2012

Linear analysis of binary data as an aid to anomaly detection

Marc Leonard Santoro

LINEAR ANALYSIS OF BINARY DATA AS AN AID TO ANOMALY DETECTION

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Marc Leonard Santoro

December 2012

LINEAR ANALYSIS OF BINARY DATA AS AN AID TO ANOMALY DETECTION

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Marc Leonard Santoro

December 2012

Approved by:

[redacted]

Dr. Keith Evan Schubert, Advisor, School of Computer Science and Engineering

11/27/12
Date

[redacted]

Dr. Ernesto Gomez

[redacted]

Dr. Richard Botting

ABSTRACT

The vast majority of contemporary methods to detect network level intrusions are primarily signature-based, require significant processing power, and rarely perform deep packet inspection. Though research has been conducted regarding the use of General Purpose Graphics Processing Units (GPGPUs) to spread the processing load across numerous stream processors, this research has focused largely on spreading packet load to increase throughput, rather than the analysis of the packets themselves. Using singular value decomposition to examine the binary structure of the individual packets, it is possible to perform frequency analysis to identify and classify data, thereby potentially allowing for a new type of paradigm for malicious packet/data identification. Combined with the use of GPGPUs, this should allow for faster, less expensive hardware solutions, and furthermore, would free up core processor resources so that additional functionality could be built in.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. Schubert for all of his assistance, his insight, and most especially his patience in helping me better understand the complex mathematical concepts contained herein, as well as pushing me to develop a deeper understanding of mathematics in general. I would like to thank Dr. Gomez and Dr. Botting for participating on my committee. I would also like to thank the entire faculty of the CSUSB School of Computer Science, as well as all of the friends and colleagues that I met during my tenure at CSUSB who played an equally integral role in my education. Finally, my deepest gratitude extends to my wife, Cassie, whose love and support made this whole journey possible.

DEDICATION

To Miles and Cassie

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Overview

Hackers. What started as a word to describe individuals with a strong skill set in computing is now a ubiquitous moniker used to describe those who would seek to use their computing skills to perpetrate malicious acts upon their fellow man. The term is so main stream now that it has become the frequent subject of nightly news reports, where hacker groups like Anonymous and LulzSec get top billing as conglomerates of supposed malcontents intent on disrupting major corporations and governmental agencies according to their capricious and convoluted agendas. Furthermore, it is now an accepted fact that various nation states are engaging in "Cyber Warfare." China has been implicated as sponsoring various groups in their efforts to penetrate Western organizations and infrastructure, and the US has all but admitted to targeting numerous facilities in the Middle East. In recent years cyber attacks have been used to among other things: shut down nuclear power plants, disrupt communications during warfare, compromise millions of consumers' financial information, and perhaps most significantly to steal intellectual property on the level of an estimated loss that numbers in the billions of US dollars [8] [12] [2] [26]. The problem has become so pervasive and serious that the US Air Force has created an entirely new command wing known as the "Air Force Cyberspace Command" [11]. In 2009, the

US Pentagon set up the US Cyber Command to defend American assets from foreign and native entities [4]. Furthermore in 2011 the Pentagon outlaid plans that defined specific acts of cyber warfare as equivalent to conventional attacks and warranting a military response of a conventional nature [14]. As cyber warfare becomes an accepted theatre of war, and viruses such as Stuxnet, Flame, and Gauss become the norm rather than exceptions, increasingly sophisticated means of detection will need to be developed to protect critical infrastructure from attack.

Cyber warfare/crime comes in many flavors. The two most common involve distributed denial of service attacks in which a targeted server is flooded with requests from clients, often acting as part of a bot net under the control of a master server, that ultimately overwhelms the servers capacity to handle requests and causes a system failure. The second most common form of attack involves attempts to gain administrative access to a target computer for purposes of data mining, destruction of software/hardware, or to gain access to infrastructure command and control devices. This type of attack can be launched through a multitude of different vectors as diverse as computer viruses like trojan horse programs and worms, SQL injection attacks on databases, cross-site scripting attacks on websites, backdoors written into software, the exploitation of improperly written and unpatched software, and a host of other means.

According to SANS, one of the leading internet security consortiums, "Attacks against web applications constitute more than 60 percent of the total attack attempts observed on the Internet" [7]. As web based attacks are the most numerous, it becomes crucially important to develop a means of protecting internet exposed systems

from outside attack. However, it is extremely important to note that when protecting web-exposed resources it is paramount that one only prevents harmful traffic from penetrating the system, as system availability is often mission critical on internet exposed systems. The backbone of internet communication is through the TCP/IP protocol stack, and its transmission of discrete "packets" of encoded binary data across networks, thus it is within this stack that the identification and nullification of threats can most effectively occur.

With the constant and rapidly changing dynamics of cyber threats, it is crucial to leverage computational abilities to assist in the identification and elimination of network level threats in as quickly and efficiently a manner as is possible. Many packet filters use keyword analysis at the application level using machine learning in combination with statistical models such as Bayesian analysis. Spam filters for your email are a popular example of this. Others focus on examining header information to analyze and filter based on IP address or protocol. Firewalls are a good example of this type of analysis. Deep packet inspection, where the contents of the payload of a packet are examined, typically requires a more sophisticated, less automated methods of analysis, typically where an individual is either hand examining a packet capture, or writing a quick script to parse a packet capture to look for a specific signature. This is an extremely time consuming process requiring a pretty high level of technical expertise. As a result, many organizations do not have this level of analysis at their disposal. The simple fact of the matter is that the practices of many security professionals tend to focus on the so-called low hanging fruit - attackers using commonplace, well established, less sophisticated methods, with the understanding

that there simply will be unavoidable intrusions from the more skilled individuals. It has repeatedly been asserted by top security professionals that there are two kinds of organizations in the wilds of the internet: those that have been hacked, and those that have been hacked but don't know it.

The proposed solution that is the focus of this research concerns analyzing network traffic based on packet inspection at layers three and four of the OSI model (the network and transport layers). At this point in communication, the ethernet and IP headers can be stripped off the packet and the underlying TCP header and more specifically the payload itself can be analyzed. By training the system, it should be possible to develop "known good" data patterns in the payload data, which can be subsequently used in the analysis process to identify good traffic versus bad traffic. While on the surface this would seem to be the equivalent of establishing a "signature" for detection, it is in fact more of a positive security model, in which a white list of acceptable data is gathered against which all incoming data can be compared to. Examining data at the binary level requires very specific mathematical analysis. There are a number of methods that can be employed, such as statistical data mining, neural networks, and machine learning based algorithms such as Bayesian analysis. The algorithm that is employed here, singular value decomposition, is borrowed from linear algebra, and is frequently used in signal analysis. Singular Value Decomposition is basically a factorization of a matrix which yields three component matrices that define the column space (range) and row space (domain) of a matrix, as well as (sigma) which is the weighting from row to column space. Packet payload data is arranged into vectors, which are then compiled into matrices. A set of known good traffic is

developed based off of the domain space after the training data has been subjected to the singular value decomposition. Subsequent traffic can be compared against this by conducting vector dot products calculations between the incoming packet data and the known good packet data. Ideally, new packet vectors that point in the same "direction" as the known good traffic vectors should be allowed to pass, while vectors that are orthogonal with regard to known good traffic should be dropped. There should be a third subset of data also, which consists of packets that don't exactly match the patterns for the good or bad traffic, which could be classified as unknown. This type of algorithm is ideally suited to stream processing via GPGPUs, as it consists of matrix operations, resulting in a binary process of classifying the data as either good or bad based off of a single set of calculations. No prior results are required, thus making the process extremely suited for parallelization.

The evolutionary arms-race between those that would try to penetrate systems for illegal purposes and those that strive to defend those systems has been ongoing for decades with the levels of sophistication growing at a rapid pace. At any given time, either one side or the other has the advantage in a particular venue. The hackers will find an exploit in a given protocol, then the defenders will patch the protocol, it goes on and on. The idea behind this research is to take yet another incremental step in furthering the ability of the defense by providing a potential new methodology off of which new tools can be developed.

1.2 History

The traditional method for protecting network infrastructure is through the use of firewalls. Most firewalls focus on the limiting of traffic by port numbers, and known traffic behavior patterns. Sadly, the only firewall truly effective at preventing all attacks is the one that blocks all traffic. Obviously, due to the increasingly interconnected nature of how businesses and individuals operate on the internet, this is not a feasible solution. Unfortunately, leaving even a single port exposed to the internet can afford attackers the opportunity to penetrate a system. Thus it has become necessary to implement secondary systems like Intrusion Detection Systems and Intrusion Prevention Systems such as SNORT, which are often integrated with firewalls and other network level devices. Current supplemental strategies that include Intrusion Detection Systems are largely utilizing negative security models (blacklists), implementing signature based detection with analysis occurring on a per-packet basis. Signature based detection is inherently flawed as it relies exclusively on analysis of prior attack vectors to create signatures that are then dispersed to clients so that they may identify and prevent any future attacks that match a given signature; therefore, they can never detect zero-day (never before seen) threats. Additionally, failure to keep the signature database consistently updated on given hardware/software effectively renders this type of protection null. Though some of the more sophisticated enterprise appliances and software platforms do employ limited use of positive security models (white lists) and heuristics to analyze the behavior of network traffic; unfortunately, even the more sophisticated solutions are somewhat statically limited in their ability to identify threats, requiring constant human directed re-training of what is and

what is not a threat to keep profiles current as threat patterns continually morph due to attackers identifying traps and threat detection methodologies, F5's "Application Security Model" is a good example of this. Positive security models and heuristically based analysis are generally more effective at uncovering new vectors of attack that simple signature blacklists would miss. It is, in fact, often the case that small changes made to existing malicious code can enable it to bypass signature detection. Since traffic and behavior patterns would remain consistent, it is easy to argue that looking at traffic patterns and behavior would be the preferred method of detection.

Essentially traffic patterns are signals, digital streams of ones and zeros that demonstrate a variety of patterns depending on what is being transmitted. Signal filtering has been around for decades. Traditionally digital signal processing has used singular value decomposition as a primary means of frequency analysis to filter out specific bands of frequencies. This is done based on analysis of the singular values of a decomposed matrix comprised of digital signals. For example, "noise" can often be identified in a signal by examining the smallest singular values of a decomposition, and the subsequent removal of these frequencies based on this identification is a tried and true method for "cleaning" up digital signals. This is in fact one of the major reasons that digital voice and video signals sent over the airwaves, phone lines, and internet networks of the world are comprehensible at all on the receiving end. In light of the success of this methodology, it was supposed that the same principals could be applied to internet traffic that was not simply voice or video in an effort not to clean up the "signals" but to use this sort of classification as a means to separate traffic into disparate categories based on the supposed "good" or "bad" nature of the traffic.

1.3 Literature Survey

The literature survey consisted of three separate fields of research. 1) The historical use of Singular Value Decomposition in Digital Signal Processing, 2) The use of different algorithms for the purpose of identifying and classifying packet traffic for purposes of providing a defensive security posture for use with Intrusion Detection Systems, and Intrusion Prevention Systems. 3) The use of GPGPUs in aiding the task of deep packet analysis, specifically with regard to the potentials for parallelization.

Research into Intrusion Detection System and Intrusion Prevention System is a rapidly expanding field, with more sophisticated methods and theories being introduced frequently. Past research has focused on largely on signature detection. Vigna, Cassell, and Fayram focused on a distributed agent based Intrusion Detection System that largely used signature based techniques, though there was some heuristical user analysis [23]. Viadya introduced and patented a dynamic signature based Intrusion Detection System in 1998 [22]. In 2001 Patton et. al. made the following statement: "The vulnerability of signature-based Intrusion Detection System to high false positive rates has been well documented but we go further to show (at a high level) how packets can be crafted to match attack signatures such that a alarms on a target Intrusion Detection System can be conditioned or disabled and then exploited" in their paper on the inherent flaws of signature based Intrusion Detection System [13]. Building off of past research, more modern theory has tended to incorporate aspects of prior research such as distributed agent networks, signature based identification, and user-level heuristical analysis; however, their is increased focus on packet-level, real time, line speed analysis, through the use of models like neural networks using

Adaptive Resonance Theory, as well as matrix decomposition through methods such as principal component analysis, and singular value decomposition [25] [15].

Traditional packet classification tends to focus almost entirely on classifying the types of traffic and traffic flows based entirely on layer 2 and 3 header information [6] [27]. Classic algorithms like the grid-of-tries, cross-producting, bit vector searching, and recursive flow classification are all examples of this [18] [5] [1] [17]. There is a substantial body of work related to these algorithms. Liu et. al. explores both the concept of parallelism for purposes of accelerating classification speeds, as well as developing a novel model for classification [10]. The technique described is largely a variation on the popular Recursive Flow Classification Algorithm, and the method of classification is entirely header based and uses a bitmap based recursive search of tables of classification data. Singh, et. al. use a variation of the popular HiCuts algorithm in their research which focuses on "multidimensional cutting" using hypercubes [16]. The model is a decision tree based analysis that is again focused on the header information of the packet.

The above models of packet classification are largely designed to aid routers in routing traffic. There is also some firewall-related value in that traffic flows can be monitored, and header based rules sets applied at fast speeds. However, the header is generally the smallest part of the packet, what about the payload. What research has been conducted examining payload information? The answer is: not much, but there are some. Wang and Stolfo describe a method called PAYL, which uses statistical analysis of byte frequency distributions for packet payloads to determine the legitimacy of packet data [24]. Thorat et. al expanded on the Payl research by ap-

plying a divide and conquer approach to Wang and Stolfo's method by partitioning large packets such as those associated with HTTP to increase the efficiency of the algorithm [19]. This however is quite a different methodology to the linear algebraic methods described herein.

With regard to linear algebra and data analysis, there is substantial evidence that eigenvalue analysis is a successful model in conducting accurate real time analysis of large data streams. Several research papers have been produced that use this particular model to increase the success rate of identifying threats [25] [15] [20]. Furthermore, there is a multi-billion dollar tech giant that utilizes this approach in their page ranking algorithm namely, Google. The hypothesis behind this page ranking algorithm is the subject of a paper by Larry Page and Sergey Brin from Stanford that is now quite legendary [3]. Eigenvalue analysis focuses on more linear independent sets than Singular Value Decomposition, and does not include substantial relevance inside the Null Space.

The focus of this research is strictly on Singular Value Decomposition analysis of packet payload data for purposes of identifying anomalies, which after much research appears to be a novel approach to this specific problem.

1.4 Contributions

It is hypothesized that using Singular Value Decomposition as a means to analyze payload data within packets at layers three and four of the OSI model that it will be possible to detect anomalous data. The purpose of this detection being the ability to use such an algorithm to establish a security model wherein a set of training

data based on known good traffic can be established, and the subsequent analysis of unknown traffic using this algorithm can classify data as either good or bad. This would provide an additional methodology to those that are existing that is based on a positive security model that utilizes white list techniques, which are far more effective at identifying new attack vectors than many of the current blacklist techniques. Using ICMP as an initial proof of concept, the algorithm shall be developed and refined as needed. ICMP stands for Internet Message Control Protocol. The ultimate goal would be to achieve an algorithm that is able to identify anomalous traffic patterns in protocols more complex than ICMP. Some of the different protocols and encoding models to be explored include: ICMP, HTTP, ASCII, and Unicode.

Should this research prove successful, it may lead to the development of a new paradigm of deep packet inspection techniques; techniques, that based on fundamentals of their mathematical algorithms, lend themselves well to parallelization and the use of stream processors, which are the predominant methods for maintaining the relevance of Moore's law with regard to increases in computing capacity.

2. THEORY

2.1 Definitions

This section will provide definitions for some of the terms that will be used throughout the rest of the paper.

Column Space of a matrix: This is the set that is defined by all of the linear combinations of the column vectors of a given matrix.

Diagonal Matrix: A matrix whose only non-zero elements lie on the main diagonal of the matrix.

GPGPU: This acronym stands for General Purpose Graphics Processing Unit. These are processors that were originally designed specifically for graphics computation within commodity hardware that can also be utilized outside of graphics processing for a multitude of mathematical computing purposes, specifically ones that can exploit the GPGPU's ability to provide a high level of parallelization for matrix operations.

HTTP: Hypertext Transfer Protocol. HTTP is the main application protocol by which information is encoded and sent across the internet. It is based on the client-server model where a client, such as a web browser, sends a request for information contained on a web page (commonly known as a GET or POST request) to a server, which then responds with the information requested. HTTP is typically associated with port 80, though it can be configured to run on different ports.

ICMP: Internet Control Message Protocol. ICMP is one of the oldest, yet still widely utilized, networking protocols. ICMP is used primarily to debug network issues. The most common usage of ICMP involves the "ping" command, which is used to check if there is a communication line open between two machines.

Intrusion Detection System (IDS): A security device or application that alerts users based on one, or a combination of the following: signatures, heuristics, or anomalous behavior. An Intrusion Detection system can be either network based (referred to as NIDS) or host based (referred to as HIDS).

Intrusion Prevention System (IPS): A security device or application that can actively deny/prevent traffic or code execution based on one or a combination of the following: signatures, heuristics, or anomalous behavior. An Intrusion Prevention System can be either network based (referred to as NIPS) or host based (referred to as HIPS).

Linear Combination: This is the method by which a space is defined in Linear Algebra. It consists of taking a term or terms and multiplying the term(s) by any constant to determine all of the possible results that the multiplication of any constant with the given term(s) can create. For example, given the formula $c + d$, where c and d are vectors, the linear combinations of this formula could be represented by the new formula $ac + bd$ where a and b are any given constant.

Matrix: A rectangular arrangement of elements. These elements can be composed of numbers, expressions, or other mathematical symbols. Matrices can be used to represent a system of equations, and can be manipulated through a multitude of operations to solve such systems.

Orthogonal: This is a property in mathematics that is used to define a specific relationship between two or more lines/vectors. If A is a line/vector, then line/vector B is orthogonal with respect to A if and only if A is perpendicular to B at their point of intersection. In other words there is an angle of $\pi/2$ radians at the intersection of A and B. Two vectors are orthogonal if there inner product is zero.

Orthogonal Matrix: A square matrix whose column and row vectors are orthonormal to each other.

OSI Model: Open Systems Interconnection model. This is a model developed by the International Organization for Standardization that is used to separate the different stages of communication between computers into multiple layers with similar functions grouped within the same layer. There are 7 different layers in the model. Layer 1 is the physical layer, which consists of the physical components such as cabling and network adapters. Layer 2 is the data link layer which describes the physical addressing of the various components using things like MAC Addresses. Layer 3 is the network layer, which provides the logical addressing of the various components using IP Addresses and handles the manipulation of traffic at the IP Address level. Layer 4 is the transport layer, which provides end to end communication through means such as the Transfer Control Protocol. Layer 5 is the session layer, which controls the communication streams over which different systems communicate. It is the layer responsible for opening, maintaining, and closing streams of data. Layer 6 is the presentation Layer, which does a lot of the syntactical translation of data between the lower layers and layer 7. Layer 7 is the application layer, which is the layer that is most visible to the user. Layer 7 is where the user actively interacts with

an application.

Row Space of a matrix: This is the set that is defined by all of the linear combinations of the row vectors of a matrix.

Signature: A unique pattern used by Intrusion Detection Systems, Intrusion Prevention Systems, Antivirus Software, and other security related software and hardware to identify a specific malicious vector of attack. This vector can be software, URL data, embedded information, etc. If a security system identifies incoming data that matches a signature in its database, it will trigger a pre-defined response, which could vary from a packet being dropped, to a file being quarantined, to an alert being sent out by the system, etc.

Singular Value Decomposition: This is a factoring method for $m \times n$ matrices that produces three results: an $m \times m$ unitary matrix U, an $m \times n$ positive semi-definite diagonal matrix Σ, and an $n \times n$ unitary matrix V. The relationship that exists between this three matrices is this: $A = U\Sigma V^H$.

