3-9-2009

Host-Based Multivariate Statistical Computer Operating Process Anomaly Intrusion Detection System (PAIDS)

Glen R. Shilland

HOST-BASED MULTIVARIATE STATISTICAL COMPUTER OPERATING PROCESS ANOMALY INTRUSION DETECTION SYSTEM (PAIDS)

THESIS

Glen R. Shilland, Major, USAF

AFIT/GOR/ENS/09-15

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/09-15

HOST-BASED MULTIVARIATE STATISTICAL COMPUTER OPERATING PROCESS ANOMALY INTRUSION DETECTION SYSTEM (PAIDS)

THESIS

Presented to the Faculty

Department of Operations Research

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Glen R. Shilland

Major, USAF

March 2009

AFIT/GOR/ENS/09-15

HOST-BASED MULTIVARIATE STATISTICAL COMPUTER OPERATING PROCESS ANOMALY INTRUSION DETECTION SYSTEM (PAIDS)

Glen R. Shilland
Major, USAF

Approved:

_____	_____
Kenneth W. Bauer, Jr., PhD (Chairman)	Date
_____	_____
Jeffrey W. Humphries, Lt Col, PhD, USAF (Reader)	Date

Abstract

Most intrusion detection systems rely on signature matching of known malware or anomaly discrimination by data mining historical network traffic. This renders defended systems vulnerable to new or polymorphic code and deceptive attacks that do not trigger anomaly alarms. A lightweight, self-aware intrusion detection system (IDS) is essential for the security of government and commercial networks, especially mobile, ad-hoc networks (MANETs) with relatively limited processing power. This research proposes a host-based, anomaly discrimination IDS using operating system process parameters to measure the "health" of individual systems. Principal Component Analysis (PCA) is employed for feature set selection and dimensionality reduction, while Mahalanobis Distance (MD) and is used to classify legitimate and illegitimate activity. This combination of statistical methods provides an efficient computer operating process anomaly intrusion detection system (PAIDS) that maximizes detection rate and minimizes false positive rate, while updating its sense of "self" in near-real-time.

To my grandfather, who taught me the virtues of integrity, service, and excellence before I had ever heard of the Air Force.

Acknowledgements

I would like to express my sincere appreciation to my faculty advisor, for his guidance and encouragement throughout the course of this thesis, especially when neither of us was entirely certain it was actually going to become a thesis. He sincerely tried to get me to enjoy myself, and though I still think multivariate statistics is one step away from magic, I think I'm finally starting to grasp how it works. Thank you to those at Air Force Research Laboratory who got me started on the path towards a possible cyber application for QuEST, and to those at the Air Force Information Operations Center who may yet turn this into something useable.

I am also deeply indebted to my classmates, without whom I would still be trying to tackle basic probability and programming. I would never have made it through this program without study groups and individual help with coding and concepts.

Thank you to my father for his editorial expertise and to my mother for her encouragement of all my educational endeavors. Of course, none of this would be possible without the love and support of my wife and children. Words cannot express my indebtedness to them for their patience and perseverance.

Glen R. Shilland

Table of Contents

List of Figures

List of Tables

HOST-BASED MULTIVARIATE STATISTICAL COMPUTER OPERATING PROCESS ANOMALY INTRUSION DETECTION SYSTEM (PAIDS)

I. Background

1.1 Introduction

Methods of cyberattack are limited only by human ingenuity, but most attacks take advantage of code vulnerabilities, the inherent trust architecture of Internet protocol, or insecure habits of users and administrators. Cyberattacks vary in form from denial of service to scanning/probing to penetration, and emanate from a wide range of sources from pranksters to organized criminals to nation states. "As systems become more complex, there are always exploitable weaknesses due to design and programming errors, or through the use of various "socially engineered" penetration techniques." (Khan, Awad, & Thuraisingham, 2007) The goal of cyberdefense in general, and an intrusion detection system (IDS) in particular, is to limit the impact of these inherent weaknesses, so that attacks are either thwarted outright or inflict the least amount of damage possible.

"The task of preventing unauthorized users from compromising the confidentiality, integrity, or availability of sensitive information, is increasingly difficult in the face of the growth in Internet use, the increasing skill levels of attackers themselves, and technological advances in their tools and methods of attack." (GAO, 1996) Thus, it is increasingly likely that attacks will be successful and go unnoticed. Assuming that any system is "crackable" by a determined adversary, the only way to mitigate an attack is through early detection and isolation. Due to the diversity of attack avenues, commercial and government administrators struggle to defend themselves in a

constantly changing environment. IDSs have emerged as essential tools to ensure security against these emerging threats.

1.2 What is an intrusion?

An intrusion is any use of a computer or network for which the user either does not have privileges or authorization. It could be a penetration by an outsider or misuse by an insider, and the result could be denial, alteration, or theft of data. The types of intrusion have changed very little since Dorothy Denning's seminal work, An Intrusion Detection Model (1986), which lists examples of intrusions and indicators:

Intrusion Type	*Possible Indicators*
Attempted break-in	• High rate of password failures with respect to a single account or the system as a whole • [High rate of port access requests on a single host (portscan) or across a network (portsweep)]
Masquerading or successful break-in	• Different login time, location, or connection type from the account's legitimate user • Different behavior pattern from legitimate user (browsing directories or executing system status commands vs. editing or compiling programs)
Penetration by legitimate user	• User executes different programs or triggers more protection violations from attempts to access unauthorized files or programs • User will have access to commands and files not normally permitted
Leakage by legitimate user	• Unusual login times or data routed to remote printers [or hosts] not normally used
Inference by legitimate user	• A user attempts to retrieve more records than usual in an attempt to aggregate or infer unauthorized data
Trojan Horse	• Behavior of planted or substituted program may differ from legitimate program in terms of its CPU time or I/O activity
Virus	• Increase in frequency of executable files rewritten, storage used by executable files, or a particular program being executed
Denial of Service	• Abnormally high resource activity with respect to a particular user while activity for all other users is abnormally low

Table 1 – Intrusions and Indicators (Denning, 1986)

What makes these intrusions so difficult to detect is that "aberrant usage can also be linked with actions unrelated to security. They could be a sign of a user changing work tasks, acquiring new skills, or making typing mistakes, software updates, or changing workload on a system." (Denning, 1986) Distinguishing between abnormal legitimate and illegitimate behavior is therefore very challenging. Also, intruders are able to test the limits of an IDS to determine its characteristics, and then attempt to appear as normal as possible to avoid tripping alarms.

"The number of incidents reported by federal agencies to the [U.S. - Computer Emergency Readiness Team] US-CERT has increased dramatically over the past 3 years, increasing from 3,634 incidents reported in fiscal year 2005 to 13,029 incidents in fiscal year 2007." (GAO, 2008) The most prevalent attacks for FY07 are presented in Figure 1, where "investigation" indicates unconfirmed incidents that are potentially malicious or anomalous activity deemed by the reporting entity to warrant further review.

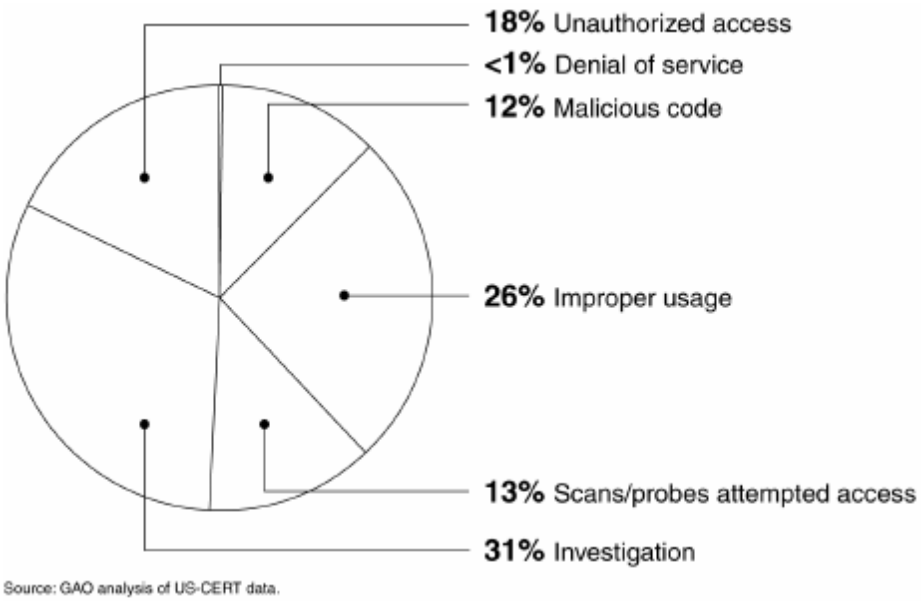


Figure 1 – Percentage of incidents reported to US-CERT in FY07 (GAO, 2008)

1.3 What is malware?

Often when an intrusion occurs, whether from an internal or external source, the intent of attacker is to install some malicious code, known as malware, which will function either autonomously or with some human interaction to perform a desired task. The task could be annoying, such as sending an advertising email to everyone in the victim's address book (a common goal of spammers) or it could be more insidious such as opening a backdoor to allow future unhindered attacks.

Like all software, the functionality of malware is only limited by the creativity of the programmer. However, malware generally falls into distinguishable categories of form and function. Each type of malware has a specific purpose, with associated strengths and weaknesses, capabilities and limitations. The discussion of cyberwarfare is easier when these terms, shown in Table 2, are used correctly (a Trojan is not a virus) but unfortunately many malware programs are combinations that defy simple classification.

Malware Type	*Defining Characteristics*
Virus	• Does not self-replicate, usually requires human interaction to spread from host to host, either through removable media or network
Worm	• Self-replicates, usually does not require human interaction to spread across a network
Malicious Mobile Code	• Lightweight programs that are downloaded from a remote system and executed locally with minimal or no user intervention
Backdoor	• Bypasses normal security controls to give an attacker access
Trojan Horse	• Disguises itself as a useful program while masking hidden malicious purposes
User-Level Rootkit	• Replaces or modifies executable programs used by system administrators and users
Kernel-Level Rootkit	• Manipulates the heart of the operating system, the kernel, to hide and create backdoors
Combination	• Combines various techniques listed above to increase effectiveness

Table 2 – Taxonomy of Malware (Skoudis & Zeltser, 2004)

1.4 Who is the threat?

Threats can come from anywhere at any time, but who is most likely to initiate a cyberattack? Table 3 presents a partial list of the potential perpetrators.

Threat source	Description
Criminal groups	There is an increased use of cyber intrusions by criminal groups that attack systems for monetary gain.
Foreign nation states	Foreign intelligence services use cyber tools as part of their information gathering and espionage activities. Also, several nations are aggressively working to develop information warfare doctrine, programs, and capabilities. Such capabilities enable a single entity to have a significant and serious impact by disrupting the supply, communications, and economic infrastructures that support military power—impacts that, according to the Director of the Central Intelligence Agency, can affect the daily lives of Americans across the country.
Hackers	Hackers sometimes crack into networks for the thrill of the challenge or for bragging rights in the hacker community. While remote cracking once required a fair amount of skill or computer knowledge, hackers can now download attack scripts and protocols from the Internet and launch them against victim sites. Thus, attack tools have become more sophisticated and easier to use.
Hacktivists	Hacktivism refers to politically motivated attacks on publicly accessible Web pages or e-mail servers. These groups and individuals overload e-mail servers and hack into Web sites to send a political message.
Disgruntled insiders	The disgruntled insider, working from within an organization, is a principal source of computer crimes. Insiders may not need a great deal of knowledge about computer intrusions because their knowledge of a victim system often allows them to gain unrestricted access to cause damage to the system or to steal system data. The insider threat also includes contractor personnel.
Terrorists	Terrorists seek to destroy, incapacitate, or exploit critical infrastructures to threaten national security, cause mass casualties, weaken the U.S. economy, and damage public morale and confidence. However, traditional terrorist adversaries of the United States are less developed in their computer network capabilities than other adversaries. Terrorists likely pose a limited cyber threat. The Central Intelligence Agency believes terrorists will stay focused on traditional attack methods, but it anticipates growing cyber threats as a more technically competent generation enters the ranks.

Source: Federal Bureau of Investigation, unless otherwise indicated.

Table 3 – Cyber Threats to Federal Systems and Critical Infrastructures (GAO, 2008)

Clearly, the list of adversaries is extensive, but how often are these attacks successful? In a 1996 report (Information Security - Computer Attacks at Department of Defense Pose Increasing Risk), the GAO reported

> DISA estimates indicate that Defense may have been attacked as many as 250,000 times last year. However, the exact number is not known because, according to DISA, only about 1 in 150 attacks is actually detected and reported. In addition, in testing its systems, DISA attacks and successfully penetrates Defense systems 65 percent of the time. According to Defense officials, attackers have obtained and corrupted sensitive information--they have stolen, modified, and destroyed both data and software. They have installed unwanted files and "back doors" which circumvent normal system protection and allow attackers unauthorized access in the future. They have shut down and crashed entire systems and networks, denying service to users who depend on automated systems to help meet critical missions. Numerous Defense functions have been adversely affected, including weapons and supercomputer research, logistics, finance, procurement, personnel management, military health, and payroll.

In the last decade, the frequency and sophistication of attack vectors has increased, while the knowledge required to use them has decreased, thus exposing computer systems to a constantly expanding threat. Recently in Estonia and Georgia, cyberattacks were used in conjunction with physical attacks to cripple the response of their respective governments. Whether these attacks were initiated by state run agencies or merely “hacktivists” is irrelevant, the important point is that they were highly effective. We are not entirely defenseless, however, and our safeguards have also increased considerably, primarily in the realm of security awareness and access control.

1.5 The first line of defense

Multiple levels of security are necessary for a truly secure network. Physical security can be considered the first layer of security. This consists of not allowing unauthorized users physical access to secure computers or networks, such as keeping them sequestered in a vault with no connection to the Internet. However, this can severely limit the utility of the system and is often not worth the inconvenience.

Another frontline measure is operational security (OPSEC), which can be as simple as not associating certain pieces of information together which, when combined, reveal a secret or vulnerability. Keeping passwords secure, limiting distribution of critical information, and restricting access to "need to know" are all examples of OPSEC. AFDD 2-5 -- Information Operations (11 January 2005) states that Operations Security is "not a collection of specific rules and instructions that can be applied to every operation, it is a methodology that can be applied to any operation or activity for the purpose of denying critical information to the adversary. Critical information consists of data and indicators that are sensitive, but unclassified. OPSEC aims to identify any unclassified activity or information that, when analyzed with other activities and information, can reveal protected and important friendly operations, information, or activities." OPSEC is everyone's responsibility, and is an important aspect of intrusion prevention.

Information security awareness training also plays a large role in maintaining the integrity of a computer network. Often, users are simply unaware that their actions are potentially harmful or may compromise a system. "Users need to know the simple things that they can do to help to prevent intrusions, cyber attacks, or other security breaches.

All users of cyberspace have some responsibility, not just for their own security, but also for the overall security and health of cyberspace." (NIAC 2003) Administrators can go a long way towards protecting their system by merely educating users on basic security measures and enforcing their implementation. Evidence that awareness of the threat is improving can be seen by comparing Figure 1 and Figure 2; the fact that more attempted intrusions and much less improper usage was reported in 2008 is encouraging.

If these doctrinal methods are insufficient, as they often are, the traditional "first layer" of defense in computer security comes into effect – electronic security measures. Some examples of these include: passwords and firewalls; authorization limits, such as administrator and user rights; and cryptologic devices to encode files and message traffic. This is the level at which people generally start thinking about network security, although most intrusions can be foiled at the physical and operational level long before they become a technological problem.

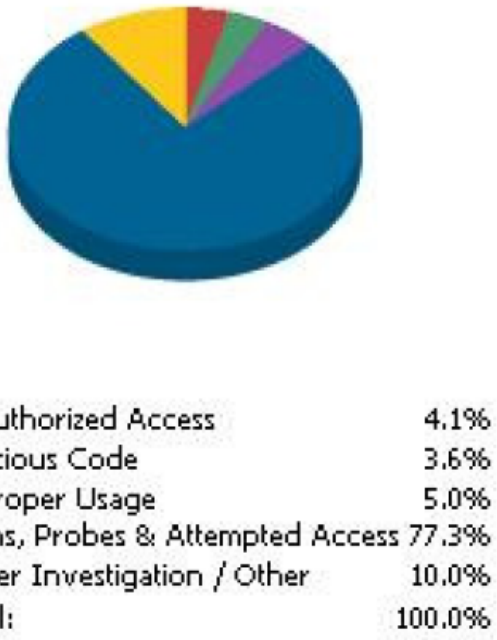


Figure 2 – Percentage of incidents reported to US-CERT in FY08 Q4 (USCERT, 2008)

1.6 What is an IDS?

Intrusion detection systems are the second (or third, or fourth) layer of defense, to notify administrators when the first line of defense has been penetrated or is being probed. The idea of an IDS is "based on the hypothesis that exploitation of a system's vulnerabilities involves abnormal use of the system; therefore, security violations could be detected from abnormal patterns of system usage." (Denning, 1986) Most IDSs attempt to parse network traffic data into a characteristic feature set that includes information such as: type and amount of data transferred; time, frequency, and depth of access; and source/destination Transport Control Protocol/Internet Protocol (TCP/IP) addresses. Although parsing network traffic is highly effective for identifying intrusions, it is increasingly difficult to characterize this data, especially in mobile ad-hoc networks (MANETs) with highly dynamic topologies.

Intrusion detection systems can either be network-based or host-based. The majority of commercial IDSs are network-based, but these are not useful for busy networks, cannot analyze encrypted data, and cannot tell if an attack was successful. (Mell, et al., 2003) The primary concerns for a host-based IDS are its susceptibility to attack and its reliance on the computing power of the device. "An attacker that is able to circumvent the security of the IDS could cause it to issue a large number of false negatives, effectively carrying out a denial of service attack, or could cause its detectors to malfunction." (Merkle, et al., 2002) Although this research is concentrated at the host level, the resulting algorithm could be used as part of a multi-layered network-based IDS.

Intrusion detection systems generally fall into two categories, anomaly detection (profile-matching) and misuse detection (signature-matching), although these types can be combined to form a specification-based system. Standard anti-virus software is typically based on signature matching of previously identified malware code, and this is still the most effective method to detect known attacks. However, since new malware is being created faster than patches can be deployed to signature libraries and much of it is polymorphic (self-mutating), regardless of how often they are updated there is no chance of predicting what the next code will look like. Only anomaly detection is capable of identifying new types of attacks, because the IDS is looking for malevolent behavior or system reactions instead of attempting to recognize known code strings or network traffic patterns. Obviously, the strongest defense would include both types of IDS at multiple layers of the network. This research will use statistical methods of multivariate analysis to identify anomalies in operating system processes, thus retaining the ability to detect intrusions regardless of the vector.

However, anomaly detection can also pose challenges, as it generally requires training at both normal and abnormal operating conditions. The anomaly detection IDS must be taught to recognize its normal "self" and distinguish it from an abnormal "other" state. The problem can be simplified by only training to one condition and assuming that anything not in that category is in the other, but this can lead to false positives if, for instance, a rare occurrence is not in the training data. "The capacity of a certain environment to be self-aware is equivalent to the capacity to detect novelties emerging inside the environment itself." (Balducelli, et al., 2007) There are various methods

proposed in the literature to classify anomalies in this way, including discriminant analysis (DA), decision trees, support vector machines (SVM), and genetic algorithms (GA). The major obstacle to building an anomaly detection system is establishing normality and finding the most effective feature set to expose abnormal conditions.

1.7 A host-based, statistical anomaly IDS proposal

Signature-based IDSs are very good at recognizing attacks which have been previously identified, but are unable to detect an attack for which no pattern has been determined. Likewise, network-based IDSs are necessary to detect distributed attacks, but often cannot ascertain what is happening at the individual component level. This research proposes that a host-based statistical anomaly IDS is necessary to defend against "zero day" attacks targeted at individual computer systems. In addition, this research will show that an IDS can be designed using local operating system process data such as: percentage of kernel and total CPU used; number of threads, handles, and windows open in a process; and number and size of read/write operations. The results of this research will be referred to as a process anomaly intrusion detection system (PAIDS).

Using this type of data is similar to measuring temperature, blood pressure, white blood count, etc. in a human patient. Although we may not be able to tell what kind of sickness (intrusion) is occurring, or where it is coming from, we can determine that something is wrong and take further action. Edward Amoroso calls this method pattern matching and suggests that it "constitutes an especially powerful approach because it provides intrusion detection capability for attacks that might not be predictable. In fact, human operators might detect subtle changes that they can neither explain nor

understand." (1999, p. 65) An added benefit to using this type of data to characterize an intrusion is that it may aid in detecting intrusions using removable media or other direct methods, which analysis of network traffic cannot do. Although some intrusions, such as rootkit attacks, may subvert some of this data, the effects of these attacks should still be measurable in a multi-dimensional feature space.

By collecting a wide range of variables, we can then use multivariate statistical methods to detect anomalies in these processes, and then classify the abnormal states as either legitimate activity or an illegitimate intrusion. Most IDSs collect features based on network traffic to classify intrusions instead of "state of health" information directly from the host operating system. By monitoring operating system behavior and hardware data, PAIDS can develop a sense of "self" and detect anomalies in this representation which is not vulnerable to standard code deception techniques generally employed by cyber attackers. This sense of "self" contains certain characteristics described by an Air Force Research Laboratory and Air Force Institute of Technology working group called the Qualia Exploitation of Sensor Technology (QuEST) headed by Dr. Steven Rogers:

1. *Continuity* – unbroken thread with a cohesive non-causal narrative of past, present and future
2. *Unity* – sensor data is diverse, but is experienced by a single "mind"
3. *Embodiment* – "mind" is anchored in a "body" that is embedded in an environment from which sensory experience is taken
4. *Sense of Free Will* – the environment can be manipulated as a result of actions taken by the "body" determined by the "mind"
5. *Reflection* – "mind" is aware of itself as a separate entity inside a world model

PAIDS assumes an uncompromised "self" at installation and during the start-up process, but this is an unavoidable requirement in any IDS.

1.8 Optimization

The most important aspect of optimizing intrusion detection is determining a subset of data to analyze that minimizes false positives (declaring an attack when there is none) and false negatives (not detecting a real attack) while maximizing true positives (detecting a real attack). Since there is a vast amount of data available to the analyst, even an automatic system is quickly overwhelmed. The challenge is to find the smallest feature set which still provides useful information. Two primary statistical methods for dimension reduction are Principal Component Analysis (PCA) and Factor Analysis (FA). Both of these methods use linear combinations of variables to cluster data and capture the most variance of the original data in the smallest subset possible. Once an anomaly has been identified in the smaller dimensional space, an analyst can investigate the incident further to determine the nature of the intrusion.

Another problem with anomaly detection in the computer environment is the highly variable range of normal operating values, which makes identification of "self" challenging. Standard anomaly detection algorithms have difficulty with this because they are based on a static measurement of historical data, which leads to a high false positive rate. "Historically, minimizing the false positive rate has been a major goal of algorithm designers." (Fox, Kiciman, & Patterson, 2004) Typically, designers are only able to attain low false positives by lowering the true positive rate as well. PAIDS is able to update its control limits and feature set in near-real-time, thus allowing it to recognize shifts in conditions caused either by normal operations or by low level attacks.

II. Literature Review

2.1 Intrusion Detection Categorization

"Hundreds of megabytes of data are collected every second that are of interest to computer security professionals. … What we need are systems that perform data mining at various levels on this corpus of data in order to ease the burden of the human analyst. … Systems that do this type of data mining for security information fall under the classification of intrusion detection systems." (Brugger, et al., 2001) Intrusion detection experts stress that an IDS is not an autonomous device, but a set of procedures to aid a trained analyst. For instance, Stephen Northcutt states, "Intrusion detection is best thought of as a capability, not a single tool. ... Even the best intrusion detection system will be blind to an attack that it is not programmed to detect." (1999, p. 16) Edward Amoroso offers, "Intrusion detection is the *process* of identifying and responding to malicious activity targeted at computing and networking resources." (1999, p. 16) (emphasis added)

Amoroso goes on to identify five methods for intrusion detection (pp. 21-25):

1. Audit trail processing
2. On-the-fly processing
3. Profiles of normal behavior
4. Profiles of abnormal behavior
5. Parameter pattern matching.

In general, however, the computer security community has defined two levels for intrusion detection – at the host or on the network, and three major types of detectors – signature, anomaly, or specification-based. There are advantages and disadvantages to

each of these, and the strongest defense is obviously multi-layered using all types of IDS. Some discussion of previously suggested IDS implementation follows.

2.1.1 Host-Based

A host-based IDS resides on individual systems, and either reports data to a consolidated analysis center to be processed, or processes the data itself and performs automatic responses to detected intrusions. Anti-virus software on a personal computer is a good example of a host-based IDS. "Performing analysis strictly at the host level has the advantage of minimizing network load. However, it has the disadvantage of not allowing the detection of broad scale attacks targeting a network of machines (for instance, an attacker sequentially hopping through a network performing brute force password guessing against each host.)" (Bace, 2000) A host-based IDS is best suited for high activity systems, or whose compromise would pose a serious risk to operations. "At the very least, host-based intrusion detection code should be deployed on all server systems, as well as corporate officers and other key personnel." (Northcutt, 1999, p. 16)

One major problem, "with the host-based IDS is the high processing overhead that they impose on their host." (Kabiri & Ghorbani, 2005) This can be significant for MANETs or wireless sensor networks where according to da Silva, et al. (2005) "nodes are designed to be cheap and small, [so] they do not have enough hardware resources. Thus, the available memory may not be sufficient to create a detection log." Various schemes have been developed to avoid this problem, such as elected monitors for a cluster of nodes (Huang & Lee, 2003), a multi-layered collaborative approach to intrusion detection at both host and network level (Yongguang, Wenke, & Huang, 2003), or

through the use of multivariate statistical techniques to reduce the dimensionality of the data to be analyzed. (Chen, et al., 2007) The latter technique will be used in this research to limit the impact on a host's processing capabilities.

Another disadvantage of a host-based IDS is "operating system vulnerabilities can undermine the integrity of host-based agents and analyzers." (Bace, 2000) In other words, attacks that target the operating system upon which the host-based IDS resides, such as a rootkit attack, might subvert the operation of the IDS itself. Of course, network IDSs are susceptible to this as well. Mell, et al. (2003) describe several attacks:

1. Sending a large amount of non-attack traffic with volume exceeding the IDS's processing capability. With too much traffic to process, an IDS may drop packets and be unable to detect attacks.

2. Sending to the IDS non-attack packets that are specially crafted to trigger many signatures within the IDS, thereby overwhelming the IDS's human operator with false positives or crashing alert processing or display tools.

3. Sending to the IDS a large number of attack packets intended to distract the IDS's human operator while the attacker instigates a real attack hidden under the "smokescreen" created by the multitude of other attacks.

4. Sending to the IDS packets containing data that exploit a vulnerability within the IDS processing algorithms. Such attacks will only be successful if the IDS contains a known coding error that can be exploited by a clever attacker. Fortunately, very few IDSs have had known exploitable buffer overflows or other vulnerabilities.

One way to mitigate the risk to individual hosts is by keeping the signature-based operating system service pack updated to "patch" known code vulnerabilities so they are no longer susceptible to known attacks. Another more difficult method of avoiding this risk is to incorporate the IDS into the operating system software itself such as the latest

attempts by Microsoft in its Vista platform, or by designing an inherently more secure operating system, such as the multi-level security access structure proposed by Bell and LaPadula. (1973) Since PAIDS works in a multivariate feature space and the feature set is capable of change in near-real-time, it is less susceptible to deception than other host-based IDSs.

2.1.2 Network-Based

A network-based IDS is one in which "a distributed system will protect the network as a whole. In this architecture the IDS might control or monitor network firewalls, network routers or network switches as well as the client machines." (Kabiri & Ghorbani, 2005) A network IDS will generally record and parse network traffic data with monitoring agents, then either let these nodes process the data or report the data to a centrally located decision node.

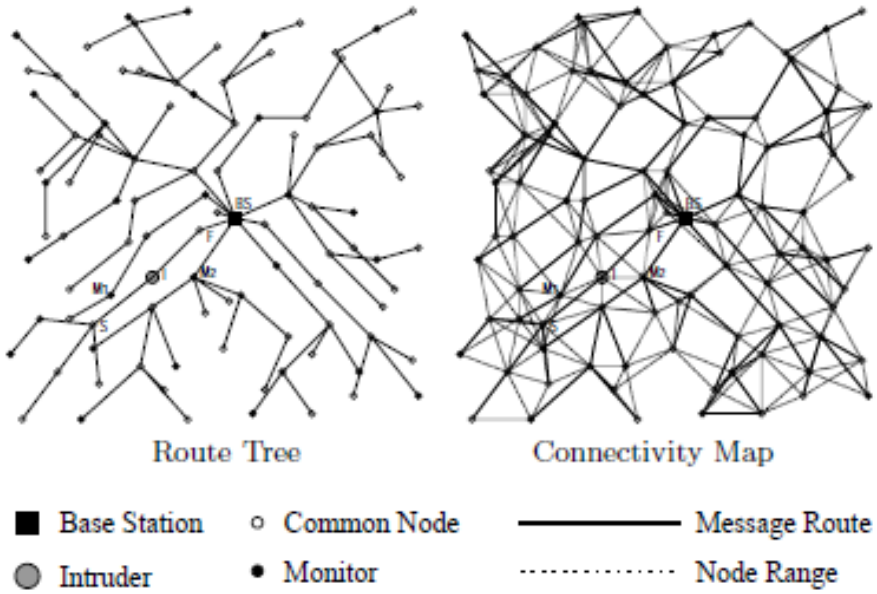


Figure 3 – Example Network Intrusion Monitor Locations (da Silva, et al., 2005)

A network-based IDS can reside either inside or outside a firewall, but as Northcutt states, “if your network-based IDS is located outside your firewall, you would never see an internal attack.” (p. 21) Depending on the level of security desired and the resources available, there will be multiple IDS monitors in a variety of locations (Figure 3) to ensure the maximum exposure while minimizing the distance between sensors and a possible intruder.

An IDS is defined by the measurements it makes and the rules it applies to these measurements. Though rules vary, there are standard intrusions which can be detected with a typical rule set such as those described in Table 4 for wireless networks.

Rule	***Description***	***Attack Detected***
Interval	Time between receipt of two consecutive messages is larger or smaller than allowed limits	• *Negligence* – Intruder does not send data from tampered node • *Exhaustion* – intruder increments sending rate to increase energy consumption of neighbors
Retransmission	Tampered node does not forward received message	• *Blackhole/Selective Forwarding* – Intruder suppresses some or all messages from retransmission
Integrity	Check to make sure message payload is the same along the entire network path	• *Modification* – Intruder aggregates or otherwise modifies contents of a received message
Delay	Retransmission does not occur before a defined timeout	• *Blackhole* • *Negligence*
Repetition	Same message can only be retransmitted only a limited number of times	• *Denial of Service* – Intruder attempts to monopolize node resource
Radio Transmission	Messages must originate from at most one hop away	• *Wormhole/Helloflood* – Intruder sends message to far located node to increase energy consumption
Jamming	Number of collisions must be lower than an expected value	• *Jamming* – Intruder introduces noise to disturb communications

Table 4 – Typical Network-based IDS Rules (da Silva, et al., 2005)

Of course, network-based IDSs also have disadvantages. "Network agents can monitor and detect network attacks (e.g. SYN flood and packet storm attacks). … [but] … Although some network-based systems can infer from network traffic what is happening on hosts, they cannot tell the outcome of commands executed on the host. This is an issue in detection, when distinguishing between user error and malfeasance." (Bace, 2000) Thus, a fully protected network will still need some form of host-based intrusion detection.

