be prevented by implementing a secure logging facility and see this as a future improvement. Even in the event that an attacker were able to access the target's log data, they would have to train themselves to use the computer as the target does. This requires not only using the same applications as the target, but also using those applications in a similar manner in regards to the number of resources used by each process. Given the that usage reflects a user's learned and instinctive habits, that the usage reflects the work role of the user, and even that it reflects the way a person works, we believe this to be a improbable attack from all but the most dedicated of attackers.

Given access to the target's log data, it is also possible that a computer program be written to mimic the target's usage. This type of attack would most likely not be detected by our approach. However, to be successful, the program would have to use the same applications the user uses, and in the same way the user uses them, to carry out some malicious behavior.

Processes which are not identified as user processes by an administrator will not be monitored with our approach. Our system identifies processes by their name so we expect that process names are uniform across every machine being monitored. For example, if Internet Explorer is named iexplore.exe on one machine, we assume it has the same name on the other machines.

# Chapter 5

# Conclusions and Future Work

## 5.1 Summary of Results

We have shown that is possible to accurately identify users by the creation and use of process profiles. The Naïve Bayes algorithm 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. We found that the main sources of false positives in the data were instances where only a few commonly used processes were in use and when there were dramatic differences in process usage during between the training and testing periods. We also identified characteristics of users who maintained low false positive rates across each of the tests we ran.

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 similar work in other environments, our work has yielded higher true positives and lower false positives than any of the previous work. We also considered analyzed 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.

## 5.2 Future Work

Our main assumption that the test data is composed only of normal data is naïve and must be addressed. We plan to examine the use of profile comparisons between users in similar roles to validate this data. For instance, by comparing several profiles of programmers and locating the unique anomalies, we can pin point activity which can then be inspected by a human administrator.

Another issue that must be addressed is that this research assumes users will not tamper with the data, attempt to circumvent the monitors, or view the data of other users. A recent paper accepted to CCS '07 may provide a solution to this problem [26]. The research uses virtualization technology which would prevent users from tampering with the monitors and data by moving them outside the operating system (OS) and into a virtual machine monitor. This work is intended for detecting malware, but we believe it would be useful in this domain as well. A unique feature of Jiang's research is that they are able to re-construct semantic views of the user's actions; by defining the data structures and functions of the guest OS in a semantic way, they are able to view the files and processes which are being used. While it is simple to monitor this information within the OS itself, they are able to do this outside the OS which adds an additional layer of protection in preventing access to the log data.

We are still collecting data from the first organization mentioned in section 3.3.2 and plan to use this data in future insider threat research.

Each of the records collected were associated with a timestamp. We did not use this information in our work, but it may be useful. For instance, we could create a simple profile that tracks the times a user normally uses their computer. This may further substantiate alerts from our system that arise from usage outside normal operating hours.

We have only made an intuitive case for being able to remove the system configuration dependencies from the logged data with by use of the handle count metric and the Relative tests. As more data is available, we would like to prove this with evidence.

On a related note, we would also like to investigate the use of other metrics. We chose the metrics used because we were trying to collect as little data as possible in order in increase the chances of a company allowing us to collect data. The Microsoft .NET framework allows for the collection of other process specific data such as: the thread count, and the amount of disk activity related to read, write, and control operations. We would

like to test this data as well.

We also would like to pursue a combination of the Naïve and updateable schemes to do classification. By combining classification results from the two, we may be able to increase the classification performance of the system.

We found that falsely classified records often appeared in succession. In some cases, several hundred consecutive records for one user were classified as another user. By combining these cases into a single record, the false positive rate would be drastically reduced. Shavlik [62, 61] required a certain number of consecutive records be identically classified before issuing an alert which may be beneficial in this research as well.

# Bibliography

[1] B. Aleman-Meza, P. Burns, M. Eavenson, D. Palaniswami, and A. Sheth. An ontological approach to the document access problem of insider threat. In *ISI*, pages 486–491, 2005.

[2] N. Amor, S. Benferhat, and Z. Elouedi. Naive Bayes vs decision trees in intrusion detection systems. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 420–424, New York, NY, USA, 2004. ACM.

[3] D. Anderson, D. Cappelli, J. Gonzalez, M. Mojtahedzadeh, A. Moore, E. Rich, J. Sarriegui, T. Shimeall, E. Weaver J. Stanton, and A. Zagonel. Preliminary system dynamics map of the insider cyber-threat problem. In *Proceedings of the 22nd International Conference of the System Dynamics Society*, 2004.

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

[5] K. Anderson, A. Carzaniga, D. Heimbigner, and A. Wolf. Event-based document sensing for insider threats. Technical Report CU-CS-968-04, University of Colorado Department of Computer Science, 2004.