TCP: Transport Control Protocol. TCP is the main protocol, in conjunction with IP (Internet Protocol), by which data is transmitted over the internet. TCP manages the information that is required to route a packet from an application on a local machine to the same application on a remote machine, including source and destination ports, sequence numbers for terminal packet reconstruction, and a number of other important parameters. On top of TCP there are hundreds of unique protocols, such as HTTP, that are associated with "port" numbers within the TCP protocol. For example, HTTP is commonly associated with port 80. TCP acts as an intermediary between IP and the higher levels of the OSI model.

U **Matrix:** One of the results of a Singular Value Decomposition on an $m \times n$ matrix. U is an orthogonal matrix that is $m \times m$ in size. The U matrix is representative of the Column Space of the decomposed matrix.

V **Matrix:** One of the results of a Singular Value Decomposition on an $m \times n$ matrix. V is an orthogonal matrix that is $n \times n$ in size. The V matrix is representative of the Row Space of the decomposed matrix.

Σ **Matrix:** One of the results of a Singular Value Decomposition on an $m \times n$ matrix. Σ is a diagonal matrix that is $m \times n$ in size. The Σ matrix describes the scaling relationship of the vectors resulting from a multiplication of a matrix A with V as they relate to the result in U.

2.2 Hypothesis

The ideal solution would be to develop an Intrusion Detection System or Intrusion Prevention System that can independently and autonomously identify and eliminate threats in real time at line speed, while adjusting threat recognition in real time to identify new anomalies and novel attack vectors. The reality is that this ideal may well be outside the scope of what is achievable given the current state of Artificial Intelligence, which is not the focus of this research. Expanding on the prior work of others, a more modest, and potentially viable solution, which will be explored herein, leverages the ability of Singular Value Decomposition to identify frequencies in a "signal." By using this algorithm, it should be possible to treat network packets as signals, and then identify them based on their individual "frequencies" as these frequencies relate to direction projection into the domain space, range space, and null

space of any given defined space. There should be characteristic spikes in magnitude within a given space for data that is non-standard or anomalous.

The theory is that packets can be analyzed at the binary level on the basis of networks and transport layer information, such as source and destination ports to determine what protocol the packets are associated with. Next, the data contained inside the payload of these packets can be analyzed via singular value decomposition in combination with the use of the dot product to determine how similar they are to packets in existing "training" data that has been specifically selected and classified as "known good" traffic, and which exists in a given defined space. It is theorized that this process will be able to take unknown traffic coming into a network and classify it based on the projection characteristics (a type of fingerprint) into three subsets. The first subset consists of known good traffic that conforms to vector directionality patterns represented in the training data. The second subset consists of truly malicious and malformed data that is atypical for the space defined by the training data, and is assumed to be intended to be used to try and penetrate or otherwise harm a network. Packet data that does not conform to the protocol specific training data that exists would also fall into this category. The third subset of data would consist of "questionable" traffic, that would require further, perhaps human level, analysis to determine the validity of it. The hope is that questions such as these can be answered: Can an SMTP based mail system be trained to only accept SMTP packets that, for example, contain no database commands or executable programs? Can we freeing up resources at endpoints by discarding all plaintext traffic or encrypted traffic before it even hits the mail server? Even ideas as simple as: can we identify the language inside

of HTTP data? This is something that can, of course, be achieved currently through other means; however, these algorithms are largely sequential in nature, slower, and also performed much further up the network stack, typically at the application level.

In the past, in order to enable a computer to identify these packets, different approaches have been taken utilizing different mathematical models, should singular value decomposition not prove fruitful, some of the other paths may be examined for usefulness and effectiveness starting with neural networks, data mining, and machine learning, which have been used prior with varying degrees of success [25] [21] [9].

Should this theory prove correct, it is hoped that the implementation of a system based on such a theory would leverage the use of GPGPUs, thus allowing stream processors to parse and categorize vast quantities of matrix-formatted data at speeds greatly increased over those of traditional sequential analysis. The goal of which would be to vastly increase the pace at which packets can be analyzed, which should add the following four benefits to incorporation within a unit such as a firewall. 1) Decrease in price due to the scalability/cost of stream processors versus regular processors, 2) reduced network bottleneck issues, and 3) the freeing up of system resources on such hardware which may enable more complex processes to be housed within a single unit, 4) It might be possible to add an additional layer of analysis at the transport layer, which would significantly increase the ability of an Intrusion Detection System or Intrusion Prevention System to detect anomalous behavior.

2.3 Solution and Proof

Five rounds of testing yielded strong positive results that singular value decomposition can indeed be used in combination with vector dot product multiplication to produce definitive "finger prints" that can be used to identify anomalous data within a given packet "space." It therefore should be possible to use these "fingerprints" to classify data as "good," "bad," or "unknown." Not all singular vectors in the Column Space of the decomposed matrices of training data were suitable candidates for use in the algorithm. However, in the ICMP test, 77 out of 100 of the singular vectors from the U matrix proved to be effective in identifying anomalous data. Furthermore, in the ICMP test, 100 percent of singular vectors associated with the Null Space were effective in identifying anomalous packets. In all tests but one, an energy shift in the strongest singular vector was able to correctly identify the bad packets. However, typically only a few singular vectors in the Column Space were effective at identifying the "bad" payload data with characteristic magnitude spikes amongst the rest of the noise of the column space. Most of the Column Space was typically noise, for example only 11 percent of Column Space singular vectors were able to identify "bad" data in the ICMP test, with far smaller percentages in all other tests. However, there does appear to be information contained in the Column Space that with refinement of the algorithm could yield useful results. It was observed that only the strongest and weakest singular vectors in the Column Space were able to correctly identify the "bad" packets in the ICMP testing data. The analysis of the Null Space was significantly more dramatic containing on average significantly less noise than the Column Space and often very clear, defined, magnitude spikes in at least a few, of

not many test sets for all testing cases.

2.4 Algorithm and Analysis

The algorithm relies on singular value decomposition to establish a set of training data based off of known good traffic which is composed of a matrix U, which represents the matrix column space (or range) of the training data, a matrix V, which represents the row space (or domain) of the training data, and a matrix composed of singular values, which represents the strength of the projections from the column space to the row space. Subsequently, unknown traffic is formatted into vectors and then multiplied with the column vectors of U from training data using the dot product which provides a value which is used to populate a chart, which can finally be used to classify whether the unknown data is "good" or "bad".

The primary step of the algorithm aggregates and formats the training data. This process begins with extracting payload data from packets after stripping them of their IP and protocol headers. The primary protocol used during initial testing was ICMP. Subsequent testing used TCP packets, specifically the HTTP protocol, of two different types. The first type consisted of HTTP content that was mostly text based, and was taken from Reddit.com, the second type was HTTP content that was solely streaming audio, and was taken from Pandora.com. For purposes of explaining the algorithm, ICMP packets will be used as a model. The algorithm is identical for the other protocols, excepting protocol header and payload sizes.

The ICMP packets used in initial testing were 84 byes in size, with 20 byes for the IP header, 8 bytes for the ICMP header, and 56 bytes for the ICMP payload itself.

The algorithm strips both the IP and ICMP header information, leaving only the payload for analysis. The payload data in binary form (448 bits total) is formatted into a column vector with 448 components.

While there was a fair amount of variance in sample size in initial testing, a sample size of 100 packets (or columns) was finally settled on for ICMP, as it was felt that this was big enough to represent the heterogeneity of the ICMP payloads that were being examined and thus be a good representation of the vector space that ICMP resides in. At the conclusion of sampling, 100 column vectors, consisting of 448 components each, were assembled. These vectors are assembled from ICMP data streaming from 16 machines running 6 Unix variants and 4 Windows variants in order to simulate a more realistic, heterogeneous environment. These column vectors represent the "known good" traffic from the test environment.

The next step in the process involves aggregating the 100 column vectors into a matrix of size 448X100. With regard to matrices, rectangular matrices have certain characteristics that made them ideal for this experiment. The rectangular nature of the matrix ensured that the matrix was not a basis for the space that the matrix mapped to, and that there would be a substantial amount of free variables, which implied a significant amount of the multiplications on the singular vectors of the matrix would result in projections within the null space. All four fundamental spaces in linear algebra contain the zero vector. Only square matrices that are a basis for the space they define contain only the zero vector. It is in the null space that the most significant results of this research were located, hence the importance of the rectangular shape of the initial matrix, as this basically guaranteed a sizeable null

space.

Once the 448X100 training matrix was composed, a singular value decomposition was performed on the matrix producing three distinct matrices, one a 448X448 matrix representation of the singular vectors of the column space called U, the second a 100X100 matrix representation of the singular vectors of the row space, called V, and finally a 100X100 matrix populated with values along the main diagonal (everything else in the matrix is zero) that is a representation of the scaling relationship between the column and row spaces of the original matrix. This final matrix is called Σ, and the values are known as the singular values. Each singular value defines a relationship in the factorization of the original matrix. The *nth* singular value in the matrix moving from top to bottom represents the how the *nth* singular vector in the column space scales to the *nth* singular vector in the row space and vice versa. Singular values of zero represent that there is no domain/range relationship between the column space and the row space. The singular values are of primary importance as they tell us two very important things: 1) the degree (or scaling factor) to which a vector in the column space scales in size to a vector in the row space, and more importantly 2)which of the singular vectors project into the null space; these are the vectors associated with singular values of zero. As would become apparent after testing, the null space is where the largest percentage of our usable data lay.

Once the data is factorized into the U,V, and Σ matrices, the next step is to use these matrices, specifically the U matrix, along with references to the Σ matrix singular values, to filter unknown data to determine the classification of the data. The initial algorithm relied on determining the mean of the non-zero components in the

singular values matrix. Next the singular vector associated with this was extracted from either the U matrix. The intent of this was to see if the mean represented an area in the column space where anomalies in packet data between the known good data and the unknown data could be identified due to our theory that this space near the mean of the singular values was representative of some sort of average magnitude of vectors within the space represented by the known good traffic. Accordingly, vectors diverging from the directionality of the vectors resulting from calculations based on the mean, should be "bad" vectors, and easily identifiable. To test the theory, the algorithm then took the selected singular vector and multiplied it against all of the unknown vectors from our testing data using the dot product. The idea was to see if the values produced by this had a noticeable spike or dip in magnitude that would identify that particular vector in the testing data as bad compared to the other vectors. After testing this, it was determined that the values returned were essentially noise, and that there was no adequate means to make a successful identification using this approach. Back to the drawing board.

The second approach was a variant of the first approach. Still using the singular vectors from the U matrix only (this is because the U Matrix is a representation of the column space, and both our testing and training data vectors were assembled into column vectors), a more comprehensive approach was considered. Instead of choosing the mean singular vector to use in the vector multiplication stage based on a statistical analysis of the singular values, we selected each column vector from the U matrix iteratively and in sequence, multiplied each of these vectors, again using the dot product, against the entire set of vectors in the matrix composed of the test

data (unknowns). The values resulting from these multiplications were plotted in a chart that tracked the magnitude on the vertical axis, and the number of the packet from the unknown data set along the horizontal axis. The theory was again that the magnitude of projections of the "bad" payload data would differ enough from those of the good payload data that classification could be achieved. The chart was a means to visualizing the data in sets large enough to see meaningful differences or patterns easily.

The complexity of the algorithm is two-fold as there are two main steps in the process. The first step concerns the development of training data. This involves gathering packets from the network (n time), reformatting them into binary and breaking out the IP Header and the Protocol Header, then creating vectors out of the payload segments, then creating matrices out of the vectors, and finally performing a singular value decomposition on this matrix.

Stage 1 - Data gathering. This is a continuous process and has a complexity of $O(n)$ since it simply captures each packet off of the line. Stage 2 - Converting the Data. This also has a complexity of $O(n)$ since it performs the conversion operation on each item of data one time. Stage 3 - Performing the Singular Value Decomposition has a complexity based on the size of the matrix. For an mxn matrix, the complexity is $O(mn^2)$, for a square matrix the complexity if $O(n^3)$. Stage 4 - Vector Multiplication. This has a complexity of $O(n^2)$ due to the sequential, two level deep, nested loop operation of the algorithm.

Most of the complexity of the different stages is $O(n)$, with the vector multiplication loop being $O(n^2)$ Since the Singular Value Decomposition is $O(n^3)$ for the worst

case scenario, it is the limiting factor for complexity, and therefore the complexity for the entire algorithm is $O(n^3)$.

2.5 Summary

The algorithm basically consisted of gathering two sets of data, training data, and testing data. The training data was composed of known good traffic for the particular protocol being tested. The testing data consisted of a separate set of known good traffic with a few bad packets interspersed. The algorithm gathered the training data into a matrix composed of vector columns, each of which represented a single packet. Next a singular value decomposition was performed on the training matrix, yielding three new matrices: a U matrix, a V matrix, and an Σ matrix. Next the testing data was also gathered into a matrix composed of mostly good packet vectors, with a few bad packet vectors. Finally, each column vector in U was multiplied using the dot product operation against each individual column vector of the testing data. The resulting values were plotted to a series of charts, with each chart consisting of the results of the multiplications of one singular vector from U against all of the vectors in the testing matrix. Finally the chart was analyzed visually to identify anomalies.

3. TESTING

3.1 Methodology

This research required the use of many different pieces of hardware and software. The individual pieces of hardware and software will be detailed along with descriptions of how everything was used in the context of the research in Appendix I.

The testing portion of the research covered six distinct training/testing sets. For initial proof of concept work, ICMP was the protocol selected. This is due to the somewhat homogeneous nature of ICMP being that the data portions of each ICMP packet are relatively similar to each other; in fact, across Windows platforms, the ICMP packet is identical for every OS version, some even spanning multiple OS releases. An additional consideration was the small size of generic ICMP payloads, with 56 bytes for Linux/Unix, and only 32 for Windows. The next set of data consisted of English ASCII and Chinese UNICODE representations of text taken from the Declaration of Independence, translated for the Chinese. This data was identical in format to how it would appear being transmitted over HTTP; however, it was extracted and converted manually from web pages, and not taken from payload data inside TCP/HTTP packets. The next set of data was actual HTTP traffic captured through tcpdump from the website Reddit.com. This site was chosen due to it's largely text-based format, in an effort to get a data set representative of payloads

over TCP/HTTP containing mostly English text. The next set of data chosen was HTTP traffic from the website Pandora.com. This site was chosen due to it's largely streaming audio based content. The next two sets of data were an intermingling of the prior sets of data, with one consisting of the Reddit training data tested against data containing the Chinese unicode character data. The final set was a comparison of packet payload data from Reddit against payload data from Pandora.

Once the data sets were populated, the first step in the testing process was to prepare an environment wherein the research would be conducted. A layer of complexity was added due to the fact that factors out of individual control required that the research be done completely offsite from CSUSB. This necessitated the creation of a remote, fully self contained environment, rather than being able to utilize CSUSB's substantial research infrastructure. Due to financial restrictions, the only feasible way to create a large enough infrastructure to facilitate a heterogeneous environment that was complex enough to conduct thorough testing was to use virtualization technology to enable the leveraging of a tiny home network consisting of two desktops and two laptops with only two different operating systems into a still somewhat small, but now adequately sized, 16 machine network, utilizing four different versions of Windows, five different versions of Linux, one version of Unix, with one final machine running OSX. The open source movement was critical to the success of this venture, as multiple free programs, as well as free operating systems, were utilized to achieve the scale and complexity that was desired.

Once the environment was up and running, work began on the programming/testing phase of the project. This phase consisted of incorporating data streaming from the

above described components into a workable program that would capture and analyze data. The first iteration of this program, which was written in Python 2.7, consisted of a remote call to tcpdump to capture packets and dump them to a packet capture file called date_tcpdump.pcap. Next the program opened that file, converted the packet data to hex, discarded the ethernet header information while keeping the ip header information and the ICMP header information, and most importantly, the payload data, and wrote that data to a new file called date_tcpdump_output.txt. Here is some sample output from the converted output file describing a single ICMP packet:

Sept_12_tcpdump_output.txt Output File example:

13:00:41.834134 IP Ubuntu1- Ubuntu2: ICMP echo request, id 3221, seq 534, length 64 0x0000: 4500 0054 0000 4000 4001 bd45 c0a8 fe06 0x0010: c0a8 fe0b 0800 1522 0c95 0216 ecab 2650 0x0020: 0000 0000 fa63 0800 0000 0000 1011 1213 0x0030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223 0x0040: 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233 0x0050: 3435 3637

Next the program read in each line of the output file, converted the hex to binary, re-printed the header information associated with each packet, counted the bytes for each portion of the packet, and broke out the binary for each segment of the packet, finally writing this data to a new file called date_tcpdump_ping_binary.txt. Here is a sample of packet data from that file:

Sept_12_tcpdump_ping_binary.txt Output File example:

Ping Packet Data For: 13:00:41.834134 IP Ubuntu1 - Ubuntu2: ICMP echo request, id 3221, seq 534, length 64

IP Header - 20 Bytes: 0100010100000000 0000000001010100 0000000000000000 0100000000000000 0100000000000001 1011110101000101 1100000010101000 1111111000000110 1100000010101000 1111111000001011

ICMP Header - 8 Bytes: 0000100000000000 0001010100100010 0000110010010101 0000001000010110

ICMP Data Portion: 1110110010101011 0010011001010000 0000000000000000 0000000000000000 1111101001100011 0000100000000000 0000000000000000 0000000000000000 0001000000010001 0001001000010011 0001010000010101 0001011000010111 0001100000011001 0001101000011011 0001110000011101 0001111000011111 0010000000100001 0010001000100011 0010010000100101 0010011000100111 0010100000101001 0010101000101011 0010110000101101 0010111000101111 0011000000110001 0011001000110011 0011010000110101 0011011000110111

Data Portion = 56.0 bytes

At this point, during the initial phase, the data portions of a file containing exclusively "known good" ICMP packet data were manually extracted, with each payload subsequently manually converted into an individual vector of binary digits 448 units long. This process occurred inside the R interpreter running on one of the CentOS boxes in the lab. For the first round of testing these vectors were assembled into matrices of size 20X448, with 20 of the previously assembled vectors inserted as column

vectors of the matrix. This matrix composition was conducted in a non-persistent R interpretive session on the same CentOS box. This matrix was to serve as the "known good" training data. After composing this matrix, a singular value decomposition was performed on the matrix, again using the non-persistent R interpreter. The singular value decomposition produced three results, a vector of singular values Σ, a U matrix, and a V matrix. The resulting singular values in the vector produced by this decomposition were examined to find a median value, and then the singular column vector from the U matrix that corresponded to the position of the median value in the vector of singular values of the same size was extracted. For example, if the value in position 5 of the singular values vector proved to be the median value, the 5th column vector of the U matrix, itself a vector 448 units long, was extracted. This extracted vector was then used as a multiplier in comparative dot product multiplications against individual vectors of both good and bad traffic types. The resulting values of these multiplications were examined to see if there were identifiers present that could illuminate whether traffic was "good" or "bad," based on the assigned metrics. This process was conducted and repeated for matrices composed of "known good" traffic, as well as "known bad traffic." The results of this experiment were inconclusive, as no identifers could be found to clearly differentiate between good and bad traffic.

After some contemplation, it was realized that larger data set would be required, as well as some sort of graphical-based visualization assistance in order for any type of concrete analysis to take place. Additionally, it was decided to switch tools with regard to the linear algebraic methods being used in the algorithm. This reasoning

behind this was two fold. First, there was the issue of the learning curve with regard to learning to write programs in the R language. While easy enough to use in a simple interpretive session, this method of usage of R was non-persistent and required a system to be up perpetually in order not to lose data cached in memory. Learning how to write persistent programs in R required substantially more complexity than simply navigating the interpreter, especially with regard to file handing. The second reason for the switch was the lack of easy integration between R and Python. If R had been maintained as the analysis language, it would have required external calls to programs outside of the main Python program, as well as some means of transferring variables between the programs. By using SciPy, a python library with the same functionality as R, complete integration with Python was achievable, greatly reducing the required code base, and providing a more stable and simplified platform for the research. The switch to SciPy also greatly accelerated testing, since tests could now be scripted once, rather than needing to be reconstructed in the R interpreter. Finally, the inclusion of the MatPlotLib graphical library, which also integrates nicely with Python, provided the means to achieve the necessary graphical visualization.