One of the major setbacks to a network IDS is the amount of network traffic generated by the monitoring agents. For instance, one of the earliest efforts at a network-based IDS, the Distributed Intrusion Detection System (DIDS), developed by Snapp, et al. (1991) had scalability issues…"the data flow between host monitors and the director agent may generate significantly high network traffic overheads." (Shyu, et al., 2007)

2.1.3 Signature-Based

Despite the best efforts of computer scientists, "any computer system or network has known vulnerabilities that an intruder can exploit. However, it is more efficient to detect intrusions that exploit these known vulnerabilities through the use of explicit expert system rules than through statistical anomaly detection." (Anderson, Frivold, & Valdes, 1995) The rules that detect previously identified exploitations are known as signatures. These signatures can be code strings, sequences of events, or other patterns indicative of malware. "Signatures are patterns corresponding to known attacks or misuses of systems." (Bace, 2000) Most anti-virus software uses a database of these signatures to detect malicious activity, and they require frequent updates since new

malware is created daily. If the exact signature does not exist in the database, then the IDS will not detect the malware.

There are "some intrusion experts [who] believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants." (Khan, Awad, & Thuraisingham, 2007) Though it is true that "script kiddies" create much of new malware by slightly altering old code in a vain attempt to avoid anti-virus software, real hackers are smart enough to alter their signatures significantly enough so that the only chance to defend against a zero-day attack will be through anomaly detection. Also, polymorphic code has progressed to a point that the required signature libraries to detect all known attacks have become too large to be useful on a system with limited memory.

2.1.4 Anomaly-Based

"In pursuit of a secure system, different measures of system behavior have been proposed, on the basis of an ad hoc presumption that normalcy and anomaly (illegitimacy) will be accurately manifested in the chosen set of system features." (Khan, Awad, & Thuraisingham, 2007) The objective of an anomaly-based IDS "is to define the normal behavior and consequently anomaly in the behavior of the system." (Kabiri & Ghorbani, 2005) This has typically been accomplished by attempting to build patterns of user activity, network traffic, or system parameters. The patterns are generally of normal activity, so any anomalies will be represented as intrusions, while patterns of abnormal conditions generally fall under the purview of signature-based detection.

The first to suggest an anomaly-based IDS was Dorothy Denning in *An Intrusion Detection Model* (1986) when she suggested several statistical methods for detecting anomalies in user behavior by monitoring a system's audit records, shown in Table 5. This "model is independent of any particular system, application environment, system vulnerability, or type of intrusion, thereby providing a framework for a general-purpose intrusion detection expert system." (Denning, 1986) The model did not go into specific applications, but merely suggested methods for implementation of future work.

Model	***Abnormality Indicators***
Operational	• New observation of random variable x exceeds fixed limits based on empirical data
Mean and Standard Deviation	• New observation of random variable x_{n+1} exceeds confidence interval that is d standard deviations away from mean of $x_1,\ldots, x_n$ • No prior knowledge is needed to establish normality and users [hosts] can have different means depending on usage behaviors
Multivariate	• Similar to Mean and Standard Deviation except that it is based on correlations among two or more metrics
Markov Process	• New observation is abnormal if its probability as determined by the previous state and the transition matrix is too low
Time Series	• New observation is abnormal if its probability as determined by order and inter-arrival time is too low

Table 5 – Anomaly-Based IDS Models (Denning, 1986)

"Statistical analysis finds deviations from normal patterns of behavior. This feature, common in research settings, is found in few commercial intrusion detection products." (Bace, 2000) This is probably because, "it is indeed very difficult to fix the threshold of alarm on a statistical variable. Too low, and the false alarm rate increases to unacceptable levels. Too high, and there is a risk of missing an alarm." (Debar, Becker, & Siboni, 1992) Also, "it is required to monitor the system within an intrusion free working environment for a while. … [and] … Since the user is a human being and

humans can be unpredictable, normal behavior modeling of the user can be a very difficult task." (Kabiri & Ghorbani, 2005) In fact, the inability to distinguish between legitimate and illegitimate activity is a significant argument against anomaly detection.

However, one of the first attempts at user behavior profiling, Next-Generation Intrusion Detection Expert System (NIDES) (Anderson, Frivold, & Valdes, 1995) was very successful, and many different useful approaches to statistical analysis have been put forward since then. "Statistical learning theory (SLT) provides a framework for the design of algorithms for classification, prediction, feature selection, clustering, sequential decision-making, novelty detection, trend analysis, and diagnosis. Its techniques are already being used in bioinformatics, information retrieval, spam filtering and intrusion detection." (Fox, Kiciman, & Patterson, 2004) It has been proposed that anomaly analysis is the only way to detect heretofore unknown attacks, and may allow detection of more complex attacks, such as those that occur over extended periods. (Bace, 2000)

Of course, anomaly-based IDSs can be prone to false alarms due to the difficulty of distinguishing between illegal and merely erroneous behavior, and an inability to deal well with changes in user activities. "Not every anomaly indicates an intrusion. This is especially true…where the system is very dynamic. … As a direct result of this uncertainty, anomaly based IDS will produce high [false positive] alarms." (Kabiri & Ghorbani, 2005) Some say it is also "relatively easy for an adversary to trick the detector into accepting attack activity as normal by gradually varying behavior over time." (Bace, 2000) This assumes that an attacker can determine what the IDS uses as an indicator and alter the attack to stay "under the radar." This research will propose that by monitoring

operating system process behavior and altering the key features to be monitored in near-real-time, the IDS can avoid spoofing or reverse engineering of this kind.

2.1.5 Specification-Based

A specification-based IDS attempts to merge the high detection rate of signature based detection with the ability to detect novel attacks of anomaly based detection. (Sekar, et al., 2002) The method basically consists of succinctly identifying the machine state then detecting undesired transitions either caused by a specific signature or anomalous condition. Hassan, Mahmoud, and El-Kassas successfully applied this technique to MANETs (2006) while Hussein and Zulkernine (2007) built the IDS into software with UML to protect individual components. This is still a relatively new area of study, so it may not offer an improvement over either of the other methods combined. In fact, (Kabiri & Ghorbani, 2005) report that, "Specification-based approach is only good when system specifications and details are known and applying limitations on the user is acceptable."

2.2 Mobile Ad-hoc Networks

"A mobile ad-hoc network [MANET] is formed by a group of mobile wireless nodes often without the assistance of fixed or existing network infrastructure…MANETs [are] much more vulnerable than wired (traditional) networking due to its limited physical security, volatile network topologies, power-constrained operations, intrinsic requirement of mutual trust among all nodes in underlying protocol design and lack of centralized monitoring and management point." (Huang, et al., 2003)

Intrusion detection in a MANET or other wireless network is exceedingly difficult due to changing topology (communication paths) and limitations on computing power. A network-based IDS must be flexible and robust, able to reconfigure itself depending on the nodes available and capable of incorporating information from multiple systems. A host-based IDS must be lightweight and able to work in near-real-time since memory and processing space are at a premium. PAIDS was designed with these limitations in mind, so every effort was made to reduce the dimensions of the collected feature set data to accommodate low memory and processing power.

2.3 Dimension Reduction

Intrusion detection systems (IDSs) deal with a "huge amount of data which contains irrelevant and redundant features causing slow training and testing process, higher resource consumption as well as poor detection rate." (Chen, et al., 2007) Thus, it is important to decrease the amount of data to be analyzed as much as possible, without losing the information needed to classify an intrusion. "Violating either of these constraints would either cause the IDS to run too slowly to detect intrusions in real-time, or would cause the network being protected to run at an unacceptably degraded level of performance." (Merkle, et al., 2002) Clearly, "feature selection/construction is the most challenging problem in building [an] IDS, regardless the development approach in use." (Lee, et al., 2000) Various multivariate statistical techniques have been proposed to solve this problem.

2.3.1 Principal Component Analysis

One method of simplifying the analysis is Principal Component Analysis (PCA), which identifies a few uncorrelated linear combinations of a select number of features from the full data set to explain the majority of variation. To calculate these components, the normalized eigenvectors of either the covariance or correlation matrix are used to rotate the original data until they are arranged along an axis of greatest variation. Generally, when the variables are measured in different units, the data are standardized and the correlation matrix is used to avoid problems of scale.

In Multivariate Analysis Methods and Applications, (1984, pp. 9-15) Dillon and Goldstein show how, given an *n* x *p* matrix **X** of *n* observations with *p* variables, the component scores are produced for the correlation matrix through the following steps. First, the centroid, μ', of each variable is calculated where $1'$ is a 1 x *n* unit row vector.

$$\boldsymbol{\mu}' = \frac{\mathbf{1}}{\mathbf{n}}\mathbf{1}'\mathbf{X} \tag{II-1}$$

Then, the *n* x *p* matrix, $\mathbf{X_d}$, centered (mean corrected) matrix can be obtained,

$$\mathbf{X_d} = \mathbf{X} - \mathbf{1}\boldsymbol{\mu}' \tag{II-2}$$

and the sample variance of each column of centered data can be calculated.

$$\mathbf{s^2} = \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{x_d^2} \tag{II-3}$$

Next, the variances are placed in a diagonal matrix, ***D***, and standardized data is computed

$$\mathbf{X_s} = \mathbf{X_d}\mathbf{D^{-1/2}} \tag{II-4}$$

Finally, the *n* x *p* matrix of component scores can be calculated by post multiplying the standardized data with a *p* x *p* matrix $\boldsymbol{\gamma}$ of normalized eigenvectors from the original data

$$\mathbf{Y} = \mathbf{X_s}\boldsymbol{\gamma} \tag{II-5}$$

These scores can then replace the original data, and in our case, be used to discriminate between normal and abnormal conditions. As mentioned previously, the data to be analyzed by an IDS can be extensive, so only a reduced number of features, or "principal" components, are kept based on a dimensionality assessment as described in section 2.3.3.

2.3.2 Factor Analysis

Factor analysis (FA) is similar to PCA, but concentrates on identifying commonalities between features to determine the essential dimensionality. It is generally used to assess "underlying relationships or dimensions in the data, and the replacement of original variables with fewer, new variables." (Wu & Zhang, 2006) FA can either be exploratory – to uncover hidden relationships, or confirmatory – to verify suspected relationships. This research will use exploratory FA to find characteristics which are most representative of a computer's reaction to an intrusion.

Dillon and Goldstein (1984, pp. 55-62) state, "while principal components analysis is best suited for deriving a small set of linear combinations of the original variables that accounts for most of the total variance, common factor-analytic techniques can better serve the functions of searching the data for qualitative and quantitative distinctions." Their basic structure for exploratory FA is shown in Figure 4.

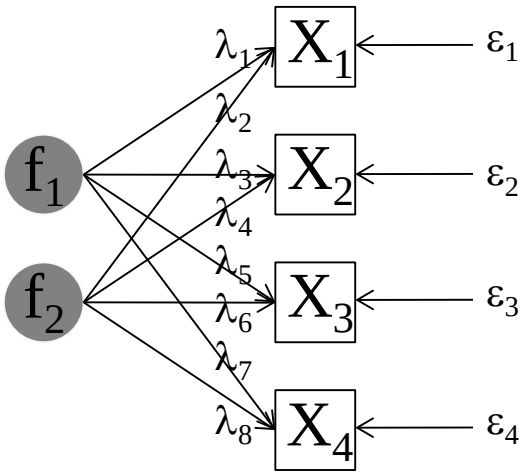


Figure 4 – Exploratory Factor Analysis diagram

Expressed as $$\mathbf{X} = \mathbf{\Lambda f} + \boldsymbol{\varepsilon} \tag{II-6}$$

where **X** *p*-dimensional vector of observed responses
f *q*-dimensional vector of unobservable common factors
ε *p*-dimensional vector of unobservable unique factors
Λ *p* x *q* matrix of unknown constants called factor loadings

There are *p* unique factors and it is generally assumed that the unique parts **Ψ** of each variable are uncorrelated with each other or with their common parts **Φ**; that is

$$\mathrm{E}(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}') = \mathrm{Cov}(\boldsymbol{\varepsilon}) = \mathbf{\Psi} = \begin{pmatrix} \Psi_1 & 0 & \dots & 0 \\ 0 & \Psi_2 & & \\ \vdots & & \ddots & 0 \\ 0 & & 0 & \Psi_p \end{pmatrix} \tag{II-7}$$

And

$$\mathrm{Cov}(\boldsymbol{\epsilon}, \mathbf{f}') = \mathbf{0} \tag{II-8}$$

Which implies the covariance matrix of the response vector **X**, can be expressed as

$$\mathrm{Cov}(\mathbf{X}) = \Sigma_{xx} = \mathbf{\Lambda \Phi \Lambda^2} + \mathbf{\Psi} \tag{II-9}$$

where the covariances (correlations) between the common factors are scaled to be

$$\mathrm{Cov}(\mathbf{f}) = \mathbf{\Phi} = \begin{pmatrix} 1 & & & \\ \phi_{21} & 1 & & \\ \vdots & & \ddots & \\ \phi_{q1} & \dots & \phi_{q,q-1} & 1 \end{pmatrix} \tag{II-10}$$

If the factors are uncorrelated, then $\boldsymbol{\Phi} = \mathbf{I}$ and (2-3) becomes

$$\Sigma_{xx} = \boldsymbol{\Lambda}\boldsymbol{\Lambda}^{\mathbf{2}} + \boldsymbol{\Psi} \qquad \text{(II-11)}$$

Dillon and Goldstein (1984, p. 62) continue by stating,

> The total number of parameters in need of estimation is the number of factor loadings, namely *pq*. There are $\frac{\mathbf{1}}{\mathbf{2}}\mathrm{p}(\mathrm{p}+1)$ separate variances and covariances in Σ_{xx}. ... Generally, the requirement for identification is that the number of parameters be less than the number of equations, so that $pq + p < \frac{\mathbf{1}}{\mathbf{2}}\mathrm{p}(\mathrm{p}+1)$ or $q < \frac{\mathbf{1}}{\mathbf{2}}(\mathrm{p}-1)$. Thus, q should be fairly small compared to p. Unfortunately, this does not guarantee that a solution will exist.
>
> It is important to note that in the case of exploratory factor analysis, if q > 1 and a solution exists, it is not generally unique. Using (II-11) we see that any orthogonal rotation of the factors in the relevant *q*-space will give a new set of factors which will also satisfy the conditions of equation (II-11).

This is important because no matter how we rotate the factor loadings to attempt discrimination, they still describe the same key variables to be monitored with the IDS.

2.3.3 Dimensionality Assessment

"The size of the feature space is obviously very large. Once the dimensions of the feature space are multiplied by the number of samples in the feature space, the result will surely present a very large number. This is why some researchers either select a small sampling time window or reduce the dimensionality of the feature space." (Kabiri & Ghorbani, 2005) The object of both PCA and FA is to reduce the dimensionality of the scores as far as possible while retaining a majority of variation from the original data.

There are numerous heuristics to accomplish this; one of the most widely used is Kaiser's Criterion in which all components associated with eigenvalues less than one are

discarded. This ensures that each retained component will have a variance greater than any single variable. Another method, commonly associated with FA, is to add factors until the average commonality is higher than a certain percentage. This research actually uses a new heuristic, keeping only those variables that load highly on the first principal component, then using eigenvectors that contain 80% of the variance to discriminate.

There are also various graphical methods to eliminate components, such as the maximum secant distance proposed by Robert Johnson (2008) or Cattell's scree test, which is named after the rubble that falls to the bottom of a cliff. In the scree test, eigenvalues are plotted and all components or factors which fall below the "scree line" are disregarded. For instance, in Figure 5, the first two or three components would be retained, and the remaining components would be dropped.

The dividing line between the retained and dropped components is not always obvious, and often more components are kept than are necessary. Johnson proposed an automatic determination of the "break point" by using the maximum Euclidean distance of the plotted eigenvalues from the log scale secant line, as shown in Figure 6.

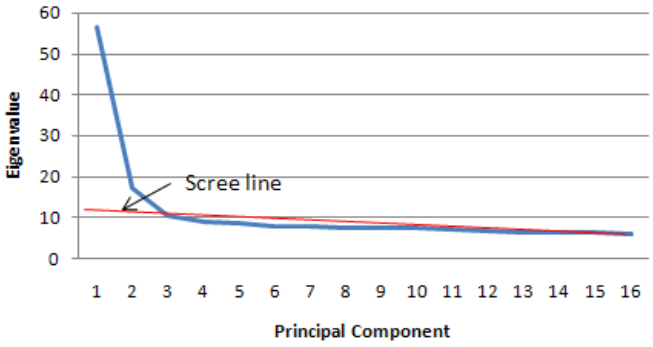


Figure 5 – Cattell's Scree Test

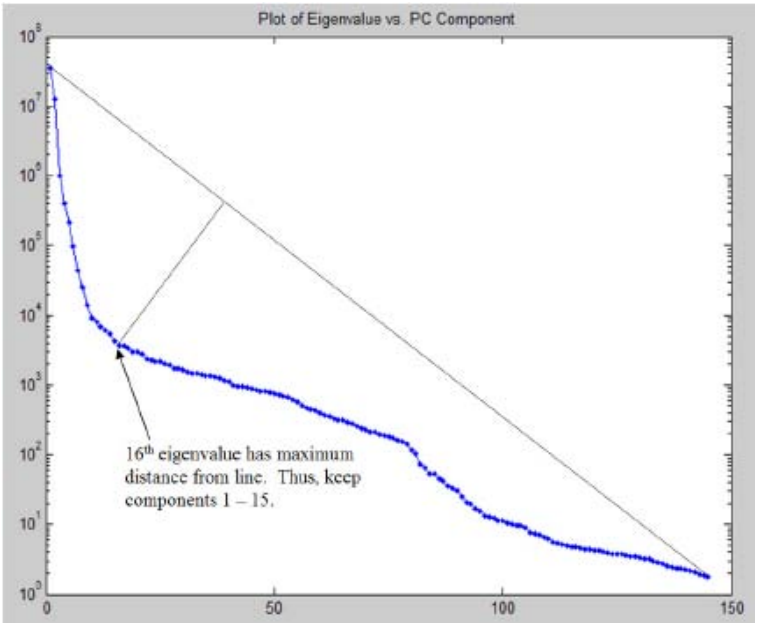


Figure 6 – Max Euclidean Distance of Eigenvalue Curve from Secant Line on Log Scale (Johnson, 2008)

Whatever heuristic is used, the dimensions of the original data needed to discriminate are reduced from *p* components to a smaller number *k* such that there is "almost as much information in the k components as there is in the original p variables." (Chen, et al., 2007) It is proposed that a lightweight IDS can be created that only analyzes the reduced data set, but still provides sufficient discrimination between normal operations and an intrusion.

2.4 Anomaly Classification

2.4.1 Discriminant Analysis

Discriminant analysis (DA) uses linear or non-linear functions to calculate a Mahalanobis distance between multiple groups, thereby creating boundaries in a multivariate space. In two dimensions, "Mahalanobis distances are calculated in units of standard deviation from the group mean. Therefore, the calculated circumscribing ellipse

formed around the training data actually defines the one standard deviation boundary of that group." (Wu & Zhang, 2006)

This technique has been used by Wong and Lai, however they "use DA to identify the important features from the training dataset and then to validate the obtained feature set with [support vector machine] (SVM)." (2006) So, instead of using DA to classify their data once important features have been identified, they use it to choose the salient features from the original variables. They also mention that "DA is rarely applied to anomaly-based network intrusion detection," (Wong & Lai, 2006) but this could be attributed to a cross-discipline misunderstanding of the most appropriate employment of this technique. This research asserts that PCA and FA are more effective methods of determining feature sets, while DA is more efficient at identifying "self" and "non-self."

Wu and Zhang also use FA to classify different types of attacks into clusters. This "clustering scheme which classifies attacks based on their factor scores' 'abnormality'" (Wu & Zhang, 2006) assumes that a particular type of attack will score highly on the same factors. The same scheme is used to some extent in this research; however, PCA is used to highlight features, and a form of quadratic discrimination is used to discriminate between normal and abnormal states.

2.4.2 Quadratic Discrimination and Mahalanobis Distance

Wu and Zhang (2006) used FA with quadratic discrimination to find anomalies by measuring the Mahalanobis distance of outliers from normal operating levels. "Generally a test sample is considered as an anomaly if it has abnormal values on one or multiple factors." A sample population with both abnormal and normal data is needed to train the

classifier. Each sample point is compared to the mean of the populations transformed by the inverse of either the pooled or individual covariance matrices of the samples. Quadratic discrimination scores are calculated by

$$\hat{d}_i^Q = -\frac{1}{2}\ln|S_i| - \frac{1}{2}(\tilde{X}_0 - \tilde{\bar{X}}_i)'S_i^{-1}(\tilde{X}_0 - \tilde{\bar{X}}_i) + \ln P_i \qquad \text{(II-12)}$$

Prior probabilities P_i of each population are beneficial but not necessary, and if used, the pooled covariance matrix S_p is calculated by

$$S_p = \frac{1}{N_1+N_2-2}(\mathbf{X}_{d1}'\mathbf{X}_{d1} + \mathbf{X}_{d2}'\mathbf{X}_{d2}) \qquad \text{(II-13)}$$

Where $\boldsymbol{X}_d$ is the sample data centered about its mean as calculated in Equation (II-2).

The calculated score is then used to classify each data point based on which population (normal or abnormal) receives the highest score. Quadratic discrimination allows for nonlinear functions and can get quite complicated in higher dimensions. However, the concept is quite direct; a linear combination of the original data or representative functions is used to find the most likely population to which a given sample belongs. The simple two-dimensional example of the probability of belonging to one of two populations in Figure 7 shows how this is accomplished.

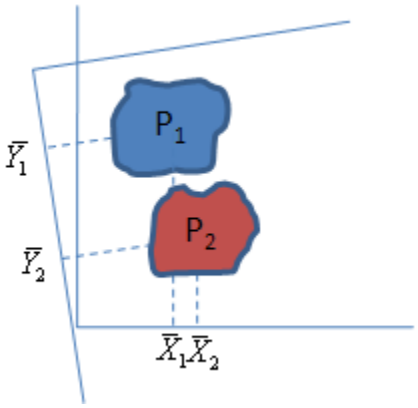


Figure 7 – Example Mahalanobis Distance

The difference between the populations on the original X axis is very small; however, when the data is transformed onto a Y axis of linear combinations of the original data, there is a marked distinction. In this case, it is possible to classify the two populations with a straight line along the midpoint of the two centroids, which is Fisher's two-group discrimination. Linear Discriminant Analysis (LDA) developed by Sir Ronald A. Fisher, in *The Use of Multiple Measurements in Taxonomic Problems* (1936) finds a "linear function of … measurements [to] maximize the ratio of the difference between the specific means to the standard deviation within (groups)." LDA is accomplished using the squared distance between the means of discriminate scores and the pooled covariance of the two populations to maximize the distance *d* between them.

$$d = \frac{\left(\mathbf{b}'\bar{\mathbf{X}}_1 - \mathbf{b}'\bar{\mathbf{X}}_2\right)^2}{\mathbf{b}'S\mathbf{b}} \tag{II-14}$$

This quantity is maximized when

$$\mathbf{b} = S^{-1}(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2) \tag{II-15}$$

Then b is used to transform the original data such that

$$\bar{\mathbf{Y}}_i = b'\bar{\mathbf{X}}_i \tag{II-16}$$

This results in a generalized Mahalanobis Distance (MD) (Mahalanobis, 1936) that takes into account the correlation between two populations

$$\bar{\mathbf{Y}}_1 - \bar{\mathbf{Y}}_2 = [S^{-1}(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2)]'(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2) \tag{II-17}$$

$$= (\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2)'S^{-1}(\bar{\mathbf{X}}_1 - \bar{\mathbf{X}}_2) \tag{II-18}$$

"Basically, MD is a measure of distance between two points in the space defined by two or more correlated variables." (Wong & Lai, 2006) MD makes the assumption that the data are nearly normally distributed and have the same covariance matrix, *S*.

Like most regression, this technique generally requires more observations than there are variables to ensure full rank in the matrix. If these conditions do not exist, the data can be manipulated, adding noise for instance, to generate a more "well behaved" dataset. This research will use MD between Principal Component scores calculated from a reduced set of system process data to distinguish between normal and abnormal activity.

The distance is used to group new data into the "closest" population, and the standard measurement of effectiveness is a "confusion matrix" built from known data. This matrix records whether data points are classified correctly by the discriminator, and can be used to calculate an apparent error rate (APER) which is slightly lower than the actual error rate (AER). For two-way discrimination, this looks like Figure 8.

		Predicted Membership		
		π_1	π_2	
Actual Membership	π_1	N_{1C}	N_{1I}	n_1
	π_2	N_{2I}	N_{2C}	n_2

Figure 8 - Confusion Matrix

where n_i total number of data points in population *i*
N_{iC} number of data points correctly classified in population *i*
N_{iI} number of data points incorrectly classified in population *i*

and APER is calculated with

$$\text{APER} = \frac{(N_{1I}+N_{2I})}{n_1+n_2} \tag{II-19}$$

2.4.3 Support Vector Machines

"The Support Vector Machine (SVM) is one of the most successful classification algorithms in the data mining area, but its *long training time limits its use."* (Khan, Awad, & Thuraisingham, 2007) (emphasis in the original) "The basic idea in SVM is to

transform the training data into a higher dimensional space and find the optimal hyperplane in the space that maximizes the margin between classes." (Wong & Lai, 2006) The simplest SVM model uses the maximal distance between margins as a kernel function, which makes it very similar to quadratic discrimination; however, any appropriate distance function can be utilized. For instance, Wong and Lai (2006) used a Gaussian radial basis function as the kernel

$$d = \mathrm{e}^{-\gamma \|\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2\|^2} \qquad \text{(II-20)}$$

Where the optimal value for parameter γ was determined with empirical test data.

Khan, Awad, and Thuraisingham attempted to improve on standard SVM by clustering large data sets. The idea is as follows: SVM computes the maximal margin separating data points; hence, only those patterns closest to the margin can affect the computations of that margin, while other points can be discarded without affecting the result. Those points lying close to the margin are called support vectors, which are then used to improve the classification limits. By using only the clusters of data points closest to the margins, the calculations needed for SVM can be dramatically reduced, thus decreasing required training time. A graphical example of this is shown in Figure 9.

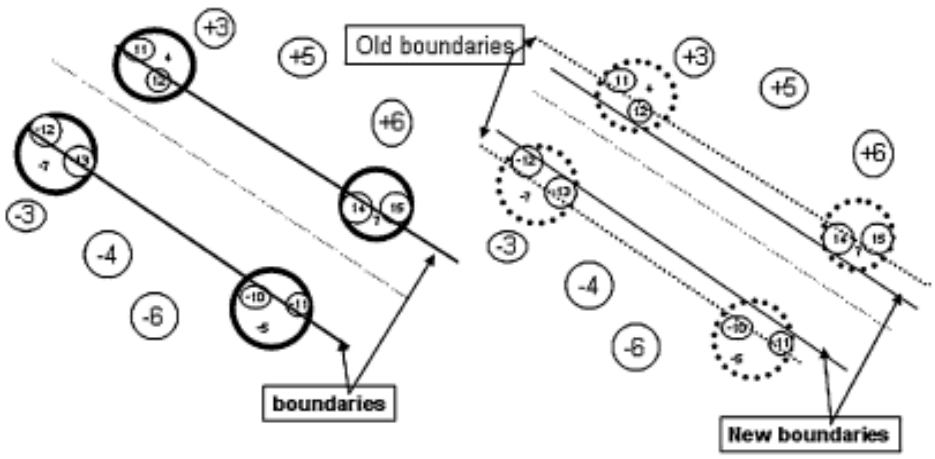


Figure 9 – Support Vector Machine Classifier (Khan, Awad, & Thuraisingham, 2007)

2.4.4 Decision Trees

Decision trees are another well-known procedure for classification. Chen, et al. used a C4.5 decision tree algorithm as a classifier between normal and abnormal conditions, "The algorithm uses a splitting criterion based on the Information Gain Ratio. The idea is to partition the training set in such a way that the information needed to classify a given example is reduced as much as possible." (2007) A decision (or classification) tree works as follows:

> Each internal node in the tree is labeled with a relational expression that compares a numeric attribute/feature of the object being classified to a constant splitting value. Each leaf is labeled to indicate whether it represents a positive or negative instance of the class of interest (e.g., failed execution). An object is classified by traversing the tree from the root to a leaf. At each step of the traversal prior to reaching a leaf, the expression at the current node is evaluated … A classification tree is constructed algorithmically using a training set containing positive and negative instances of the class of interest. (Francis, et al., 2004)

Although decision trees can be efficient, they are most useful for clustering attacks into distinct types (denial of service, probes, buffer overflows, etc.) to elicit an appropriate response. Also, they can be computationally demanding if partitions are not

effectively established. Since this research is only concerned with proving the concept of identifying an intrusion using process data and not on identifying the type of attack, the computationally simple and efficient Mahalanobis distance between populations will be employed to classify intrusions instead.

2.4.5 Genetic Algorithms

A genetic algorithm (GA) has been used to improve SVM so that it "is not only able to select [an] 'optimal feature set' but also is able to figure out 'optimal parameters' for [the] SVM classifier." (Kim, Nguyen, & Park, 2005) GA builds a database of rules or signatures based on their fitness in classifying training data, which is subsequently used by the IDS. "A GA is essentially a type of search algorithm which is used to solve a wide variety of problems. The goal of a GA is to create optimal solutions to problems. Potential solutions are encoded as a sequence of bits, characters or numbers. This unit of encoding is called a gene, and the encoding sequence is known as a chromosome. The GA begins with a set of these chromosomes and an evaluation function that measures the fitness of each chromosome. It uses reproduction, such as crossover and mutation to create new solutions, which are then evaluated." (Pillai, Eloff, & Venter, 2004) Although GA is effective, it is computationally expensive, and is not feasible for a lightweight IDS.

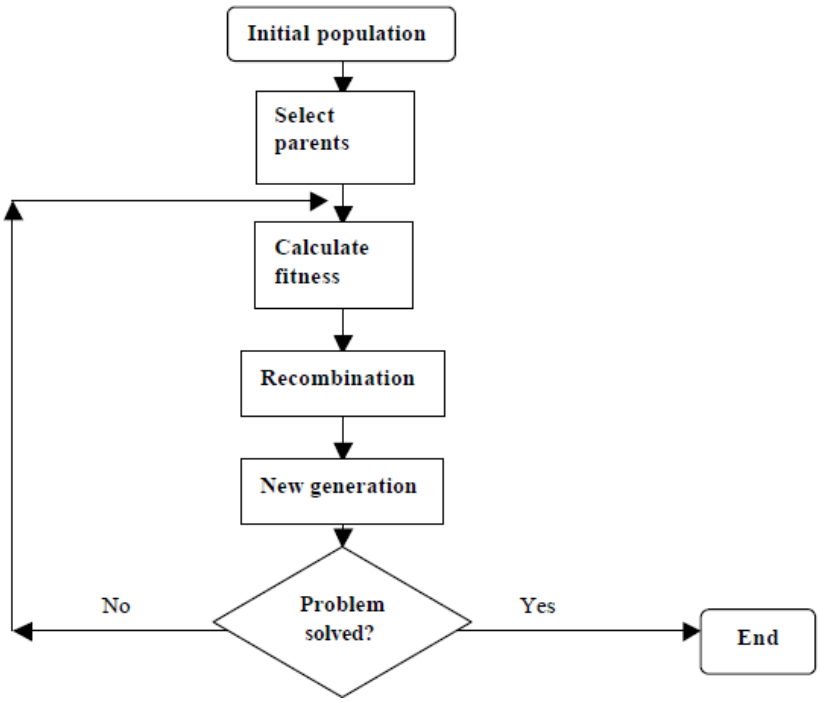


Figure 10 – Basic Iteration of a Genetic Algorithm (Pillai, et al., 2004)

2.4.6 Neural Networks

Another computationally expensive method for classification is a neural network. A neural network transforms an input vector into a weighted output vector through the use of hidden interconnected nodes. For an IDS, the input vector would be a particular feature set, and the output vector would be a classification of "normal" or "abnormal". As usual, there are advantages and disadvantages to this system. On the plus side, "Data in the audit trail may be incomplete, a field is sometimes missing, or the accuracy of the measure is low. ... An intruder may try to alter the audit records to hide its illegal activity. However, the neural network will be able to cope with this kind of problem." (Debar, Becker, & Siboni, 1992) On the other hand, Balducelli, et al. warn, a "neural encoder needs to be trained (for many hours) with data collected during more days and weeks." (2007) This is not practical for a system with low processing power. "Also, since

recurrent neural networks are retroactive systems, unstable configurations appear," (Debar, Becker, & Siboni, 1992) and, it does not allow for updates in real-time.