[6] R. Battistoni, E. Gabrielli, and L. Mancini. An extended access control system for Windows XP, 11 2003.

[7] R. Battistoni, E. Gabrielli, and L. Mancini. A host intrusion prevention system for Windows operating systems. In *ESORICS*, pages 352–368, 2004.

[8] W. Bhukya, S. Kommuru, and A. Negi. Masquerade detection based upon GUI user profiling in Linux systems. In Iliano Cervesato, editor, *ASIAN*, volume 4846 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2007.

[9] S. Bleha, C. Slivinsky, and B. Hussien. Computer-access security systems using keystroke dynamics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(12):1217–1222, 1990.

[10] Bloomberg. Societe Generale reports EU4.9 billion trading loss. http://www.bloomberg.com/apps/news?pid=20601087&sid=azUPx3TKR8zs &refer=home, 2008.

[11] P. Bradford, M. Brown, J. Perdue, and B. Self. Towards proactive computer-system forensics. In *ITCC (2)*, pages 648–652, 2004.

[12] P. Bradford and N. Hu. A layered approach to insider threat detection and proactive forensics. In *Annual Computer Security Applications Conference*, 2005.

[13] J. Calandrino and S. McKinney. Detection of undesirable insider behavior, 2007.

[14] D. Cappelli, A. Moore, and T. Shimeall. Protecting against insider threat. *ttp://www.sei.cmu.edu/news-at-sei/columns/security_matters/2007/02/security-matters-2007-02.htm*, February 2007.

[15] R. Cathey, L. Ma, N. Goharian, and D. Grossman. Misuse detection for information retrieval systems. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 183–190, New York, NY, USA, 2003. ACM.

[16] S. Chakrabarti, M. Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[17] P. Costa, D. Barbara, K. Laskey, E. Wright, G. Alghamdi, S Mirza, M. Revankar, and T. Shackleford. DTB Project: A behavioral model for detecting insider threats. In *International Conference on Intelligence Analysis*, 2005.

[18] P. Costa, K. Laskey, and G. Alghamdi. Bayesian ontologies in AI systems. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, 2006.

[19] InfoSec Research Council. Infosec Research Council. *http://www.infosec-research.org/*, 2004.

[20] S. Dash, K. Reddy, and A. Pujari. Episode based masquerade detection. In S. Jajodia and C. Mazumdar, editors, *ICISS*, volume 3803 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2005.

[21] 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.

[22] D. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, 1987.

[23] C. Feng, J. Peng, H. Qiao, and J. Rozenblit. Alert fusion for a computer host based intrusion detection system. In *ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 433–440, Washington, DC, USA, 2007. IEEE Computer Society.

[24] A. Garg, R. Rahalkar, S. Upadhyaya, and K. Kwiat. Profiling users in GUI based systems for masquerade detection. In *2006 IEEE Information Assurance Workshop*, pages 48–54, Washington, DC, USA, 2006. IEEE Computer Society.

[25] M. Hall. How to get detailed accuracy for updateable classifiers. Wekalist mailing list, February 2008.

[26] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection through VMM-based "out-of-the-box" semantic view reconstruction. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 128–138, New York, NY, USA, 2007. ACM.

[27] K. Killourhy and R. Maxion. Toward realistic and artifact-free insider-threat data. In *ACSAC '07: Proceedings of the 23rd Annual Computer Security Applications Conference*, Miami Beach, Fl, USA, 2007.

[28] H. Kim and S. Cha. Empirical evaluation of SVM-based masquerade detection using UNIX commands. *Computers & Security*, 24(2):160–168, 2005.

[29] E. Kowalski, T. Conway, S. Keverline, M. Williams, D. Cappelli, B. Willke, and A. Moore. Insider threat study: Illicit cyber activity in the government sector. Tech-

nical report, Carnegie Mellon Software Engineering Institute / United States Secret Service, January 2008.

[30] K. Laskey, G. Alghamdi, X. Wang, D. Barbara, T. Shackleford, E. Wright, and J. Fitzgerald. Detecting threatening behavior using Bayesian Networks. In *Proceedings of the Conference on Behavioral Representation in Modeling and Simulation*, 2004.

[31] M. Latendresse. Masquerade detection via customized grammars. In K. Julisch and C. Krügel, editors, *DIMVA*, volume 3548 of *Lecture Notes in Computer Science*, pages 141–159. Springer, 2005.

[32] W. Lee, S. Stolfo, and K. Mok. Algorithms for mining system audit data. pages 166–189, 2002.

[33] L. Li and C. Manikopoulos. Windows NT one-class masquerade detection. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 82–87, Washington, DC, USA, 2004. IEEE Computer Society.