These developments led to the second phase of research, in which the first tangible positive results were obtained. With the integration of SciPy and MatPlotLib it was now possible to populate binary data directly from the dump files into SciPy vector and matrix variables, and then subsequently perform singular value decomposition on the data as described previously. The major difference between the first phase, and the second phase was that the second phase incorporated a greater degree of completeness in the analysis of the singular values achieved by iteration over the entire matrix of

singular vectors, rather than just selecting the one associated with the median singular value as in the first phase. In the second phase of testing, a loop was added to the program that allowed for testing of all of the singular vectors, by selecting iteratively each of the columns in the U matrix, and then multiplying all of those columns using the dot product within SciPy against each vector in two different sets of vectors that were fed to the program. The first set of vectors contained only "known good" vectors, which were composed of actual binary data extracted from ICMP packets collected from selected nodes on the network. The second set contained mostly good vectors, with a few bad vectors, that were artificially produced using either hping or ColaSoft Packet Builder. The results of these dot product operations were stored into a "results" array within the Python program. Finally, the program iterated over the "results" array, plotting two lines. The first line was labeled "Training", and it consisted of the results of multiplications of the "Known Good" singular vectors produced from the singular value decomposition with the original "Known Good" matrix prior to decomposition. Essentially the first line, labeled as "Training" is testing the known good traffic against itself. The second line, labeled "Testing" consisted of the results of multiplying the "unknown" traffic which consisted of a mix of mostly known good packets that were independent of the training data mixed a few bad packets against the singular vectors of the training data. By comparing the two lines it was now possible to see discrepancies between the "known good" traffic and the traffic that had "bad" packets mixed in with a unique set of known good traffic.

As mentioned previously, there were 6 different data sets. While the algorithm

was largely the same for each data set, some differences in structure of the different sets should be noted.

3.1.1 Set 1: ICMP

The ICMP data was composed of two subsets of data, one composed entirely of Windows packets, and one composed of linux, unix, and mac packets. ICMP is a mutable with regard to packet size, in that packets of different lengths can be constructed by providing additional arguments; however, there is a default size that is constructed with a standard call. The testing dealt only with the default packet sizes for each platform. Windows uses it's own proprietary implementation of the ICMP protocol, which changed slightly after Windows XP. Windows ICMP packets are 32 bytes in length, with no differences in the contents of the binary payload data from one machine to the next, there packets, excepting the headers, are essentially static checksums. The Linux/Unix/Mac packets all use the BSD derived version of ICMP, which constructs packets that have payloads that are 56 bytes in length with small discrepancies in the payload contents based off of checksums that include source and destination information in the data. In the ICMP data set the training data was composed of 100 standard size known good packets. The training data was composed of 97 standard size known good packets that were unique from the training data, with 3 packets inserted that were standard size; however, their payloads had been manipulated by inserting SQL query data into the payload.

3.1.2 Set 2: English ASCII vs. Chinese Unicode

This data set consisted of two subsets of data, English ASCII characters converted to binary, and Chinese Unicode characters converted to binary. The source document was the Declaration of Independence of the United States of America. The text was grabbed from www.archives.gov/exhibits/charters/declaration_transcript.html and a portion of it was converted first to ASCII using the converter located at: textop.us/Text-Convert/Ascii then to binary using the converter located at: www.binaryhexconverter.com/ascii-text-to-binary-converter. Two sets of text were created, one to act as the training data, one to use for the testing data. The Chinese Unicode set was created by grabbing a different part of the Declaration of Independence, and using Google Translate located at: translate.google.com to convert the text to Chinese characters, which were then input into the converter located at www.pinyin.info/tools/converter/chars2uninumbers.html to convert them to Unicode, finally the Unicode was converted to binary by using the converter located at: bennettroesch.com/Tools/BinaryUnicodeConverter/. In this set the training data consisted of 50 columns with 300 individual bit components derived from the English ASCII. The testing data consisted of 47 columns of 300 individual bit components derived from English ASCII, and 3 columns of 300 individual bit components derived from Chinese unicode.

3.1.3 Set 3: Reddit TCP/HTTP

This data set was composed of payload data that was extracted from packet captures taken from communication with the website Reddit.com. Only incoming ACK data

packets were used. Reddit.com was selected on the basis of it's composition of largely text-based content. Non-text based content was excluded from the data set as much as possible by only visiting pages that displayed mostly text. The payloads were extracted from standard 1500 KB MTU packets, and were 1452 bytes in size after the header information was removed. The data sets were 50 column vectors composed of 11621 individual bit components. The training data consisted of 50 known good vectors of this size. The testing data consisted of 47 known good vectors unique from the training data, and 3 vectors generated using ColaSoft's packet generation tool that were of equal 11621 bit component length, but contained artificially generated garbage, basically randomly typed hex characters.

3.1.4 Set 4: Pandora TCP/HTTP

This data set was composed of two subsets. The first subset was composed of payload data that was extracted from packet captures taken from communication with the website Pandora.com. Only incoming ACK data packets were used. Pandora.com was selected on the basis of it's composition of largely streaming audio-based content. The payloads were extracted from standard 1500 KB MTU packets, and were 1452 bytes in size after the header information was removed. The data sets were 50 column vectors composed of 11621 individual bit components. The training data consisted of 50 known good vectors of this size. The testing data consisted of 47 known good vectors unique from the training data, and 3 vectors generated using ColaSoft's packet generation tool that were of equal 11621 bit component length, but contained artificially generated garbage, basically randomly typed hex characters. The second

set was identical to the first set, with the exception that an additional "bad" vector composed of roughly half known good bits, and half 1s was added into the testing data.

3.1.5 Set 5: Reddit vs. Chinese

This was the first of two hybrid sets that were tested. Reddit training data that was using largely ASCII-based English text translations was tested against the Chinese Unicode data. The training data consisted of the 50 known good vectors from the Reddit training data, these were chopped from 11621 bits to 300 bits to conform to the size of the Chinese data vectors. The testing data consisted of 44 known good vectors from a set of data unique to the training data and two sets of "bad" data. The first set of "bad" data was 3 vectors from the original "bad" data from the first Reddit test chopped down to 300 bits in size. The second set was 3 vectors from the Chinese "bad" vectors from the first language test that were each 300 bits in size.

3.1.6 Set 6: Reddit vs. Pandora

This was the second of the two hybrid sets that were tested. The training data consisted of the Pandora training data from the previous homogeneous Pandora test, which was comprised of 200 vectors based off of payload captures of 11621 bit lengths. The testing data was composed of 197 vectors of "known good" payload data from Pandora of 11621 bit lengths, and three "bad" vectors taken from the "known good" vectors of the Reddit test that were 11621 bits in length.

3.1.7 *Conclusion*

If the algorithm is sound, the identification of the goodness of each type of payload should be easily established. The more homogeneous payloads are expected to be more successful in this regard. For example, ICMP is a simple protocol that is largely homogenous in packet complexity and the packets should be easily distinguished as valid or not based on the contents of the payload. Packets that are either too big, too small, or contain erroneous header information can be easily filtered out by a router, while packets that fit typical standard norms in all the above categories should be fairly safe to allow through so that they may be subjected to internal analysis. Classification of packets will be threefold initially, where packets are identified as "good", "bad", or "unknown." Based on classification, an implementation using this algorithm could then either drop the packets, allow the packets into the network, or place the packets into quarantine for further, human or computer based, examination.

3.2 *Analysis of Data*

There were multiple phases of testing, each yielding unique data sets that rather quickly identified the accuracy of the test being run. While clear, positive, results were obtained from five of the six data sets, the nature of the data implied that there were additional subtleties contained within the data that could mostly likely be exploited for future research and refinement of the algorithm. Within each testing set, there were two different sets of data which can be classified as either input or output.

The Input data was exclusively raw binary data. This data consisted of binary

digits collected from either raw packets or from translated ASCII and Unicode. This data was then assembled into individual column vectors, one for each packet or grouping. These column vectors were then aggregated into matrices whose size varied on the length of the data in the initial vectors (number of rows) and the sample size (number of columns). The raw packet data was obtained from 16 different operating systems, two different protocols, and three different websites. Below is a sample vector from the ICMP "known good" data set:

1000110101011111001001000101000000000000000000000000000000000001011000111
0111000000101100000000000000000000000000000000000000000000010000000100010001
0010000100110001010000010101000101100001011100011000000110010001101000011 0
1100011100000111010001111000011111001000000010000100100010001000110010010 0
0010010100100110001001110010100000101001001010100010101100101100001011010 0
1011100010111100110000001100010011001000110011001101000011010100110110001 1
0111

The Output data consists of sets of values that were computed by using the vector dot product operation on the vectors from the "unknown" data set and the singular vectors from the U matrix, which were derived from the singular value decomposition of the matrix comprised of the known good data. Although there were some generalities, this output data was unique for each testing set.

3.2.1 Set 1: ICMP

ICMP testing took part in two phases as it was the initial set to be tested, and consequently the algorithm was tested and refined during these initial periods of

testing. In phase one of the ICMP testing, the results were comprised of a single set that consisted of 20 values. Each value represented the result of multiplying the one singular vector from the U matrix that was associated with the mean singular value from the "known good" training data with each of the 20 unknown vectors from the testing set. Within the the unknown data set that was tested were three "bad" vectors that had spoofed SQL queries, created using hping. The data was completely inconclusive. There were no spikes in magnitude associated with the bad packets; furthermore, the magnitude of the values seemed to vary at random across all 20 packets. Although data from this phase was not plotted to a graph, the same phenomena reappeared in subsequent testing in the column space in and around the mean singular values. Figure 1 is an example taken from the second phase of testing, yet consistent with the inconclusive results seen with the first phase. This is a pattern that was seen again in every future phase of testing when looking at vectors associated with the mean singular values, though in a different context each time.

In phase two of the ICMP testing, the results were comprised of 100 sets of data. Each set containing 100 values, with each set of values associated with a unique singular vector from the U Matrix. Each value represented the result of that set's unique singular vector from the U matrix multiplied against each of the 100 vectors from the unknown data set. Due to the large superset of resulting values, it was important to use a visual aid in order to discern patterns in the data. Accordingly, the values from each set were plotted onto 100 graphs, with each graph representing the results from one set of calculations.

After reviewing the results from phase two, two things became immediately ap-

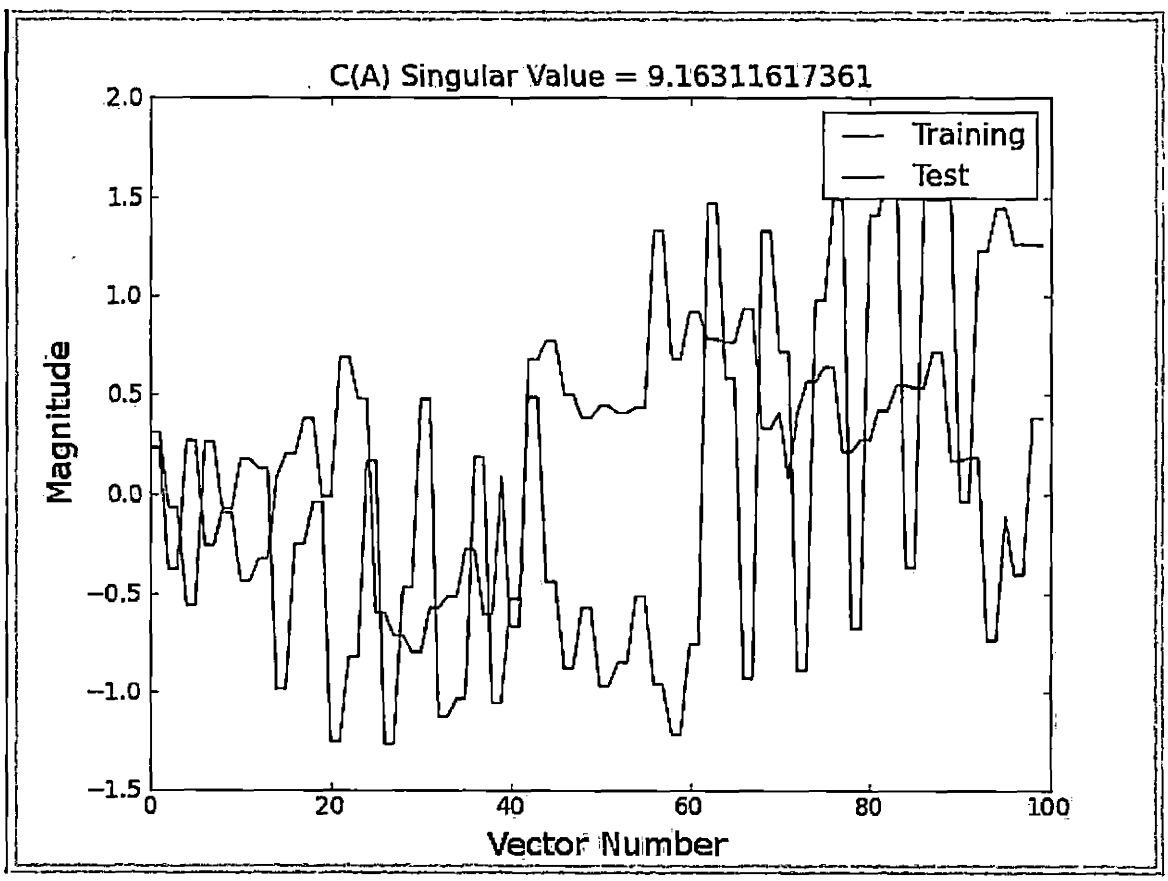

Fig. 3.1: Example of an inconclusive result from phase 1

parent. 1) The initial expectation that the mean singular value and it's associated vector would hold meaningful data was incorrect, and in fact quite the opposite was true. The magnitude shifts that were expected were only expressed at either extreme end of the singular values, or in the null space. Figure 2 shows a plotting of the first 50 singular values for an ICMP testing set (the last 50 are all zeros) from the second phase of testing.

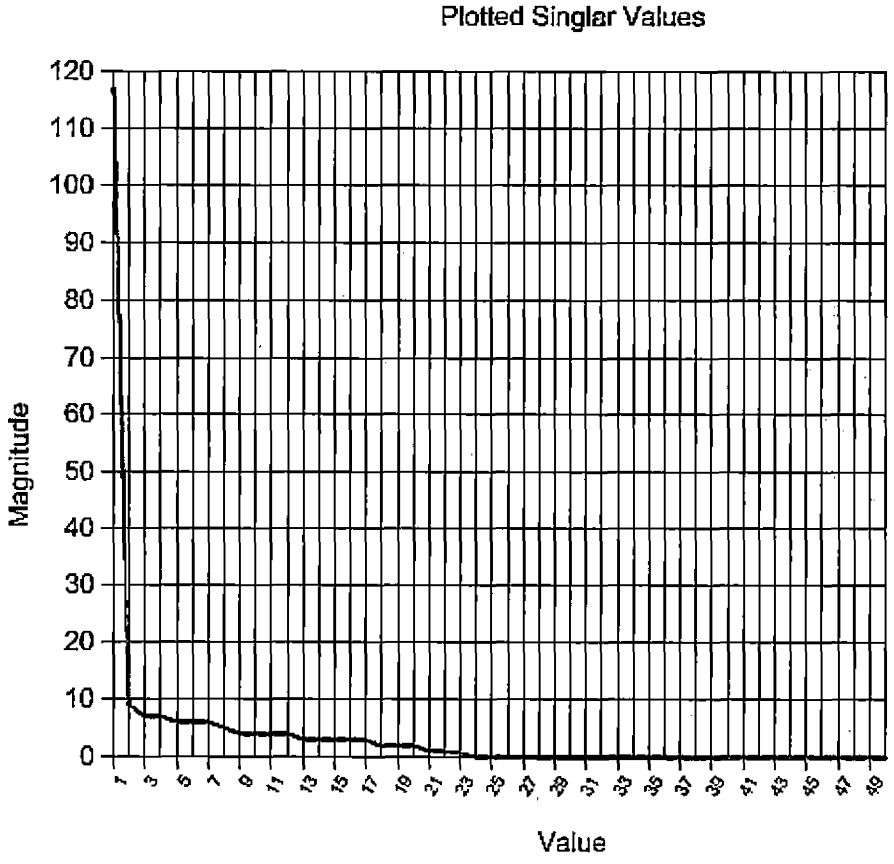

Fig. 3.2: Plotted Singular Values

The mean of this set is the singular value 4. An associated singular vector analysis

for one vector associated with a value near 4 is shown in Figure 3. By looking at the visualizations of the set represented in Figure 3, it is very difficult to get a clear picture of where the bad data sits in the set. In contrast, looking at the extreme ends of the value spectrum provides valuable data. Figure 4 is a plotting of the values associated with the Highest Singular Vector in the Linux ICMP tests. Figure 5 is a plotting of the values associated with the smallest Singular Vector.

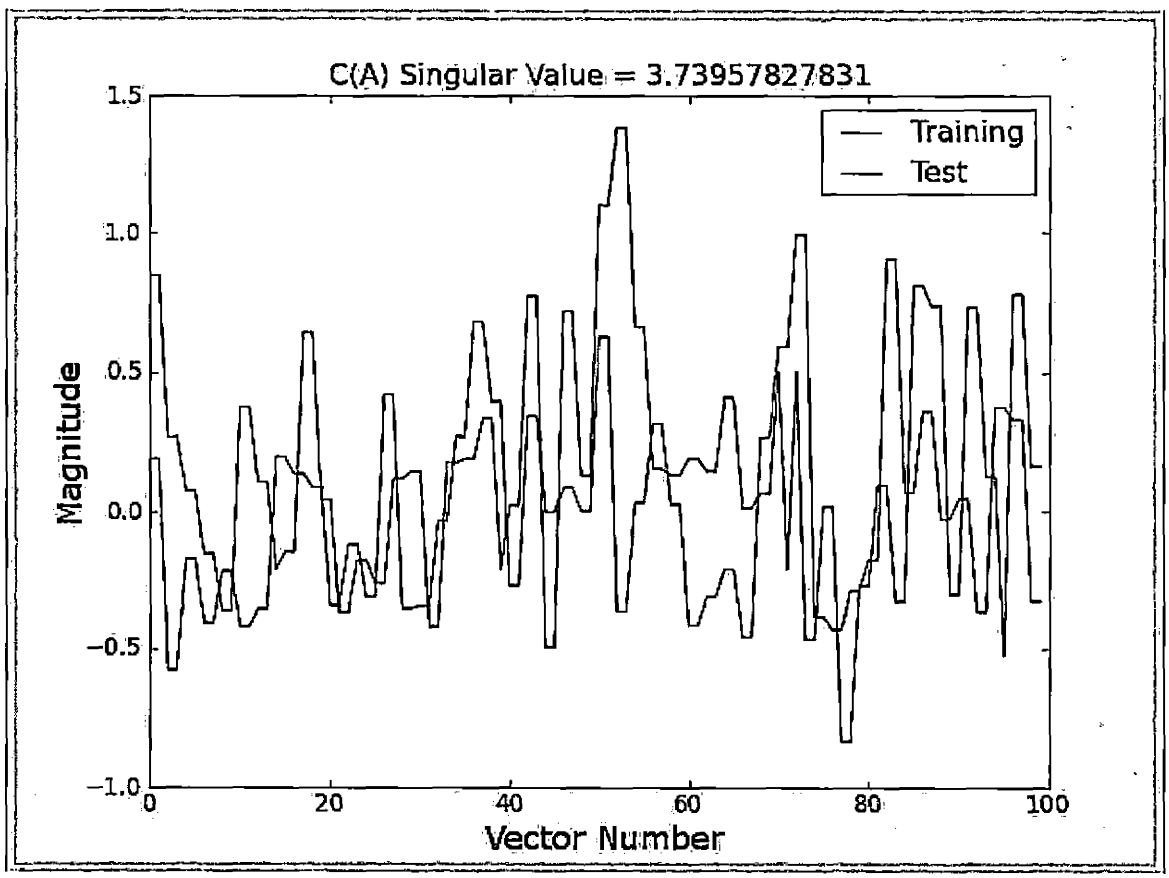

Fig. 3.3: Singular vector plotting for value near 4

Using the extreme singular values on either end of the spectrum it is quite easy to distinguish where the bad packets are by observing the significant spikes in magnitude.