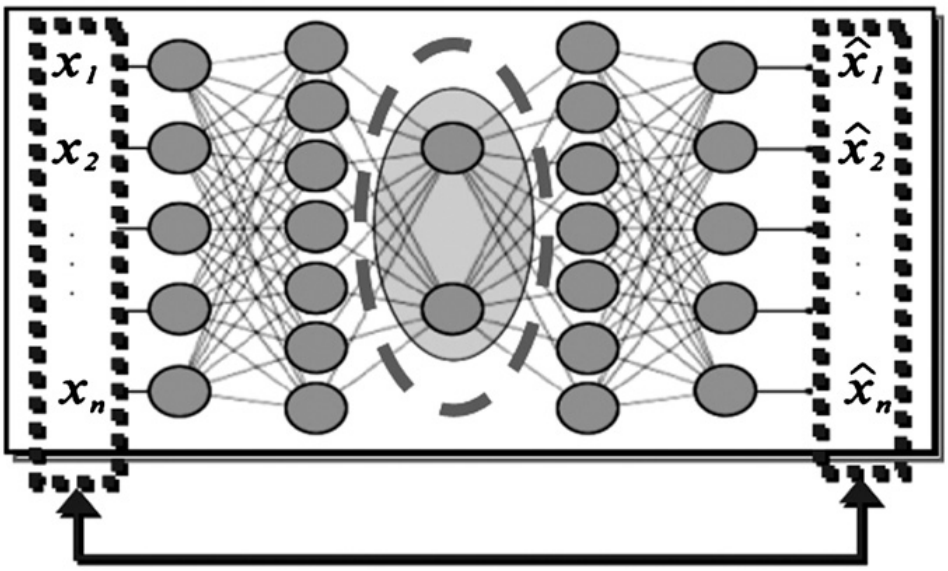


Figure 11 – Neural Network (Balducelli, et al.)

2.4.7 Immune System Algorithms

A different approach to anomaly detection through identification of "self" is to simulate an immune system by generating a string of detectors that do not match any of the protected data, then "monitor the protected data by comparing them with the detectors. If a detector is ever activated, a change is known to have occurred." (Forrest, et al., 1994) While this may be useful in protecting static data, it is not applicable to detecting intrusion in a highly variable environment, or in detecting attacks that simply steal or redirect data without altering it.

Another approach to using the immune system as a model is "based essentially on mathematical models extracted as an abstract of general principles of information processing by natural immune systems." (Tarakanov, 2008) This method uses training

data to map “antibodies” (rules or signatures) that are then used to classify other “molecules” as shown in Figure 12. In this case, normal conditions can be recognized, and anything other than that is classified as abnormal.

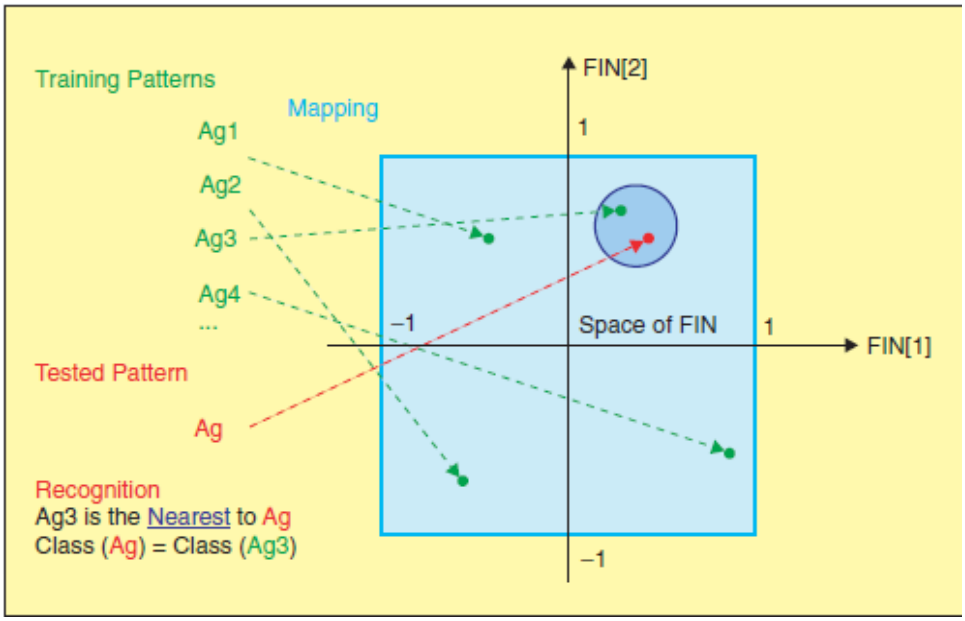


Figure 12 – Artificial Immune Intrusion Detection System (Tarakanov, 2008)

2.5 Data Generation

2.5.1 Repeatable, Sanitized, and Realistic

Data generation is a major problem in IDS development, specifically “the need for a properly labeled set of normal training data from which to construct standard statistical measures or train advanced systems. Statistical analysis requires such a set to ascertain what constitutes normal activity and classification based systems that can be trained require the set for their training.” (Brugger, et al., 2001) Some problems may arise when training and testing an anomaly detection IDS due to the difficulty in ensuring repeatable, sanitized, realistic traffic. (Mell, et al., 2003)

The importance of repeatable data is a basic tenet of scientific experimentation, however the other two characteristics warrant some explanation. Sanitization ensures there are no unknown viruses in the data to be analyzed since a "problem with real background data is that it may contain attacks about which we know nothing," (Mell, et al., 2003) which can cause problems in assessing "false" detections. Realistic data is important to ensure the IDS is not simply successful for a unique, simplified situation, but is universally applicable. "This problem hasn't been well addressed by the existing literature. The accepted practice seems to be to have a few weeks of training data that is thoroughly analyzed and labeled by a human analyst. Many of the research projects to date have used the data provided as part of the 1998 DARPA intrusion detection system shootout …While such data sets are essential for comparing different IDS, they are highly labor intensive to produce and may not be totally accurate." (Brugger, et al., 2001) Another common frame of reference is the KDD 1999 Cup data (Chen, et al., 2007) (Kabiri & Ghorbani, 2005) (Kim, Nguyen, & Park, 2005) (Wong & Lai, 2006) but these datasets, though they have faults and benefits, only provide TCP dumps and other characteristics of network traffic with no information about the hosts under attack, so they are not usable for this research.

"Many evaluations test IDSs using no background traffic as a reference condition. In such experiments, an IDS is set up on a host or network on which there is no activity. Then, computer attacks are launched on this host or network to determine whether or not the IDS can detect the attacks. This technique can determine an IDS hit rate but can say nothing about false positives." (Mell, et al., 2003) This is the approach used for this

research, using a crossover cable connector between two laptops to simulate an Internet connection. In this way, the attack can be controllable and repeatable, even if realism is somewhat sacrificed.

2.5.2 Trustworthiness

Another major problem with the implementation of any security measure is the uncertain "guarantee that the data to be protected are uncorrupted at the time that the detectors are generated." (Forrest, et al., 1994) Like sanitized test data, it is vital that users trust their system is uninfected before they apply an IDS. This research used stand-alone computers on a closed network to run experiments and data analysis to avoid the possibility of inadvertent contamination. However, some of the software used was downloaded from open Internet sources, so it is possible additional unwanted code was included. To mitigate this threat, older, more benign versions were collected from an established source to limit potential exposure.

III. Methodology

3.1 Problem Definition

3.1.1 Assumptions and Hypotheses

Most intrusion detection systems attempt to stop attacks before they happen by parsing network traffic data and applying code signatures or behavioral rule sets to catch malware in the delivery process. However, if an attack has an unknown pattern it is likely to bypass most of the existing IDS software. Thus, if one is to detect new or polymorphic malware, one must be able to distinguish between normal and abnormal activity on the attacked system rather than rely on signature matching.

Many solutions to this problem have been proposed, which have been discussed in the previous section, but none of them rely exclusively on operating system parameters to discriminate between normal and abnormal. It is possible this is because many modern attacks alter the operating system itself, or because once an operating system shows signs of an attack the damage may already be done. It is the premise of this research that any attack will have a measurable effect on the operating system in at least one of the proposed 18 dimensions, despite attempts by a hacker to deceive the victim, and that notification of an attack in progress is better than none at all.

Process Anomaly IDS (PAIDS) is intended as a last chance discovery of malware at the host level that has managed to avoid detection by a robust multi-layered security system. PAIDS detects backdoor intrusion based solely on operating system performance and can adapt to changing conditions. This is a valuable tool since backdoors are notoriously difficult to detect once they are resident and active on computer systems.

3.1.2 Properties of PAIDS

"Computer networks have a dynamic nature in a sense that information and data within them are continuously changing. Therefore, detecting an intrusion accurately and promptly, the system has to operate in real time…not just to perform the detection in real time, but to adapt to the new dynamics in the network." (Kabiri & Ghorbani, 2005) Establishing a "normal" baseline at startup is not sufficient to detect intrusions, because once legitimate activity occurs on the system, the "normal" conditions change. This poses problems for an anomaly detection system, because identifying rare legitimate activity as abnormal can lead to false positives, and incorporating illegitimate activity into a normal model can lead to false negatives.

For instance, in behavioral pattern matching, if a user only accesses a program once a month, this activity may be considered abnormal and identified as an intrusion by an IDS. Difficulty incorporating anomalous legitimate behavior into a "normal" baseline model has always been the bane of anomaly detection systems. Another problem with anomaly detection is a hacker can make his/her intrusion appear normal by testing the limits of the IDS and attacking at the "boundary" layer to either push the "normal" baseline past a vulnerable point, or determine the applicable feature set and avoid it.

PAIDS avoids the first problem by periodically analyzing data samples, and reestablishing a baseline of normality based on existing conditions of legitimate activity. Also, since it uses a linear combination of 18 different characteristics of the operating processes, it is less vulnerable to false positives generated by rare legitimate activity. To combat the second problem, the use of Principal Component Analysis (PCA) helps

PAIDS avoid deception and reverse engineering by hackers, because classification happens in an abstract "feature space" indiscernible to an outside observer. Finally, PAIDS alters the key variables it uses based on the factor loading scores from PCA each time it resets the baseline, so it is nearly impossible to determine the "boundary" used for classification. This also helps PAIDS learn what is normal and which processes are most important at the time to discriminate between normal and abnormal activity, so the algorithm evolves with changing conditions.

3.2 Tools

3.2.1 MATLAB®

A powerful piece of numerical processing software developed by The Mathworks Inc., MATLAB has been the "language of technical computing" for decades. It was designed to perform analytical operations on large matrices and it is capable of compiling user defined programs as well as offering a wide array of pre-defined functions. This research used version 7.6.0.324 (R2008a) for pre-processing data on the laptop, and for some of the post-processing as well. Most of the post-processing was performed using version 7.4.0.287 (R2007a) on the AFIT LAN. Programs written in MATLAB code for this research can be easily compiled into other languages to improve performance in an actual IDS.

3.2.2 TaskInfo

TaskInfo is shareware developed by Igor M. Arsenin to "combine and improve features of Task Manager and System Information tools. It visually monitors (in text and graphical forms) different types of system information in any Windows system in real

time.” (Arsenin, 2008) This research used version 7.2.0, though a more recent version was released on 10 Nov 08. Some of its professed capabilities are:

> ▪ List of all running processes and threads (including system threads), with detailed information about each process: CPU and memory usage, path, all opened files and modules (DLLs), command line, environment variables, opened connections and more
> ▪ List most of the processes that want to be invisible like worms, keyloggers and other spy software
> ▪ Total CPU(s), memory (physical, virtual, cache and swap) usage
> ▪ Detailed information about installed CPU(s) and operating system
> ▪ Data rates on local disks, network server/client, Dial-Up I/O
> ▪ All opened files, drivers and TCP/IP, VPN connections with details
> (Arsenin, 2008)

The first capability, detailed process information, was the most useful to this research, since the primary purpose of using TaskInfo was to record as much raw data on running processes as possible. An example of the visual output from TaskInfo can be found in Appendix C – TaskInfo Screenshot. If it had simply been a GUI, the data would be worthless for processing, but fortunately TaskInfo also allows you to:

> ▪ Copy all information to clipboard or text file
> ▪ Run/stop processes and shutdown/restart the system
> ▪ Use it from command line
> ▪ Automatically show different low-resource alerts
> ▪ "Free" physical memory on demand (Arsenin, 2008)

The first capability, copying information to a text file, provided a useable output and the third capability, command line use, enabled convenient access to TaskInfo through a batch file designed for data collection. An example of the output file, converted to an Excel format, appears in Appendix A – Output Data from TaskInfo in Excel Format.

3.2.3 Sub7

Also known as SubSeven, this is one of the best known, most widely distributed backdoor programs on the Internet. “It is mainly used for causing mischief, such as

hiding the computer cursor, changing system settings or loading up pornographic websites. However, it can also be used for more serious criminal applications, such as stealing credit card details [or passwords] with a keystroke logger." (Wikipedia, 2008) Other nefarious features include the ability to upload or download files, alter system data, or completely destroy the hard drive of the infected computer.

A backdoor allows an unauthorized user entry to a system by circumventing legitimate access controls. Generally, a backdoor must either be installed prior to initial use of the system, or a legitimate user must be tricked into installing it through the use of a Trojan. A Trojan is a malware executable disguised as something the user may be interested in or curious about, and is the classic method of socially engineering the spread of viruses, worms, backdoors and other malware. The Sub7 Trojan establishes a server on the victim computer that subsequently notifies the intruder when and where it has been activated through the use of ICQ, email, or other instant messaging service. The hacker then uses a client program to contact the server, thus providing unauthorized access to the victim computer.

Sub7 has been around since the mid-1990s, so most anti-virus software can now detect its existence, though new variations are always being developed and deployed. It is very user friendly, with a straight forward GUI for a multitude of malicious activities, shown in Appendix B – SubSeven Command Screens. This research used version 1.5, developed in 1999 by a hacker named mobman, which was downloaded from www.hackpr.net/~sub7/downloads.shtml, a mirror for www.sub7.net. While there is no

guarantee that the code was pristine since it came from the "wild", it is a very old version and less likely to contain other hidden code since it was archived for historical purposes.

3.3 Experimental Design

3.3.1 Factors

To design an experiment properly, the factors to be studied must be identified. Obviously, the goal of developing an IDS is to identify an intrusion, so the primary variable is whether the system has malware functioning on it or not. To distinguish the effects of malware from the effects of legitimate activity, one must compare these factors and their interactions, so the next variable is the level of legitimate activity on the system at any given time. For this research, three levels of activity were used:

- **low** – no programs active except those needed to collect data
- **medium** – one or two programs running (i.e. Word and PowerPoint)
- **high** – multiple programs running requiring large amounts of memory (i.e. Word, PowerPoint, Excel, MATLAB)

An additional factor of active or inactive connection to the Internet was considered; however, for security purposes, this option was not implemented. Also, to ensure statistically significant results were obtained, multiple replications of the experiment at each factor level were required.

3.3.2 Test Runs

Four formal tests were run to collect data in order to develop and test the algorithms used in PAIDS. The first experimental runs simply recorded data under three

conditions to determine if discrimination was even possible: low activity without malware present, medium activity without malware present, and low activity with malware present. These data sets were collected with no consideration of log-on session or warm-up times. Length of collection varied from 500 seconds to 2000 seconds, and samples were at two second intervals due to limitations in code speed.

Once the concept was proven, DesignExpert® Version 7.1.5 was used to build a full factorial experiment with five replications at two levels of malware (present or absent) and three levels of activity (low, medium, high) which produced the test plan shown in Appendix J – Nov21 Test Plan. This plan was used to ensure all possible interactions between the factors were studied without inducing unintended interactions due to the order of runs. These data sets were collected under a single log on session without resetting either system except to delete the malware from the victim computer between runs. Each replication was 1000 seconds long with samples taken at one second intervals.

Additional runs were collected afterwards which considerably simplified the interactions, with data containing (activity/malware):

- **Run 1** – ½ low / no ; ½ low / yes
- **Run 2** – ½ low / no ; ½ high / no
- **Run 3** – ⅓ low / no ; ⅓ high / no ; ⅓ high / yes

Each of these runs was collected on a separate log-on session by recycling power on the victim computer with a five minute warm-up period after start-up. These last runs lasted 2000 seconds with a one second sample rate, and were used for the majority of experimentation.

3.3.3 Data Collection

The basic methodology remained consistent throughout this experimentation; output was produced with TaskInfo for a certain period of time without intrusion or legitimate activity, then malware was introduced into the system, legitimate activity was started, or both depending on the requirements of the run. Times were recorded throughout the experiments so the resulting datasets could be divided into periods of intrusion and levels of legitimate activity.

Data was collected by calling TaskInfo with a simple batch file

```
cd "c:\Program Files\Iarsn\TaskInfo 8.x\"
TaskInfo pl "c:\Thesis\data\1.txt"
TaskInfo pl "c:\Thesis\data\2.txt"
…
TaskInfo pl "c:\Thesis\data\2000.txt"
```

Although there is certainly a more elegant way to record data into successively named and time-stamped output files, this worked well enough for the research. The data was later read by an import function written specifically for the output file format.

The import data function (Appendix C – Import Data) initially required MATLAB to be running, which limited data collection to once every two seconds due to operation of the code. However, this severely affected the quality of the output, because MATLAB uses a huge amount of memory and data is not collected fast enough. By using the batch file to call TaskInfo, data could be collected once per second and did not require MATLAB to be running during data collection. This not only streamlined the process, but limited the effect of monitoring software on the operating system. Any actual application of PAIDS will require much more efficient data collection, described further in Conclusions and Future Research.

3.4 Implementation

3.4.1 Hardware Environment

One of the difficulties in performing cyberwarfare experimentation is harvesting malware and analyzing it without adversely affecting your own systems. In this way it is akin to biological warfare, which requires sequestered clean rooms with stringent protocols in handling viral and bacteriological weapons. This necessity is summarized well by the European Institute for Computer Antivirus Research (EICAR), "Using real viruses for testing in the real world is rather like setting fire to the dustbin in your office to see whether the smoke detector is working." (EICAR, 2006) The Laboratory for Information System Security/Assurance Research and Development (LISSARD) at AFIT's Center for Cyberspace Research provides this environment, where researchers can download and test malware and anti-virus programs alike.

One possibility for testing the effects of malware on systems is to set up virtual machines using VMware or simulate networks with OpNet, QualNet, ns2, LARIAT, or a similar platform. This allows the experimenter to employ the same attacks on the same system repeatedly without danger of affecting a real operating system. However, these simulators are unable to provide the granularity of host-level responses this research was interested in, namely operating characteristics such as CPU and memory usage, number of handles, threads, and windows open, and read/write operations. In order to collect this type of information, real systems were required.

To further protect the LAN and provide a more sterile environment for experimentation, this research used standalone laptops. As a proof of concept, this

research used two laptops which were Intel® Core™ 2 Duo CPU, T7300 @ 2.00GHz with 1.99GB RAM running Microsoft Windows XP Professional 2002 with Service Pack 2. One laptop was designated the attacker and one the victim, and they were linked together with a generic category 5E crossover cable to simulate a network connection. To accomplish this, the IP address of the defender was established as 10.1.1.1 and the attacker as 10.1.1.2, both on subnet 255.255.255.0. Although this does not allow for background network traffic, which is a limitation described in Section 2.5.1, the data collected does not factor in network traffic anyway, so it was considered an acceptable loss of fidelity.

3.4.2 Software Environment

There are many useful tools to test for computer vulnerabilities and simulate malware effects. One widely used application is Nessus, which scans a system for weaknesses in access control, checks for current patch updates, and probes ports with known attack profiles. Metasploit is a well known framework for developing and decoding exploitations, and is often used to test IDS software. EICAR has also developed a simple text file to test anti-virus software, which can be found at http://eicar.org/anti_virus_test_file.htm, but will not be reproduced here as it causes anti-virus software to quarantine any file containing the known character string (including Word documents). These are valuable because they allow researchers to simulate effects without actually infecting systems with real malware. Although many of these tools were investigated, they were ultimately supplanted by actual malware, which was more applicable to this research.

The search for appropriate malware that predictably and appreciably affects a victim system yet is easily removable is a non-trivial task. Though there are numerous blogs, websites, and businesses dedicated to collecting information about malware and publishing fixes (Astalavista.com, Milw0rm.com, bleepingcomputer.com, etc.) a guaranteed location to find reliably unaltered source code for malware online could not be found. Initially, a popular virus ironically called Antivirus XP 2008 was a prime candidate since its effects and procedures for removal are widely documented, but it was not possible to find a dependable copy of the actual virus without a danger of infection by other malware. After a considerable amount of investigation, older copies of the popular Sub7 program were discovered, which provided an ideal platform to test intrusion while limiting the danger of additional attached malware. Fortunately, the service pack (2) loaded on the laptops was susceptible to Sub7, otherwise they would have had to be reloaded with an older patch to allow the exploit to work.

To enhance security, hardware restrictions were applied as described previously, and exposure to hacker sites was kept to a minimum while retrieving the necessary malware. In addition, full scans were completed with Symantec Antivirus 2005, Version 10.0.2.2000, Scan Engine 71.1.0.11 with Virus Definition File 3/29/2007 rev. 32 after downloading the malware, as well as between data collection runs, to ensure the laptops were as pristine as possible.

The anti-virus software did pose some problems, however, as the auto-protect feature deleted Sub7 files as soon as they were downloaded. Therefore, the auto-protect feature had to be disabled while the malware was being harvested and while tests were

being run. Also, Symantec is very interested in protecting users from inadvertent lapses in security, so the auto-protect feature reengages itself every 30 minutes, which required constant vigilance during test runs. Since the 1.5 version of Sub7 is an older malware program, the anti-virus software had no problems finding it. Also, Sub7 allows the user to design their own server with specific capabilities on what programs it can attach to, where it "hides" on the victim system, and how it contacts the hacker. The server developed for this research was intentionally designed to be as transparent as possible after installation to enable easy removal. However, the ease of detection by existing anti-virus software did not necessarily make it easier for PAIDS since PAIDS is looking for anomalous activity in system processes rather than for a specific code string like Symantec does.

TaskInfo was practical to collect data for research, but it would be unrealistic to rely on text output for a true IDS, because the minimum data collection rate is only once every ½ second which is still far too slow to be effective at detecting attacks which can happen in milliseconds. Also, the Heisenberg uncertainty principle applies to analyzing an operating system with a program that is working on the same operating system (i.e. measurements of the system are affected by the monitoring program.) Instead, the necessary data should be gathered directly from the operating system and analyzed by an embedded PAIDS, preferably at the kernel level or lower using primitive hardware monitors or software not resident on the operating system being monitored.

3.4.3 Data Acquisition and Formatting

At the start of the project, it was hoped an existing data set could be analyzed to prove the concept of identifying the operation of malware through system anomalies rather than the standard method of parsing network traffic. This would allow comparison to previous IDS performance, and simplify the research since a new data set takes time to generate. The DARPA 1998 dataset produced by MIT Lincoln Labs was promising, but did not contain the granularity of host level responses required. Likewise the KDD 1999 Cup data was unusable, as it merely contained packet data from TCP dumps and not operating system measurements. Thus, like many other IDS research projects, the data had to be produced locally.

Although the Center for Cyberspace Research did not have specific software to record system process parameters, it was relatively easy to procure with a quick Internet search. TaskInfo, a shareware program designed by Igor Arsenin, was an ideal platform for collecting the desired test data. Unfortunately, the program is primarily a GUI, and text output from the program (Appendix A – Output Data from TaskInfo in Excel Format) was not accessible in real time by the preferred analysis software, MATLAB. Therefore, to generate time sequenced data, test runs were recorded into an output folder as described in Data Collection, and then transferred into a useable array format as described below.

TaskInfo actually collects a great deal of information, but many of the fields are non-numeric. These fields could be used for classification by assigning numeric values

to the categorical information, but the numerical data already present provided enough discrimination so they were dropped. The remaining data fields are listed in Table 6.

VARIABLE	*DESCRIPTION*
PID	Process Identification Number
% CPU	Percentage CPU used by each process
% K CPU	Percentage kernel CPU used by each process
Sw/s	Number of switches to execution of process/second
InMem KB	Physical memory used by process in KB
Private KB	Virtual memory used by process in KB
Total KB	Total virtual address space used by process in KB
Th	Number of threads currently running in process
Handles	Number of handles opened by process
Windows	Number of windows opened by process
USER Obj	Number of user objects opened by process
GDI Obj	Number of GDI objects opened by process
Reads	Number of read operations issued by process
Read KB	Data read by process in KB
Rd Rate B/s	Read data rate in bytes/sec
Writes	Number of write operations issued by process
Write KB	Data written by process in KB
Wr Rate B/s	Write data rate in bytes/sec

Table 6 – Characteristics Collected for each Operating System Process

Converting the output into a numerical array was not a trivial task. Each line of the text file had to be read and decomposed individually, then converted to a single row vector representing the various characteristics for every process during each sample. The resulting vector starts with the first characteristic and lists the corresponding values for each process, then continues to the next characteristic until all 17 are recorded (Table 9).

Two other idiosyncrasies of computer process operations compelled further data manipulation. First, the number of processes running at any particular moment changes as they are started and stopped, either by recurring automatic activities (disk scans, autosaves, etc.) or by normal legitimate activity. Next, the order processes are put in the

"stack" is different every time a computer is turned on. Therefore, it was necessary to develop a method to standardize the number of processes to be recorded, and the data had to be scrubbed to ensure the same processes were being compared in the same way between data collection sets. Otherwise, the correlation matrices produced by PCA and FA would be meaningless.

The first problem actually has a simple solution because each new process gets assigned a process identification (PID) when it is started. When the PID number gets too large, the PIDs cycle back to one, so no two concurrently running processes will ever have the same number. To ensure the same processes are recorded from one sample to the next, PAIDS compares the PIDs from the first two samples of collected data and retains only the identical PIDs. These become "static" processes, and the characteristics in Table 6 are collected for only these processes. The number of processes beyond the "static" ones is also recorded as an additional characteristic, which is added to the end of the row vector for that sample. For this research, there were generally 52 or 53 static processes, and since 18 characteristics are retained for each process, the matrices were generally at least 1000 x 936. In fact, the sample size of 1000 seconds was chosen to ensure the matrices were full rank and statistically significant.

A simple example will show how this works. Assume the data in Table 7 is the first sample taken, while data in Table 8 is the second sample, with only nine characteristics recorded for each process. If one were to look only at the Process information, they appear identical, however if the PIDs are compared, it is seen that from the first sample to the next, the first instances of cmd.exe and TaskInfo.exe have been

dropped and the next instances have moved up in the stack to replace them. At the same time, new instances of the same processes have been added.

Process	PID	% CPU	% K CPU	Sw/s	InMem KB	Private KB	Total KB	Th	Handles
alg.exe	2660			0	3,560	1,200	33,196	6	105
Dot1XCfg.exe	2044			40	14,624	9,716	156,808	15	297
wuauclt.exe	2932			0	4,080	2,220	36,852	3	206
EXCEL.EXE	3116	0.72%		91	39,612	20,368	172,308	10	694
cmd.exe	3584			0	1,204	1,448	13,636	1	19
TaskInfo.exe	2812	2.17%	1.44%	179	10,320	7,224	49,952	5	150
cmd.exe	3668			0	1,328	1,448	13,636	1	19
TaskInfo.exe	2808			0	8,008	5,940	46,808	5	95

Table 7 – Sample 1 Notional Data

Process	PID	% CPU	% K CPU	Sw/s	InMem KB	Private KB	Total KB	Th	Handles
alg.exe	2660			0	3,560	1,200	33,196	6	105
Dot1XCfg.exe	2044			48	14,624	9,716	156,808	15	297
wuauclt.exe	2932			0	4,080	2,220	36,852	3	206
EXCEL.EXE	3116	0.72%	0.72%	135	39,612	20,368	172,308	10	694
cmd.exe	3668			0	1,328	1,448	13,636	1	19
TaskInfo.exe	2808	2.90%	0.72%	766	10,324	7,224	49,952	5	150
cmd.exe	2332			0	1,260	1,448	13,636	1	19
TaskInfo.exe	2368			0	8,808	6,804	46,808	5	97

Table 8 – Sample 2 Notional Data

It is easy to see that if these two samples had been compared to one another in a covariance or correlation matrix, or by any standard of measure for that matter, anomalies would most certainly arise, but they would be meaningless. Thus, for this example, only the first four processes would be kept for analysis, although the total number of processes running at each moment would be recorded as an additional characteristic. The data is then read in order of columns into a row vector for each sample as shown in Table 9.

% CPU				% KCPU				Sw/s				InMemKB				Private KB				Total KB				Threads				Handles				#
0	0	0	0.72	0	0	0	0	0	40	0	91	3560	14624	4080	39612	1200	9716	2220	20368	33196	156808	36852	172308	6	15	3	10	105	297	206	694	8
0	0	0	0.72	0	0	0	0	0	48	0	135	3560	14624	4080	39612	1200	9716	2220	20368	33196	156808	36852	172308	6	15	3	10	105	297	206	694	8

Table 9 – Row Vectors for Sample Notional Data

This is only a valid procedure assuming the computer has not been infected prior to start-up. If the computer is already infected, then the malware effects will probably be incorporated into the initial "normal" baseline and PAIDS will most likely not detect an anomalous condition caused by the malware in the future. However, this is an unavoidable weakness in any anomaly detection IDS.

The second problem, process order, is only applicable to historical data collected for this research. This will not be a problem in the actual PAIDS, because the baseline dataset will be reset every time a computer is turned, and the order of the "static" processes will not change during that session. The only time it may be a problem is if a process is started before PAIDS makes it "static" assessment, and is dropped afterwards. For instance, in the example Excel.exe is regarded as "static"; however, if the user quits Excel the same number of processes is still recorded to maintain constant dimensionality, thus cmd.exe would be recorded as "static" in the second sample. In this case, although the dimensions of the matrices match, the last process in every subsequent sample will change, and the analysis will be meaningless.

Even after the data is successfully stored in a numerical array, it requires some slight modifications to be analyzed using multivariate methods. Specifically, if a column contains no variation, when the correlation matrix is calculated, these columns produce infinite values when they are inverted, the matrix becomes singular, and the method fails. In any other data set, these columns would be dropped as they obviously don't affect the outcome since they don't change and cannot contain any anomalous condition. However, in this case, columns that have no variation vary between data sets, so if they were

dropped, comparisons would be made between the wrong characteristics and dimensions often would not match. For instance, the number of write operations in one dataset might be compared to the amount of data written in another dataset, which clearly doesn't make sense.