[34] D. Lin. Computer-access authentication with neural network based keystroke identity verification. In *International Conference on Neural Networks*, pages 174–178, June 1997.

[35] 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.

[36] T. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 59–66, April 1988.

[37] L. Ma and N. Goharian. Query length impact on misuse detection in information retrieval systems. In *SAC*, pages 1070–1075, 2005.

[38] CSO Magazine, United States Secret Service, CERT, and Microsoft. 2007 E-crime watch survey, 09 2007.

[39] G. Magklaras and S. Furnell. Insider threat prediction tool: Evaluating the probability of IT misuse. *Computers & Security*, 21(1):62–73, 2002.

[40] G. Magklaras and S. Furnell. A preliminary model of end user sophistication for insider threat prediction in IT systems. *Computers & Security*, 24(5):371–380, 2005.

[41] 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.

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

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

[44] Microsoft. Maximum NT user handles per process is 10,000 in Windows XP. http://support.microsoft.com/kb/327699/en-us, 2007.

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

[46] N. Nguyen, P. Reiher, and G. Kuenning. Detecting insider threats by monitoring system call activity. In *IAW*, pages 45–52, 2003.

[47] J. Park and J. Giordano. Role-based profile analysis for scalable and accurate insider-anomaly detection. In *Proceedings of the 25th IEEE International Performance Computing and Communications Conference*, pages 463–469, 2006.

[48] J. Park and S. Ho. Composite role-based monitoring (CRBM) for countering insider threats. In *ISI*, pages 201–213, 2004.

[49] A. Peacock, X. Ke, and M. Wilkerson. Typing patterns: A key to user identification. In *IEEE Security and Privacy*, pages 40–47, September 2004.

[50] J. Peng, C. Feng, H. Qiao, and J. Rozenblit. An event-driven architecture for fine grained intrusion detection and attack aftermath mitigation. In *ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 55–62, Washington, DC, USA, 2007. IEEE Computer Society.

[51] S. Pramanik, V. Sankaranarayanan, and S. Upadhyaya. Security policies to mitigate insider threat in the document control domain. In *Proceedings of the 20th Annual*

*Computer Security Applications Conference (ACSAC'04)*, pages 304–313, Washington, DC, USA, 2004. IEEE Computer Society.

[52] H. Qiao, J. Peng, C. Feng, and J. Rozenblit. Behavior analysis-based learning framework for host level intrusion detection. In *ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 441–447, Washington, DC, USA, 2007. IEEE Computer Society.

[53] M. Randazzo, M. Keeney, E. Kowalski, D. Cappelli, and A. Moore. Insider threat study: Illicit cyber activity in the banking and finance sector. Technical Report CMU/SEI-2004-TR-021, Carnegie Mellon Software Engineering Institute, June 2005.

[54] B. Robinson. Expert: Spies driven by money and thrill. *http://abcnews.go.com/US/story?id=94034*, February 2001.

[55] 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.

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

[57] E. Schultz. A framework for understanding and predicting insider attacks. *Computers & Security*, 21(6):526–531, 2002.

[58] 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.

[59] Y. Seo and K. Sycara. Cost-sensitive access control for illegitimate confidential access by insiders. In *ISI*, pages 117–128, 2006.

[60] 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.

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

[62] 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.

[63] S. Smaha. Haystack: An intrusion detection system. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44, December 1988.

[64] Nueber Software. Common Windows processes. http://www.neuber.com/taskmanager/process/, 2008.

[65] L. Spitzner. Honeypots: Catching the insider threat. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 170, Washington, DC, USA, 2003. IEEE Computer Society.

[66] S. Symonenko, E. Liddy, O. Yilmazel, R. Zoppo, E. Brown, and M. Downey. Semantic analysis for monitoring insider threats. In *ISI*, pages 492–500, 2004.

[67] Uniblue. Uniblue process library. http://www.liutilities.com/products/wintaskspro/processlibrary/, 2008.

[68] K. Wang and S. Stolfo. One class training for masquerade detection. In *3rd IEEE Conf Data Mining Workshop on Data Mining for Computer Security*, Florida, USA, 2003. IEEE Computer Society.

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

[70] O. Yilmazel, S. Symonenko, N. Balasubramanian, and E. Liddy. Improved document representation for classification tasks for the intelligence community. In *Proceedings of the AAAI Spring Symposium Series*, 2005.

[71] O. Yilmazel, S. Symonenko, N. Balasubramanian, and E. Liddy. Leveraging one-class SVM and semantic analysis to detect anomalous content. In *ISI*, pages 381–388, 2005.