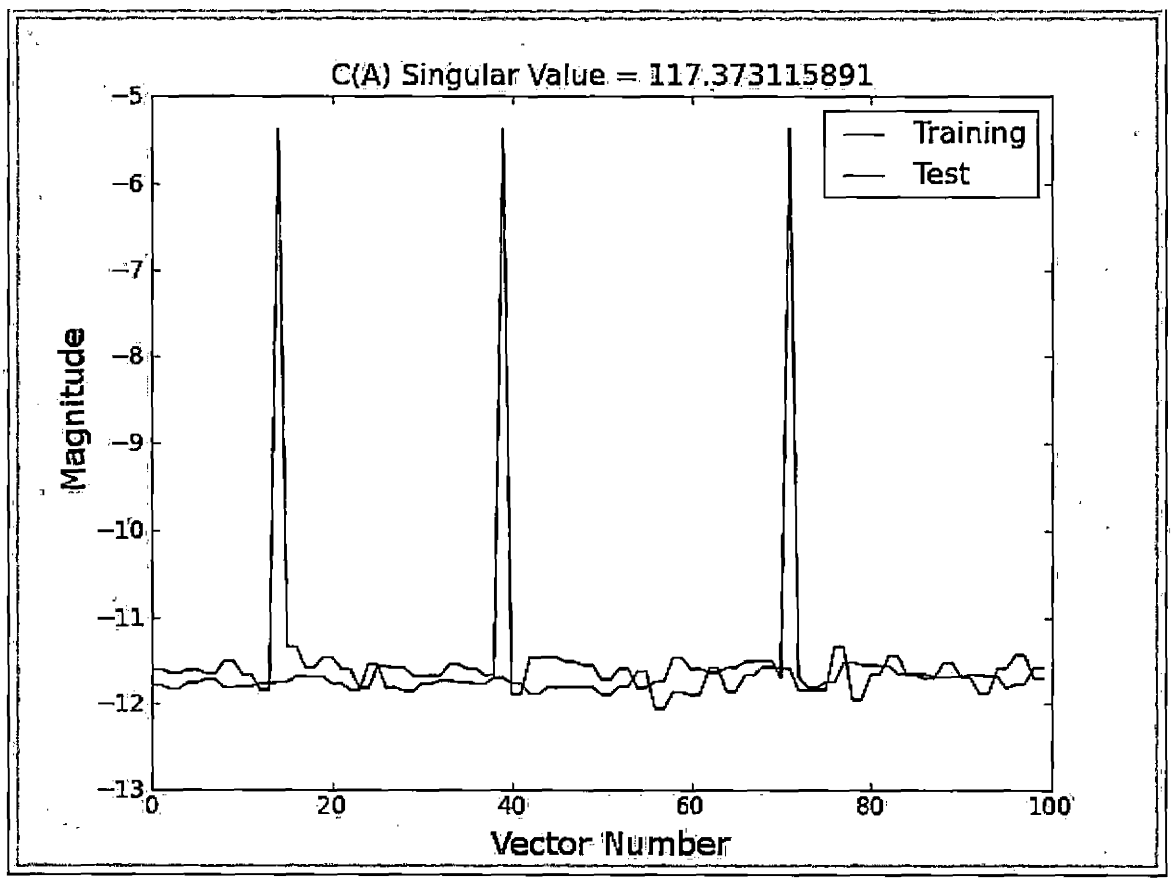

Fig. 3.4: ICMP highest singular value

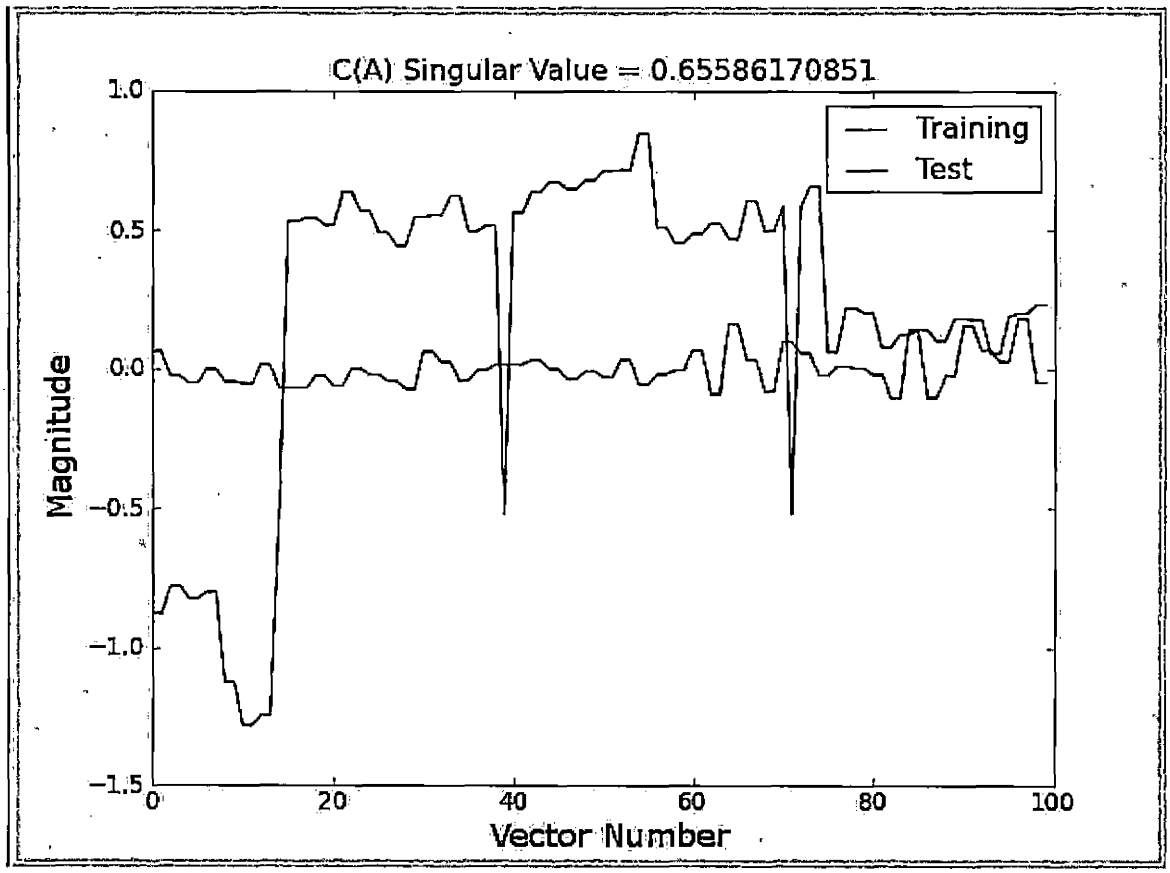

Fig. 3.5: ICMP lowest singular value

Additionally, looking at the sets of values associated with the Null Space where the singular value is zero, it is even easier to distinguish the location of the "bad" packets. Figures 6 and 7 are examples of values derived from Singular Vectors in the Null Space.

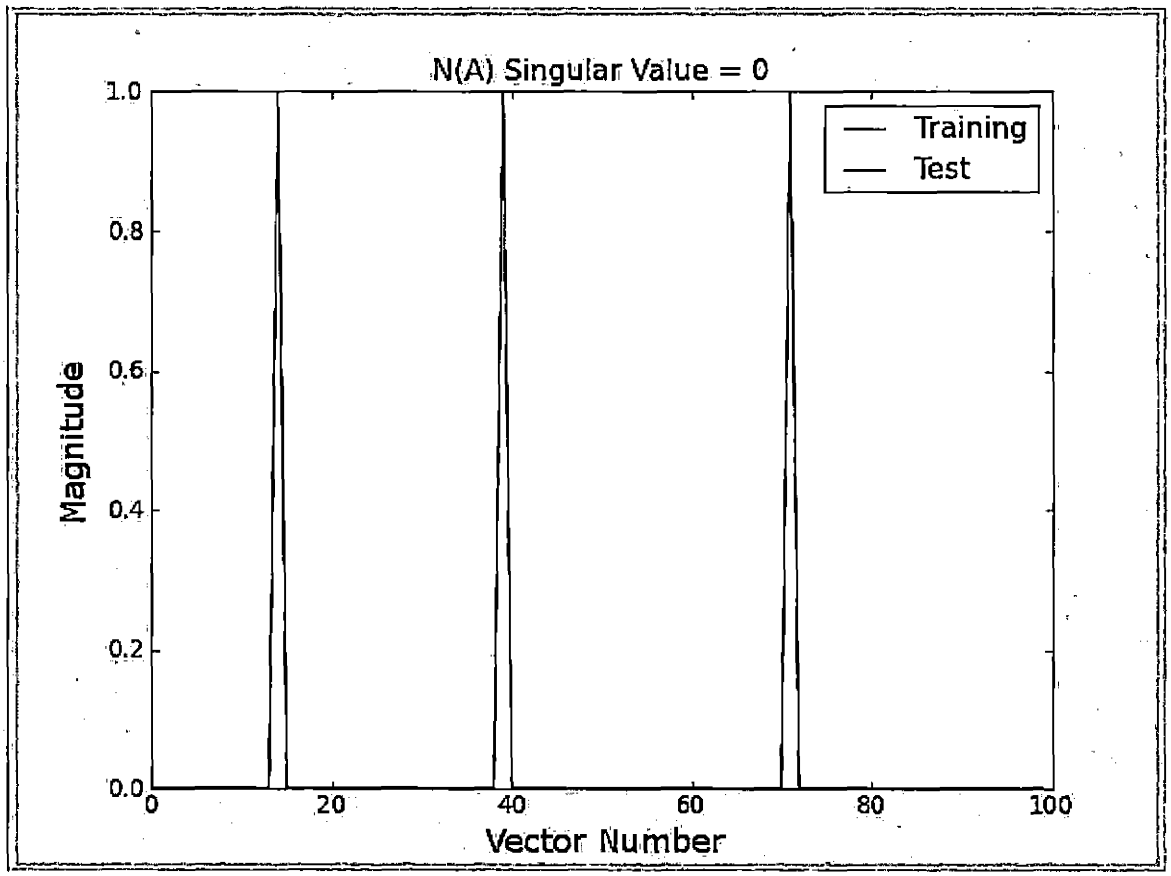

Fig. 3.6: ICMP Null space singular value 1

Clearly the Null Space has the most obvious results; however, it is interesting to note that the clarity observed at the extreme ends of the distribution of the singular values and their associated vectors blurred almost immediately when moving towards the mean. The reason for this is likely because the domain of the column space vectors of U does not cross into the Null Space, whereas, the anomalous vectors reside almost

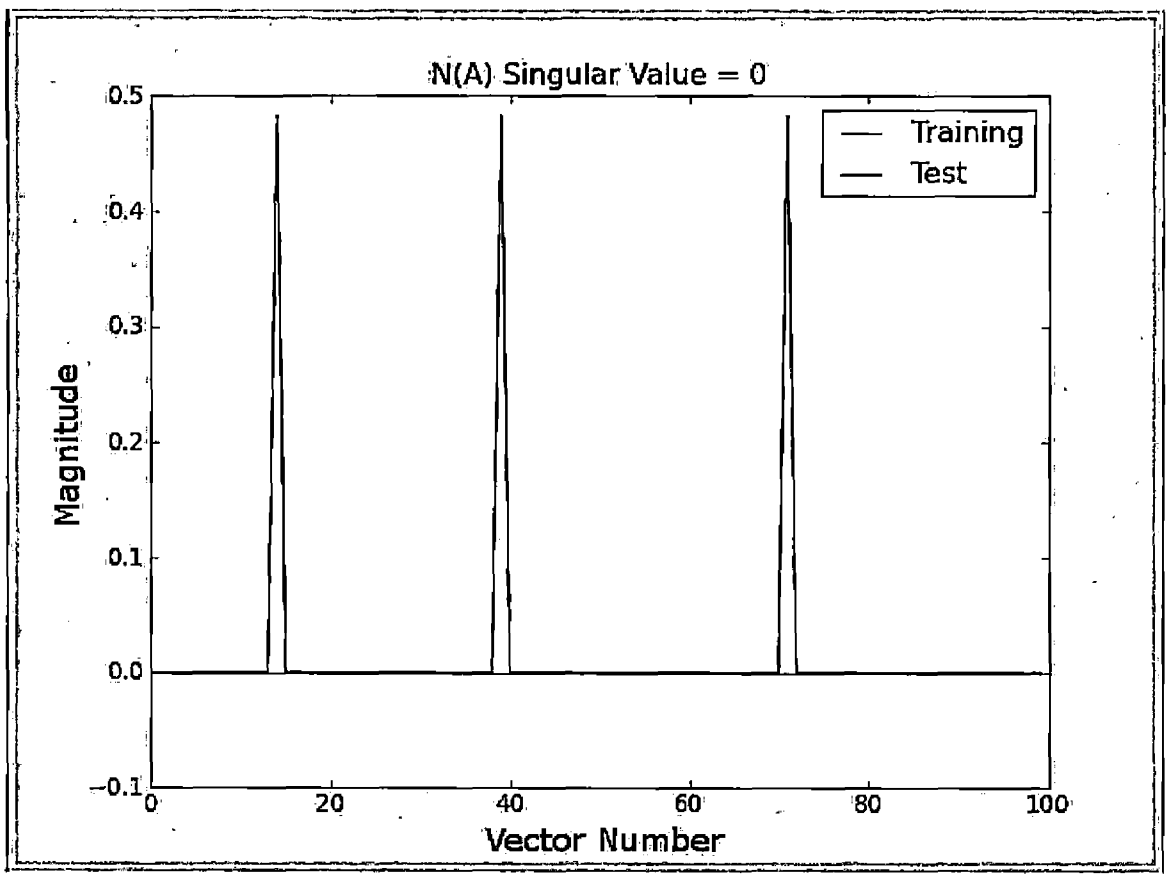

Fig. 3.7: Null space singular value 2

exclusively in the Null Space. The projections of the "bad" vectors that can be seen in the domain space are most likely due to a loss of energy in that space as the "bad" packets are projecting into the Null Space.

The Windows testing was all but worthless as the complete lack of any variation in the packets between the training data, and the testing data with the exception of the "bad" packet made it a little too easy to detect the anomalous data. As such this avenue of testing, and the accompanying platforms were quickly abandoned.

3.2.2 Set 2: English ASCII vs. Chinese Unicode

Testing for the this set again provided strong positive results both in the Column Space and the Null Space. The pattern held true from ICMP proof of concept testing that the areas where "bad" vectors could be identified were in the strongest projection of the Column Space, as well as throughout Null Space. This was the one test that was conducted that did not use actual packet data; however, the data it used was analogous to what would have been extractable from the payload of a packet. The reason for not using strictly packet data was so that a direct correlation could be drawn between ASCII English and Unicode Chinese. Had packet data been used, the payload might have been polluted slightly with non ASCII or Unicode binary information. In Figure 8, the strongest projection into the Column space identified the three bad packets quite clearly. However, as before, the characteristic pattern immediately fades when looking at even the second largest singular vector plotting, as demonstrated in Figure 9; however this was somewhat of an odd pattern compared to what was seen in the ICMP testing. The third vector into the Column space returns

to visualizations consistent with what was seen in the ICMP testing (see Figure 10). Looking at the middle of the Column space provides no better indication of which vectors are anomalous as indicated in Figure 11. Unlike the ICMP testing, when the Null Space is first entered, it remains somewhat unclear which of the packets are bad (see Figure 12). However, once the 5th vector in the Null Space is reached, the easily identifiable pattern emerges once again (see Figure 13). This trend continues with the 6th Null Space projection (see Figure 14). It is also evident in many of the rest of the Null Space projections. Figure 15 is a projection from the middle of the Null Space.

The data from the Chinese vs English testing was very conclusive. It is clearly possible to distinguish the Chinese Unicode data from the English ASCII data. There were a couple of interesting observations when examining the date. First, the initial projection into the Null Space did not provide an immediate conclusive result as it had in the ICMP testing. Second, unlike the ICMP testing which had positive results in the entirety of the null space, there were sets within the Null Space where it was not possible to identify the "bad" data. Furthermore, sets of data such as those expressed in Figures 16-18 posed questions as to what, if anything, is meant by the directional variation in energy projections.