Instead of dropping columns with no variance, they are replaced with random noise. This has little effect on the analysis, since by its nature the noise does not contain significant anomalies, and the method retains all columns so the dimension of the array remains constant. One problem encountered with this technique is in the calculation of zero variance. When MATLAB calculates the standard deviation of a column, a floating point error often produces values that are extremely small (on the order of e^{-17}) instead of zero and these columns are not replaced with noise. Thus, the workaround seen on line 27 of Appendix F – PCA/Mahalanobis Distance was developed to get the results desired.

At the same time, all columns are normalized to values between 0 and 1 by dividing each value by the largest value in the column, so the matrix is not subject to scale problems caused by measurements on different orders of magnitude (i.e. number of threads open vs. kilobytes of data read.) When this is complete, the matrix generated is normalized, numerical, nonsingular, and positive definite. Finally, the data is ready for multivariate statistical analysis.

3.5 Statistical Methods

3.5.1 Factor Analysis

This research began with the premise that FA would be a better tool to aid in discrimination between normal and abnormal because the technique finds a linear

combination of unobservable factors based on commonalities and uniqueness of the data. Also, factor rotation could be used to find the most effective axis of discrimination. Thus, this technique seems ideal to find anomalies and distinguish between the populations. In practice, several difficulties were encountered.

First, the data in this research is not very well behaved. The arrays are sparse, not well scaled, contain many variables with no variation, and are very large. The problems and solutions to importation and manipulation have been described in 3.4.3, and most of them were developed in response to singularity issues encountered when performing FA. For instance, in order to calculate standardized data for a dataset ***X***, the inverse square root of the diagonal of the covariance matrix is needed.

$$\mathrm{Cov}(\mathbf{X}) = \begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1p} \\ S_{21} & S_{22} & & \\ \vdots & & \ddots & \\ S_{p1} & & & S_{pp} \end{pmatrix} \tag{III-1}$$

$$\mathbf{D}^{-\frac{1}{2}} = \begin{pmatrix} {S_{11}}^{-\frac{1}{2}} & 0 & \cdots & 0 \\ 0 & {S_{22}}^{-\frac{1}{2}} & & \\ \vdots & & \ddots & \\ 0 & & & {S_{pp}}^{-\frac{1}{2}} \end{pmatrix} \tag{III-2}$$

However, when the inverse of this matrix is computed with the original data, the columns with zero variance create infinite values and the matrix becomes singular. Obviously, the same problem occurs when the correlation matrix is calculated.

Once the singularity problems are fixed with random noise, the number of factors to be retained must be determined. This is accomplished with the Principal Factor method. Starting with equation (II-11) we can estimate the correlation matrix **R** with **R***

$$\mathbf{R} = \mathbf{\Lambda\Lambda^2} + \mathbf{\Psi} \quad \text{(III-3)}$$

$$\mathbf{R}^* = \mathbf{h^2} + \mathbf{\Psi} \quad \text{(III-4)}$$

Then $\mathbf{h^2}$ can be calculated by squaring the factor loading matrix $\mathbf{\Lambda}$ where

$$\mathbf{\Lambda} = \sqrt{\boldsymbol{\lambda}}\boldsymbol{\gamma} \quad \text{(III-5)}$$

$\boldsymbol{\lambda}$ is a diagonal matrix of eigenvalues from the original correlation matrix

$\boldsymbol{\gamma}$ is the matrix of corresponding eigenvectors arranged in columns

The commonality estimates $h_i{}^2$ become the diagonal of **R***

$$\mathbf{R}^* = \begin{pmatrix} h_1^2 & \Gamma_{12} & \cdots & \Gamma_{1p} \\ \Gamma_{21} & h_2^2 & & \\ \vdots & & \ddots & \\ \Gamma_{p1} & & & h_p^2 \end{pmatrix} \quad \text{(III-6)}$$

R* grows by iterating until the estimated average commonality $\sum h_i^2/p$ is greater than a specified value. In this research the value was set at 80%. Once the number of factors, *p*, has been established, the eigenvalues, $\boldsymbol{\lambda}$, can be resized to a *p* x *p* matrix and factor scores can be calculated for the remaining data.

Since **R*** accounts for all of the factors, and $\mathbf{h^2}$ accounts for the commonality in the factors, if **R*** is normalized then from (III-4)

$$\mathbf{\Psi} = \mathbf{1} - \mathbf{h^2} \quad \text{(III-7)}$$

Factor scores are then calculated using, $\mathbf{\Psi}$, the estimate of factor uniqueness. Various methods have been proposed to calculate this score.

General Least Squares: $\mathbf{f} = (\mathbf{\Lambda'\Psi^{-1}\Lambda})^{-1}(\mathbf{\Lambda'\Psi^{-1}})\mathbf{X_d'}$ (III-8)

Min Mean Square Error: $\mathbf{f} = \mathbf{\Lambda'}(\mathbf{\Lambda'\Lambda} + \mathbf{\Psi})\mathbf{X_d'}$ (III-9)

General Regression: $\mathbf{f} = \mathbf{X_s'}(\mathbf{R})^{-1}\mathbf{\Lambda}$ (III-10)

In this research, $\mathbf{X_d}$ and $\mathbf{X_s}$ start with the data to be analyzed then adjusts by the mean of the baseline data as described in section 2.3.1. In this way, the new data is compared against the existing sense of "normality" before incorporating any changes into the model. Also, if known changes do occur, for instance when legitimately opening a program, a new baseline can be established, so anomalies will only occur from unknown conditions such as an intrusion.

Once a p x 1 factor score, $\boldsymbol{f}$, has been obtained, Mahalanobis Distance, **MD**, is calculated for each of n samples using (II-18) with p x p retained eigenvalues, $\boldsymbol{\lambda}$, instead of the covariance matrix. Assuming the average factor score for baseline data is zero

$$\boldsymbol{MD} = \mathbf{f}'\boldsymbol{\lambda}^{-1}\mathbf{f} \qquad \text{(III-11)}$$

The MD vector is then used for discrimination between normal and abnormal.

All of the above methods for calculating factor scores were tried in this research, however, none of them performed as well as Principal Component Analysis. The reasons for this are described further in Results and Analysis.

3.5.2 Principal Component Analysis

In PCA, a smaller weighted set of linear combinations that describes most of the variation of the original characteristics is calculated to reduce the dimensionality of the problem. In this research, PCA was used in two stages of the problem. First, it was used to select a representative feature set and establish a baseline data set. Next, it was used to compare this baseline to new collected data and calculate a Mahalanobis Distance (MD) between Principal Component Scores. The MD is then used to determine normal or abnormal activity.

When PCA is performed, a loading matrix, **L**, is calculated from the product of eigenvectors, $\boldsymbol{\gamma}$, and the square root of the eigenvalues, $\boldsymbol{\lambda}$, from either the correlation or covariance matrix of original data

$$\mathbf{L} = \boldsymbol{\gamma}\sqrt{\boldsymbol{\lambda}} \tag{III-12}$$

It is important to note that the absolute values produced are always less than one and that covariance and correlation matrices produce different values. This research used the correlation matrix because the characteristics were measured on widely different scales, although the resulting matrices were eventually normalized to solve singularity problems.

Once the loading matrix is calculated, the first principal component (first column) is evaluated to determine which characteristics load greater than a certain absolute value; this research used an empirically generated value of 0.2 as a cutoff point. The values for only these characteristics are retained as the new baseline data and future data collection is only performed on these characteristics until a new baseline is established. There is not necessarily a mathematical reason this method should work; however, there are four reasons it is advantageous:

1. It is a unique method for dimensionality reduction, typically decreasing the amount of data to be retained and compared by over two-thirds
2. Retaining highly loaded characteristics from only the first component provides optimum discrimination between the baseline and future data
3. This method for choosing the feature set is completely opaque to an outside observer, and thus highly resistant to reverse engineering
4. The feature set changes every time a baseline is established, further reducing a hacker's chances of reproducing "normal" looking conditions

The choice of using characteristics from only the first principal component was also empirically determined. At first, *n* components were retained until they explained 80% of the variance of the *p* characteristics, such that

$$\frac{\sum_1^n \lambda_i}{p} = 0.8 \qquad \text{(III-13)}$$

Then, all characteristics that loaded higher than 0.2 on every retained component were kept. However, the same characteristics load repeatedly on different components, and the number of uniquely loaded characteristics rapidly decreases as the number of retained components is increased. Next, the heuristic of retaining only those eigenvalues greater than one was applied (Kaiser's criterion) but there were still too many retained characteristics. Finally, it was determined that keeping only the first component's characteristics was sufficient to discriminate while providing the minimum number of retained values. In this research, the number of retained characteristics was generally about 310-330 out of 936-954 original characteristics.

For example, Table 10 shows an example loading matrix for the first six principal components with 18 characteristics. These six components would account for (8.287+3.682+1.506+0.937+0.928+0.772)/18 = 89.51% of the variance. In this case, the 3, 4, 6, 7, 8, 10, 11, 12, 14, 16, 17 and 18 characteristics would all be retained, which is only a reduction from 18 to 12 dimensions. If the 80% rule is used, then the first four eigenvalues are kept and we are left with nine characteristics, which is a 50% reduction. If Kaiser's criterion is applied, only the first three components are kept, and the retained characteristics are reduced slightly to seven. This is a significant reduction in dimensionality, but it has been found empirically that keeping only the first component,

and in this example only one characteristic, still provides enough discrimination to be useful.

Component	1	2	3	4	5	6
Eigenvalue	**8.287**	**3.682**	**1.506**	**0.937**	**0.928**	**0.772**
Characteristic						
1	-0.15384	-0.07401	0.096476	-0.1563	0.049696	-0.02757
2	0.06142	-0.12557	0.050501	0.169607	0.177942	0.043259
3	-0.0336	-0.17164	0.544852	-0.08679	-0.36458	-0.11488
4	0.091929	0.485113	0.144811	0.179635	-0.08225	0.424709
5	-0.04926	0.086957	-0.04442	0.075297	-0.0772	-0.02613
6	0.071546	0.155932	0.101687	-0.25174	0.246949	0.040774
7	-0.10697	-0.00649	-0.2533	-0.00158	0.047893	0.066287
8	0.185519	-0.02182	-0.70137	0.266576	0.374742	-0.14233
9	0.005743	-0.0604	0.12758	-0.13675	-0.08696	0.105364
10	-0.05692	-0.05561	-0.19612	-0.13258	0.244851	-0.03042
11	0.144862	0.008105	-0.17055	0.365728	0.248483	0.207989
12	-0.17734	-0.07986	-0.24029	0.064084	0.315741	0.120701
13	0.079405	-0.08628	-0.14164	0.094099	0.052438	0.008133
14	0.175093	0.247552	-0.27632	0.106642	0.288483	-0.20954
15	0.039833	0.137801	0.014215	-0.02605	-0.01455	0.179396
16	-0.07113	-0.06733	-0.05017	-0.15586	0.242786	-0.13835
17	-0.21841	-0.21717	-0.27561	-0.14524	-0.02712	-0.14616
18	-0.03333	-0.11565	0.033592	-0.09906	0.044157	0.220843

Table 10 – Example PCA Load Matrix

After the baseline of "normality" has been established, it must be compared to the incoming data. Again, PCA is employed using the correlation matrix (Appendix F – PCA/Mahalanobis Distance) but during this stage only the eigenvalues which contain 80% of the variance are retained. As before, various methods of dimensionality reduction were tested, and this time the 80% rule provided optimum discrimination. The number of retained components after this stage was typically 20-30, which is a 98% reduction in dimensionality from 936-954.

The loading scores are calculated as shown in Section 2.3.1 where $\mathbf{X_s}$ is calculated by subtracting the mean of the baseline data from the new data, then

standardizing with the inverse square root of the baseline's covariance matrix. As stated previously, the eigenvalues used to calculate the loading scores are from the correlation matrix. Finally, the loading scores and retained eigenvectors are used to calculate a Mahalanobis distance for each sample as described in the previous section.

3.5.3 Quadratic Discrimination

Although it is easy for a human to visualize the effectiveness of PAIDS when the Mahalanobis Distance (MD) is plotted, it is useful to generate an actual numerical value in terms of percentage true and false positive to compare results against other IDSs. Quadratic Discrimination (QD) conveniently provides these values, although it is unorthodox to use QD in this case since MD is part of the QD calculation. However, QD was performed simply to prove the success of the PAIDS algorithm. The QD procedure and measurement of effectiveness is described in Section 2.4.2.

To perform QD with this data, the MD values from the initial baseline are stored as "normal" data, and they are compared to the incoming data in a two-way classifier. If PAIDS were to be implemented in a real system, the first time legitimate activity occurred the system could be prompted to record additional "normal" data to be used in a three-way classifier. In this way, the system could learn how to measure "self" and "other". Anything that registers in the three-way classifier as something different than the initial baseline or designated legitimate activity would be labeled an intrusion.

IV. Results and Analysis

This research consisted of four main experiments, presented here in chronological order. As the research progressed, the analysis techniques changed slightly to pursue avenues that seemed to hold promise for anomaly discrimination. The progression is depicted as clearly as possible to show how assumptions and results evolved throughout the process.

4.1 Oct 31 Test – Component Scores

At first, it was hoped a simple plot of the first principal component scores would offer some distinction between normal and abnormal populations. At the very least, it provided some insight into the complexity of the problem, and some potential areas of improvement for future experiments. In this first test, MATLAB was still being used to run code, so the sample rate was limited to once every two seconds. Also, Internet connection was still being considered as a possible variable. Finally, the "intrusion" mechanism in this case was the EICAR standard anti-virus test file and the quirks of working with (and against) Symantec were still being worked out.

Three runs were completed in this first experiment: normal activity with no internet connection, normal activity with internet connection, and abnormal activity with internet connection. The timelines for each run are shown in Table 11 through Table 13. Sample number was used to maintain consistency between experiments and charts, because in this case each sample was two seconds apart while the interval in later experiments was one second.

Sample Number	*Activity on victim computer*
1 – 50	Opened Explorer without Internet connection Opened c:\ prompt, “pinged” from attacking computer
51-450	No activity, Explorer window still open
451-500	Opened Word, saved file, closed Word
501- 550	Opened Excel, saved file, closed Excel
650-700	Opened c:\ prompt, “pinged” from attacking computer
750-900	Opened Word, Excel, Outlook, and PowerPoint Copied/Pasted between programs Opened old files, saved and closed new files
901-1000	No activity, no windows open

Table 11 – Oct 31 Run1 (normalnoactivity) Timeline

Sample Number	*Activity on victim computer*
1 – 1000	Opened Explorer with Internet connection Opened two tabs, searched for possible virus to use during next phase of experiment

Table 12 – Oct 31 Run2 (normalinternet) Timeline

Sample Number	*Activity on victim computer*
1 – 59	Downloaded EICAR test file from eicar.org/anti_virus_test_file.htm Symantec auto-protect instantly deleted file Disabled Symantec auto-protect
60-69	Ran EICAR test file
110-119	Ran EICAR test file
160- 169	Ran EICAR test file
210-219	Ran EICAR test file with Symantec enabled
260-269	Ran EICAR test file with Symantec enabled
310-319	Ran EICAR test file with Symantec enabled
360-369	Ran EICAR test file
410-419	Ran EICAR test file
460- 469	Ran EICAR test file
510	Ran Symantec quick scan
560	Enabled Symantec auto-protect
700	Found and quarantined EICAR files
910	Deleted EICAR files

Table 13 – Oct 31 Run3 (abnormalinternet) Timeline

The first run, labeled “no activity”, actually had a significant amount of activity, which was not taken into account during the analysis. For this reason, the dataset was deemed unusable, but it highlighted the importance of recording an exact account of the activity during the data collection process. The second run, labeled “normalinternet”, was taken while connected to the Internet, so there is no guarantee the data is malware free except that no files were downloaded and it was run on a military connection which is presumed to be (relatively) secure. Also, the first time malware was intentionally introduced from the Internet it was caught and deleted instantly by Symantec auto-protect. Workarounds, described in Section 3.4.2, were used to avoid Symantec throughout the rest of the research.

After the data from these runs was cleaned and processed, the principal components scores from the first two components were plotted to see if any discrimination could be made. Figure 13 shows there was indeed a distinction, particularly between the second and third runs, though there was still some overlap.

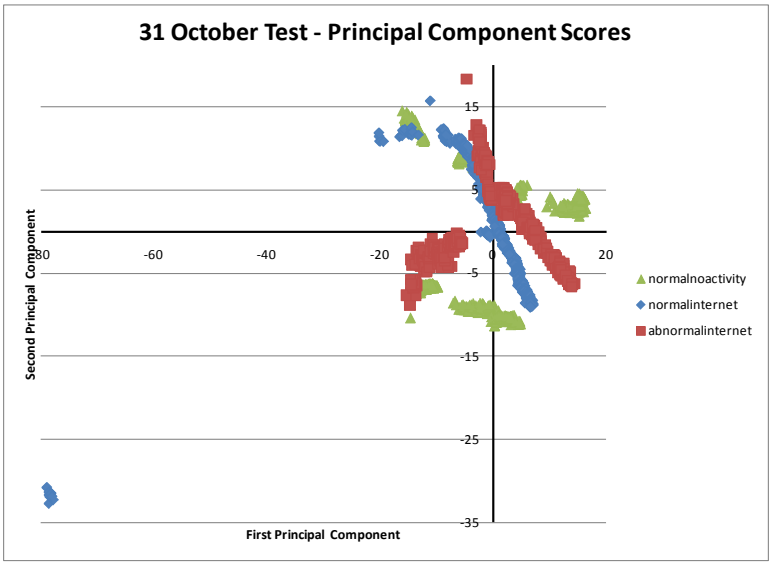


Figure 13 – Oct 31 First Two Principal Component Scores

The fact that discrimination was apparent in two dimensions was promising, since it was known the final discrimination would be accomplished in at least 17 dimensions. It also demonstrated the strength of the first principal component for discrimination, which was used later to facilitate dimension reduction. This technique definitely did not give the clarity required for an IDS; plus, the analysis was faulty since "static" processes had not been identified yet, and columns with zero variance were dropped instead of being replaced with noise. It was also assessed that a more significant "attack" was needed to get a measurable result.

Interestingly, when this data was analyzed with the PCA-PCA-MD technique developed later, the presence of the EICAR file was readily distinguishable (Figure 14). However, the difference between the operation of this file and normal activity would not be significant enough to register in quadratic discrimination.

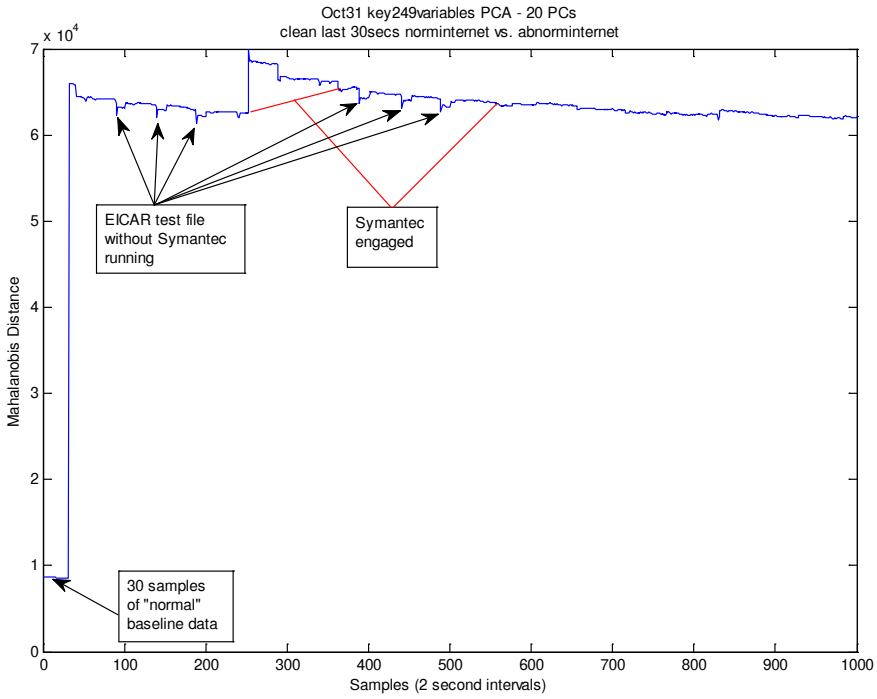


Figure 14 – Oct 31 Evaluated with PCA-PCA-MD Technique

4.2 Nov 7 Test – Component Scores vs. Time

Five runs were accomplished during this test, with 1000 samples of data at a rate of once per second for each run. The object of this test was to work out the data problems identified in the first test, so no malware was introduced and no activity was conducted during any run. Many of the data manipulation techniques used throughout the research were developed as a result of this test.

The biggest problem with the data collected during these first tests was that columns with zero variance caused the PCA and FA techniques to fail. These columns were initially thrown out, however this caused dimensionality discrepancies (comparing apples to oranges) so the technique of replacing the columns with random noise was developed as described in Section 3.4.3. Also, the data was badly scaled, since some measurements were taken in kilobytes while others were a simple count of "number of threads open by process" so the entire matrix was normalized from 0 to 1 to correct this problem. These techniques were continued throughout the research.

Also, the monitoring code required MATLAB to be open, which greatly affected the operating system, memory used, and the speed at which measurements were taken. Changes were made so a batch file could call TaskInfo directly, but data collection still required the operation of these two programs (TaskInfo and a command prompt). However, using the batch file, the data could be collected twice as fast with less impact on the operating system, so this method was used for the rest of the tests.

Finally, the highly variable nature of the data became readily apparent during the first two tests. Both between runs and during a single run, system processes were added

and dropped when programs were open and closed and when automatic operating system processes (disk scans, etc.) started and stopped. This not only caused problems comparing data sets, but also with the statistical methods used due to singularity problems. This problem was corrected by identifying "static" processes as described in Section 3.4.3 and only recording data for these processes.

However, the first run in this test showed the "static" process technique may also induce problems. For instance, a DOSscan was occurring during the first eight seconds of this run, so it was identified as a "static" process, but when the scan was complete it dropped out of the stack and the algorithm was stuck recording too many processes. This caused dimension problems and gave the system a false sense of self. This discrepancy can be seen in the wildly different scores for run 1 plotted in Figure 15.

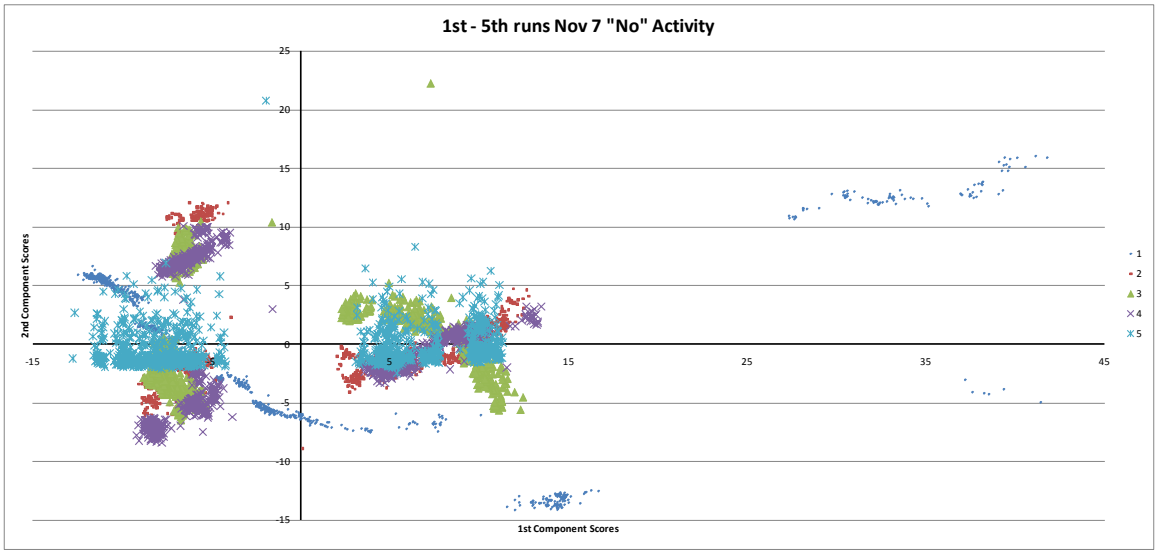


Figure 15 – Nov 7 First Two Principal Scores All Runs

Based on these observations, it is imperative that the total number of processes recorded by PAIDS is established at startup after all beginning transient processes have closed and before recurring automatic processes begin. This technique also assumes the

system is not infected at startup since any intrusion at that point would be incorporated into the sense of self.

Another conclusion from this test was that a plot of PCA scores over time would not be useful. It was hoped some type of variable control limit could be placed on the values of these scores which would be exceeded under anomalous conditions such as an intrusion. Instead, as can be seen in the figures below, it was found that this data by itself was too highly variable to be of value. Thus, the idea of calculating a Mahalanobis Distance between component scores was introduced.

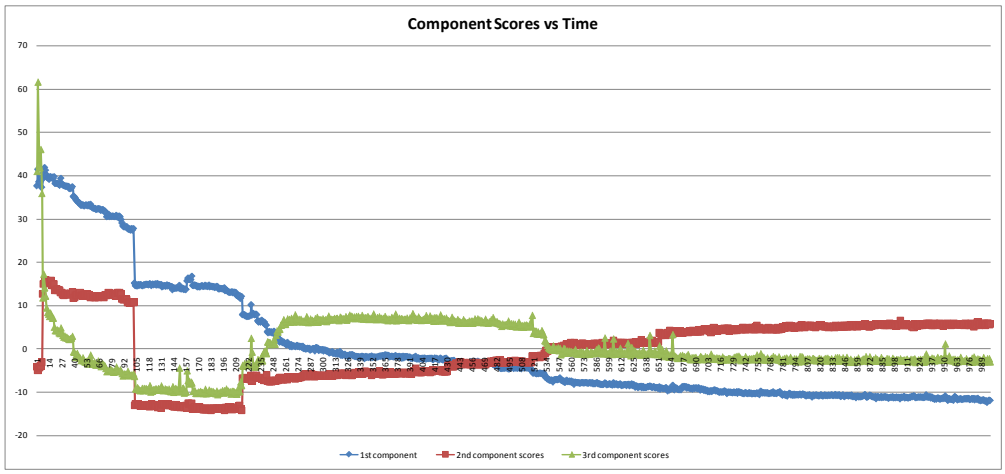


Figure 16 – Nov 7 Run1 First Three PCA Scores vs. Time

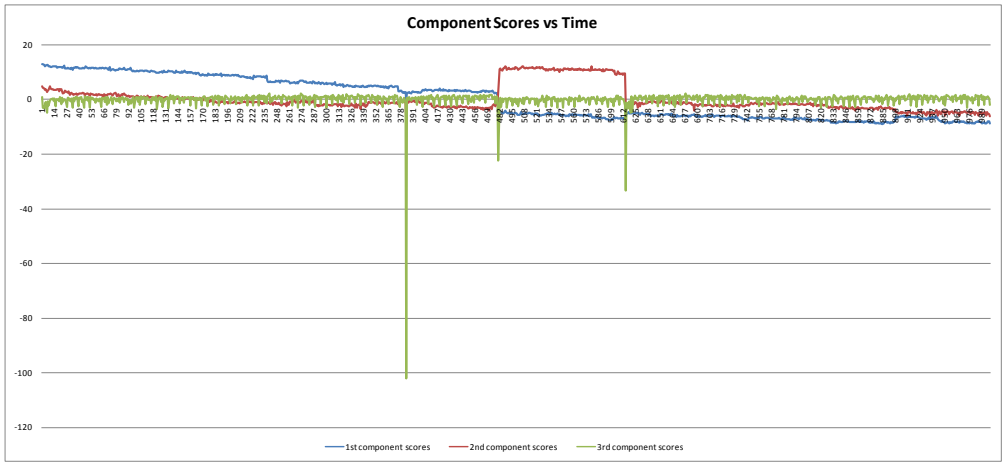


Figure 17 – Nov 7 Run2 First Three PCA Scores vs. Time

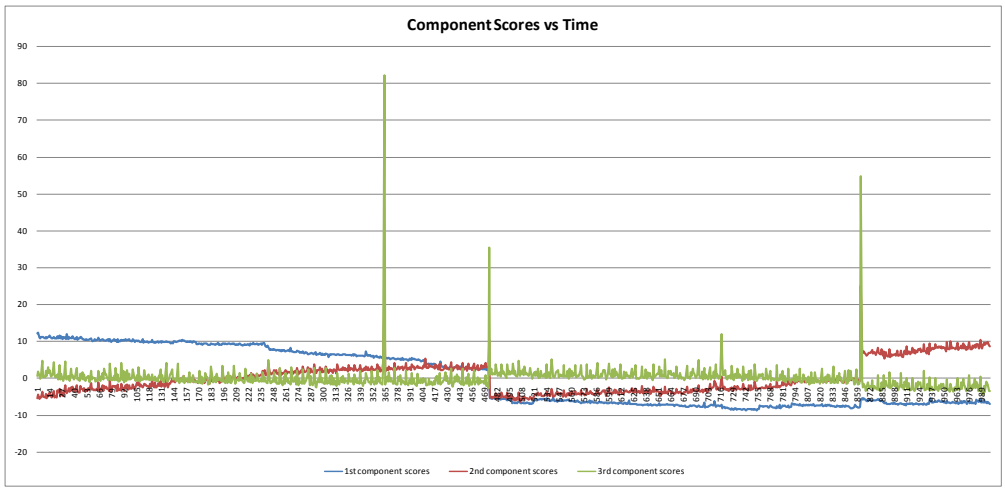


Figure 18 – Nov 7 Run3 First Three PCA Scores vs. Time

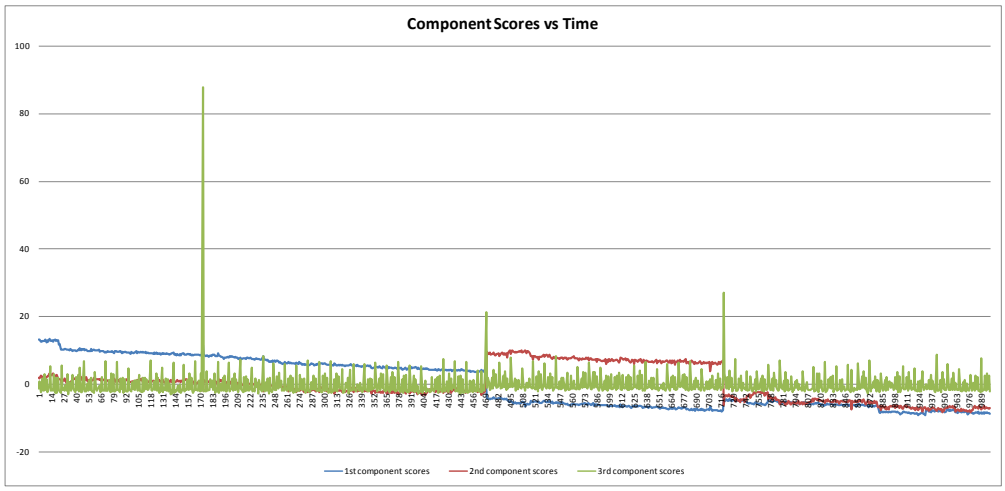


Figure 19 – Nov 7 Run4 First Three PCA Scores vs. Time

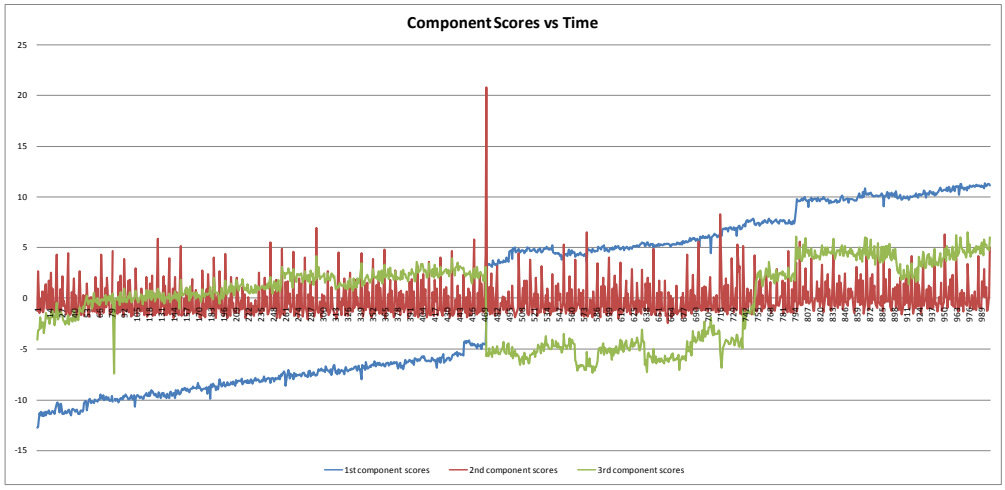


Figure 20 – Nov 7 Run5 First Three PCA Scores vs. Time

4.3 Nov 21 Test – Principal Component Analysis-Mahalanobis Distance (PCA-MD)

Once the data problems had been resolved, a new, more realistic, attack had to be found. Various hacker websites were visited, as described in Section 3.4.2, before the backdoor Trojan, Sub7, was identified as an ideal candidate. This exploitation was used throughout the rest of the research. During this test, the design of experiment presented in Appendix J – Nov21 Test Plan was used during a single log-on session. It was hoped that using a single log-on would decrease the chances of problems caused by a different number or order of "static" processes.