[72] Y. Yu and T. Chiueh. Display-only file server: a solution against information theft due to insider attack. In *DRM '04: Proceedings of the 4th ACM workshop on Digital Rights Management*, pages 31–39, New York, NY, USA, 2004. ACM.

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

# Appendices

# Appendix A

# User Processes

1. 3Ds Viz
2. AutoCad
3. AutoCad Help
4. Adobe Acrobat
5. Adobe Acrobat Help
6. Adobe Acrobat Distiller
7. Adobe Acrobat Reader
8. Adobe Acrobat Reader Help
9. Adobe Collaboration Sync
10. Apple Mobile Device Helper
11. Microsoft Calculator
12. Command Prompt
13. Disk Defragmenter
14. DVDLauncher
15. DWG True View
16. Editors700 (Engineer application)
17. Excel
18. Free Download Manager
19. Firefox
20. Internet Explorer
21. Adobe Illustrator
22. Adobe ImageReady
23. Adobe InDesign

24. Windows CardSpace
25. iTunes
26. LaserTrack Pro
27. Microsoft Management Console
28. Windows Installer
29. Maxwell Lighting Renderer
30. Maxwell Material Editor
31. Maxwell Studio
32. Notepad
33. Windows 16-bit Virtual Machine
34. Outlook
35. PDF Save
36. Photoshop
37. PeerNet Image Printer
38. PowerPoint
39. Print Scout
40. QuickTime Player
41. Raysat VIZ2008
42. Remote Desktop Copy and Paste
43. Revit
44. SQL Anywhere Database
45. Volume Control
46. Windows Task Manager
47. Toolbox (Engineer application)
48. Trace (Engineer application)
49. Trilogy Encrypted Chat
50. Micro Station
51. WinRAR
52. WINWORD
53. WINZIP32
54. WISPTIS (Tablet pen input)
55. Windows Media Player
56. Autodesk Communication Center
57. xcopy

# Appendix B

# Naïve Bayes Example

Table B.1 contains twelve records for two users. There are four processes represented: Internet Explorer, Outlook, Word, and Acrobat. If a process has a value of 1, then the process was running when the record was collected, and if it has the value 0, it was not running. Given this data set, we would like to use Naïve Bayes to find out who was using the computer when all four processes were running simultaneously.

Table B.1: Naïve Bayes Example Training Set.

| Record # | IE | Outlook | Word | Acrobat | User |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | A |
| 2 | 0 | 1 | 0 | 0 | A |
| 3 | 1 | 1 | 1 | 0 | A |
| 4 | 1 | 0 | 1 | 1 | A |
| 5 | 0 | 1 | 0 | 0 | A |
| 6 | 1 | 1 | 0 | 1 | A |
| 7 | 1 | 0 | 1 | 0 | B |
| 8 | 1 | 0 | 0 | 1 | B |
| 9 | 1 | 1 | 0 | 0 | B |
| 10 | 0 | 1 | 0 | 1 | B |
| 11 | 1 | 1 | 0 | 1 | B |
| 12 | 1 | 1 | 0 | 0 | B |

Recall the formula for using Naïve Bayes in classifications:

$$classify(a_1, ..., a_n) = argmax_{c_k} p(C = c_k) \prod_i p(A_i | C = c_k)$$

We must first compute the probabilities that each process is running per user. These can be obtained from the table and $P(IE|A)$ should be interpreted as "the probability that Internet Explorer is running given that A is using the computer."

$$
\begin{aligned}
P(A) &= 1/2 \\
P(IE|A) &= 1/2 \\
P(Outlook|A) &= 2/3 \\
P(Word|A) &= 5/6 \\
P(Acrobat|A) &= 1/2
\end{aligned}
$$

$$
\begin{aligned}
P(B) &= 1/2 \\
P(IE|B) &= 5/6 \\
P(Outlook|B) &= 2/3 \\
P(Word|B) &= 1/6 \\
P(Acrobat|B) &= 1/2
\end{aligned}
$$

We can now use the above formula to determine which user is more likely to be using the computer:

$$
\begin{aligned}
P(A|IE, Outlook, Word, Acrobat) &= P(A)P(IE|A)P(Outlook|A)P(Word|A)P(Acrobat|A) \\
&= 0.046
\end{aligned}
$$

$$
\begin{aligned}
P(B|IE, Outlook, Word, Acrobat) &= P(B)P(IE|B)P(Outlook|B)P(Word|B)P(Acrobat|B) \\
&= 0.023
\end{aligned}
$$

Since the probability for user A is greater than that of user B, we predict that user A is using the computer when all four processes are running.

The steps are the same for Updateable Naïve Bayes, but after the classification is made, the newly classified record is added to the set of records in table B.1 and future predictions use the record as if it were in the training set.