3.2.3 Set 3: Reddit.com TCP/HTTP data

The Data from the Reddit TCP/HTTP test set was conclusive; however, there were again some variations between this set and the prior testing. First, this set was the first set where there were no conclusive results in the Column Space. Not only that,

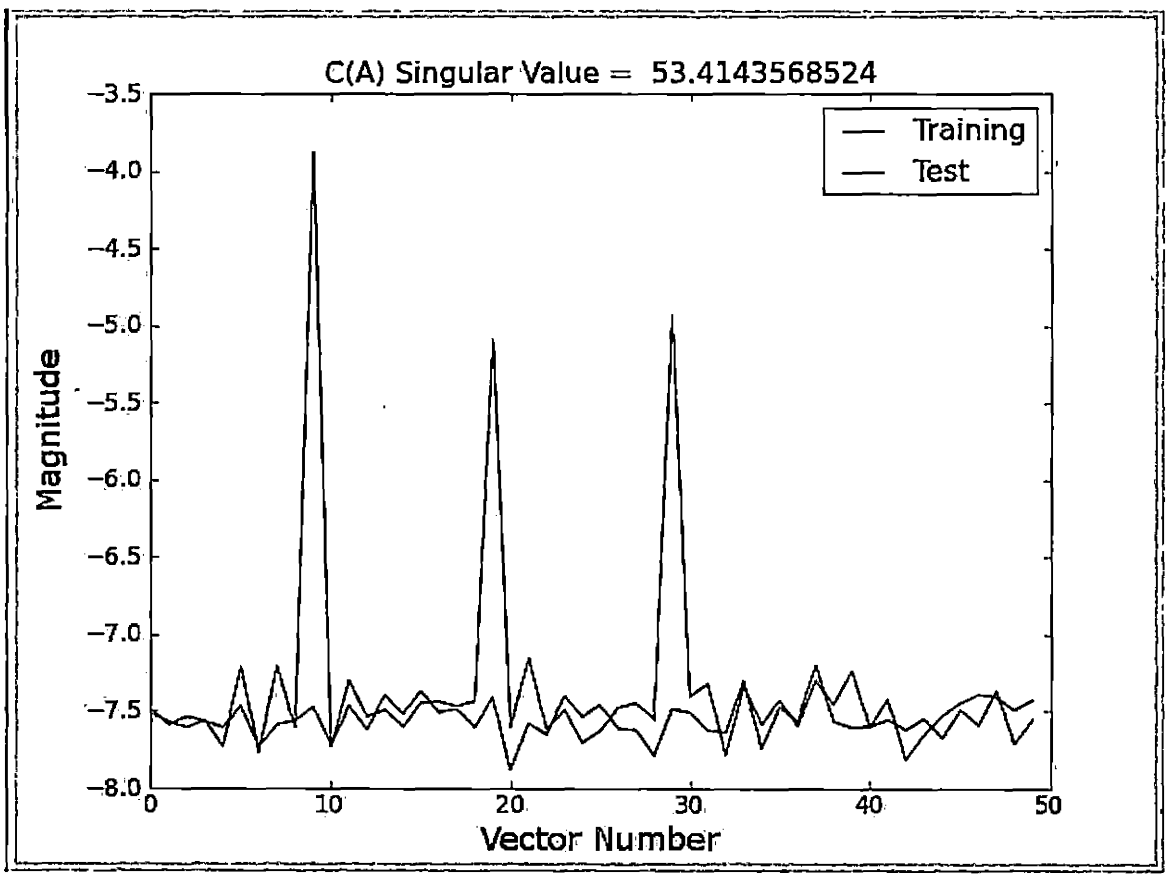

Fig. 3.8: Largest column space singular vector - English vs. Chinese

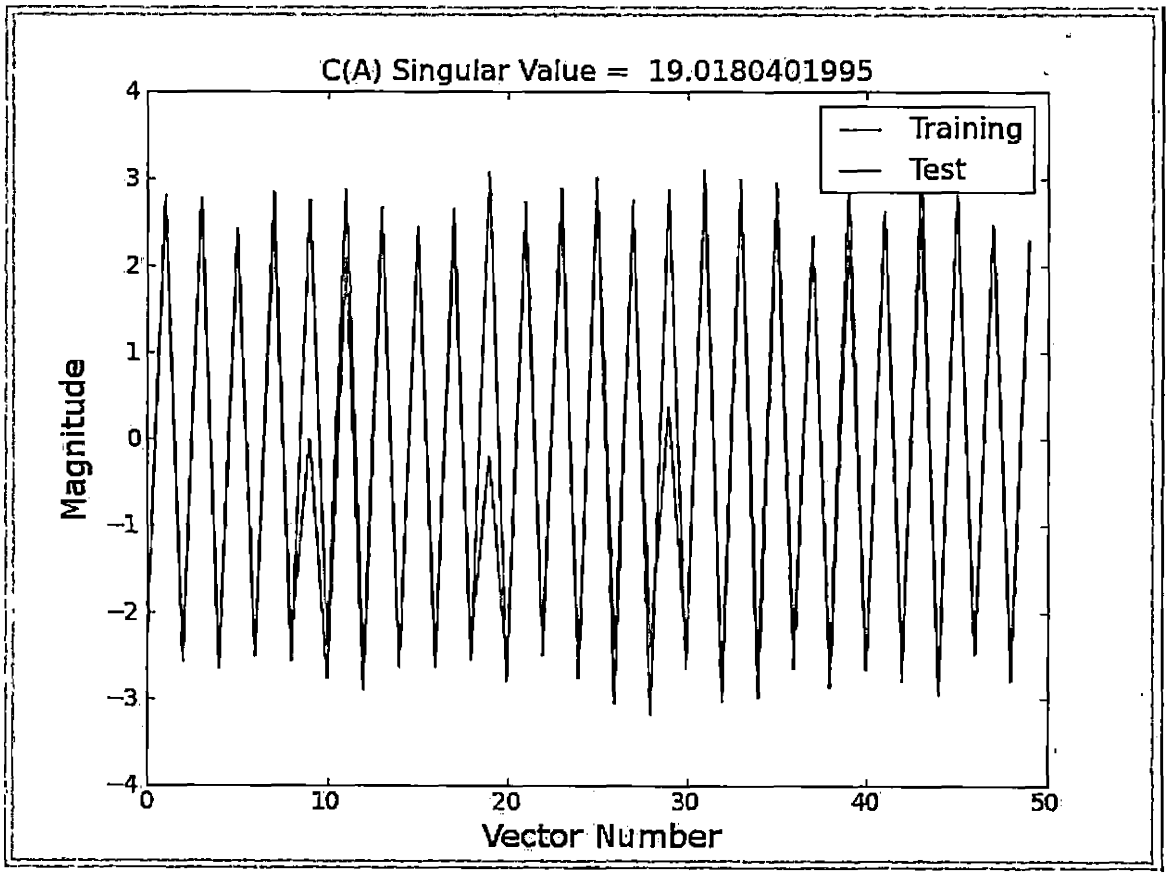

Fig. 3.9: 2nd largest column space singular vector - English vs. Chinese

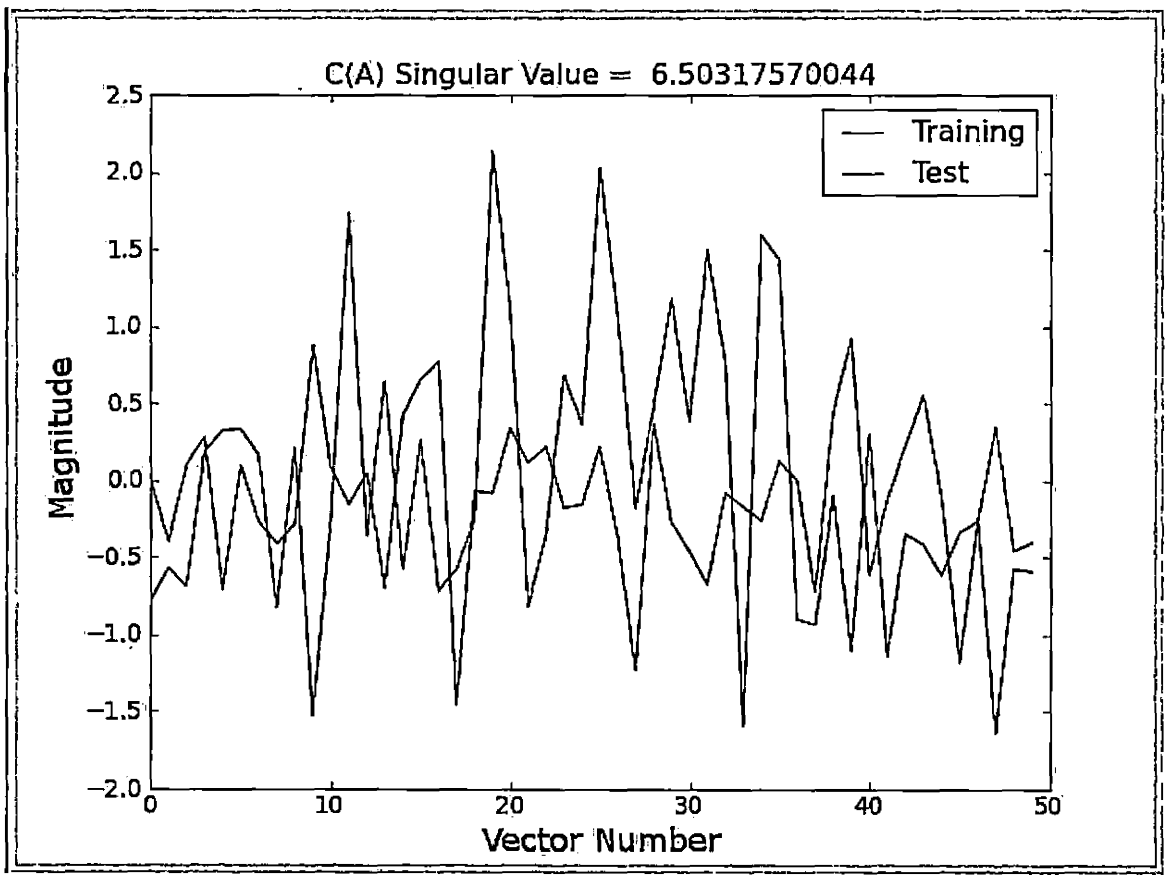

Fig. 3.10: 3rd largest column space singular value - English vs. Chinese

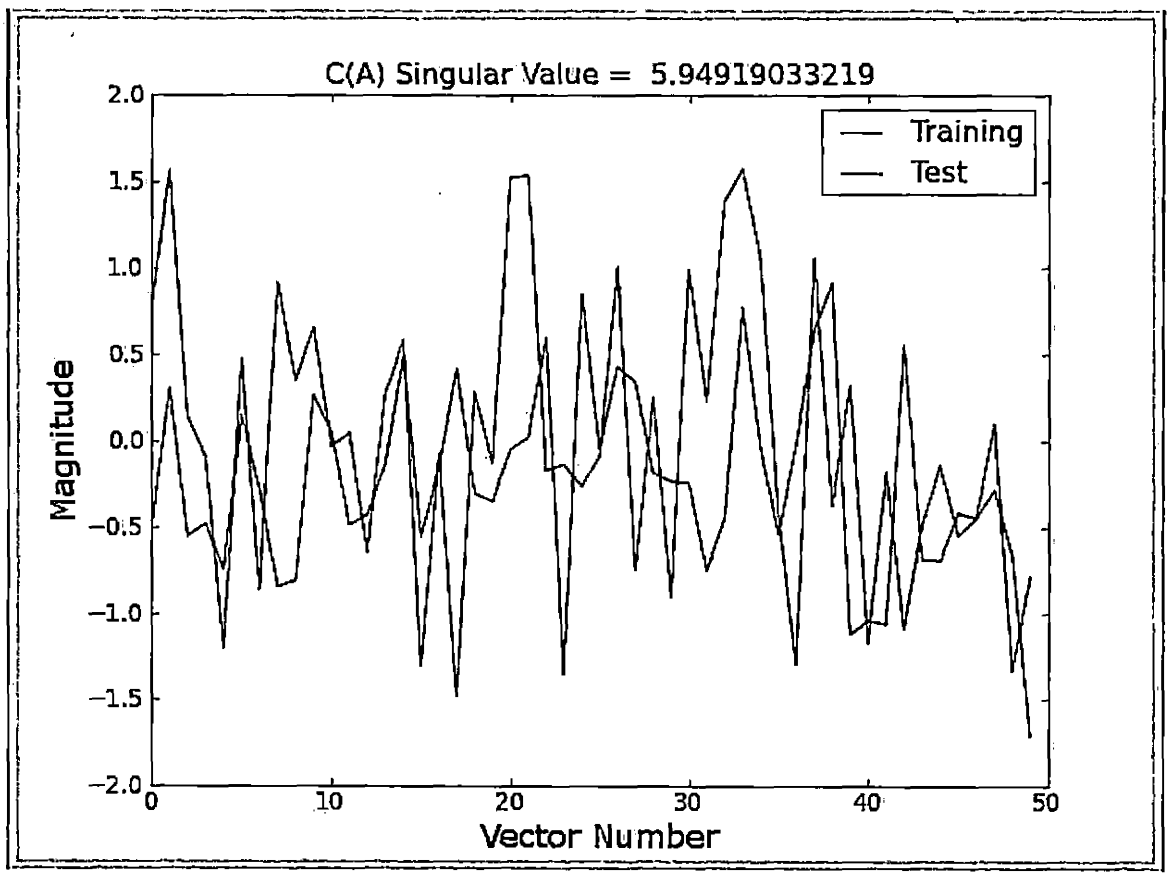

Fig. 3.11: Middle column space singular vector - English vs. Chinese

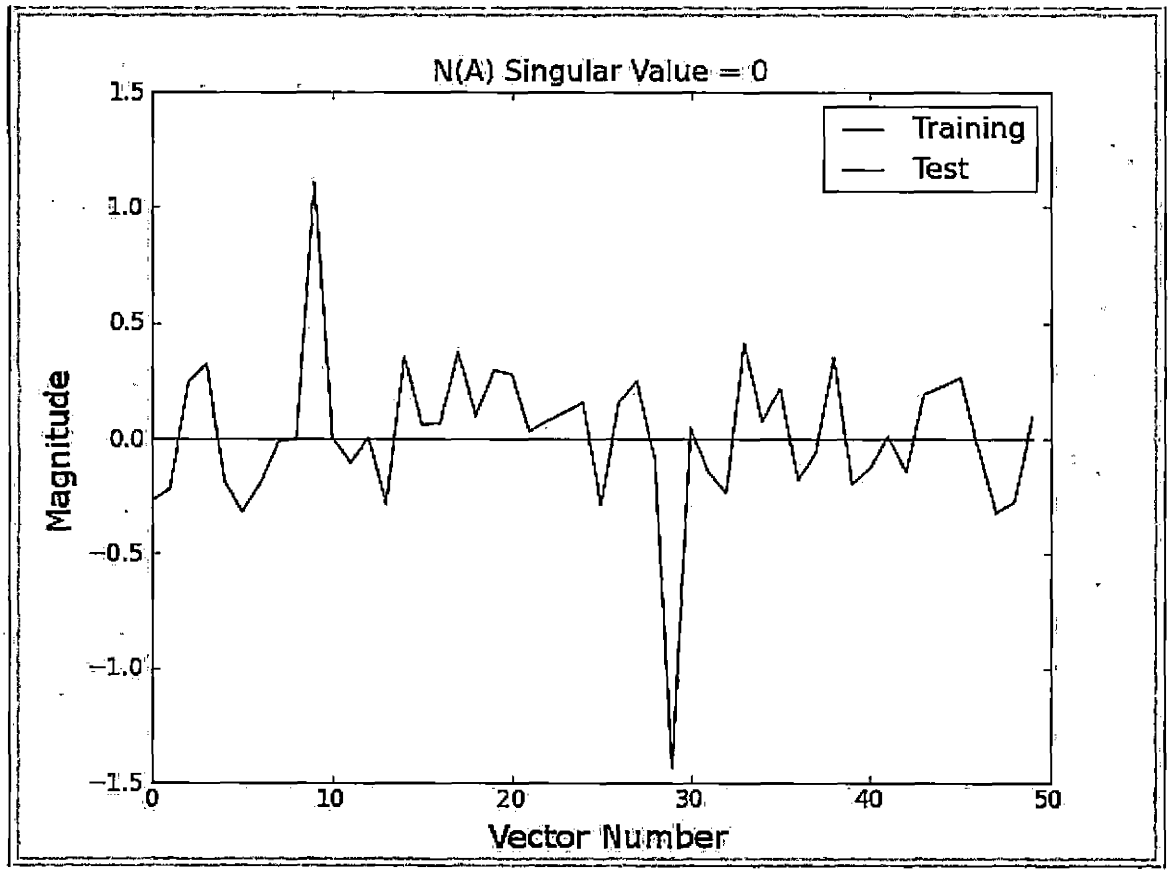

Fig. 3.12: 1st null space singular value plotting - English vs. Chinese

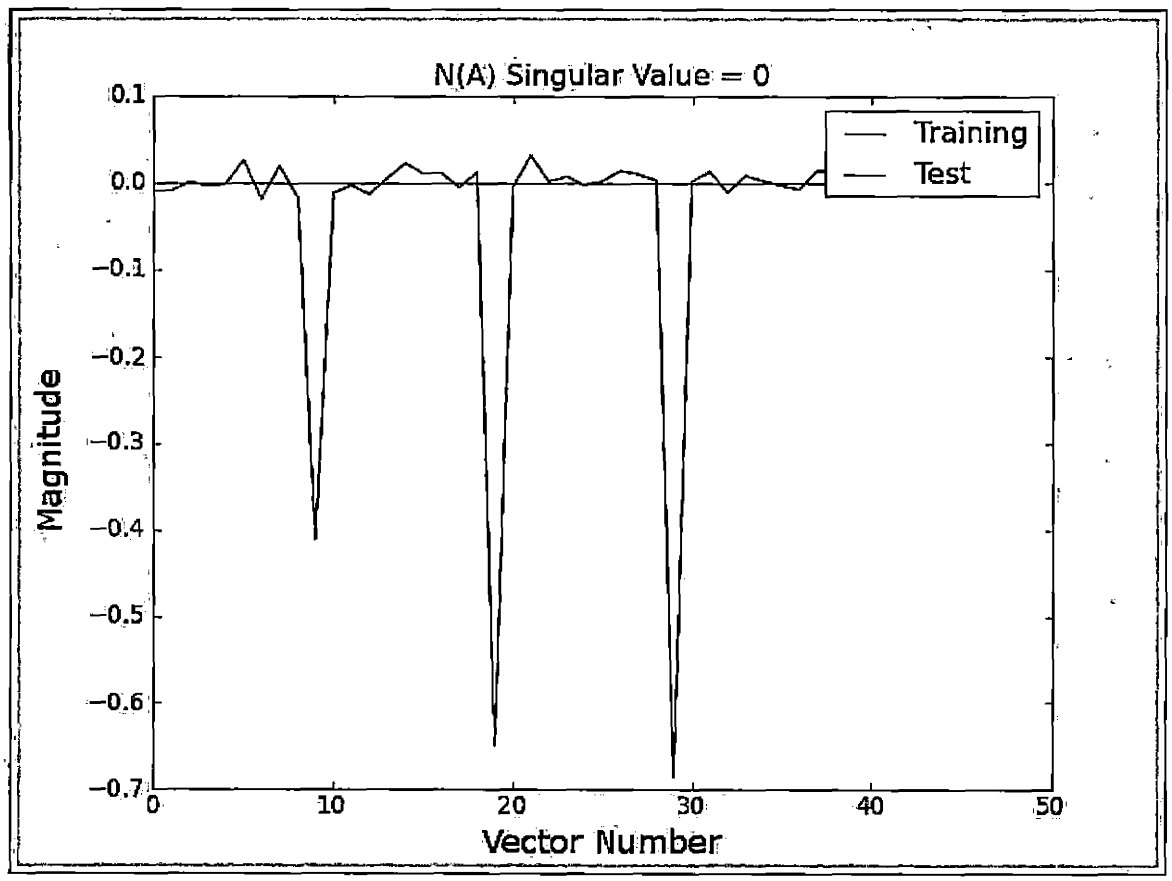

Fig. 3.13: 5th null space column space singular value - English vs. Chinese

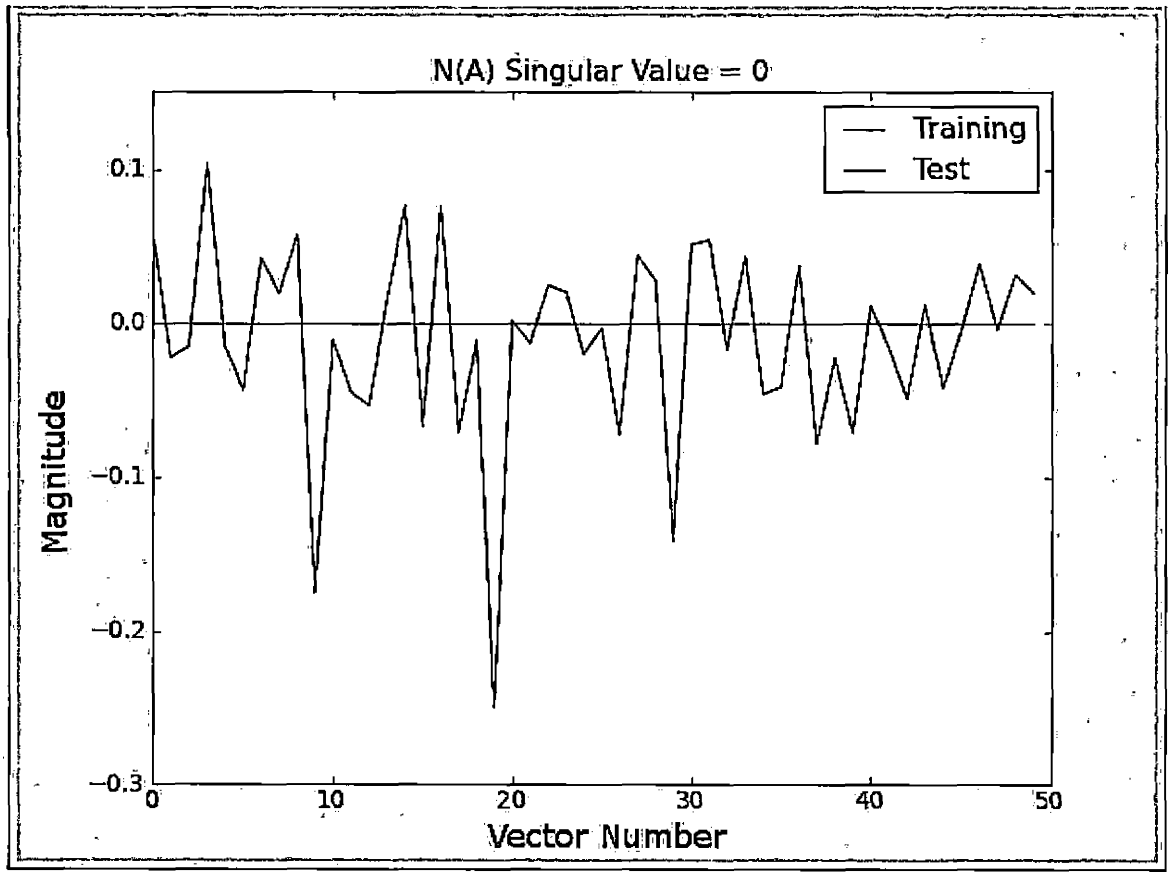

Fig. 3.14: 6th largest null space singular value - English vs. Chinese

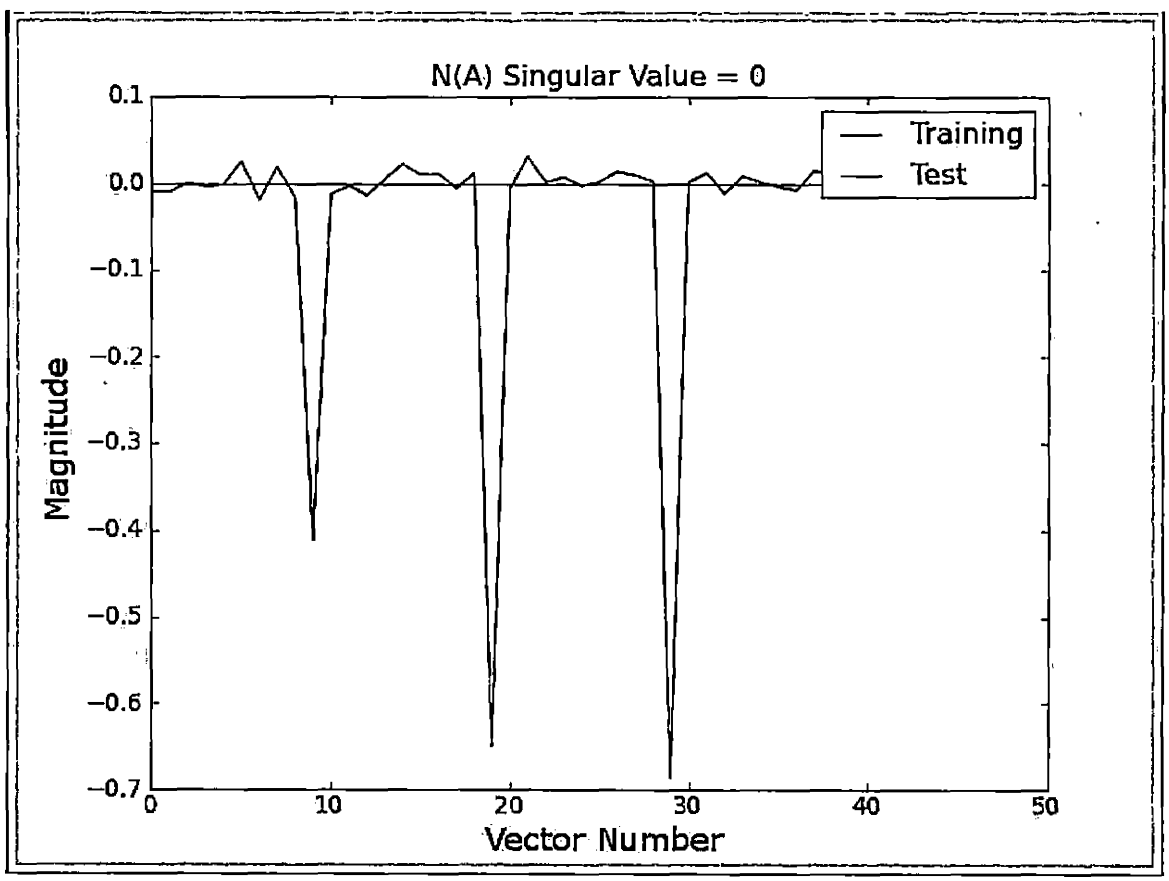

Fig. 3.15: Middle null space projection - English vs. Chinese

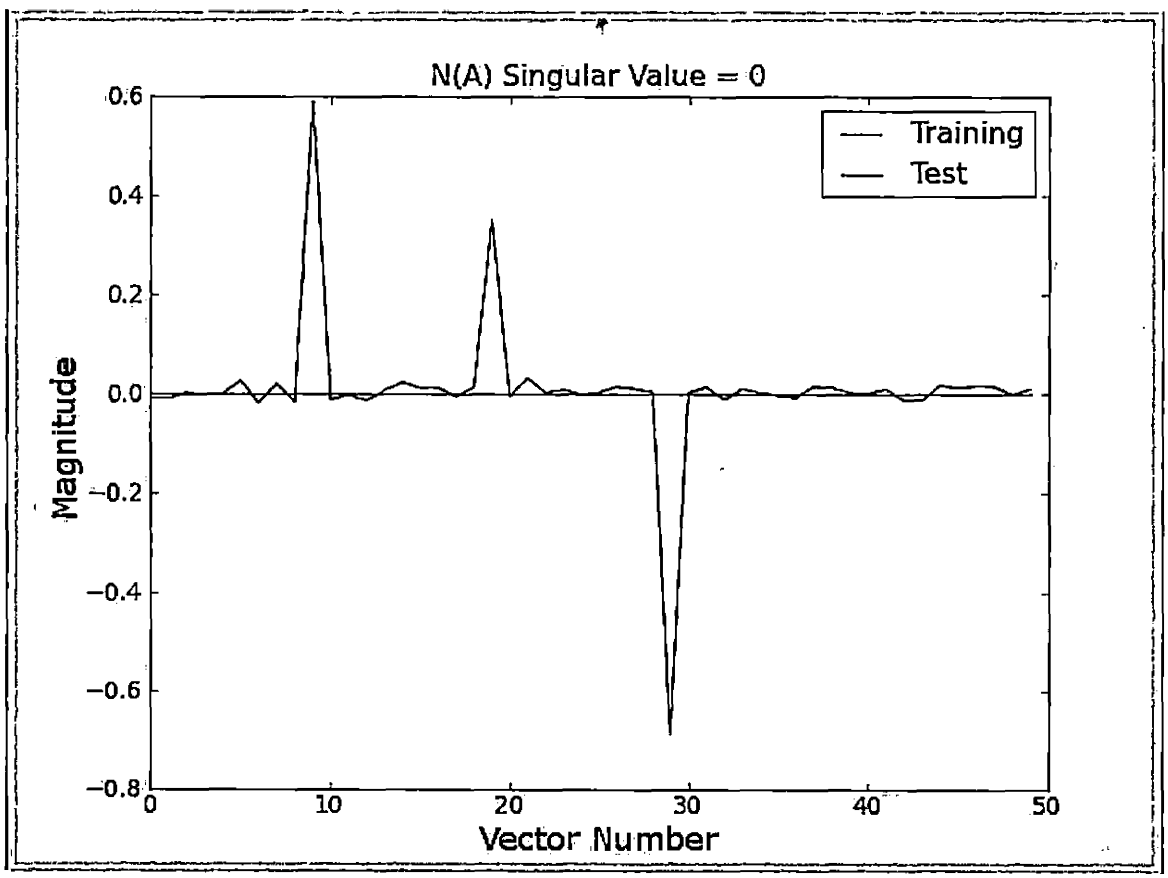

Fig. 3.16: Odd null space projection 1 - English vs. Chinese

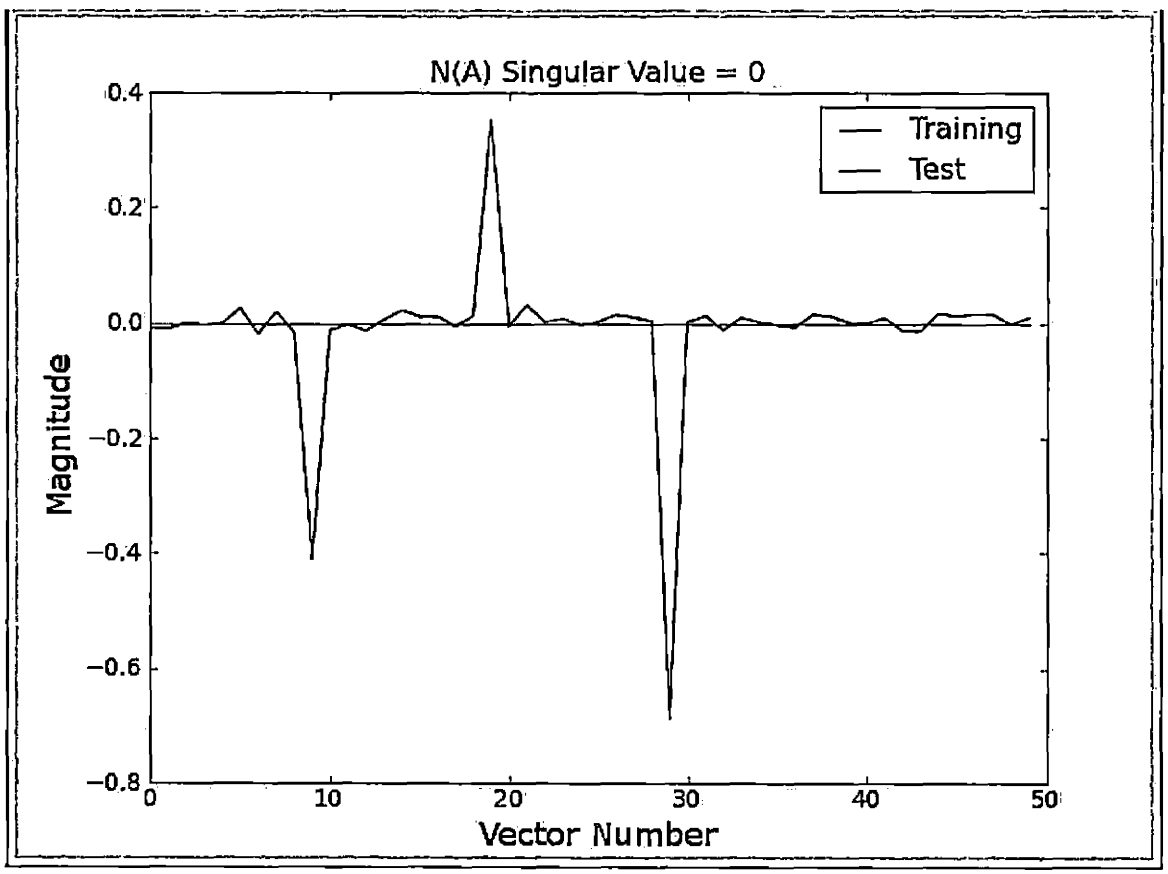

Fig. 3.17: Odd null space projection 2 - English vs. Chinese

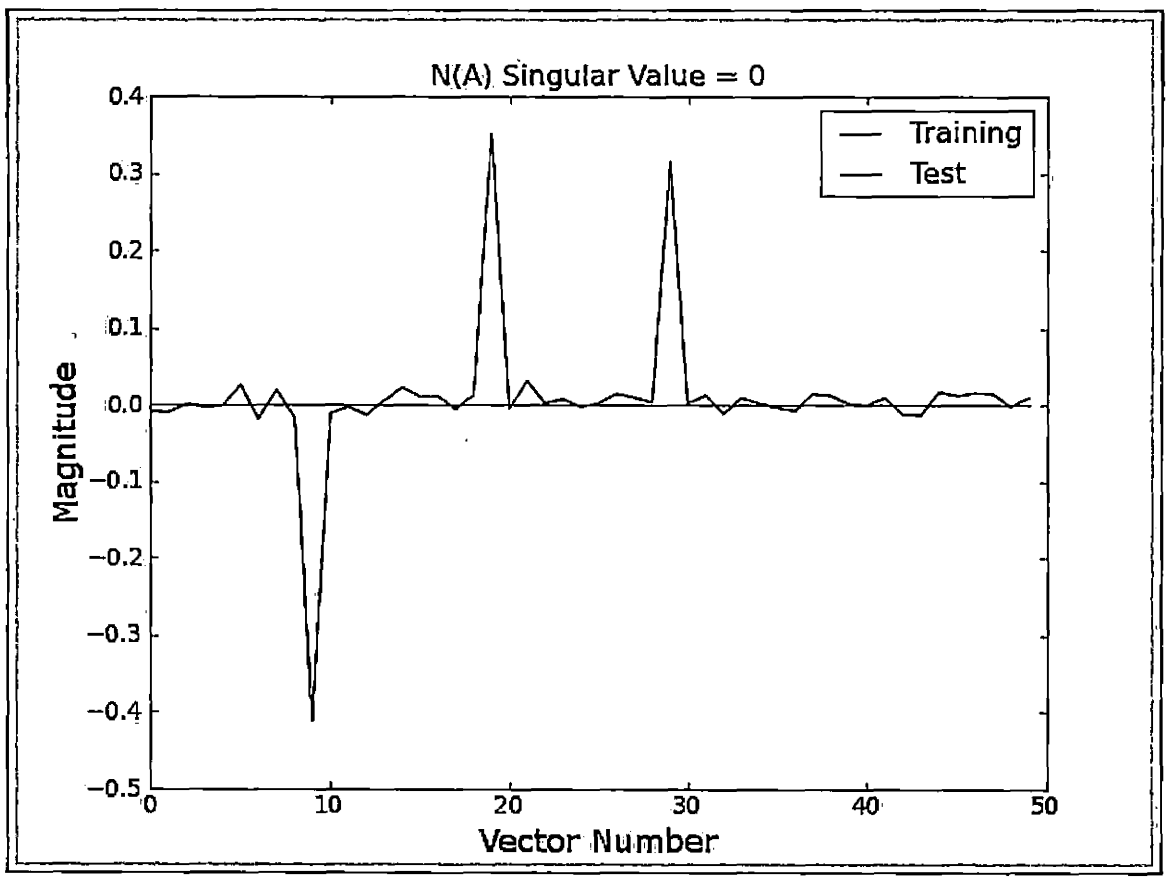

Fig. 3.18: Odd null space projection 3 - English vs. Chinese

but like the Chinese vs. English data tests, the positive results were not present in the first few sets of results in the Null Space. It took delving deeper into the Null Space before the characteristic magnitude spikes emerged. Like the Chinese vs. English results, not every projection in the Null Space provided clear illumination of the "bad" vectors. This particular set was important due to two factors. First, this was the first set where large data (Standard MTU sizes) taken from actual packets in the wild was