Even with the precautions taken, the first 11 runs had an extra process running, which had to be deleted from the historical record before analysis could be performed. This system process is described at www.ProcessLibrary.com (Uniblue, 2008):

ccapp Stands for Common Client Application and is the executable responsible for checking emails and auto-protect facilities as part of the Norton Antivirus suite. The ccapp.exe file is used as the Norton Antivirus' real-time scanner executing behind the scenes, scanning your system for Trojan Horses, viruses, and worms.

So even after auto-protect had been disabled, there was still a process running in the background attempting to detect malware. This demonstrated how convoluted background processes can be and the importance that "static" processes established at start-up are not actually transient.

Once the data had been scrubbed to ensure the same processes were being compared against each other, it was necessary to determine which sets of data to compare. First, some terms were required.

- *Clean* – data with no activity and no malware
- *Dirty* – data with activity
- *Infected* – data with malware

To prove the Mahalanobis Distance technique was valid, a first attempt was made to compare a run to itself, which should in theory result in a very low MD, though not necessarily zero. Therefore, Run7 (clean) was used with the entire dataset as the "baseline" with the results presented in Figure 21. This showed that, while highly variable, the MD was relatively low for a dataset compared to itself as expected. However, there was an unusual result when compared to another clean dataset (Run10) using the entirety of Run7 as a baseline. Since both these runs were at level one activity with no malware, it was expected that a similarly low MD would result, however, Figure 22 shows this was not the case (notice the scale of the MD on the Y axis).

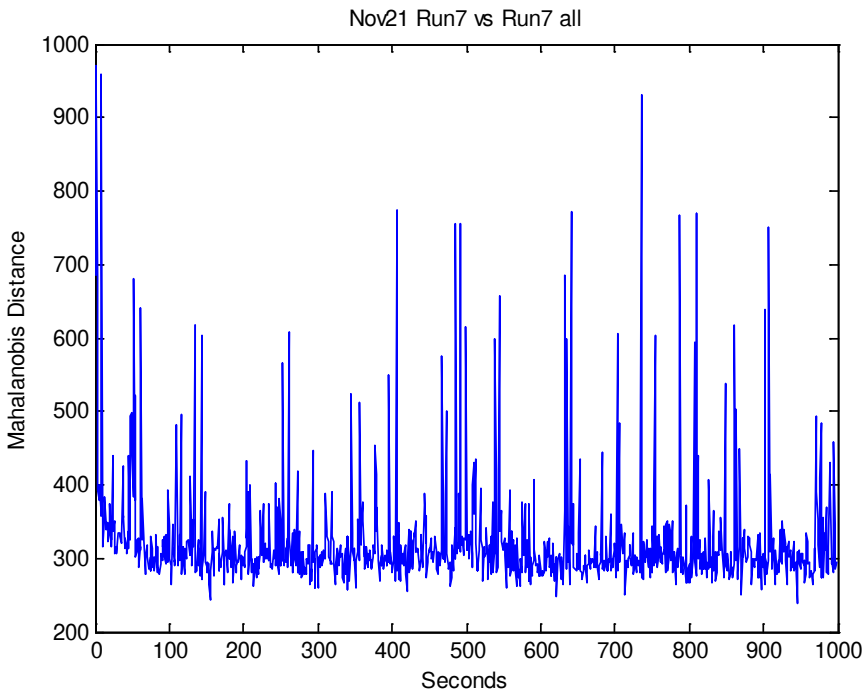


Figure 21 – Nov 21 Run7 (clean) All vs. Run7 (clean) All

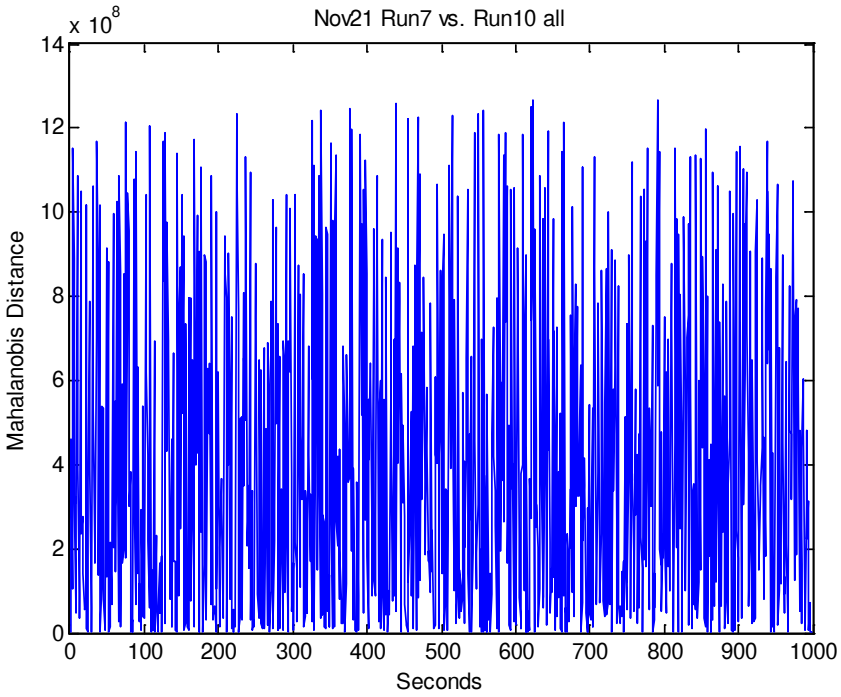


Figure 22 – Nov 21 Run7 (clean) All vs. Run10 (clean) All

These results suggest that the dataset used as a baseline makes a great deal of difference both in terms of when the window for data collection occurs and how long the window lasts. The first problem is caused by changing conditions of the operating system, and can be solved with a rolling window, periodic updates, or event driven updates to the baseline. The latter issue affects the memory and processing capacity required, since it takes more of both to analyze a longer window of collected data without improving the result. It also suggests the introduction of noise into the data may have an effect, and that the number of variables to be analyzed might need to be reduced.

At this point, it was possible to distinguish one dataset from another, but this did little to discriminate normal from abnormal let alone legitimate from illegitimate activity. For instance, when Run7 (clean) was concatenated with Run13 (dirty) and Run12 (infected) to simulate continuous data, the result was inconclusive (Figure 23). It was obvious there was a difference between the baseline data (first 1000 seconds) and other data, but there was still no distinction between dirty (second 1000 seconds) and infected data (last 1000 seconds).

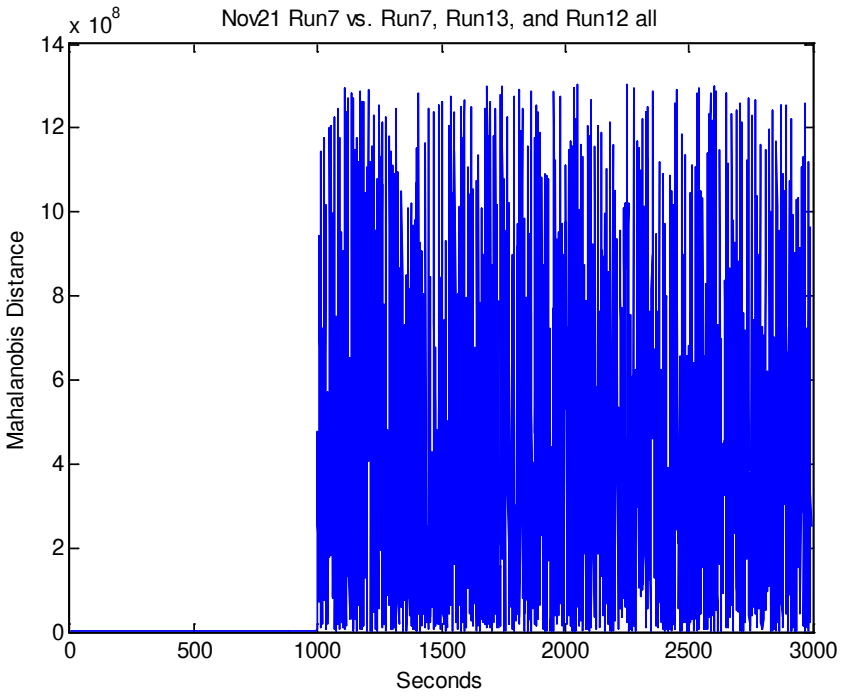


Figure 23 – Nov 21 Run7 vs. Run7(clean), Run13(dirty), and Run12(infected)

In fact, after some investigation, it was determined that it was only possible to distinguish between the window of data used as the baseline and any other data. It was clear from these results that distinctions could be made between datasets; however, the time and data required to make such discriminations was considerable. For instance, MATLAB took 56.7 seconds to generate Figure 22 and 107.9 seconds for Figure 23, while a 1000x936 matrix (1000 seconds of 936 variables) takes up approximately 5,000 KB of memory. Obviously, if PAIDS was to be an effective deterrent it would have to react faster to an intrusion, and if it were to be used on a MANET it would need to use less memory. Thus, less data had to be collected and analyzed while still providing the same or better level of discrimination in less time. During the course of the next test, alterations to the algorithm and observations of the resulting data provided significant improvements in memory load, processing speed, and accuracy of discrimination.

4.4 Nov 25 Test – PCA-PCA-MD

The next test was conducted to reduce the complexity of the data and to refine the data collection process so a more efficient algorithm could be designed. It concentrated on finding the key variables needed to discriminate between datasets and fine tuning the dimension and sample reduction techniques necessary to considerably decrease the amount of data required. There were only three runs during this test, and the victim computer was shutdown and cleaned between each run with a five minute "warm-up" period after being turned on. Sample numbers were determined by analysis of the resulting data files which showed when programs were opened and closed.

Sample Number	*Activity on victim computer*
1 – 999	No activity, except for batch file
1000 - 2000	Introduced Sub7 v1.5 into system from thumb drive

Table 14 – Nov 25 Run1 (clean/infected) Timeline

Sample Number	*Activity on victim computer*
1 – 1002	No activity, except for batch file
1003 – 2000	Opened programs and data as follows:
1015	Notepad – looked at .txt files
1146	MATLAB – imported data
1299	Excel – opened file with links to other files
1448	Word – opened file with embedded macros
1599	Powerpoint – opened file and manipulated it
1748	Explorer – opened without Internet connection
1897	TaskInfo – opened and took screenshot

Table 15 – Nov 25 Run2 (clean/dirty) Timeline

Sample Number	*Activity on victim computer*
1 – 666	No activity, except for batch file
667 – 1334	Opened programs and data as in Run2
1335 (1405) – 2000	Introduced Sub7 v1.5 into system from thumb drive

Table 16 – Nov 25 Run3 (clean/dirty/infected) Timeline

As with the Nov 21 test, Symantec AntiVirus auto-protect had to be disabled throughout the test, but in this case the data did not need to be massaged afterwards because the "static" processes were the same. Programs once started were not stopped, and when the backdoor was opened, all of the various activities available were conducted to simulate an intrusion. During the last run, the computer was running so slowly from all the open programs that effects from the intrusion did not appear for nearly half a minute on the victim computer after they were initiated on the attacking computer (thus the sample number in parentheses) and eventually caused runtime errors which shutdown MATLAB and Excel.

While performing analysis of this data, singularity problems appeared again, and it was at this point that the floating point error was discovered, requiring the use of $<10^{-11}$ instead of =0 (Appendix E – Principal Component Analysis Baseline – Line 20) as well as the need to reintroduce noise during the PCA process in addition to the import process. This was because smaller sample sizes were taken from the original data, and some columns once again contained all zeros, where in the larger data set there was at least one number to prevent the introduction of noise. In an actual application of PAIDS, the noise will only need to be introduced one time after the sample is taken.

In an attempt to reduce the amount of data recorded, an effort was made to find the key variables to discrimination. Initially, all eigenvectors with eigenvalues less than one were thrown out (Kaiser's Criterion) but in some cases only a single component remained, which did not provide any discrimination (Figure 24). A graphical method to determine the cutoff point such as Johnson's secant method is possible, but it takes much

less time and effort to retain eigenvalues that explain a certain percentage of the variance. This research used 80% with good results (Figure 25); however, future work might attempt a graphical solution or investigate different percentages to see if they provide better discrimination. Also, using the MATLAB command COV on standardized data instead of CORR on centered data (Appendix F – PCA/Mahalanobis Distance – Line 55) provided an order of magnitude decrease in processing time.

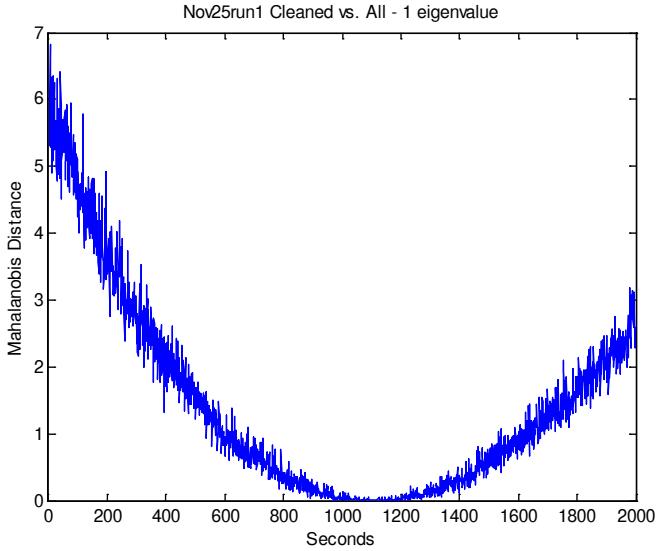


Figure 24 – Nov 25 Run1 (clean) vs. Run1 (clean/infected) - Kaiser's Criterion

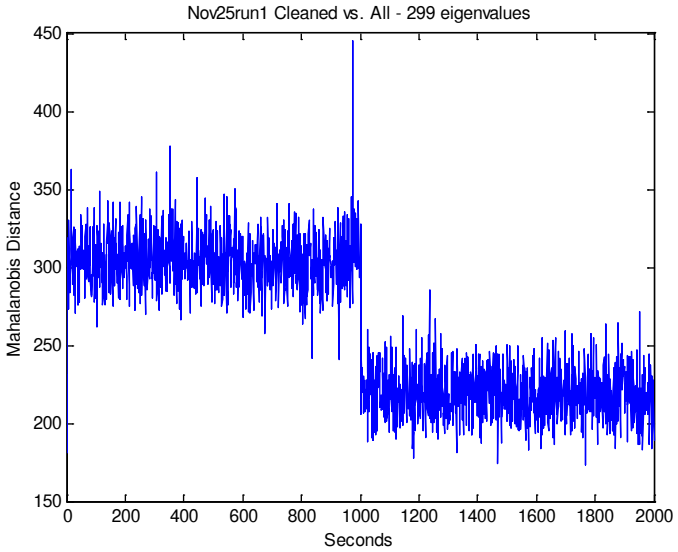


Figure 25 – Nov 25 Run1 (clean) vs. Run1 (clean/infected) - 80% variance

A major breakthrough was made in data reduction when it was discovered that variables which loaded heaviest on the first principal component of the baseline data were sufficient to discriminate between subsequent datasets. Originally, the factor loading matrix was observed to determine if a smaller common set of variables were important to intrusion detection. Details of the development of this technique are found in Section 3.5.2 and results are described in Section 4.3. The most important result was using these factor loadings to select key variables to collect and analyze.

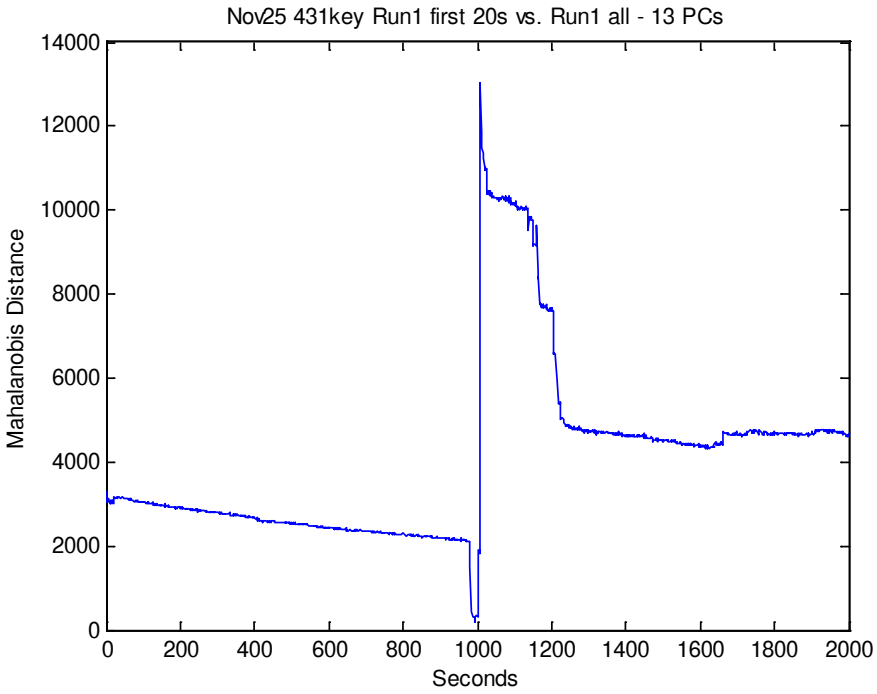


Figure 26 – Nov 25 Run1 (clean) vs. Run1 (clean/infected) - Key Variables

Now, instead of comparing all 900+ variables, PCA is performed on the baseline dataset, and only those variables which load higher than the absolute value of 0.2 on the first principal component are retained. This eliminates the noisy and low variability data, then PAIDS only collects data on the reduced set. When PCA is performed again on the

reduced data set, the number of retained components is less than 30, which is more than a 97% reduction in dimensionality, and the discrimination is much better (Figure 26).

Another major effort to reduce the amount of data was in determining an optimum period of time data that should be recorded (the sample window) to establish the baseline. Initially when developing the model, the entire period of known "normal" data was used as the baseline, but this was simply to prove the concept of distinction. During this test, three minute, two minute, one minute, 30 second, 20 second and 10 second intervals from the start of known periods of inactivity were used as a baseline. The procedure was:

- Run Appendix E – Principal Component Analysis Baseline on the first period of data taken during the run (i.e. first 10, 20, or 30 seconds, etc.)
- Reduce the database to the key variables identified in the baseline process
- Use these reduced datasets in Appendix F – PCA/Mahalanobis Distance to produce final results

Discrimination was not as good as later tests which used rolling and periodic windows, but it indicated the length of the baseline window was significant, and suggested variable lengths might be beneficial. Recurrence of data collection was addressed in another test.

Results varied slightly for all categories (processing time, components retained, etc.) but typical results of these comparisons are depicted on the following pages. Based on these results, it was determined 20 seconds was the optimum amount of data to retain as a baseline in terms of memory used, time needed to record and analyze data, and most importantly accuracy in discrimination.

Sample Length	Baseline Processing Time (Sec)	Key Variables Retained	Discrimination Processing Time (Sec)	Principal Components Retained	Baseline Database Size (KB)
10 sec	3.875830	572	0.979272	7	30
20 sec	3.826673	427	0.513489	13	41
30 sec	3.724152	320	0.337810	18	45
60 sec	3.419391	202	0.216404	25	52
120 sec	3.421965	114	0.127510	14	39
180 sec	3.117276	104	0.092684	4	35

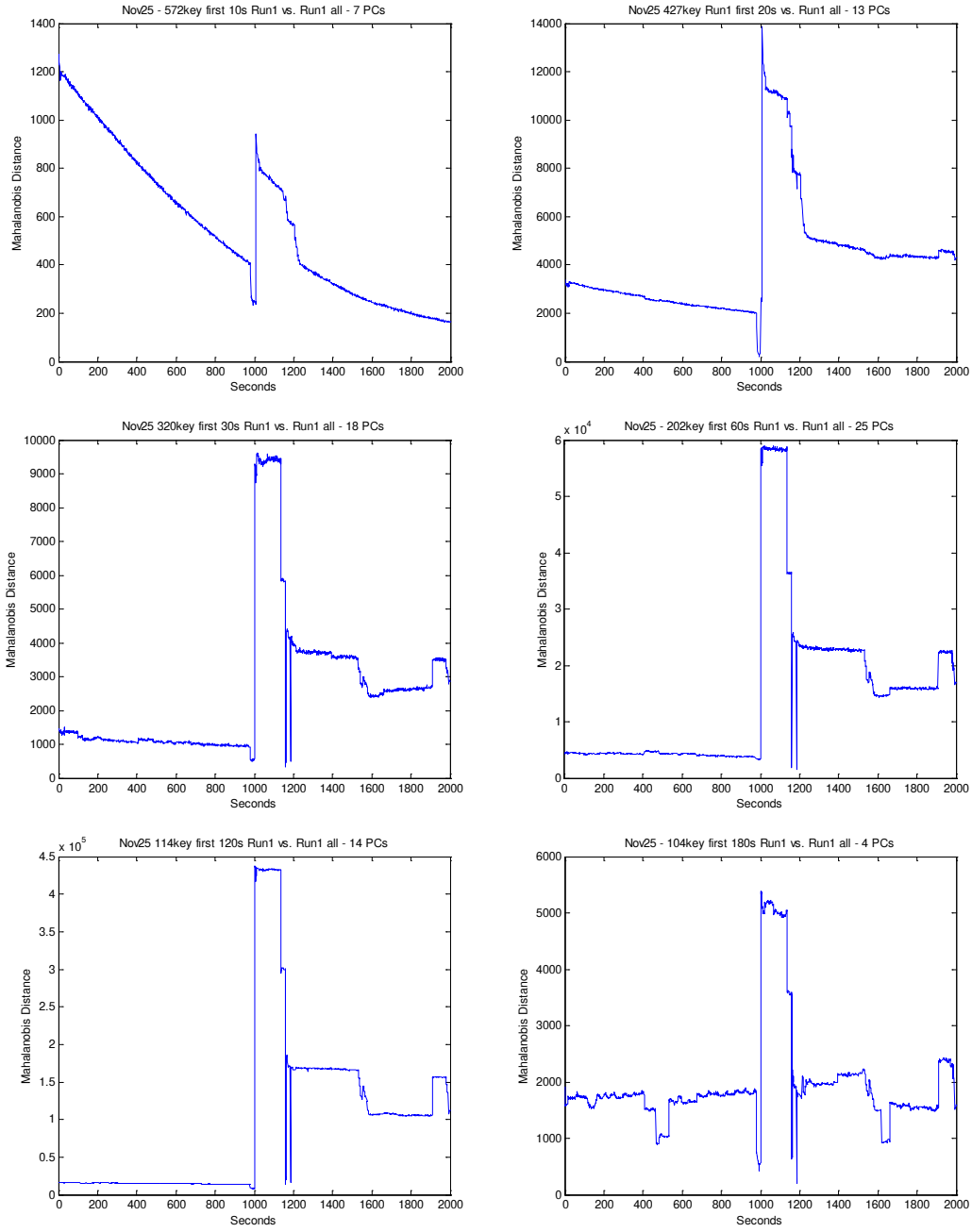


Figure 27 – Nov25 Run1 (clean/infected) PCA-PCA-MD Comparisons

Sample Length	Baseline Processing Time (Sec)	Key Variables Retained	Discrimination Processing Time (Sec)	Principal Components Retained	Baseline Database Size (KB)
10 sec	3.904401	585	1.048655	7	31
20 sec	3.825657	410	0.477722	13	41
30 sec	3.732138	326	0.347049	18	45
60 sec	3.361985	244	0.230664	19	47
120 sec	3.424280	156	0.118346	4	35
180 sec	3.138091	149	0.109284	3	34

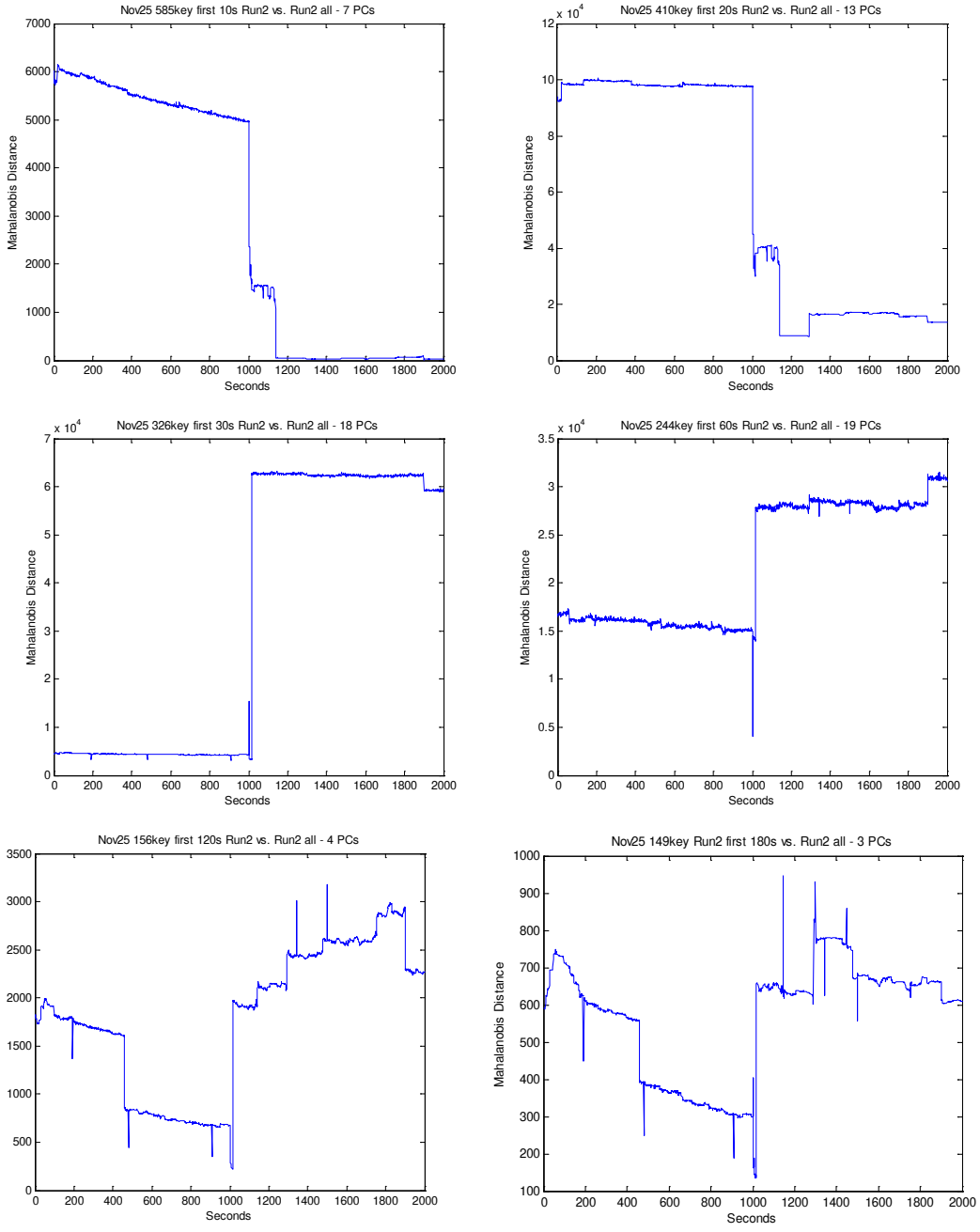


Figure 28 – Nov25 Run2 (clean/dirty) PCA-PCA-MD Comparisons

Sample Length	Baseline Processing Time (Sec)	Key Variables Retained	Discrimination Processing Time (Sec)	Principal Components Retained	Baseline Database Size (KB)
10 sec	3.945497	585	1.033670	7	28
20 sec	3.837077	431	0.525982	13	39
30 sec	3.764498	354	0.395639	19	46
60 sec	3.324500	209	0.228809	23	49
120 sec	3.522625	131	0.116491	9	35
180 sec	3.172045	117	0.099162	4	34

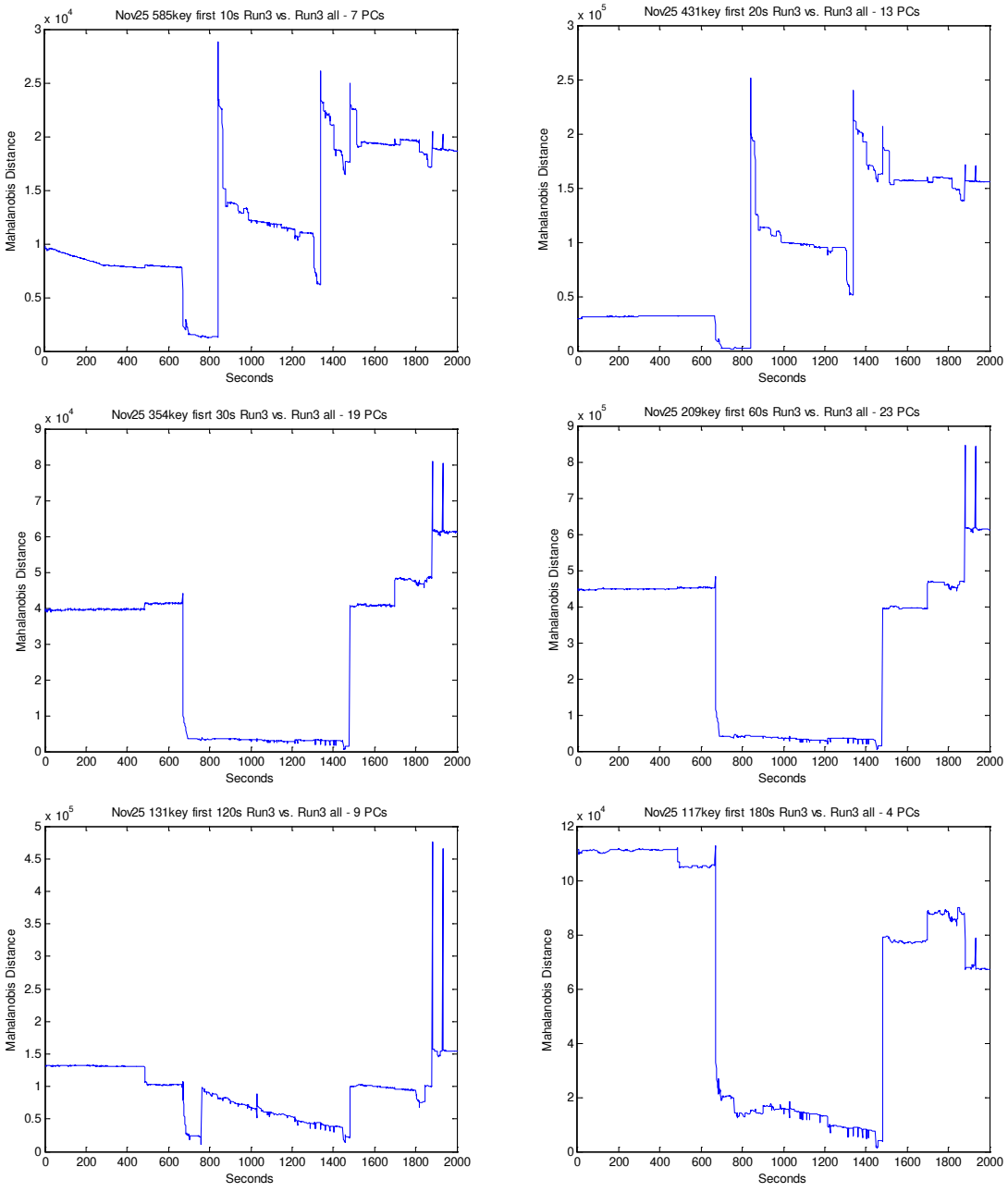


Figure 29 – Nov25 Run3 (clean/dirty/infected) PCA-PCA-MD Comparisons

The last decision to be made was how often the baseline should be updated. Some possibilities were:

- *Session-driven* – a single baseline established at start-up
- *Time-driven* – once every two minutes (or other periodic interval)
- *Event-driven* – based on user input when MD change exceeds limits
- *Continuous* – latest sample replaces earliest after being compared to existing baseline for anomaly (rolling window)

To decide the optimum update rate, comparisons were made between different baseline lengths and times to see how they affected discrimination. In other words, for a given dataset of clean, dirty, and infected data, the baseline was established at different times (at start-up, after a certain period of time, or immediately before and after transition to an altered state) and the discrimination between data was observed. The data from the Nov 21 test was primarily used for these comparisons.

Before embarking on this experiment, some assumptions had to be checked. If only the first 20 seconds of Run7 are used as the baseline, it was expected that the first 20 seconds would have a lower MD (since they are in effect being compared to themselves) and the remaining time would either be close to the same or slowly increasing as conditions changed. The results are presented in Figure 30, and though the low MD values last for 21 seconds instead of 20 as expected, the increase in MD is sharp and much greater than expected.

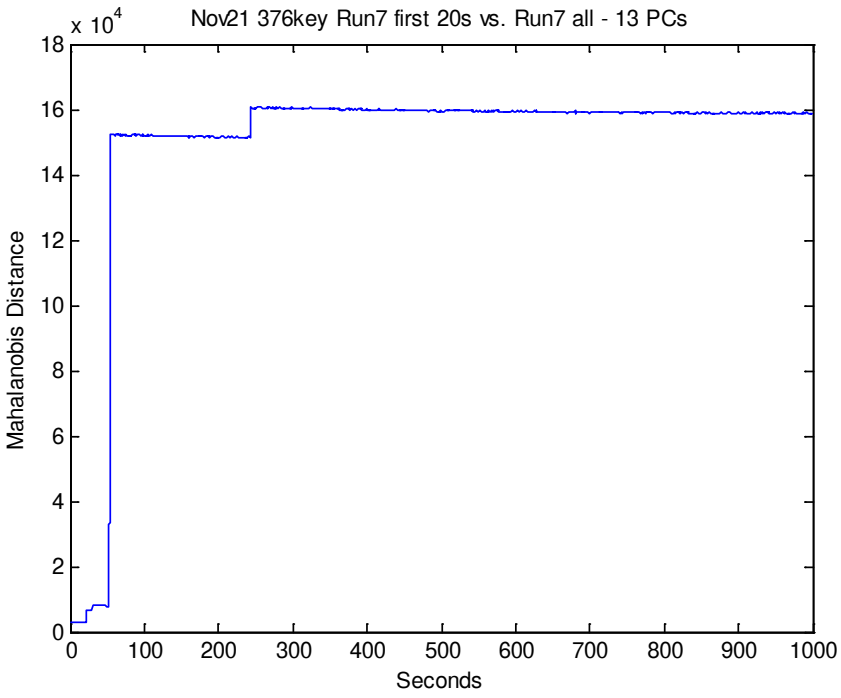


Figure 30 – Nov 21 Run7 First 20s vs. Run7 (clean) All

These intuitive results are far from typical. When the same experiment was completed on the other "clean" datasets, similar results were expected; however, the MD actually starts high and gradually decreases to near zero (Figure 31). Interestingly, when the baseline window was increased in an attempt to "even" the MD out, the change only exacerbated the spikes in the data and did nothing to decrease the initial value. In fact, none of the runs achieved the expected form, but runs 7 and 15 were considered most useable since they didn't contain large spikes. Clearly, the window size made a difference, but it was not entirely clear why the MD decreased towards zero instead of the other way around. Despite these counterintuitive results and the occasional spikes in MD, the results showed a dramatic difference between clean, dirty and infected data, which boded well for the next step: discrimination.

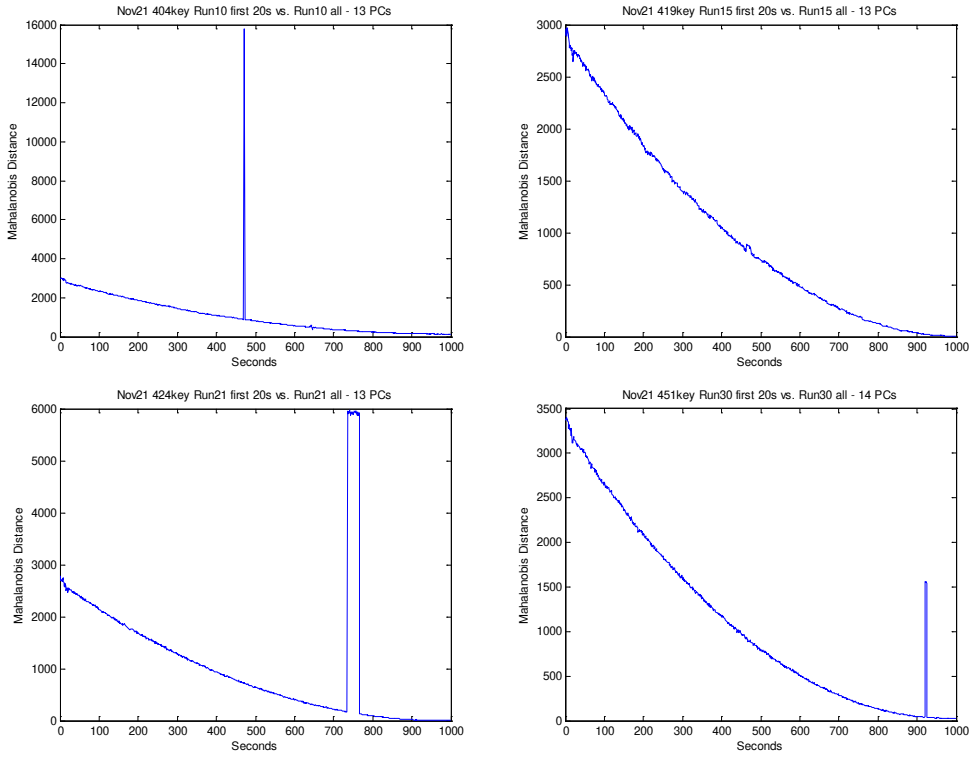


Figure 31 – Nov 21 Clean Runs First 20s vs. All Data

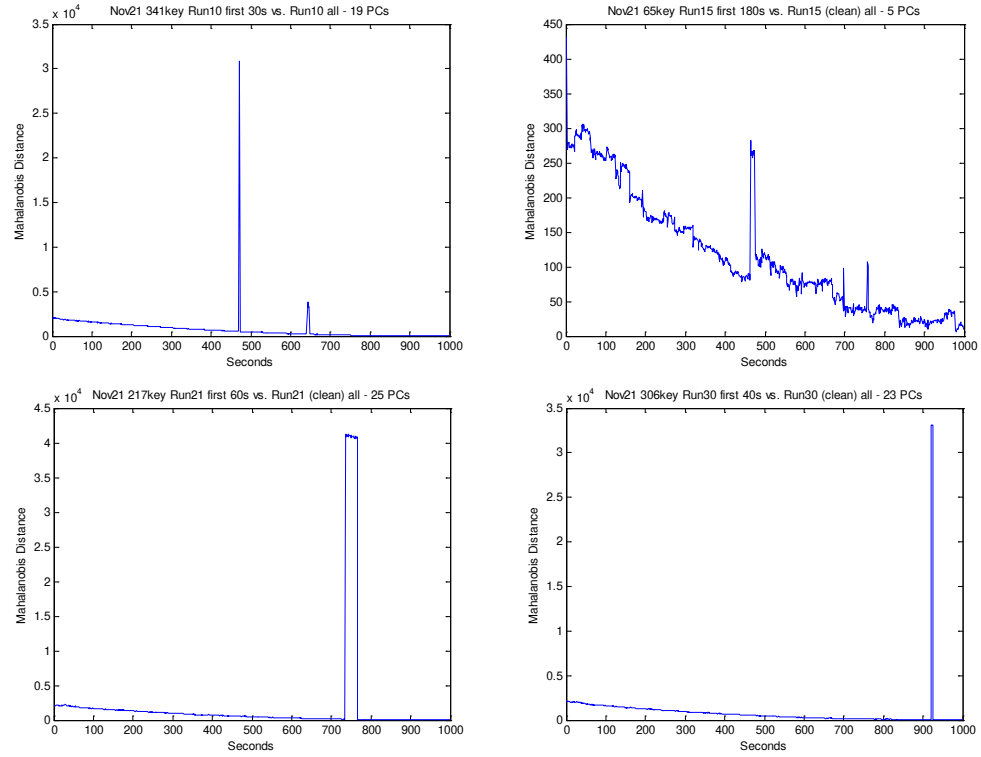


Figure 32 – Nov21 Clean Runs with Various Baseline Window Lengths

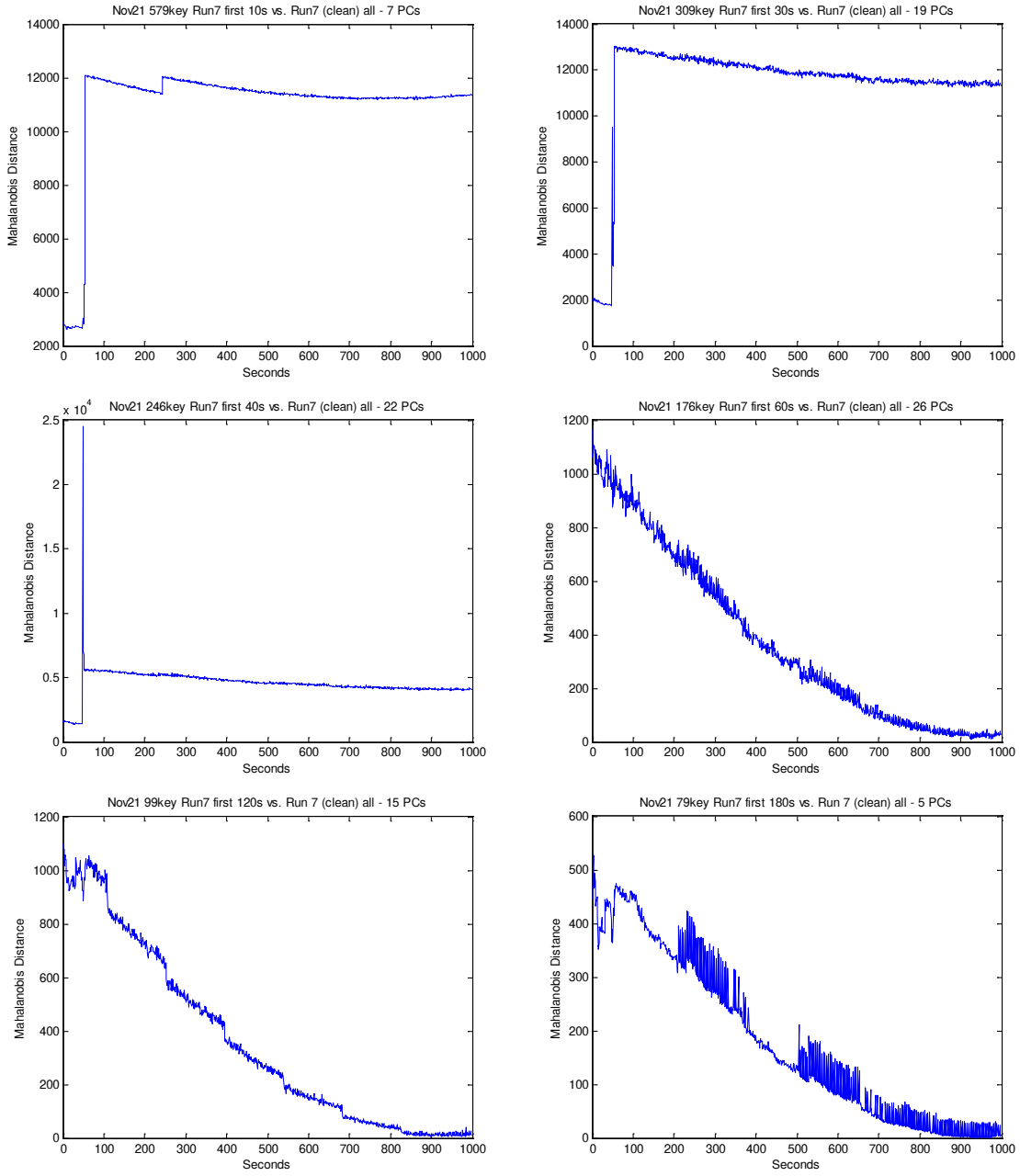


Figure 33 – Nov21 Run7 with Various Baseline Window Lengths

It was expected that a measurement of clean data would have a lower MD when compared to itself or other clean data than when compared to dirty or infected data. In some cases, such as Figure 34 this was the case, while in others, such as Figure 35 it was not. Again, it was unclear why this should be the case, though it seemed highly

dependent on baseline window size; however, it can easily be seen that discrimination was visually possible between clean, dirty, and infected datasets.

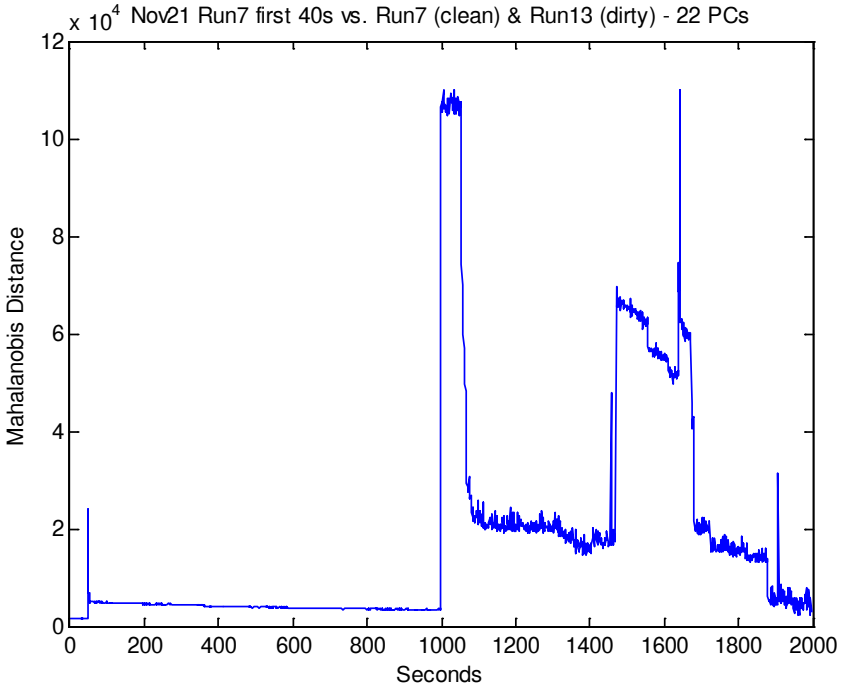


Figure 34 – Nov 21 Run7 First 40s vs. Run7 (clean) and Run13 (dirty)

The case of interest, of course, was determining whether activity was legitimate or illegitimate, so a clean dataset (Run7) was concatenated with dirty (Run13) and infected datasets (Run12) to simulate continuous data from the same system under normal use and intrusion. Figure 35 shows the results of using a 20 second baseline generated at (simulated) start-up, while Figure 36 shows the effect of updating the baseline just prior to the transition to dirty data, and Figure 37 shows the effect of updating the baseline just after legitimate activity has begun. If the window for the baseline was doubled to 40 seconds after legitimate activity began, the result was dramatically improved (Figure 38). These examples emphasized the need to re-establish a sense of self when system conditions change and re-iterated the efficacy of an event-driven baseline update;

however, they also suggest the baseline window size may need to change depending on conditions.

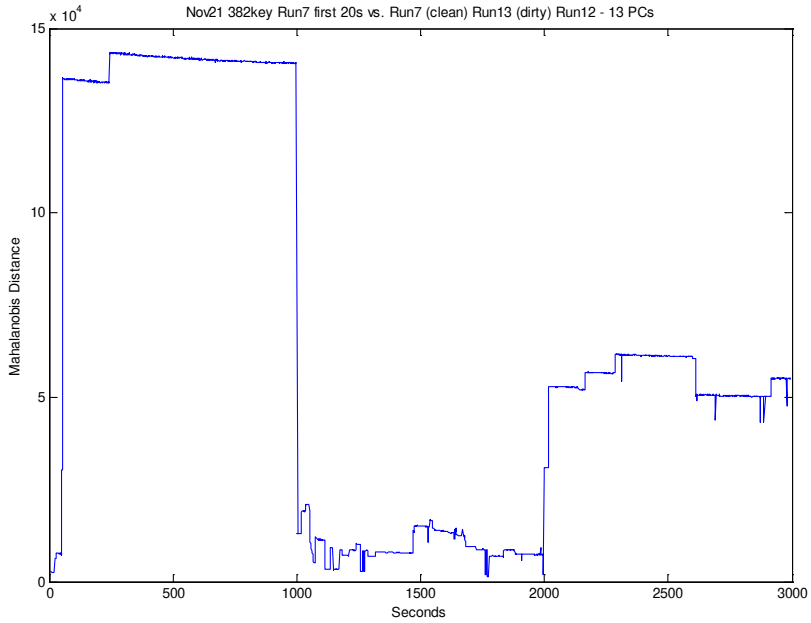


Figure 35 – Nov 21 Run7 first 20s vs. Run7 (clean) Run13 (dirty) and Run12 (infected)

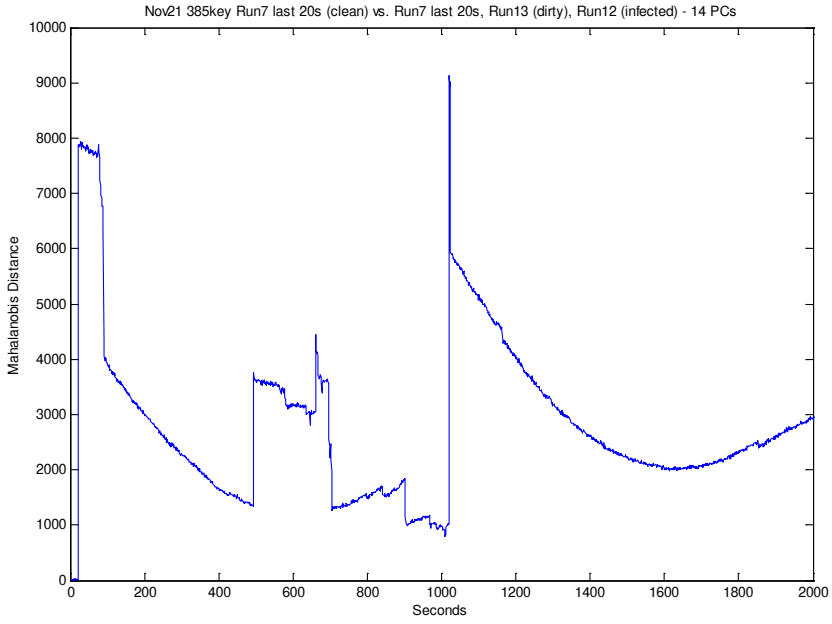


Figure 36 – Nov 21 Run7 last 20s vs. Run7 last 20s (clean) Run13 (dirty) Run12 (infected)

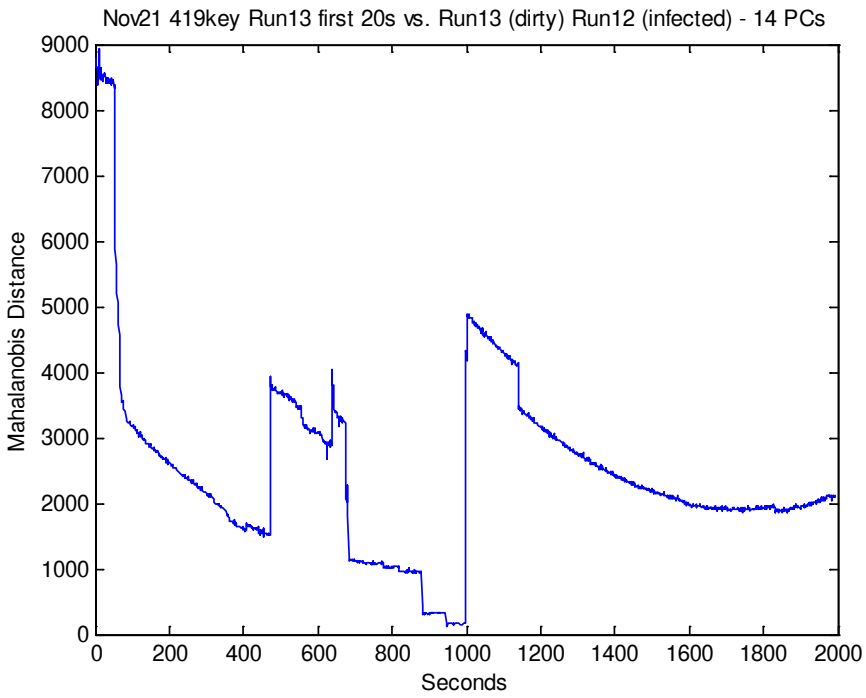


Figure 37 – Nov 21 Run13 first 20s vs. Run13 (dirty) and Run12 (infected)

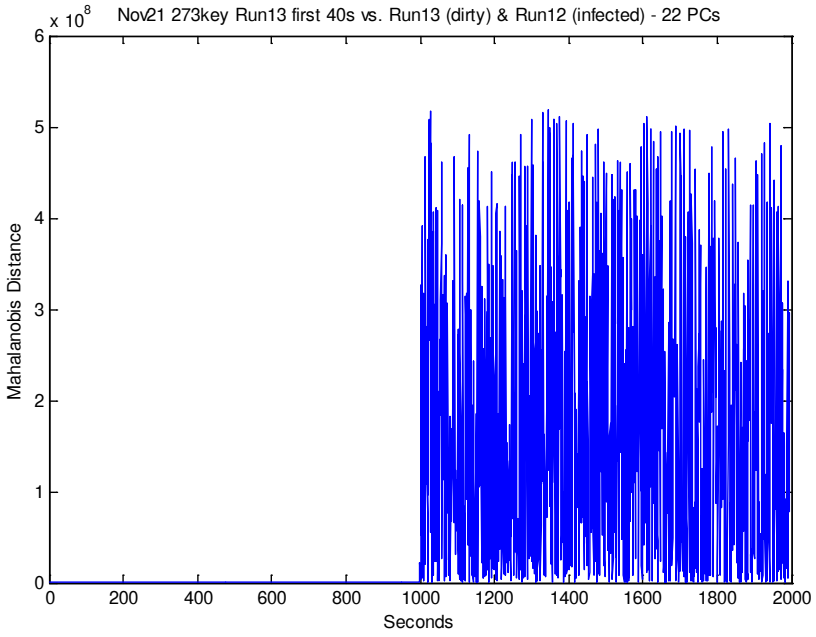


Figure 38 – Nov 21 Run13 first 40s vs. Run13 (dirty) and Run12 (infected)

4.5 PCA-PCA-MD-QD

The values for the Mahalanobis Distance (MD) between Principal Component scores provide a valid method for discrimination between legitimate and illegitimate activity, however a measurement of effectiveness is needed to compare the results against other IDSs. Quadratic Discrimination (QD) provides a means to test the efficiency of this method as described in Section 3.5.3.

When the MDs from the Nov25 tests (Figure 27 - Figure 29) were input to quadratic discriminators, using the first 20 seconds as a baseline and comparing all the data, the results were as follows:

25-Nov Run 1		Predicted Membership: clean	infected	Total data points	APER
Actual Membership	clean	981	18	999	0.01802
	infected	7	992	999	0.00701

25-Nov Run 2		Predicted Membership: clean	dirty	Total data points	APER
Actual Membership	clean	999	0	999	0
	dirty	0	999	999	0

25-Nov Run 3		Predicted Membership: clean	dirty	infected	Total data points	APER
Actual Membership	clean	645	21	0	666	0.031532
	dirty	1	642	23	666	0.036036
	infected	0	6	660	666	0.009009

Figure 39 – Nov25 Quadratic Discrimination with Individual Covariance Matrices

25-Nov Run 1		Predicted Membership clean	infected	Total data points	APER
Actual	clean	999	0	999	0
Membership	infected	252	747	999	0.25225

25-Nov Run 2		Predicted Membership clean	dirty	Total data points	APER
Actual	clean	999	0	999	0
Membership	dirty	2	997	999	0.002

25-Nov Run 3		Predicted Membership clean	dirty	infected	Total data points	APER
Actual	clean	666	0	0	666	0
Membership	dirty	184	442	40	666	0.336336
	infected	3	0	663	666	0.004505

Figure 40 – Nov25 Quadratic Discrimination with Pooled Covariance Matrices

From these results, it is easy to see that using a pooled covariance matrix instead of individual covariance matrices provides better discrimination. Predictably, the worst error was encountered when trying to distinguish legitimate activity (dirty) from intrusion (infected) during Run3. The false positive rate (identifying legitimate activity as illegitimate) was 40/666 = 6.01% but the false negative rate (identifying illegitimate activity as legitimate) was only 3/666 = 0.45% which is outstanding. Similar results were obtained with other test data. For instance, Run7 (clean), Run13 (dirty), and Run12 (infected) concatenated data produced no false positives and a 98.29% detection rate:

21-Nov Runs 7, 13, 12		Predicted Membership clean	dirty	infected	Total data points	APER
Actual	clean	941	53	0	994	0.05332
Membership	dirty	0	994	0	994	0
	infected	0	17	977	994	0.017103

Figure 41 - Nov21 Quadratic Discrimination with Pooled Covariance Matrices

4.6 Comparison to other IDSs

Although it is always difficult and dangerous to compare different data sets and techniques, PAIDS compares well to other IDSs, such as NIDES which had a 77.7% true positive rate and 1% false positive rate (Anderson, Frivold, & Valdes, 1995) and a genetic algorithm tested on KDD 1999 Cup data with a 95.47% true positive and 10.63% false positive rate (Kabiri & Ghorbani, 2005). Other anomaly-based algorithms (Figures 42-44) performed better against network attacks with respect to false positives but were not adaptable to changes in real-time.

Some research suggests that if "the cost of acting on a false positive [is] sufficiently low" (Fox, Kiciman, & Patterson, 2004) that the number of false positives is irrelevant, but it is always a good idea to try to reduce them as much as possible while still providing an acceptable true positive rate. An extensive inspection of issues in false positive rates for IDSs was performed by Mell, et al. in 2003. By adjusting the control limits placed on Mahalanobis Distance in PAIDS, it would be possible to decrease the false positive rate while maintaining its high true positive rate.

Method Used	TN (%)	FP (%)	FN (%)	TP (%)
Ours	99.58	0.42	9.93	90.07
Ohn [10]	99.55	0.45	9.08	90.92
Sung [7]	99.52	0.48	9.18	90.82
Ambwani [16]	99.57	0.43	9.26	90.74
KDDCup'99 Winner	99.45	0.55	8.19	91.81

Figure 42 - IDS effectiveness rates (Wong & Lai, 2006)

Attack	Detection Rate	False Alarm Rate
Intrusion I	85%	0.97%
Intrusion II	98%	0.89%
Intrusion III	99%	0.95%
Intrusion IV	87%	0.98%

Figure 43 - IDS effectiveness rates (Huang & Lee, 2003)

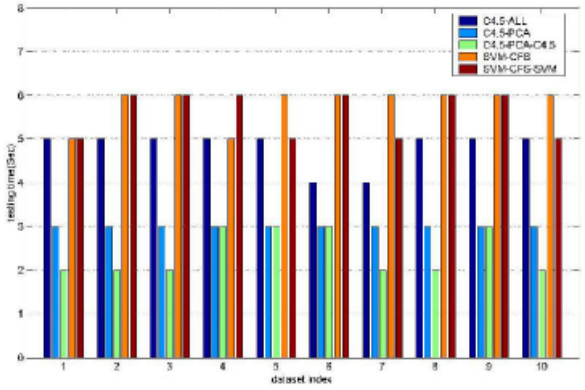


Figure 2. System testing time vs. dataset index

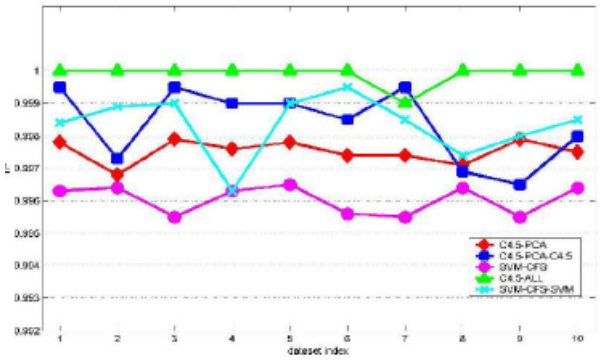


Figure 3. True positive rates vs. dataset index

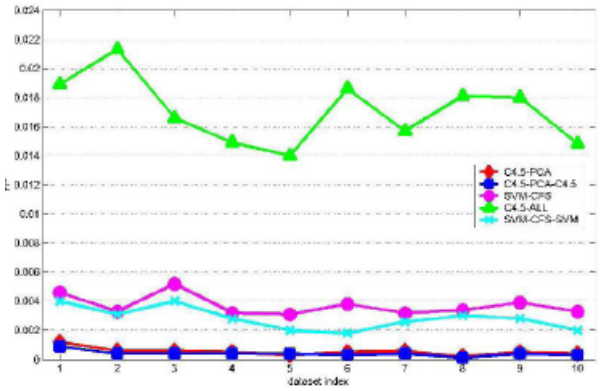


Figure 4. False positive rates vs. dataset index

Figure 44 - Comparison of IDS effectiveness (Chen, Dai, Li, & Cheng, 2007)

4.7 Key Operating System Processes

In an attempt to sell anti-virus and scanning software, companies often resort to vague and ominous threats. Uniblue offers a typical pitch: "In the recesses of your computer, 20-30 invisible processes run silently in the background. Some hog system resources, turning your PC into a sluggish computer. Worse yet, other useless processes harbour spyware and Trojans - violating your privacy and giving hackers free reign on your computer. Uniblue ProcessLibrary is an invaluable resource for anyone who wants to know the exact purpose of every single process." (Uniblue, 2008) However, more often than not, the scanning software offered contains spyware and phishing programs designed to advertise more anti-virus software and instill fear of further infection. Unfortunately, the operating system software has become so complex that even experts are often unsure of what a process really does.