analyzed. Second, this data set had direct implications to indicate usefulness in security applications. Unlike the ICMP data, which was constructed entirely in the lab, and had somewhat contrived "bad" payload data for its testing set, data that would not likely ever occur in the wild inserted into the payload, this data set had characteristics in the data that was representative of actual past attack vectors where garbage data is inserted into the payload portion. An example of an attack that would match this pattern would include a certain set of Distributed Denial of Service attacks, where the contents of the payload are meaningless junk and the goal is simply to overwhelm a network or server with data. Looking directly at the data, as previously mentioned, the first Singular Vector in the data set provided no clear results (see Figure 19), nor did the 2nd largest Singular Vector (see Figure 20), nor did the middle of the Column Space (see Figure 21), nor did any vector in the Column Space. Furthermore, like the Language tests, the first vector in the Null Space was also inconclusive (see Figure 22). However, further analysis of the Null space did provide very conclusive results as identified in Figures 23 and 24. These results were perhaps the most exciting of the entire range of testing as they proved conclusively that data taken directly from the wild could have the algorithm applied

to it with positive results. The implications of what could be done with this type of analysis given future research refining the algorithm appear to be far reaching, and will be discussed in the Future Directions section.

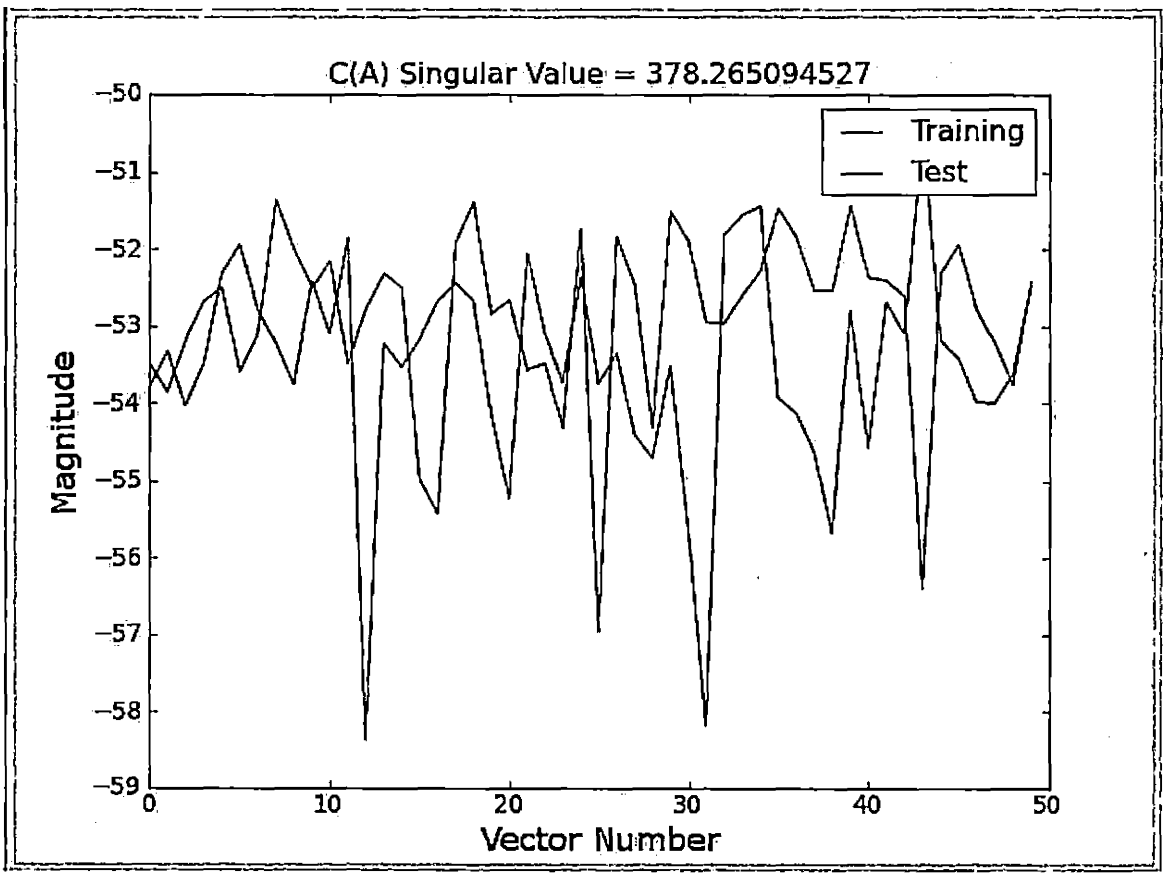

Fig. 3.19: Largest singular vector in the column space - Reddit Data

3.2.4 Set 4: Pandora TCP/HTTP

The data for the payload data from the website Pandora.com was the least successful, with the algorithm only able to detect a blatant manipulation of the data representative of something somewhat unlikely to appear in the wild. This test was run in two