Principal Component Analysis was used to highlight which processes were most significant in discriminating normal from abnormal systems. Using empirical data, any loads with absolute values greater than 0.2 were considered highly loaded. The four processes which loaded highly on the first principal component from clean data for all three runs of the Nov 25 test data were: cmd.exe, csrss.exe, lsass.exe and svchost.exe. Although these are only a few of the process characteristics retained, they are indicative of the processes that are relevant to intrusion detection and some of the efforts by hackers to avoid discovery. The website www.ProcessLibrary.com (Uniblue, 2008) describes these processes as having the following functions.

cmd Allows access to the Microsoft Windows Command Prompt, also known as Microsoft DOS. To-date, cmd.exe is a 32-bit command prompt used in

Windows NT, 2000, and XP and offers disk and file maintenance functions to your computer as well as network functions. This program is a non-essential system process, but should not be terminated unless suspected to be causing problems.

csrss The Microsoft Client Server Runtime Server subsystem utilizes the process csrss.exe for managing the majority of the graphical instruction sets under the Microsoft Windows operating system. As such Csrss.exe provides the critical functions of the operating system, and its termination can result in the Blue Screen of Death being displayed.
Csrss.exe controls threading and Win32 console window features. Threading is where the application splits itself into multiple simultaneous running tasks. Threads supported by csrss.exe are different from processes in that threads are commonly contained within the process, with various threads sharing resources within the same process. The Win32 console is the plain text window in the Windows API system (programs can use the console without the need for image display).
In mobile devices such as notebooks and laptops, the process csrss.exe is closely dependent on power management schemes implemented by the system as defined under the Control Panel option.

lsass The process lsass.exe serves as the Local Security Authentication Server by Microsoft, Inc. It is responsible for the enforcement of the security policy within the operating system. This process checks whether a user's supplied identification is valid or not whenever he or she tries to access the computer system.
With the execution of the file lsass.exe, the system acquires security by preventing the access of unwanted users to any private information. The file lsass.exe also handles the password modifications done by the user.
The process lsass.exe mainly operates in the system through its ability to create access tokens. These tokens encapsulate the file's security descriptor, which contains the necessary information to process user authentication such as data on which user holds access to the system and whether the access is mandatory or discretionary.

svchost The file svchost.exe is the Generic Host Process for Win32 Services used for administering 16-bit-based dynamically linked library files (DLL files) including other supplementary support applications.
As operating systems became more complex Microsoft decided to run more software functionality from a dynamic link library (DLL) interface. However DLLs are unable to launch themselves and require at least one executable program, i.e. svchost.exe, is needed to bridge between the library process and the operating system.
Through the solitary file svchost.exe, the DLLs efficiently contain and dispense Win32 services as well as neatly facilitate the execution of svchost.exe's own operations. Acting as a host, the file svchost.exe creates multiple instances of

> itself. The multiple executions of the file svchost.exe contribute to the stability and security of the operating system by reducing the possibility of a crashing process that causes a domino effect on its neighbor processes, thereby creating a system-wide crash in the machine.

However, even when a process appears legitimate, "determining whether [a process] is a virus or a legitimate Windows process depends on the directory location it executes or runs from." (Uniblue, 2008) Hackers often make minor alterations in legitimate process names to "hide" in the operating system. Uniblue lists lsasss.exe, svhost.exe and csrsc.exe as three of the top five security threats. (Uniblue, 2008) These are malware versions of legitimate processes associated with the w32/Sasser.E worm, W32.mydoomI@mm worm and W32.spybot.cf backdoor trojan respectively. You will notice that they are very close to three of the four processes highlighted by PCA as being significant for detecting intrusion.

V. Discussion

5.1 Conclusions

PAIDS monitors "static" processes running at the host level to develop a sense of "self" which can be analyzed for anomalies to detect intrusions by malware. This offers a last layer of defense in a multi-layered intrusion detection system which should also include physical, operational, and network security. The baseline data representing "normality" can be updated in near-real-time and since the recorded variables change over time and are discriminated in an 18 dimensional feature space, the IDS is highly resistant to reverse engineering or other hacking efforts. PAIDS would be most efficient on a platform outside the operating system to be monitored, such as a hardware primitive or software at the kernel level or below.

Data collection for the PAIDS algorithm is sensitive to window length and recurrence, as both these variables significantly change the results of Principal Component Analysis and Mahalanobis Distance. Empirical data implies that 20 seconds of baseline data is sufficient for discrimination while limiting the use of memory and processing time and that an event-driven update of the baseline is most appropriate. Mahalanobis Distance between selected Principal Component scores from host-level process characteristics over time is adequate for an analyst to detect an intrusion, though some known uninfected historical data is necessary to train the IDS to recognize legitimate activity. Preliminary data has shown that PAIDS can reliably produce greater than a 98% success rate in discriminating between legitimate and illegitimate activity with less than a 6% false positive rate.

5.2 Limitations

As is true with most IDSs, PAIDS is a purely reactive program, and will only detect an intrusion after it has begun. Thus, it is a last resort line of defense, notifying the user when the actual attack is occurring, rather than preventing the attack from happening. PAIDS assumes the system is not infected at start-up, since the intrusion would then be incorporated into the sense of "self" and operation of the malware would not be detected as an anomaly. Also, different operating systems may require different implementations of the algorithm, which would have to be developed separately. Finally, PAIDS uses multivariate statistical analysis methods which require relatively powerful mathematical programming, which may or may not be available in lightweight software or hardware primitives functioning outside the monitored operating system.

The methods used in this research required the output of one program to be analyzed by another program; however, any efficient application of PAIDS will have to take measurements directly from the operating system instead of relying on additional software to provide the data. This is primarily because the collection and analysis rate of any software is too slow to detect an anomaly in time to make a difference. The secondary reason is for security purposes, so a kernel rootkit or similar attack will not be able to subvert the collected data. It is not known if this technique is fully resistant to an attack that manipulates the process data itself, however, if the monitors are outside the operating system this would not be a factor. Multi-context primitive hardware monitors such as those suggested by Mott (2007) and Hart (2007) would be able to record the appropriate data, but software working outside the monitored operating system would probably be required to analyze the data.

5.3 Contributions

Very few IDSs use host-level system processes to identify intrusions. Using anomalies in hardware characteristics is a powerful tool because it does not rely on signature databases and it is not subject to standard spoofing like packet analysis can be. Instead, PAIDS can detect the operation of malware using subtle changes in the operating system that the user may not be aware of and that other anti-virus software is not designed to recognize. This is analogous to an expert observer "knowing" something is wrong with the computer due to sluggish or altered operation, yet not being able to identify exactly what is causing the problem. With PAIDS, the user no longer needs to be a computer analyst to know there is something wrong with the computer, but will still need help finding the actual malware and cleaning the system.

Also, most anomaly IDSs rely on historical data which is updated once per session at best, and often no faster than daily or weekly. PAIDS updates its baseline in near-real-time so that legitimate changes in the operating system can be incorporated into the definition of normality. This may require some user input when opening legitimate programs, but this is a small price to pay for security. Not only does the sense of self update in near-real-time, but the feature set it uses is completely opaque to an outside observer and also changes from baseline to baseline, so it is highly resistant to reverse engineering and hacking.

The opacity in the IDS is primarily due to the use of Mahalanobis Distance between Principal Component scores to distinguish between legitimate and illegitimate activity, which is another valuable contribution which has not been used in any other IDS

before. Also, the use of PCA to identify key variables loaded on the first principal component as a dimension reduction method, though unorthodox, is highly effective and increases the security of the IDS by changing the feature set in near-real-time while providing significant discrimination.

Adding random noise to columns of zero variance to avoid singularity and infinite value problems was an original solution to a difficult idiosyncrasy in matrix comparisons. This technique is only applicable if individual variables cannot be thrown out for reasons of data and dimension consistency such as existed in this research. In other words, if datasets from different populations must be compared, but columns of zero variance occur in different characteristics of each population, then this technique is valid. Also, the dataset must be normalized, or the noise must be scaled to naturally occurring levels of activity, otherwise the noise may show up as anomalous and result in a false positive.

Finally, setting up a "laboratory" to investigate malware is always a unique experience due to the potential volatility and infective nature of many computer viruses. The use of stand-alone laptops connected with a crossover cable solved a variety of problems, such as the need to measure hardware characteristics, which was not possible using virtual machines, and the requirement for a network connection to run the backdoor client-server application without subjecting the victim system to a real network with possible unknown intrusion attempts. Though undoubtedly this system configuration has been used before, this research further validated this particular technique for testing the effects of known malware on an IDS.

5.4 Future Research

Obviously, this method must be tested using other types of malware and intrusion techniques to ensure it is valid. Also, PAIDS could be tested on different operating systems (LINUX, UNIX, etc.) by analyzing the applicable process data or other available hardware information. Some consideration was given to recovering additional information from the hardware, such as battery life as suggested by Buennemeyer, et al. (2007) or GPS location and velocity (Zhang, Lee, & Huang, 2003), but this would add significant complexity to the experiment. However, this data could easily be incorporated into the proposed PAIDS, and would be a valuable addition for discrimination in mobile ad-hoc networks.

Adding this type of detector to primitive hardware monitors or non-resident software is key to the successful implementation of PAIDS or any truly effective host-based IDS. This would ensure integrity of the IDS as well as provide the necessary system speed to catch an intrusion before too much damage occurred. Obviously, the methods demonstrated in this research are only a proof of concept and they would have to be incorporated into specially designed hardware or software. However, this is beyond the ability of the author and will have to be accomplished by an actual computer engineer.

Optimizing the data collection times and dimension reduction techniques would also be important to fielding this IDS. An experiment determining whether a rolling or periodic window is more appropriate than an event-driven one might help improve the PAIDS solution. Although 10 seconds was definitely too short and 120 seconds too long,

20 seconds was chosen because it seemed to provide a good solution in a limited amount of time, and was not necessarily based on solid mathematical analysis. A rolling window has the danger of incorporating anomalies into the normal baseline, while a periodic window may not be responsive enough to changing conditions; however, it might eliminate the need for user input which has been a major complaint of security systems such as the one found on Microsoft Vista. A variable sized window might also be effective based on the existing conditions of the system.

Dimension reduction could be based on Johnson's secant method instead of retaining data with 80% of the variance, or a smaller percentage may provide the same or better discrimination. Also, Factor Analysis could be performed on the key variables to identify which processes or characteristics are most valuable to anomaly detection. This may also provide insight into the function of many intrusions. Comparison of results when using a covariance versus a correlation matrix when performing Principal Component Analysis might be valuable. Finally, an investigation to prove process characteristics and samples are truly independent, or to prove that introduction of noise into the data does not significantly affect the solution would validate some assumptions made in this research.

Appendix A – Output Data from TaskInfo in Excel Format

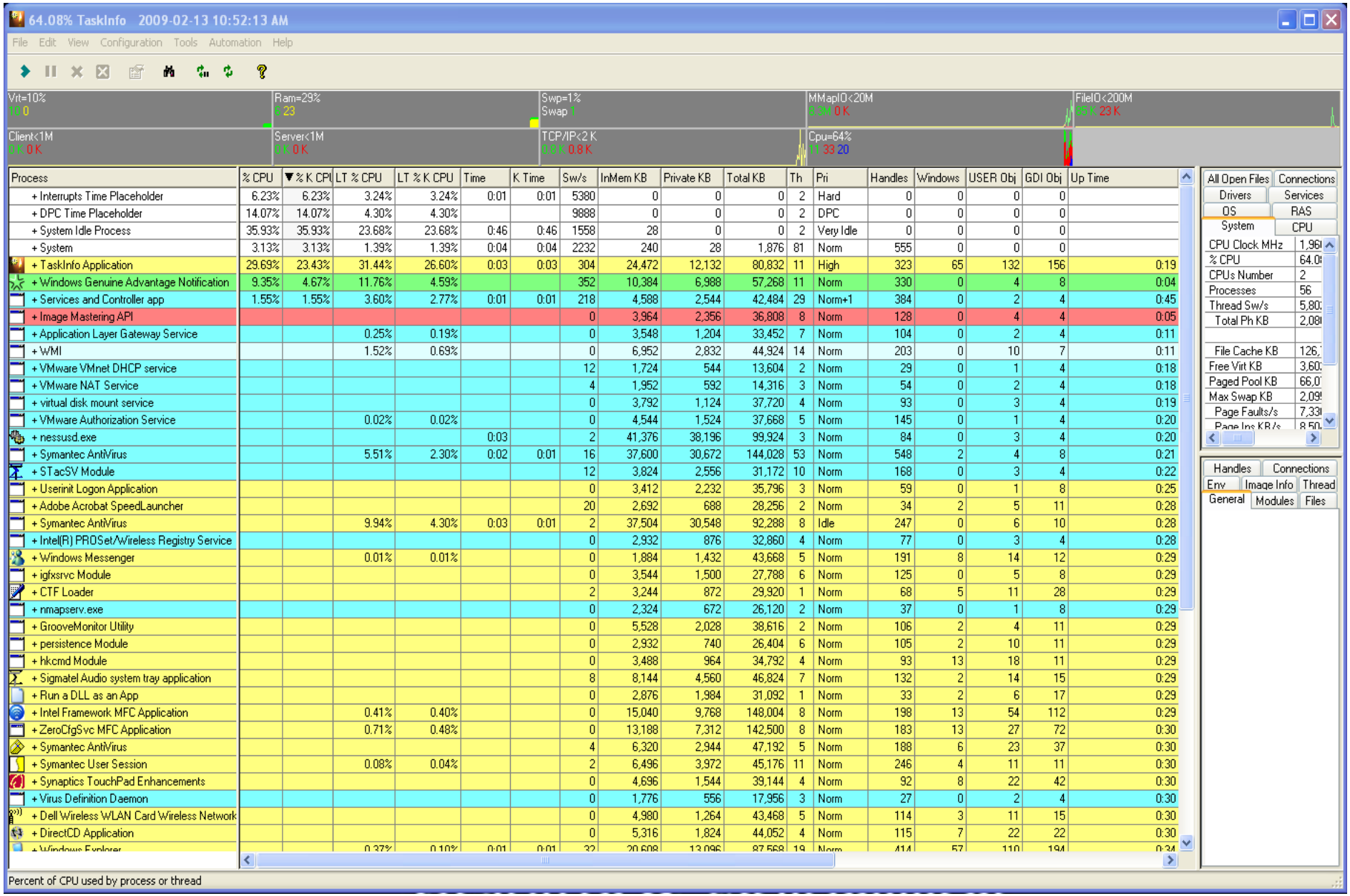
Process	PID	% CPU	% K CPU	Time	K Time	Sw/s	InMem KB	Private KB	Total KB	Th	Pri	OS Ve	State	Handles	Windows	USER Obj	GDI Obj	Start Time	Up Time	Session ID
Interrupts Time				0:03	0:03	149	0	0	0	2	Hard	5.1		0	0	0	0			0
DPC Time				0:01	0:01	31	0	0	0	2	DPC	5.1		0	0	0	0			0
Idle		93.49%	93.49%	32:53:00	32:53:00	3063	28	0	0	2	Very Idle	5.1		0	0	0	0			0
System		0.72%	0.72%	0:09	0:09	938	240	28	1,876	80	Norm	5.1		842	0	0	0			0
smss.exe	1892					0	392	176	3,800	3	ANorm+1	5.1	32 NT N	21	0	0	0	10/31/2008 8:52	34:43:00	0
csrss.exe	232			0:03	0:02	29	4,096	1,788	26,500	12	High	5.1	32 NT N	639	0	63	68	10/31/2008 8:52	34:41:00	0
winlogon.exe	660			0:01	0:01	0	4,580	6,728	54,484	21	High	4	32 GUI	537	0	16	50	10/31/2008 8:52	34:38:00	0
services.exe	756	2.17%	2.17%	0:20	0:16	793	4,472	2,328	38,012	17	Norm+1	4	32 GUI	357	0	2	4	10/31/2008 8:52	34:38:00	0
lsass.exe	768					50	6,768	3,988	42,840	24	Norm+1	4	32 GUI	421	0	2	4	10/31/2008 8:52	34:38:00	0
svchost.exe	1052					11	5,096	2,832	62,472	21	Norm	4	32 GUI	218	0	1	4	10/31/2008 8:52	34:37:00	0
svchost.exe	1112					0	4,628	2,068	38,144	10	Norm	4	32 GUI	417	0	1	4	10/31/2008 8:52	34:36:00	0
svchost.exe	1764			0:02	0:01	13	24,204	15,588	105,112	66	Norm	4	32 GUI	1,566	0	30	11	10/31/2008 8:52	34:36:00	0
EvtEng.exe	1868					0	12,372	8,792	193,604	11	Norm	4	32 GUI	187	0	10	4	10/31/2008 8:52	34:36:00	0
S24EvMon.exe	724	0.73%	0.72%	0:13	0:11	998	11,700	9,224	151,976	8	Norm	4	32 GUI	282	2	13	13	10/31/2008 8:52	34:34:00	0
WLKeeper.exe	924					0	13,032	8,856	139,824	5	Norm	4	32 GUI	189	2	6	12	10/31/2008 8:52	34:34:00	0
svchost.exe	1216					64	2,940	1,156	28,412	6	Norm	4	32 GUI	62	0	1	4	10/31/2008 8:52	34:34:00	0
svchost.exe	1460					0	4,484	1,764	37,972	13	Norm	4	32 GUI	217	0	1	4	10/31/2008 8:52	34:34:00	0
ccSetMgr.exe	1688					0	4,112	3,796	44,992	7	Norm	4	32 GUI	198	0	1	4	10/31/2008 8:53	34:33:00	0
ccEvtMgr.exe	1840					0	2,812	3,952	43,624	19	Norm	4	32 GUI	305	0	1	4	10/31/2008 8:53	34:33:00	0
WLTRYSVC.EXE	1920					0	1,632	432	15,208	2	Norm	4	32 Con	35	0	1	4	10/31/2008 8:53	34:33:00	0
bcmwltry.exe	1996					4	6,560	2,828	48,948	4	Norm	4	32 GUI	156	3	15	15	10/31/2008 8:53	34:33:00	0
spoolsv.exe	176					0	5,188	3,356	45,836	12	Norm	4	32 GUI	128	0	4	4	10/31/2008 8:53	34:33:00	0
SCardSvr.exe	312					7	2,644	928	27,932	7	Norm	4	32 Con	85	0	2	4	10/31/2008 8:53	34:33:00	0
Explorer.EXE	600			0:02	0:02	30	20,732	13,292	91,180	12	Norm	4.1	32 GUI	429	55	118	256	10/31/2008 8:53	34:13:00	0
DirectCD.exe	328					0	5,252	1,800	43,540	4	Norm	4	32 GUI	113	7	21	19	10/31/2008 8:53	34:12:00	0
WLTRAY.exe	1036					3	5,008	1,248	41,476	3	Norm	4	32 GUI	116	3	12	15	10/31/2008 8:53	34:12:00	0
SynTPEnh.exe	1336			0:01		0	4,736	1,544	39,144	4	Norm	4	32 GUI	91	8	22	42	10/31/2008 8:53	34:12:00	0

Appendix A – Output Data from TaskInfo in Excel Format (cont.)

User ID	Reads	Read KB	Rd Rate B/s	Writes	Write KB	Wr Rate B/s	Version	Description	Company
	0	0	0	0	0	0		Interrupts Time Placeholder	
	0	0	0	0	0	0		DPC Time Placeholder	
	0	0	0	0	0	0		System Idle Process	
	78,054	65,451	0	1,287	4,562	0		System	
NT AUTHORITY \ SYSTEM	47	26	0	4	0	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Windows NT Session Manager	Microsoft Corporation
NT AUTHORITY \ SYSTEM	15,361	1,845	0	0	0	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Client Server Runtime Process	Microsoft Corporation
NT AUTHORITY \ SYSTEM	525	2,505	0	131	11	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Windows NT Logon Application	Microsoft Corporation
NT AUTHORITY \ SYSTEM	172	21	0	646	2,714	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Services and Controller app	Microsoft Corporation
NT AUTHORITY \ SYSTEM	6,499	969	1,617	5,874	548	2,048	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	LSA Shell (Export Version)	Microsoft Corporation
NT AUTHORITY \ SYSTEM	92	321	0	13	0	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Generic Host Process for Win32 Services	Microsoft Corporation
NT AUTHORITY \ NETWORK S	100	321	0	21	0	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Generic Host Process for Win32 Services	Microsoft Corporation
NT AUTHORITY \ SYSTEM	5,173	8,420	5,538	3,071	4,570	5,682	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Generic Host Process for Win32 Services	Microsoft Corporation
NT AUTHORITY \ SYSTEM	276	1,065	0	43	21	0	11.1.0.4	Intel(R) PROSet/Wireless Event Log	Intel Corporation
NT AUTHORITY \ SYSTEM	134	538	0	9	0	0	11, 1, 0, 9	Wireless Management Service	Intel Corporation
NT AUTHORITY \ SYSTEM	130	530	0	7	0	0	11.1.0.4	WLANKEEPER	Intel(R) Corporation
NT AUTHORITY \ NETWORK S	5	0	0	3	0	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Generic Host Process for Win32 Services	Microsoft Corporation
NT AUTHORITY \ LOCAL SERV	16	23	0	14	0	0	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Generic Host Process for Win32 Services	Microsoft Corporation
NT AUTHORITY \ SYSTEM	1,971	456	0	28	18	0	103.5.6.3	Symantec Settings Manager Service	Symantec Corporation
NT AUTHORITY \ SYSTEM	3,618	442	0	52	2	0	103.5.6.3	Symantec Event Manager Service	Symantec Corporation
NT AUTHORITY \ SYSTEM	3	0	0	3	0	0		WLTRYSVC.EXE	
NT AUTHORITY \ SYSTEM	100	331	0	3	1	0	4.100.15.8	Dell Wireless WLAN Card Wireless Network Controller	Dell Inc.
NT AUTHORITY \ SYSTEM	19	23	0	18	0	0	5.1.2600.2696 (xpsp_sp2_gdr.050610-1519)	Spooler SubSystem App	Microsoft Corporation
NT AUTHORITY \ LOCAL SERV	23,724	833	400	11,862	309	148	5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)	Smart Card Resource Management Server	Microsoft Corporation
LISXP4913LT \ user	1,050	4,853	0	28	6	0	6.00.2900.3156 (xpsp_sp2_gdr.070613-1234)	Windows Explorer	Microsoft Corporation
LISXP4913LT \ user	67	30	0	5	0	0	5.3.5.10	DirectCD Application	Roxio
LISXP4913LT \ user	3	23	0	2	0	0	4.100.15.8	Dell Wireless WLAN Card Wireless Network Tray Applet	Dell Inc.
LISXP4913LT \ user	1	23	0	0	0	0	8.2.4.6 08Mar06	Synaptics TouchPad Enhancements	Synaptics, Inc.

Appendix B – SubSeven Command Screens

Appendix C – TaskInfo Screenshot



Appendix D – Import Data

```
% This function records system data from freeware product TaskInfo for
% a hardcoded length of time (in seconds) then determines how many
% processes are "static" and collects data for only those processes
% then puts the data into a 2-dimensional array called "cleanoutput"
% which is normalized data with all columns of zero variance changed to % noise.
"rawoutput" is raw data without columns changed.
% ROWS are snapshots in time - intervals are 1 second
% COLUMNS are recorded values of each "static" process in vector format
% for 18 system characteristics in order as follows:
%   1 - % CPU used by each process
%   2 - % Kernel CPU used by each process
%   3 - # Switches to execution of process/second
%   4 - Physical memory used by process in KB
%   5 - Virtual memory used by process in  KB
%   6 - Total virtual address space used by process in KB
%   7 - # Threads currently running in process
%   8 - # Handles opened by process
%   9 - # Windows opened by process
%   10 - # User objects opened by process
%   11 - # GDI objects opened by process
%   12 - # Read operations issued by process
%   13 - Data read by process in KB
%   14 - Read data rate in bytes/sec
%   15 - # Write operations issued by process
%   16 - Data written by process in KB
%   17 - Write data rate in bytes/sec
%   18 - # of processes beyond the "static" processes running at time

function [rawoutput,cleanoutput] = import_data3

for i = 1:2002
  j = num2str(i);
  filename = strcat(j,'.','txt');
      fileID = fopen(filename); % reads output file in order

      % This subroutine written by Maj Larry Nance
        %need to seek down three carriage returns to start on 4th line
      temp = fgets(fileID); %reads first line (don't need this line)
      temp = fgets(fileID); %reads second line (don't need this line)
      temp = fgets(fileID); %reads third line (don't need this line
      %now we are on the third line and start recording values
      quit = 0;
      row = 0;
      while quit == 0
        row = row + 1;
        line = fgets (fileID);  %read the next line of the text file
        if line == -1  %if we have reached the end of file, quit
          quit = 1;
        end

        %Now need to parse the line and assign numbers to variables
        quit1 =0;
        column = 0;
        while quit1 == 0 && quit == 0
          [data,line] = strtok(line,9);  %go to each tab

          %get rid of % signs if they are there
          leng = size(data,2);
          if leng > 0
            if data(leng) == '%'
              data=data(1:leng-1);
            end
          end
```

```
            % get rid of commas
            commaindex = findstr(data,',');
            if ~isempty(commaindex)
              left = data(1:commaindex-1);
              right = data(commaindex+1:leng);
              data = strcat(left,right);
 70         end
            column = column + 1;  %columns

            % put numbers into proper rows and columns
            temp = str2double (data);
            if ~isnan(temp)
              if size(temp,2) == 1
                A(row,column) = temp;
              else
                A(row,column) = 0;
 80           end
            else
              A(row,column) = 0;
            end
            if isempty(data)
              quit1 = 1;
            end
          end %while
        end %while
    % closes output.txt file
 90 close = fclose(fileID);

    % put process ID numbers from first two samples into temp array
    if i == 1
        static = zeros(100,2);
    end
    if i <= 2
        static(1:size(A,1),i) = A(:,2);
    end
        % determine number of "static" processes by comparing PIDs
100 if i == 2
        processes = find(logical(static(:,2)-static(:,1)),1,'first')
    end
    if i > 2

        % remove unwanted columns from matrix
        temp = [A(:,3:4),A(:,7:11),A(:,15:18),A(:,23:28)];
        % calculate and record number of processes running in sample
        temp(1,18)=size(temp,1);
        % remove unwanted rows from matrix
110     temp = temp(1:processes,:);
        % add data as row vector to output matrix in order of processes
        % and characteristics (i.e. starting with first characteristic)
        out(i,:) = reshape(temp,1,[]);
    end

    end;

    % save raw data delete first two lines because they are blank.
    rawoutput = out(3:size(out,1),:);
120
    % create normalized matrix, if no variance exists in column, change to
    % random noise to correct singularity problems later
    maxout = max(rawoutput);
    stand = std(rawoutput);
    for n = 1:size(rawoutput,2)
        if stand(n) (n) <= 0.00000000001
            cleanoutput(:,n) = rand(size(rawoutput,1),1);
        else
            cleanoutput(:,n) = rawoutput(:,n)/maxout(n);
130     end
    end
```

Appendix E – Principal Component Analysis Baseline

```
% Perform Principal Component Analysis on given data matrices, then
% find highly loaded key variables to determine feature set to be
% collected thus reducing dimensionality of original data

% Inputs required:
% X     = data matrix you want to establish baseline from
%         (for instance, first 30 seconds of collected data)

% Outputs:
% key   = key variables with |load|>.2
% z     = Eigenvalues of Correlation Matrix

function [z,key] = PCA_baseline(X)
tic

%create normalized matrices, if no variance exists in column, change
%to random noise to correct singularity problems in corr matrix
maxout = max(X);
stand = std(X);
for n = 1:size(X,2)
    if stand(n) <= 0.00000000001
        X(:,n) = randn(size(X,1),1);
    else
        X(:,n) = X(:,n)/maxout(n);
    end
end

% Find Mean (M) of baseline data then center (XD) and standardize
(XS) data
M = mean(X);
E = diag(ones(size(X,1)));
XD = X - E * M;
D = inv(sqrt(diag(diag(cov(X)))));
XS = XD *D;

% calculate Correlation matrix (cor) eigenvalues (LR) and
eigenvectors (AR)
cor = cov(XS);
[AR,LR] = eig(cor);
AR = fliplr(AR);
LR = fliplr(flipud(LR));

% calculate loading matrix (LM)
LM = AR * sqrt(LR);

% determine key variables and save in vector (key)
n = 1;
for i=1:size(LM,1)
    if abs(LM(i,1)) >= .2
       key(n) = i;
       n = n + 1;
    end
end
% display eigenvalues in descending order
z = diag(LR)';
toc
end
```

Appendix F – PCA/Mahalanobis Distance

```
% Perform Principal Component Analysis on given data matrices then
% find Mahalanobis Distance between feature space over time and plot

% Inputs required:
% base  = baseline data to be compared against
% data  = data matrix if you want to compare to nominal

% Outputs:
% Y     = Component Scores Matrix
% COR   = Indicator Correlation Matrix
% AR    = Eigenvectors of Correlation Matrix
% z     = Eigenvalues of Correlation Matrix
% MD    = Mahalanobis Distance between Principal Component Scores

function [Y,COR,AR,z,MD,key] = PCA_MD(base,data)
tic
% if only one variable is assigned, assume no baseline data matrix
if nargin == 1
    data=base;
end

% create normalized matrices and if no variance exists in column,
% change to random noise to correct singularity problems later
maxout = max(base);
stand = std(base);
for n = 1:size(base,2)
    if stand(n) <= 0.00000000001
       base(:,n) = randn(size(base,1),1);
    else
       base(:,n) = base(:,n)/maxout(n);
    end
end
maxout = max(data);
stand = std(data);
for n = 1:size(data,2)
    if stand(n) <= 0.00000000001
        data(:,n) = rand(size(data,1),1);
    else
        data(:,n) = data(:,n)/maxout(n);
    end
end

% Find Mean (M) of baseline data, then center and standardize data
% for baseline (XSB) and data to be analyzed (XSD)
M = mean(base);
EB = diag(ones(size(base,1)));
ED = diag(ones(size(data,1)));
XDB = base - EB * M;
XDD = data - ED * M;
D = inv(sqrt(diag(diag(cov(base)))));
XSB = XDB *D;
XSD = XDD *D;
```

```
   % calculate Correlation matrix (cor) and find eigenvalues (LR) and
   % eigenvectors (AR)
   cor = cov(XSB);
   [AR,LR] = eig(COR);
   AR = fliplr(AR);
   LR = fliplr(flipud(LR));

60 % determine key variables and save in vector (key)
   n = 1;
   for i=1:size(LM,1)
       if abs(LM(i,1)) >= .2
           key(n) = i;
           n = n + 1;
       end
   end

   % display eigenvalues in descending order
70 z = diag(LR)';

   % retain eigenvectors that contain 80% of variance
   r = 0;
   p = 0;
   zs = sum(z);
   while p <= .8
       r = r + 1;
       p = p + z(1,r)/zs;
   end
80 AR = AR(:,1:r);
   LR = LR(1:r,1:r);

   % calculate loading scores (Y) for retained eigenvalues
   Y = XS * AR;

   % calculate Mahalanobis Distance(MD) for each instantiation and plot
   for i = 1:size(Y,1)
       MD(i) = Y(i,:)*inv(LR)*(Y(i,:)');
   end
90 plot(MD, 'DisplayName', 'MD', 'YDataSource', 'MD'); figure(gcf)

   hold on
   xlabel('Seconds');
   ylabel('Mahalanobis Distance');
   hold off
   toc
   end
```