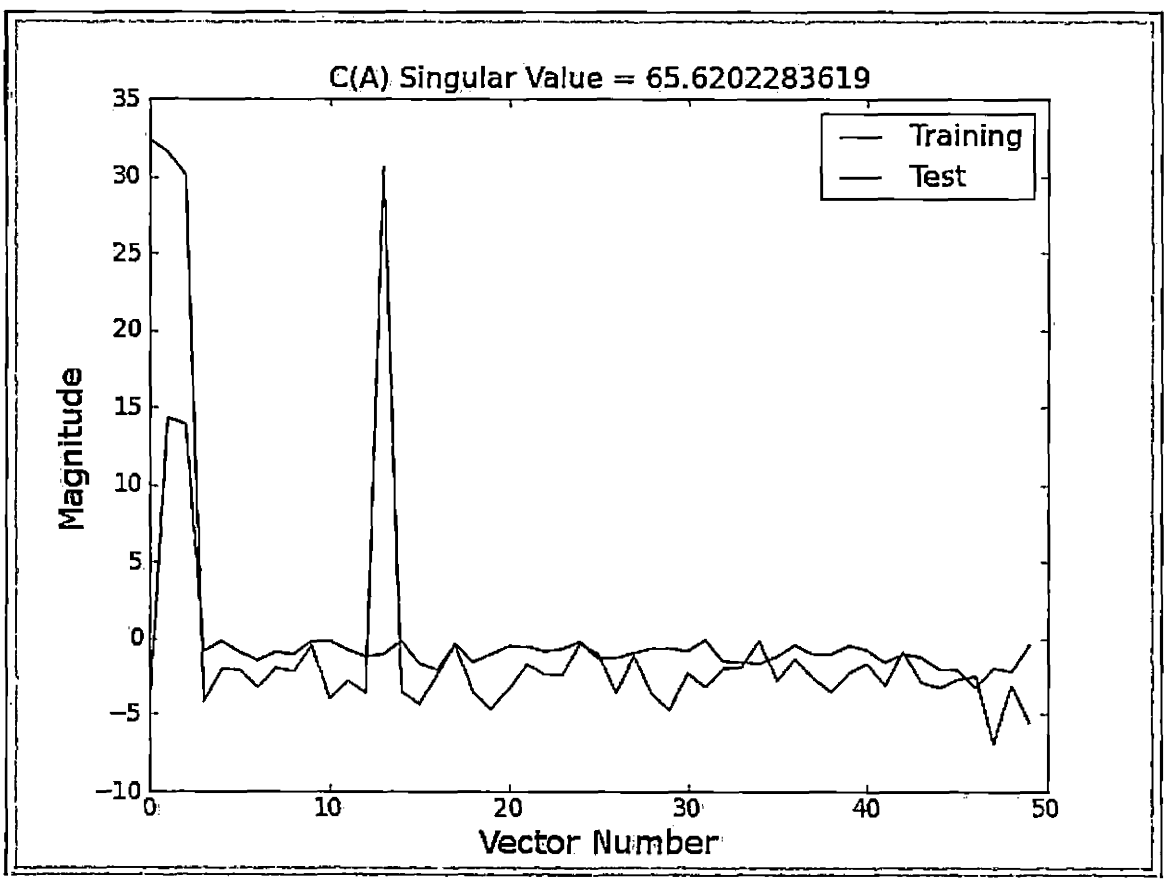

Fig. 3.20: 2nd largest singular vector in the column space - Reddit Data

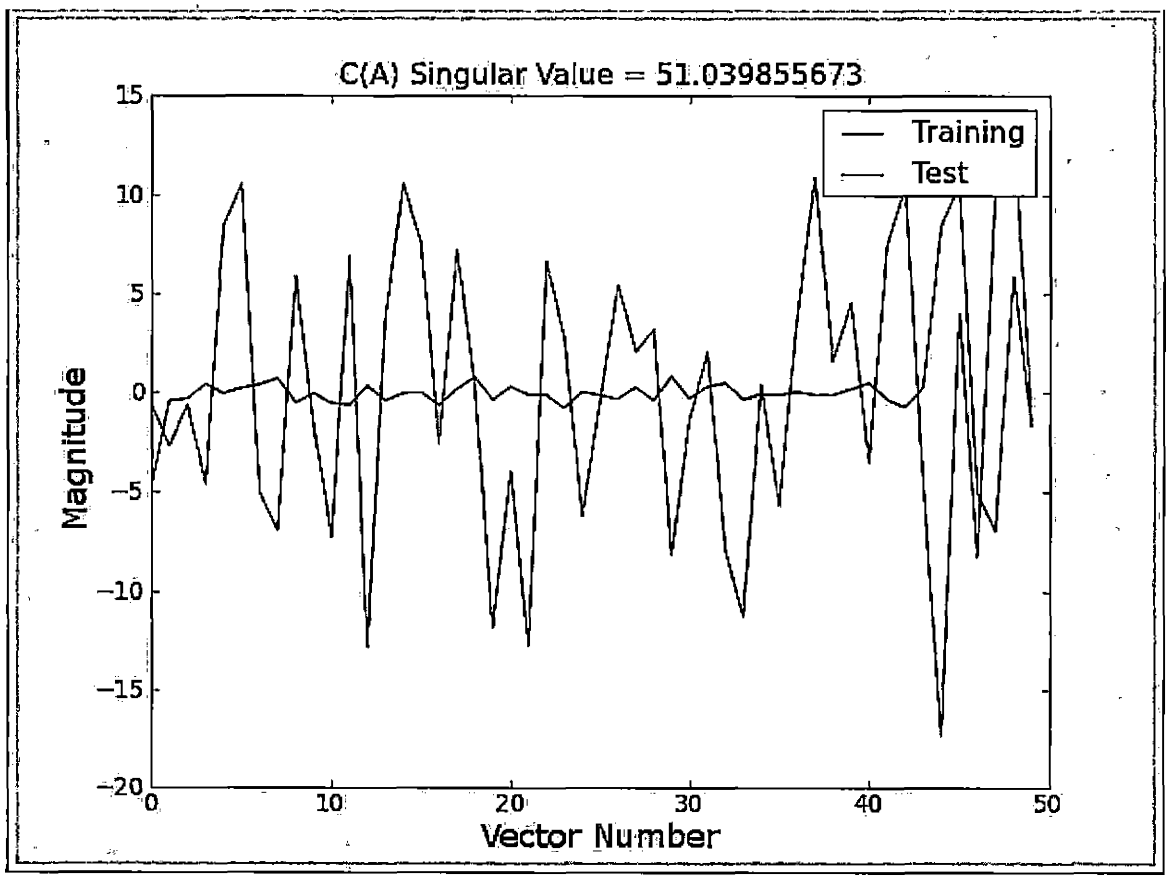

Fig. 3.21: Middle singular vector in the column space - Reddit Data

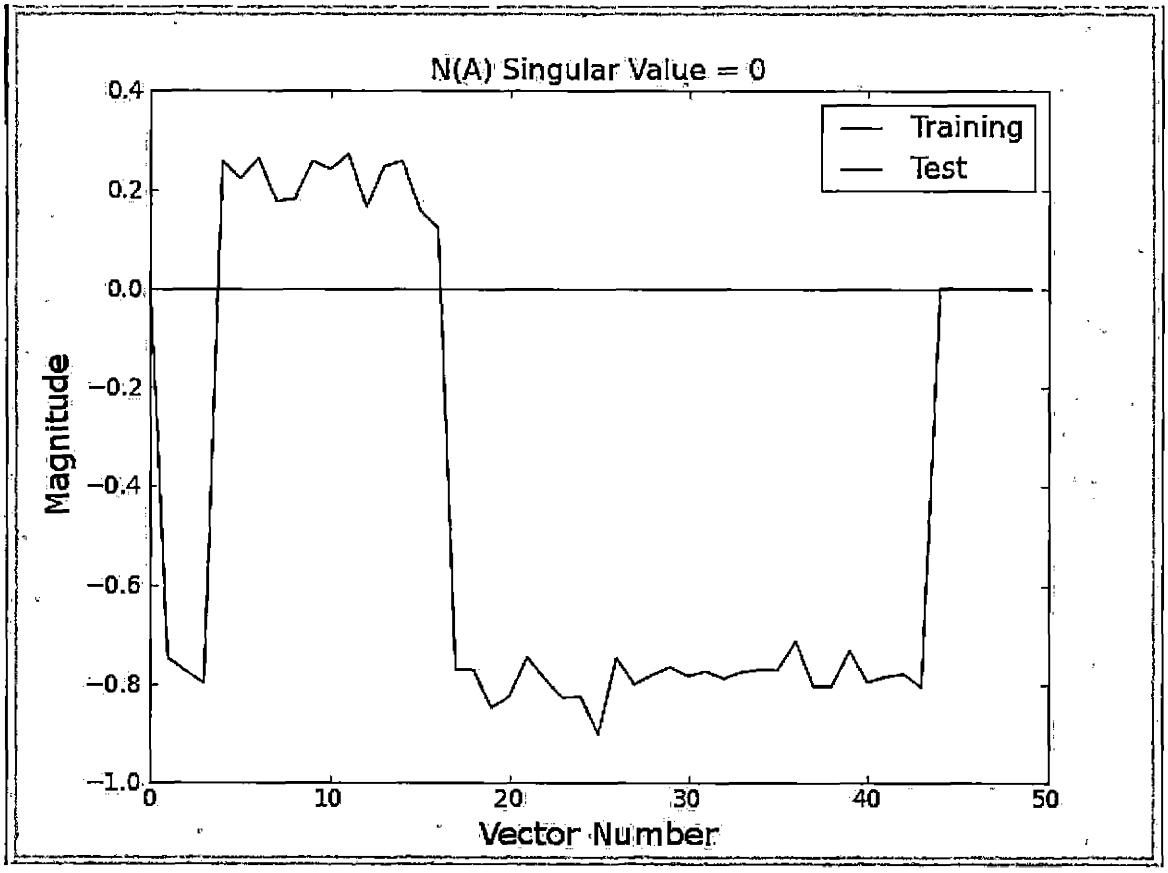

Fig. 3.22: 1st singular vector in the null space - Reddit Data.

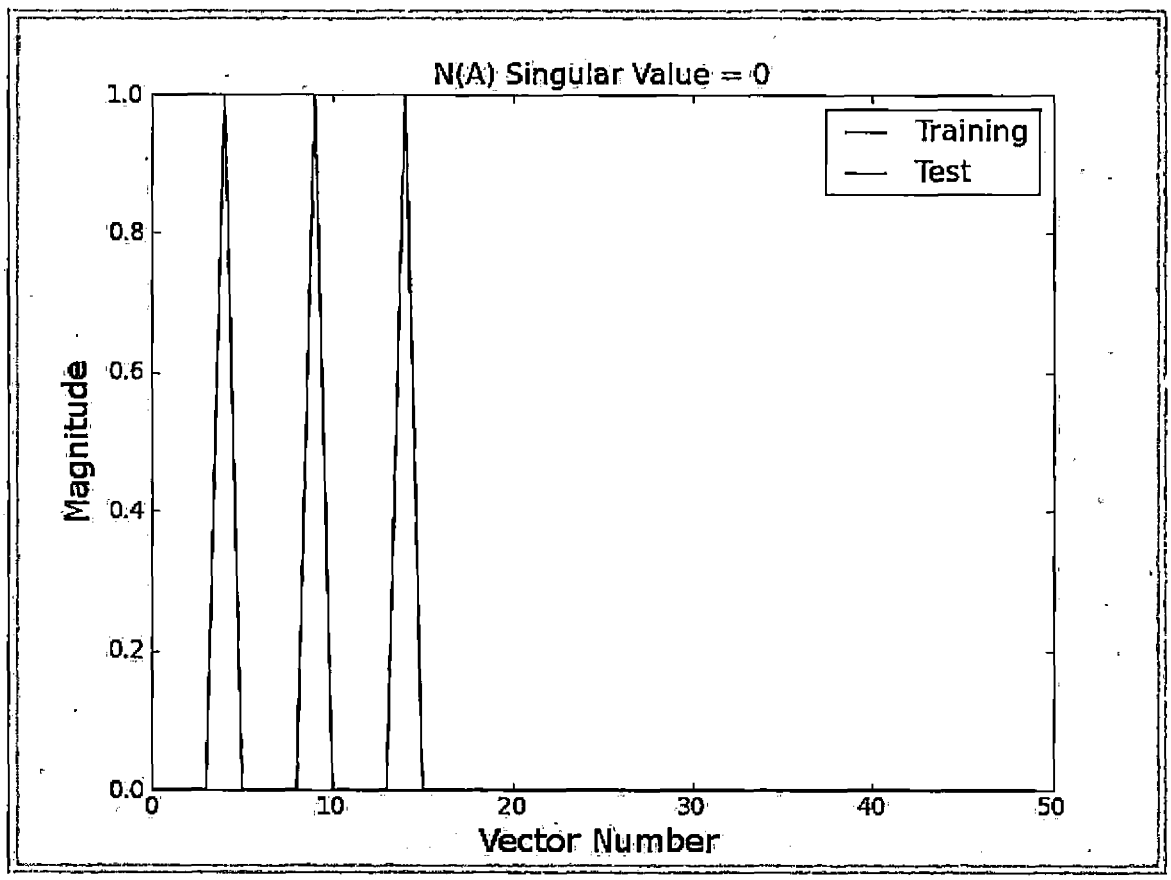

Fig. 3.23: Singular vector in the null space 1 - Reddit Data

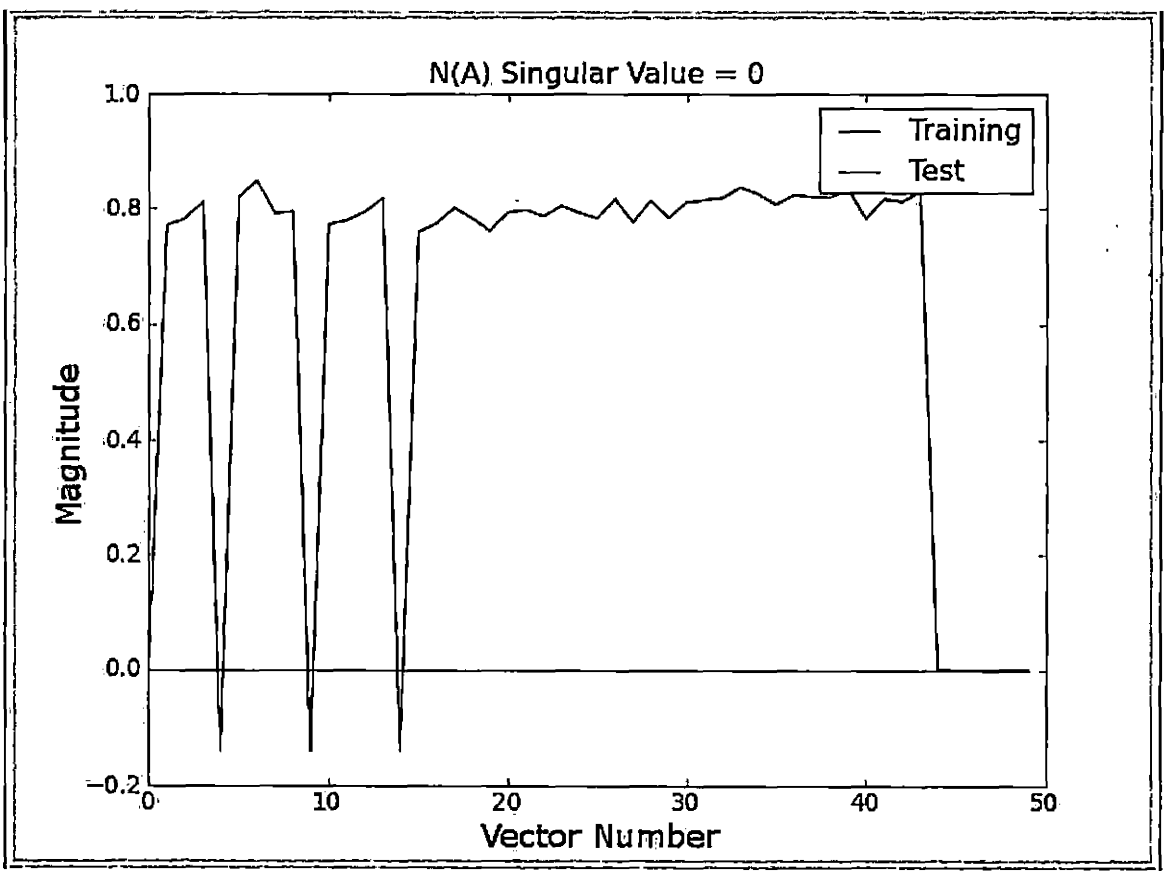

Fig. 3.24: Singular vector in the null space 2 - Reddit Data

phases, as the first phase was completely inconclusive. The first phase consisted of testing 200 packets of training data against 200 packets of testing data that contained three packets whose payloads consisted of arbitrary hex garbage similar to those used in the Reddit testing. The results of the first phase were that there were no sets where the "bad" packets were identified. After these disappointing results, a second phase was initiated where an additional "bad" packet consisting of 800 bytes of just 1s in addition to the regular garbage from the previous phase was included. This did yield a positive result; however, even with such a blatantly artificially created packet, there was only one set in the data where the algorithm was able to identify the packet; namely, the largest singular value in the Column Space (see Figure 25). The last singular value was inconclusive (see Figure 26), as was the middle of the Column Space (see Figure 27). The Null Space proved no more illuminating either in the first value in the Null Space (see Figure 28), or in the middle of the Null Space (see Figure 29), or anywhere in the Null Space. The fact that the Null Space had such an abundance of arbitrary strong projections unrelated to the "bad" packets would seem to indicate that the reason for failure in this space was due to too small of a data set used for testing, indicating the entirety of the domain was not covered by the training data, this may have been leakage of Column Space vectors appearing incorrectly in the Null Space calculation sets. Further testing with greatly expanded data sets could prove fruitful, and the fact at least a very skewed packet was detectable provides hope that further testing could prove successful.