Appendix G – Factor Analysis/Mahalanobis Distance

```
% Perform Factor Analysis on given data matrices
% and find Mahalanobis distance between data sets
% Inputs required:
% X     = data matrix

% Outputs:
% FS    = Factor Scores Matrix
% FL    = Factor Loading Matrix
% h2    = Commonality Matrix
% PSI   = Uniqueness Matrix
% R     = Actual Correlation Matrix
% Rhat  = Estimation of Correlation Matrix
% Res   = Residual Correlation Matrix
% LR    = Eigenvalues of Correlation Matrix
% AR    = Eigenvectors of Correlation Matrix
% f     = number of retained factors

function [FS,FL,h2,PSI,R,Rhat,Res,LR,AR,f] = FA_MD(base,data)
tic
% if only one variable is assigned, assume no baseline data matrix
if nargin == 1
    data=base;
end

% create normalized matrices and if no variance exists in column,
% change to random noise to correct singularity problems later
maxout = max(base);
stand = std(base);
for n = 1:size(base,2)
    if stand(n) <= 0.00000000001
        base(:,n) = rand(size(base,1),1);
    else
        base(:,n) = base(:,n)/maxout(n);
    end
end
maxout = max(data);
stand = std(data);
for n = 1:size(data,2)
    if stand(n) <= 0.00000000001
       data(:,n) = rand(size(data,1),1);
    else
       data(:,n) = data(:,n)/maxout(n);
    end
end

% Find Mean (M), then center (XD) data:
M = mean(base);
E = diag(ones(size(data,1)));
XD = data - E * M;

% calculate Covariance matrix (R) and find eigenvalues (LR) and eigenvectors (AR)
R = corr(base);
[AR,LR] = eig(R);
AR = fliplr(AR);
LR = fliplr(flipud(LR));

% display eigenvalues in descending order
z = diag(LR)';

% retain eigenvectors that contain 80% of variance
r = 0;
p = 0;
zs = sum(z);
```

```
   while p <= .8
       r = r + 1;
       p = p + z(1,r)/zs;
   end
   AR = AR(:,1:r);
   LR = LR(1:r,1:r);

70 % determine how many factors to keep (f) by iterating until average
   % commonality (h2) is greater than 0.8 or all factors are included
   [m,n] = size(LR);
   h2 = sparse(n,n);
   f = 1;
   while (trace(h2))/n < .8 && n >= f

       % calculate loading matrix (FL) and rotate to maximize variance
       for j = 1:f
           FL(:,j) = sqrt(LR(j,j))*AR(:,j);
80     end
       if f > 1
           FL = rotatefactors(FL,'method','orthomax','maxit',2000);
       end

       % calculate Estimated Correlation Matrix(Rhat), commonality(h2), Uniqueness(PSI)
       h2 = FL*FL';
       PSI = eye(size(h2)) - diag(diag(h2));
       Rhat = h2 + PSI;

90  % calculate factor scores (FS) and Residual Matrix (Res)
    % try different methods until you get a non-singular matrix

    FS = inv(FL'*inv(PSI)*FL)*(FL'*inv(PSI))*XD';%General Least Squares
    %FS = XS*inv(R)*FL;                          %Regression Techniques
    %N = length(X);
    %FS = XS*inv((1/N)*(XS'*XS))*FL;
    %FS = FL'*inv(FL*FL'+PSI)*XD';               %Min Mean Square Error
    FS = FS';
    Res = R - Rhat;
100
    f = f + 1
    end
    f = f - 1;

    % resize eigenvalues to match factor scores
    LR = LR(1:f,1:f);

    % calculate Mahalanobis Distance (MD) for each Factor Score
    for i = 1:size(FS,1)
110 MD(i) = FS(i,:)*inv(LR)*(FS(i,:)');
    end

    % Calculate mean of Mahalanobis Distance and make into vector
    o = ones(size(FS,1));
    MDmean = mean(MD)*o(1,:);

    % Plot MD and MDmean and label
    plot(MD, 'DisplayName', 'MD', 'YDataSource', 'MD'); figure(gcf)
    hold on
120 plot(MDmean, 'DisplayName', 'MDmean', 'YDataSource', 'MD','Color','r' ); figure(gcf)
    xlabel('Seconds');
    ylabel('Mahalanobis Distance');
    legend('MD','Mean');
    text(size(MDmean,1),(MDmean(1)),['MD Mean =
    ',num2str(MDmean(1))],'HorizontalAlignment','center',...
    'BackgroundColor',[0 1 0],'Margin',2);
    hold off
    toc
    end
130
```

Appendix H – 2-way Quadratic Discrimination

```
% Perform Quadratic Discriminant Analysis on given two matrices
% If more than two matrices are required, they must be added to code

% Inputs required:
% X1,X2     = Indicator Data Matrices
% P1,P2     = Prior Probabilities

% Outputs:
% CP        = Indicator Pooled Covariation Matrix
% CM        = Confusion Matrix

function [CP,CM,CM1,DL] = QuadDisc(X1,X2,P1,P2)
tic
% attempt to dummy proof and avoid infinite loops, if input is not specified
if nargin < 4, P2 = .5; end
if nargin < 3, P1 = .5; end
n1=size(X1,1);
n2=size(X2,1);

% Find Mean (M) of each matrix then center (XD):
M1 = mean(X1);
E1 = diag(ones(n1));
XD1 = X1 - E1 * M1;
M2 = mean(X2);
E2 = diag(ones(n2));
XD2 = X2 - E2 * M2;

% create normalized matrices, if no variance exists in column, change to
% random noise to correct singularity problems later
maxout = max(XD1);
stand = std(XD1);
for n = 1:size(XD1,2)
    if stand(n) <= 0.00000000001
       XD1(:,n) = rand(size(XD1,1),1);
    else
       XD1(:,n) = XD1(:,n)/maxout(n);
    end
end
maxout = max(XD2);
stand = std(XD2);
for n = 1:size(XD2,2)
    if stand(n) <= 0.00000000001
        XD2(:,n) = rand(size(XD2,1),1);
    else
        XD2(:,n) = XD2(:,n)/maxout(n);
    end
end

% Calculate Pooled Covariance matrix(CP), build confusion matrices(CM, CM1)
CP = (1/(n1+n2-2))*((XD1'*XD1)+(XD2'*XD2));
CM = [0,0;0,0];
CM1 = [0,0;0,0];

% precalculate to speed up code
ICP = inv(CP);
DCP = log(det(CP));
LP1 = log(P1);
```

```
    LP2 = log(P2);

    % Calculate quadratic discriminant using pooled covariance matrix
    % and compare values to fill in confusion matrix
    for i=1:n1
        d11(i)=(-1/2)*DCP-(1/2)*(X1(i,:)-M1(1,:))*ICP*(X1(i,:)-M1(1,:))'+LP1;
        d12(i)=(-1/2)*DCP-(1/2)*(X1(i,:)-M2(1,:))*ICP*(X1(i,:)-M2(1,:))'+LP2;
        if d11(i) > d12(i)
            CM(1,1)=CM(1,1)+1;
        else
            CM(1,2)=CM(1,2)+1;
        end
    end
    for i=1:n2
        d21(i)=(-1/2)*DCP-(1/2)*(X2(i,:)-M1(1,:))*ICP*(X2(i,:)-M1(1,:))'+LP1;
        d22(i)=(-1/2)*DCP-(1/2)*(X2(i,:)-M2(1,:))*ICP*(X2(i,:)-M2(1,:))'+LP2;
        if d21(i) > d22(i)
            CM(2,1)=CM(2,1)+1;
        else
            CM(2,2)=CM(2,2)+1;
        end
    end

    % precalculate to speed up code
    IC1 = inv(cov(X1));
    IC2 = inv(cov(X2));
    DC1 = log(det(cov(X1)));
    DC2 = log(det(cov(X2)));

    % Calculate quadratic discriminant using individual covariance matrices
    % and compare values to fill in confusion matrix
    for i=1:n1
        d11(i)=(-1/2)*DC1-(1/2)*(X1(i,:)-M1(1,:))*IC1*(X1(i,:)-M1(1,:))'+LP1;
        d12(i)=(-1/2)*DC2-(1/2)*(X1(i,:)-M2(1,:))*IC2*(X1(i,:)-M2(1,:))'+LP2;
        if d11(i) > d12(i)
            CM1(1,1)=CM1(1,1)+1;
        else
            CM1(1,2)=CM1(1,2)+1;
        end
    end
    for i=1:n2
       d21(i)=(-1/2)*DC1-(1/2)*(X2(i,:)-M1(1,:))*IC1*(X2(i,:)-M1(1,:))'+LP1;
       d22(i)=(-1/2)*DC2-(1/2)*(X2(i,:)-M2(1,:))*IC2*(X2(i,:)-M2(1,:))'+LP2;
       if d21(i) > d22(i)
          CM1(2,1)=CM1(2,1)+1;
       else
          CM1(2,2)=CM1(2,2)+1;
       end
    end

    %calculate discriminant loadings
    b = ICP*(M1-M2)';
    Dbx = inv(sqrt(b'*CP*b));
    Dx = inv(sqrt(diag(diag(CP))));
    DL = Dbx*Dx*CP*b;
    toc
    end
```

Appendix I – 3-way Quadratic Discrimination

```
% Perform Quadratic Discriminant Analysis on three given matrices
% If more than three populations exist, the code must be altered
% Inputs required:
% X1,X2,X3  = Indicator Data Matrices
% P1,P2,P3  = Prior Probabilities

% Outputs:
% CP        = Indicator Pooled Covariation Matrix
% CM        = Confusion Matrix
% DL12      = Discriminant Loadings Group 1 vs. Group 2
% DL23      = Discriminant Loadings Group 2 vs. Group 3
% DL13      = Discriminant Loadings Group 1 vs. Group 3

function [CP,CM,CM1,DL12,DL23,DL13] = QuadDisc3(X1,X2,X3,P1,P2,P3)

%attempt to dummy proof and avoid infinite loops, if inputs are not specified
if nargin < 6, P3 = 1/3; end
if nargin < 5, P2 = 1/3; end
if nargin < 4, P1 = 1/3; end
n1=length(X1);
n2=length(X2);
n3=length(X3);

% Find Mean (M) of each matrix then center (XD):
M1 = mean(X1);
E1 = diag(ones(n1));
XD1 = X1 - E1 * M1;
M2 = mean(X2);
E2 = diag(ones(n2));
XD2 = X2 - E2 * M2;
M3 = mean(X3);
E3 = diag(ones(n3));
XD3 = X3 - E3 * M3;

%Calculate Pooled Covariance matrix(CP), confusion matrices(CM, CM1)
CP = (1/(n1+n2+n3-3))*((XD1'*XD1)+(XD2'*XD2)+(XD3'*XD3));
CM = [0,0,0;0,0,0;0,0,0];
CM1 = [0,0,0;0,0,0;0,0,0];

% Calculate quadratic discriminant using pooled covariance matrix
% and compare values to fill in confusion matrix
for i=1:n1
    d11=(-1/2)*log(det(CP))-(1/2)*(X1(i,:)-M1(1,:))*inv(CP)*(X1(i,:)-M1(1,:))'+log(P1);
    d12=(-1/2)*log(det(CP))-(1/2)*(X1(i,:)-M2(1,:))*inv(CP)*(X1(i,:)-M2(1,:))'+log(P2);
    d13=(-1/2)*log(det(CP))-(1/2)*(X1(i,:)-M3(1,:))*inv(CP)*(X1(i,:)-M3(1,:))'+log(P3);
    if d11 > d12 && d11 > d13
       CM(1,1)=CM(1,1)+1;
    elseif d12 > d11 && d12 > d13
       CM(1,2)=CM(1,2)+1;
    else
       CM(1,3)=CM(1,3)+1;
   end
end
for i=1:n2
    d21=(-1/2)*log(det(CP))-(1/2)*(X2(i,:)-M1(1,:))*inv(CP)*(X2(i,:)-M1(1,:))'+log(P1);
    d22=(-1/2)*log(det(CP))-(1/2)*(X2(i,:)-M2(1,:))*inv(CP)*(X2(i,:)-M2(1,:))'+log(P2);
    d23=(-1/2)*log(det(CP))-(1/2)*(X2(i,:)-M3(1,:))*inv(CP)*(X2(i,:)-M3(1,:))'+log(P3);
    if d21 > d22 && d21 > d23
        CM(2,1)=CM(2,1)+1;
    elseif d22 > d21 && d22 > d23
        CM(2,2)=CM(2,2)+1;
    else
        CM(2,3)=CM(2,3)+1;
    end
end
for i=1:n3
    d31=(-1/2)*log(det(CP))-(1/2)*(X3(i,:)-M1(1,:))*inv(CP)*(X3(i,:)-M1(1,:))'+log(P1);
    d32=(-1/2)*log(det(CP))-(1/2)*(X3(i,:)-M2(1,:))*inv(CP)*(X3(i,:)-M2(1,:))'+log(P2);
    d33=(-1/2)*log(det(CP))-(1/2)*(X3(i,:)-M3(1,:))*inv(CP)*(X3(i,:)-M3(1,:))'+log(P3);
    if d31 > d32 && d31 > d33
        CM(3,1)=CM(3,1)+1;
    elseif d32 > d31 && d32 > d33
        CM(3,2)=CM(3,2)+1;
    else
```

```
        CM(3,3)=CM(3,3)+1;
      end
   end
   % Calculate quadratic discriminant using individual covariance matrices
   % and compare values to fill in confusion matrix
80 for i=1:n1
      d11=(-1/2)*log(det(cov(X1)))-(1/2)*(X1(i,:)-M1(1,:))*inv(cov(X1))*(X1(i,:)-M1(1,:))'+log(P1);
      d12=(-1/2)*log(det(cov(X2)))-(1/2)*(X1(i,:)-M2(1,:))*inv(cov(X2))*(X1(i,:)-M2(1,:))'+log(P2);
      d13=(-1/2)*log(det(cov(X3)))-(1/2)*(X1(i,:)-M3(1,:))*inv(cov(X3))*(X1(i,:)-M3(1,:))'+log(P3);
      if d11 > d12 && d11 > d13
          CM1(1,1)=CM1(1,1)+1;
      elseif d12 > d11 && d12 > d13
          CM1(1,2)=CM1(1,2)+1;
      else
          CM1(1,3)=CM1(1,3)+1;
90    end
   end
   for i=1:n2
      d21=(-1/2)*log(det(cov(X1)))-(1/2)*(X2(i,:)-M1(1,:))*inv(cov(X1))*(X2(i,:)-M1(1,:))'+log(P1);
      d22=(-1/2)*log(det(cov(X2)))-(1/2)*(X2(i,:)-M2(1,:))*inv(cov(X2))*(X2(i,:)-M2(1,:))'+log(P2);
      d23=(-1/2)*log(det(cov(X3)))-(1/2)*(X2(i,:)-M3(1,:))*inv(cov(X3))*(X2(i,:)-M3(1,:))'+log(P3);
      if d21 > d22 && d21 > d23
          CM1(2,1)=CM1(2,1)+1;
      elseif d22 > d21 && d22 > d23
          CM1(2,2)=CM1(2,2)+1;
100   else
       CM1(2,3)=CM1(2,3)+1;
      end
   end
   for i=1:n3
      d31=(-1/2)*log(det(cov(X1)))-(1/2)*(X3(i,:)-M1(1,:))*inv(cov(X1))*(X3(i,:)-M1(1,:))'+log(P1);
      d32=(-1/2)*log(det(cov(X2)))-(1/2)*(X3(i,:)-M2(1,:))*inv(cov(X2))*(X3(i,:)-M2(1,:))'+log(P2);
      d33=(-1/2)*log(det(cov(X3)))-(1/2)*(X3(i,:)-M3(1,:))*inv(cov(X3))*(X3(i,:)-M3(1,:))'+log(P3);
      if d31 > d32 && d31 > d33
          CM1(3,1)=CM1(3,1)+1;
110   elseif d32 > d31 && d32 > d33
          CM1(3,2)=CM1(3,2)+1;
      else
          CM1(3,3)=CM1(3,3)+1;
      end
   end

   %calculate discriminant loadings
   %Group 1 vs group 2
   CP12 = (1/(n1+n2-2))*((XD1'*XD1)+(XD2'*XD2));
120 b = inv(CP12)*(M1-M2)';
   Dbx = inv(sqrt(b'*CP12*b));
   Dx = inv(sqrt(diag(diag(CP12))));
   DL12 = Dbx*Dx*CP12*b;

   %Group 2 vs group 3
   CP23 = (1/(n2+n3-2))*((XD2'*XD2)+(XD3'*XD3));
   b = inv(CP23)*(M2-M3)';
   Dbx = inv(sqrt(b'*CP23*b));
   Dx = inv(sqrt(diag(diag(CP23))));
130 DL23 = Dbx*Dx*CP*b;

   %Group 1 vs group 3
   CP13 = (1/(n1+n3-2))*((XD1'*XD1)+(XD3'*XD3));
   b = inv(CP13)*(M1-M3)';
   Dbx = inv(sqrt(b'*CP13*b));
   Dx = inv(sqrt(diag(diag(CP13))));
   DL13 = Dbx*Dx*CP*b;

   end
140
```

Appendix J – Nov21 Test Plan

Run #	Activity Level	Malware Present
1	2	N
2	2	N
3	1	Y
4	2	Y
5	1	Y
6	2	N
7	1	N
8	2	Y
9	2	N
10	1	N
11	2	Y
12	1	Y
13	3	N
14	3	Y
15	1	N
16	3	N
17	3	N
18	2	Y
19	3	Y
20	2	Y
21	1	N
22	3	Y
23	1	Y
24	2	N
25	1	Y
26	3	Y
27	3	N
28	3	N
29	3	Y
30	1	N

Bibliography

AFDD 2-5. (11 January 2005). *Information Operations.*

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

Anderson, D., Frivold, T., & Valdes, A. (1995). *Next-generation Intrusion Detection Expert System (NIDES): A Summary.* Department of Navy, Space and Naval Warfare Systems Command, Computer Science Laboratory. Menlo Park, CA: SRI International.

Arsenin, I. (2008). *IARSN - High Quality System Software*. Retrieved Oct 15, 2008, from IARSN - High Quality System Software: http://www.iarsn.com

Bace, R. (2000, January 4). *An Introduction to Intrusion Detection and Assessment.* Retrieved November 19, 2008, from ICSA Labs: http://www.icsalabs.com/icsa/docs/html/communities/ids/whitepaper/Intrusion1.pdf

Balducelli, C., Bologna, S., Lavalle, L., & Vicoli, G. (2007). Safeguarding information intensive critical infrastructures against novel types of emerging failures. *Reliability Engineering and System Safety , 92* (9), 1218-1229.

Bell, D. E., & LaPadula, L. (1973). *Secure Computer Systems: Mathematical Foundations.* Bedford, MA: MITRE Corp.

Brugger, S. T., Kelley, M., Sumikawa, K., & Wakumoto, S. (2001). Data Mining for Security Information: A Survey. *8th Association for Computing Machinery Conference on Computer & Communications Security.* Philadelphia, PA: U.S. Department of Energy - Lawrence Livermore National Laboratory.

Buennemeyer, T. K., Gora, M., Marchany, R. C., & Tront, J. G. (2007). Battery Exhaustion Attack Detection with Small Handheld Mobile Computers. *International Conference on Portable Information Devices* (pp. 144-148). Orlando, FL: Institute for Electrical and Electronics Engineers.

Chen, Y., Dai, L., Li, Y., & Cheng, X.-Q. (2007). Building lightweight intrusion Detection System Based on Principal Component Analysis and C4.5 Algorithm. *9th International Conference on Advanced Communication Technology* (pp. 2109-2112). Gangwon-Do, South Korea: Institute of Electrical and Electronics Engineers Inc.

da Silva, A. P., Martins, M., Rocha, B., Loureiro, A., Ruiz, L., & Wong, H. (2005). Decentralized Intrusion Detection in Wireless Sensor Networks. *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless*

and mobile networks (pp. 16- 23). Montreal, Canada: Association for Computing Machinery.

Debar, H., Becker, M., & Siboni, D. (1992). A Neural Network Component for an Intrusion Detection System. *1992 IEEE Symposium on Security and Privacy* (p. 240). Institute of Electrical and Electronics Engineers.

Denning, D. (1986). An Intrusion Detection Model. *Symposium on Security and Privacy* (pp. 118-131). Oakland, CA: Institute of Electrical and Electronics Engineers.

Dillon, W. R., & Goldstein, M. (1984). *Multivariate Analysis Methods and Applications.* New York, NY: John Wiley & Sons.

EICAR. (2006, September 7). *The Anti-Virus or Anti-Malware Test File*. Retrieved October 31, 2008, from EICAR: http://eicar.org/anti_virus_test_file.htm

Fisher, S. R. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics , 7*, 179-188.

Forrest, S., Allen, L., Perelson, A. S., & Cherukuri, R. (1994). Self-nonself discrimination in a computer. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (pp. 202-212). Oakland, CA: Institute of Electrical and Electronics Engineers Inc.

Fox, A., Kiciman, E., & Patterson, D. (2004). Combining Statistical Monitoring and Predictable Recovery for Self-Management. *Proceedings of the 1st ACM SIFSOFT workshop on self-managed systems* (pp. 49-53). Newport Beach, CA: Association for Computing Machinery.

Francis, P., Leon, D., Minch, M., & Podgurski, A. (2004). Tree-Based Methods for Classifying Software Failures. *Proceedings - International Symposium on Software Reliability Engineering* (pp. 451-462). Saint-Malo, France: Institute of Electrial and Electronics Engineers Inc.

GAO. (1996). *Information Security - Computer Attacks at Department of Defense Pose Increasing Risk.* Accounting and Information Management Division, Defense Information Security Agency. Washington D.C.: United States Government Accountability Office.

GAO. (2008). *Information Security - Progress Reported, but Weaknesses at Federal Agencies Persist.* United States Government Accountability Office. Washington D.C.: United States Government Accountability Office.

Hart, S. (2007). *APHID: Anomaly Processor in Hardware for Intrusion Detection.* AFIT, ENG. Wright-Patterson AFB, OH: Air Force Institute of Technology.

Hassan, H., Mahmoud, M., & El-Kassas, S. (2006). Securing the AODV Protocol Using Specification-Based Intrusion Detection. *Q2SWinet 2006: 2nd ACM International Workshop on Quality of Service and Security in Wireless and Mobile Networks* (pp. 33-36). Terromolinos, Spain: Association for Computing Machinery.

Huang, Y.-a., & Wenke, L. (2003). A Cooperative Intrusion Detection System for Ad Hoc Networks. *1st ACM Workshop Security of Ad-hoc and Sensor Networks* (pp. 135-147). Fairfax, VA: Association for Computing Machinery.

Huang, Y.-a., Fan, W., Lee, W., & Yu, P. (2003). Cross-Feature Analysis for Detecting Ad-Hoc Routing Anomalies. *The 23rd IEEE International Conference on Distributed Computing Systems* (pp. 478-487). Providence, RI: Institute of Electrical and Electronics Engineers.

Hussein, M., & Zulkernine, M. (2007). Intrusion Detection Aware Component-Based Systems: A Specification-Based Framework. *Journal of Systems and Software , 80* (5), 700-710.

Johnson, R. (2008). *Improved Feature Extraction, Feature Selection, and Identification Techniques That Create a Fast Unsupervised Hyperspectral Target Detection Algorithm.* Wright-Patterson AFB, OH: Air Force Institute of Technology (Air University Press).

Kabiri, P., & Ghorbani, A. (2005). Research on Intrusion Detection and Response: A Survey. *International Journal of Network Security , 1* (2), 84-102.

Khan, L., Awad, M., & Thuraisingham, B. (2007). A New Intrusion Detection System Using Support Vector Machines and Hierarchical Clustering. *The VLDB Journal , 16* (4), 507-521.

Kim, D. S., Nguyen, H.-N., & Park, J. S. (2005). Genetic Algorithm to Improve SVM Based Network Intrusion Detection. *Proceedings - 19th International Conference on Advanced Information Networking and Applications. v 2*, pp. 155-158. Taipei, Taiwan: Institute of Electrical and Electronic Engineers Inc.

Lee, W., Nimbalkar, R., Yee, K., Patil, S., Desai, P., Tran, T., et al. (2000). A Data Mining and CIDF Based Approach for Detecting Novel and Distributed Intrusions. In H. Debar, L. Me, & S. F. Wu (Ed.), *Recent Advances in Intrusion Detection, 3rd International Symposium. 1907*, pp. 49-65. Toulouse, France: Springer.

Mahalanobis, P. C. (1936). On the Generalised Distance in Statistics. *Proceedings of the National Institute of Sciences of India , 2*, 49-55.

Mell, P., Hu, V., Lippmann, R., Haines, J., & Zissman, M. (2003). *An Overview of Issues in Testing Intrusion Detection Systems.* Defense Advanced Research Projects Agency. National Institue of Standards and Technology.

Merkle, L. D., Carlisle, M. C., Humphries, J. W., & Lopez, D. W. (2002). EA-Based Approach for Detecting Stealthy Attacks. *Proceedings of the 2002 IEEE Workshop on Information Assurance.* West Point, NY: Institute of Electrical and Electronic Engineers Inc. .

Montgomery, D. C. (1991). *Introduction to Statistical Quality Control* (2 ed.). New York, NY: John Wiley & Sons.

Mott, S. (2007). *Exploring Hardware-based Primitives to Enhance Parallel Security Monitoring in a Novel Computing Architecture.* AFIT, ENG. Wright-Patterson AFB, OH: Air Force Institute of Technology (Air University Press).

NIAC, N. I. (2003, February). The National Strategy to Secure Cyberspace. http://www.whitehouse.gov/pcipb/.

Northcutt, S. (1999). *Network Intrusion Detection - An Analyst's Handbook.* Indianapolis, IN: New Riders.

Pillai, M. M., Eloff, J. H., & Venter, H. S. (2004). An Approach to Implement a Network Intrusion Detection System Using Genetic Algorithms. *Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries. 75*, pp. 221-228. Stellenbosch, South Africa: South African Institute of Computer Scientists and Information Technologists.

Sekar, R., Gupta, R., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., et al. (2002). Specification Based Anomaly Detection: A New Approach for Detecting Network Intrusions. *Proceedings of the 9th ACM Conference on Computer and Communication Security* (pp. 265-274). Washington D.C.: Association for Computing Machinery.

Shyu, M.-L., Quirino, T., Xie, Z., Chen, S.-C., & Chang, L. (2007). Network Intrusion Detection Through Adaptive Sub-Eigenspace Modeling in Multiagent Systems. *ACM Transactions on Autonomous and Adaptive Systems , 2* (3), 9:1-37.

Skoudis, E., & Zeltser, L. (2004). *Malware - Fighting Malicious Code.* Upper Saddle River, NJ: Prentice Hall.

Snapp, S., Brentano, J., Dias, G., Goan, T., Heberlein, L. T., Ho, C.-L., et al. (1991). DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and an Early Prototype. *Proceedings of the 14th National Computer Science Conference,* (pp. 167-176). Washington D.C.

Tarakanov, A. (2008, May). Immunocomputing for Intelligent Intrusion Detection. *IEEE Computational Intelligence Magazine* , 22-30.

Uniblue. (2008). *Uniblue ProcessLibrary.com*. Retrieved December 4, 2008, from Processes Directory: http://www.processlibrary.com/directory/

Uniblue. (2008). *Uniblue www.uniblue.com*. Retrieved December 4, 2008, from Products: http://www.liutilities.com/products/wintaskspro/processlibrary/

USCERT. (2008, Nov 7). *US Computer Emergency Response Team.* Retrieved Jan 23, 2009, from Quarterly Trends and Analysis Report: http://www.us-cert.gov/press_room/trendsanalysisQ408.pdf

Wikipedia. (2008, December 13). *Sub7*. Retrieved December 16, 2008, from Wikipedia: http://en.wikipedia.org/wiki/Sub7

Wong, W.-T., & Lai, C.-Y. (2006). Identifying important features for intrusion detection using Discriminant Analysis and Support Vector Machine. *International Conference on Machine Learning and Cybernetics. v 2006*, pp. 3563-3567. Dalian, China: Institute of Electrical and Electronics Engineers Computer Society.

Wu, N., & Zhang, J. (2006). Factor Analysis Based Anomaly Detection and Clustering. *Decision Support Systems , 42* (1), 375-389.

Yongguang, Z., Wenke, L., & Huang, Y.-a. (2003). Intrusion Detection Techniques for Mobile Wireles Networks. *Wireless Networks , 9* (5), 545-556.

Zhang, Y., Lee, W., & Huang, Y.-a. (2003). Intrusion Detection Techniques for Mobile Wireless Networks. *Wireless Networks , 9* (5), 545-556.

Vita

Major Shilland graduated from the University of Michigan with a BS in Aerospace Engineering in 1992. He entered the Air Force a year later as a distinguished graduate of Officer Training School. His first assignment was with the 1st Space Operations Squadron as a satellite operations officer; he performed command and control for launch, early orbit, and station keeping on: DSP, DMSP, GPS, and TAOS spacecraft. He attended Undergraduate Navigator Training with the Navy at Pensacola NAS, and transitioned to the flying world as a navigator on the B-52H. After upgrading to Radar Navigator (bombardier) and earning distinguished graduate honors at Squadron Officer School, he volunteered as an Air Liaison Officer and was a distinguished graduate again at the Joint Firepower Control Course.

He then trained with the Special Forces and deployed with 5th Group during the first days of Operation IRAQI FREEDOM, for which he was awarded a Bronze Star. He has also been deployed as Director of Operations with the 19th Air Support Operations Squadron in the 101st Airborne Division Headquarters, Mosul, Iraq, after which he returned to flying as an instructor and flight commander. His most recent deployment was as Assistant Director of Operations with the 23rd Expeditionary Bomb Squadron in support of Operation ENDURING FREEDOM, flying 327 combat hours out of Diego Garcia, British Indian Ocean Territories.

Major Shilland was accepted to the Air Force Institute of Technology as an 18-month Intermediate Developmental Education student in the Operations Research department in 2007. He was recently promoted to Lieutenant Colonel and starts studies at the School of Advanced Air and Space Studies in Montgomery, Alabama in July 2009.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
02-26-2009	**Master's Thesis**	Jul 2007 - Mar 2009

4. TITLE AND SUBTITLE

HOST-BASED MULTIVARIATE STATISTICAL COMPUTER OPERATING PROCESS ANOMALY INTRUSION DETECTION SYSTEM (PAIDS)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Shilland, Glen R., Major, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Street, Building 642
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/09-15

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Steven K. Rogers
Air Force Research Laboratory
Sensor and Information Directorate
2241 Avionics Circle
WPAFB OH 45433-7334

Dr. N. Adam Fraser
Air Force Information Operations Center
Information Operations Technologies
102 Hall Blvd, Suite 345
San Antonio, TX 78243-7038

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RY, AFIOC/IOT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT Most intrusion detection systems rely on signature matching of known malware or anomaly discrimination by data mining historical network traffic. This renders defended systems vulnerable to new or polymorphic code and deceptive attacks that do not trigger anomaly alarms. A lightweight, self-aware intrusion detection system (IDS) is essential for the security of government and commercial networks, especially mobile, ad-hoc networks (MANETs) with relatively limited processing power. This research proposes a host-based, anomaly discrimination IDS using operating system process parameters to measure the "health" of individual systems. Principal Component Analysis (PCA) is employed for feature set selection and dimensionality reduction, while Mahalanobis Distance (MD) and is used to classify legitimate and illegitimate activity. This combination of statistical methods provides an efficient computer operating process anomaly intrusion detection system (PAIDS) that maximizes detection rate and minimizes false positive rate, while updating its sense of "self" in near-real-time.

15. SUBJECT TERMS host-based computer intrusion detection system, multivariate statistical anomaly detection, Mahalanobis Distance, Principal Component Analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kenneth W. Bauer, Jr., PhD, ENS
U	**U**	**U**	**UU**	142	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636 x4328 kenneth.bauer@afit.edu