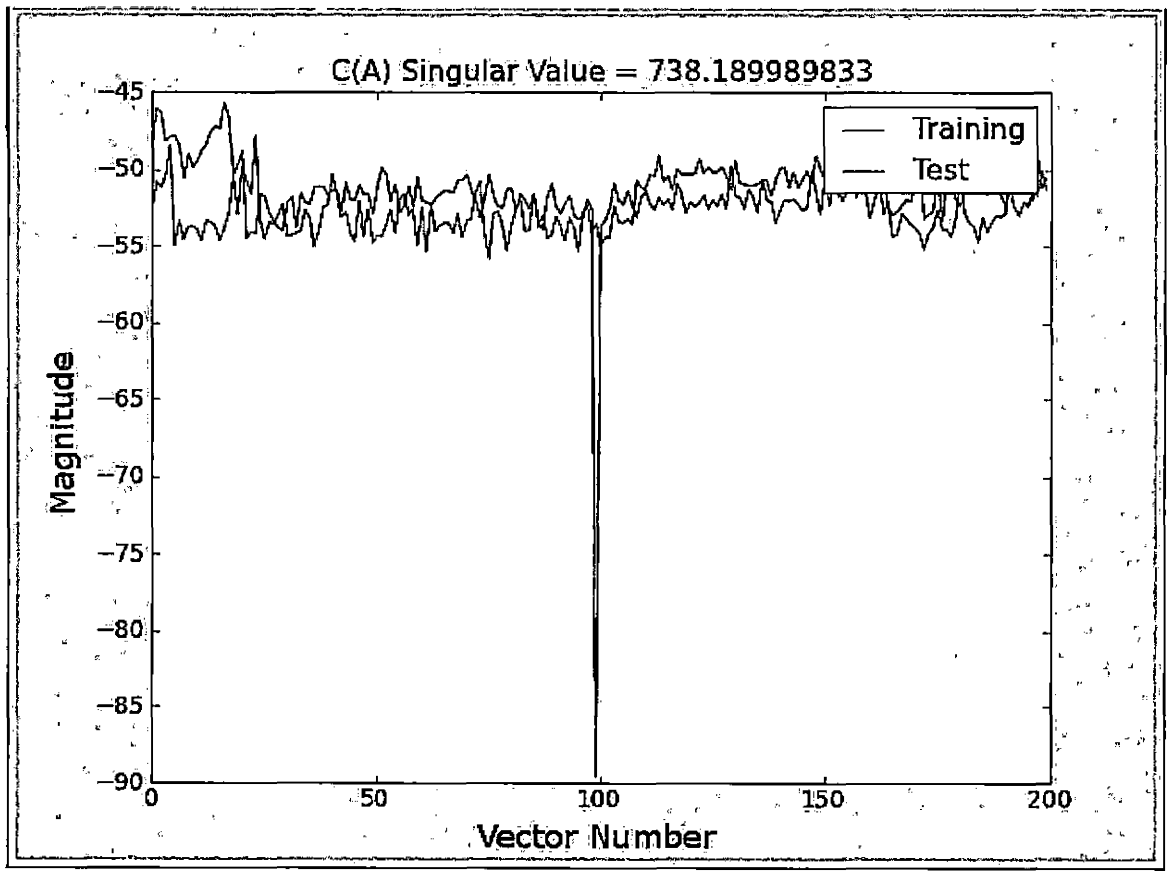

Fig. 3.25: 1st singular vector in the column space - Pandora Data

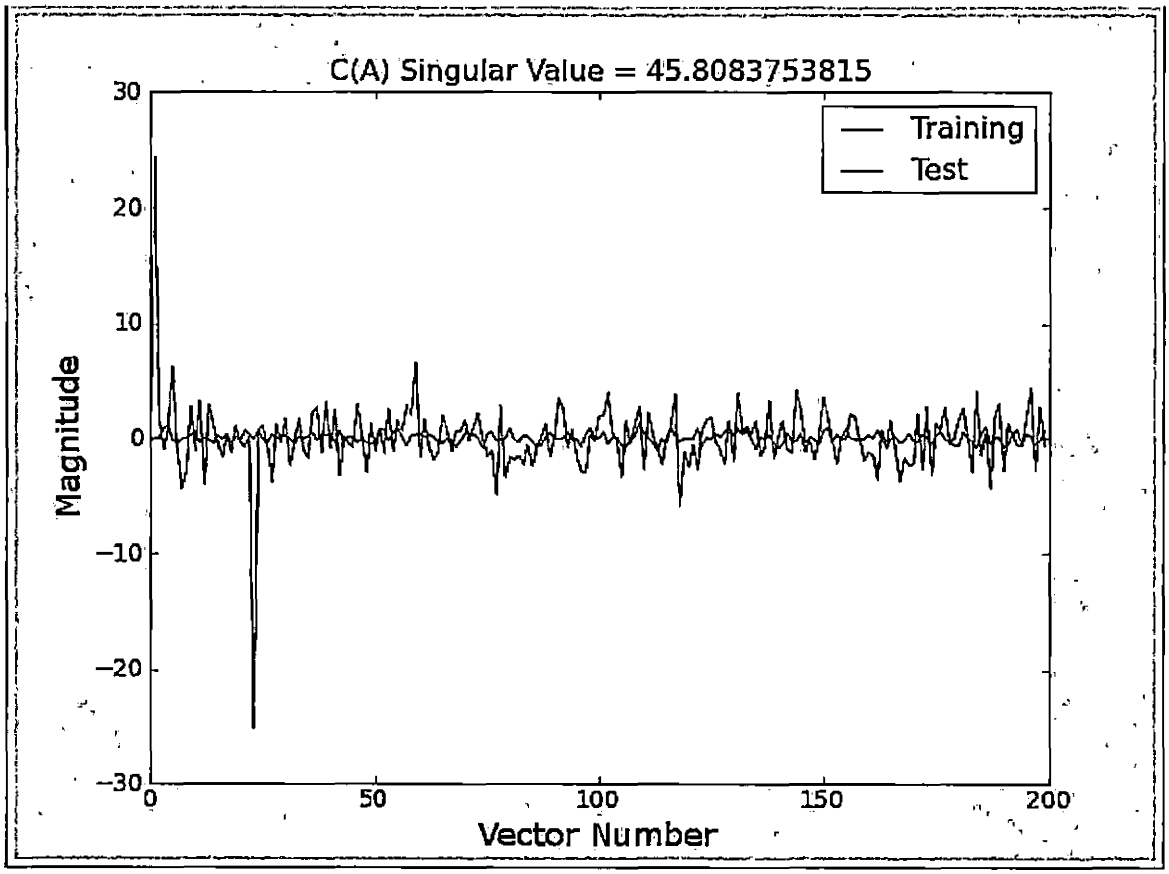

Fig. 3.26: Last singular vector in the column space - Pandora Data

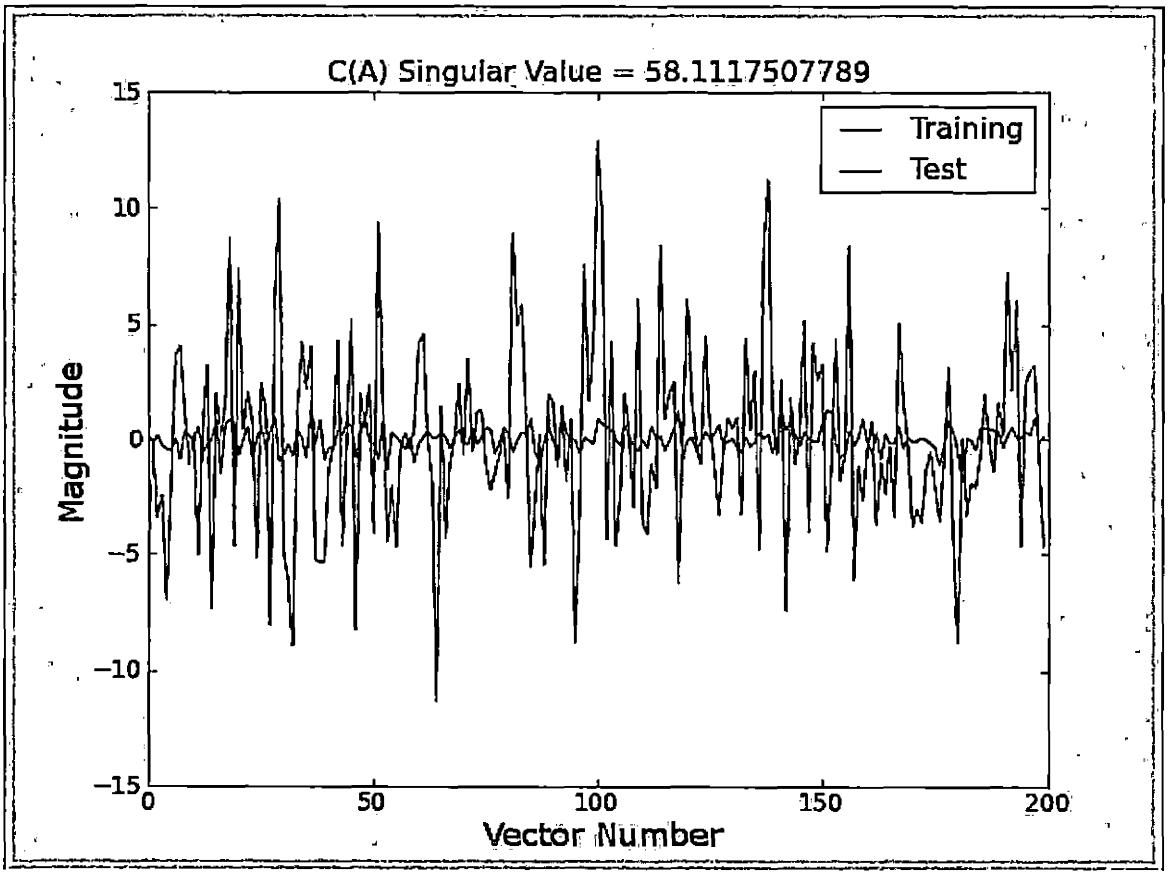

Fig. 3.27: Middle singular vector in the column space - Pandora Data

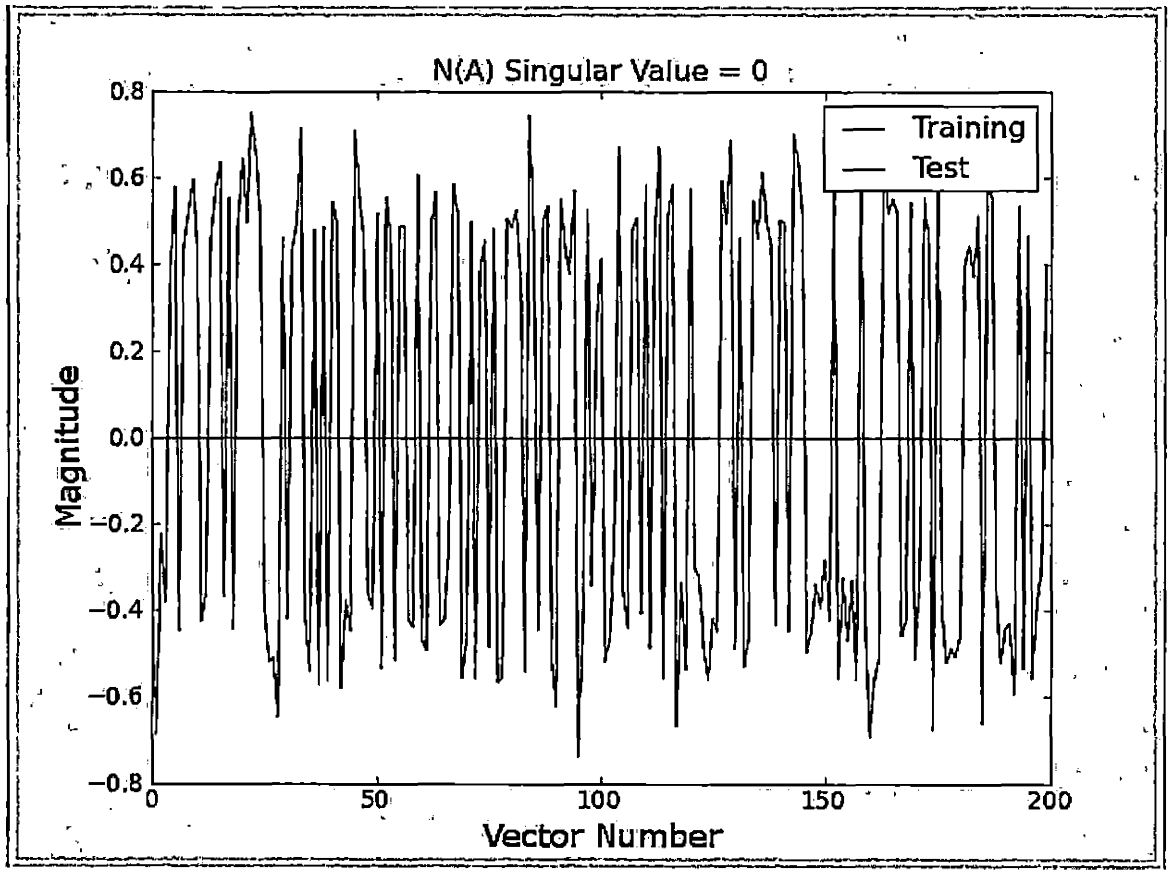

Fig. 3.28: 1st singular vector in the null space - Pandora Data

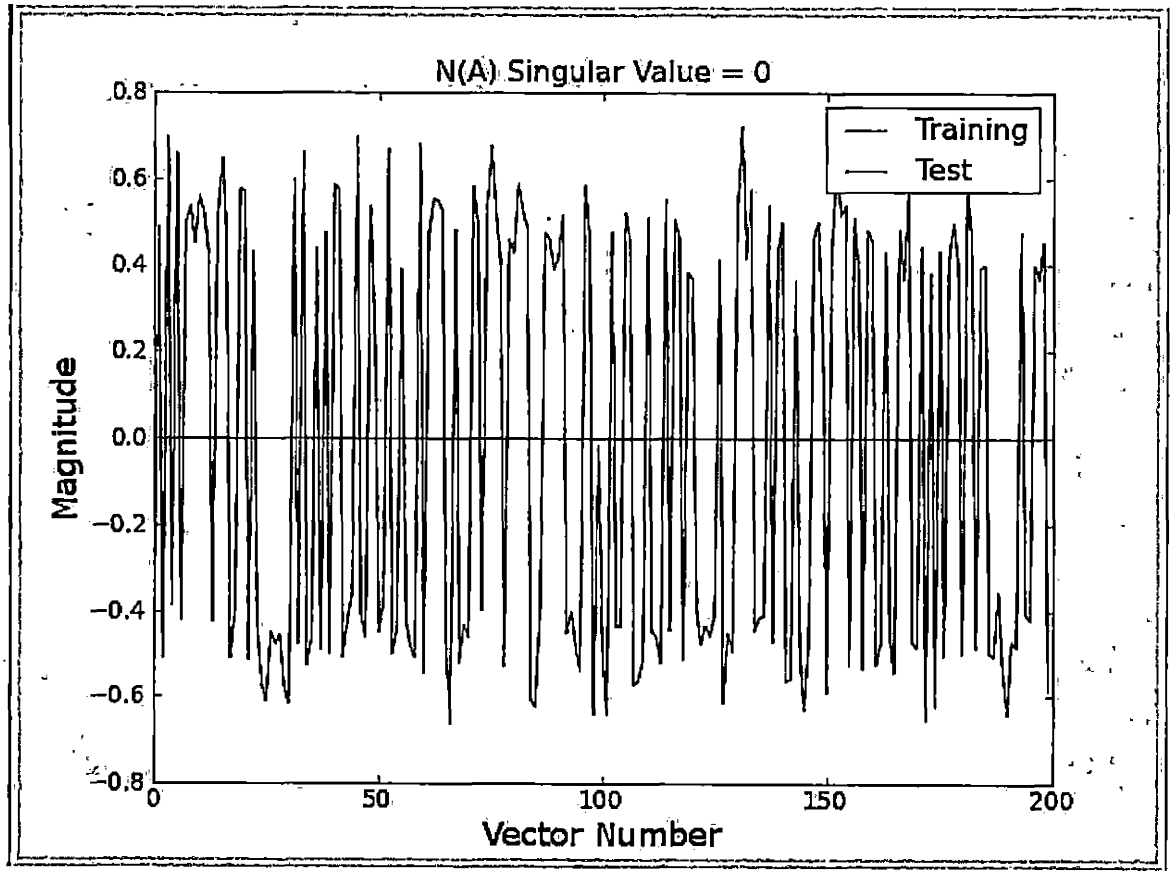

Fig. 3.29: Middle singular vector in the null space - Pandora Data

3.2.5 *Set 5: Reddit.com vs. Chinese Unicode*

This was a hybrid test which combined data from two previous tests in order to see if further conclusions could be drawn. In this test, the previous Reddit.com data was truncated to match the data size of the Chinese Unicode test set. It was thought a valuable exercise to see if the largely English ASCII packet data gathered in the wild from Reddit.com could be tested against a testing set that contained additional Reddit.com data, as well as the Chinese Unicode from the previous test. The initial "bad" packets from the first testing set were included as well to provide a contrast in "bad" packet data to see if there was any valuable information to be gathered with regard to how one "bad" packet was identified compared to a completely different kind of "bad" packet. The answer was that there was no obvious distinction between the two types of "bad" packets. The results were again quite conclusive with positive results showing in both the Column Space and the Null Space. Note that there are now six spikes (three from the old garbage data, three from the new Chinese Unicode data) in this first set from the largest singular vector in the Column Space (see Figure 30). Holding to a pattern seen in earlier testing, the clarity disappears when moving to the second largest of the Singular Vectors in the Column Space (see Figure 31). The third Singular Vector in the Column Space proved no more fruitful (see Figure 32). However, once entering the Null Space, the "bad" vectors were again easily identifiable in a number of instances. The first singular vector in the Null Space was able to detect all six "bad" vectors (see Figure 33). The second Singular Vector was also able to identify all three "bad" vectors (see Figure 34). As was the third (see Figure 35). There were several other instances in the Null Space where all six "bad"

vectors were easily identifiable, as well as some very strange behavior where some, but not all of the vectors were identified (see Figures 36, 37, 38). All told, the algorithm was very successful in identifying the Chinese Unicode characters (in binary form) mixed amongst the Reddit.com data, as well being able to still identify the initial garbage data. There did appear to be some differences in delineating between the Chinese Unicode and the garbage data, as evidenced by some of the stranger results in Figures 36-38; however, the algorithm was equally successful in identifying the entire group of anomalous data.

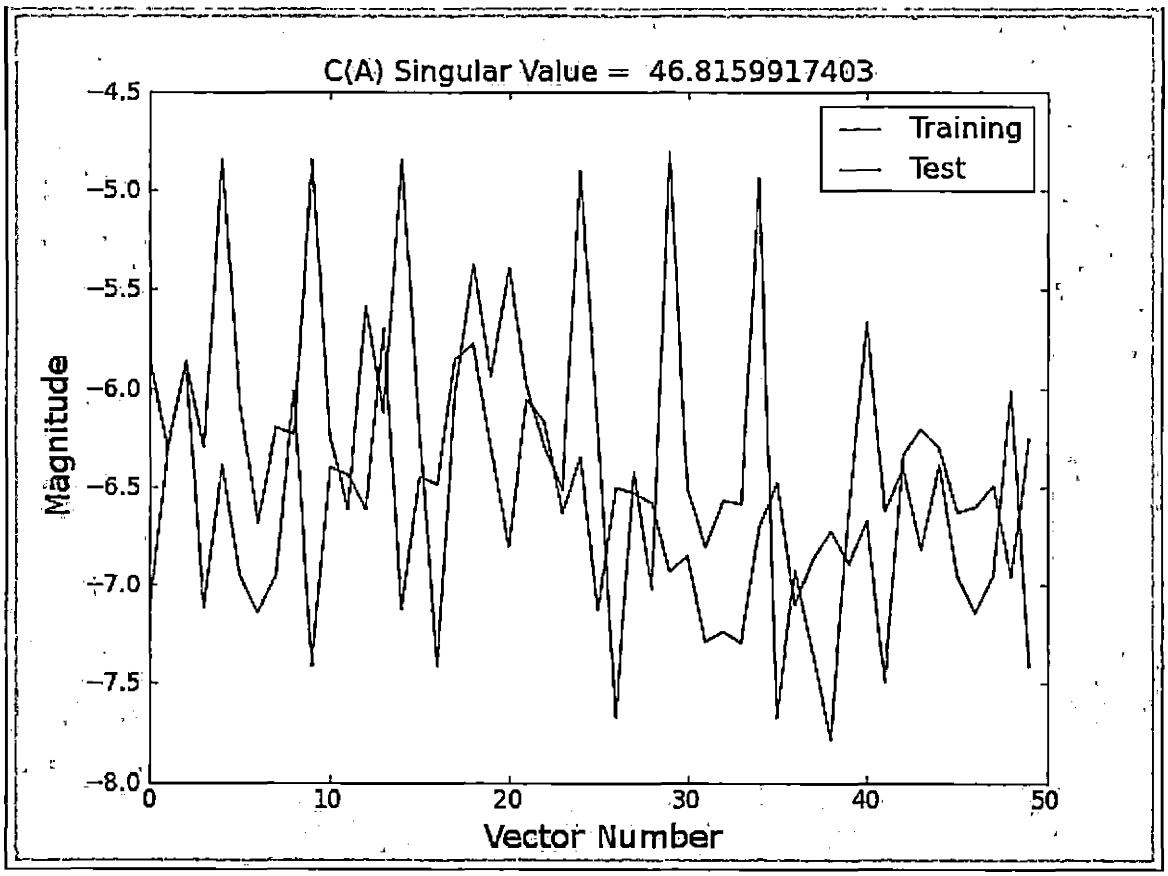

Fig. 3.30: Largest singular vector - Reddit vs. Chinese

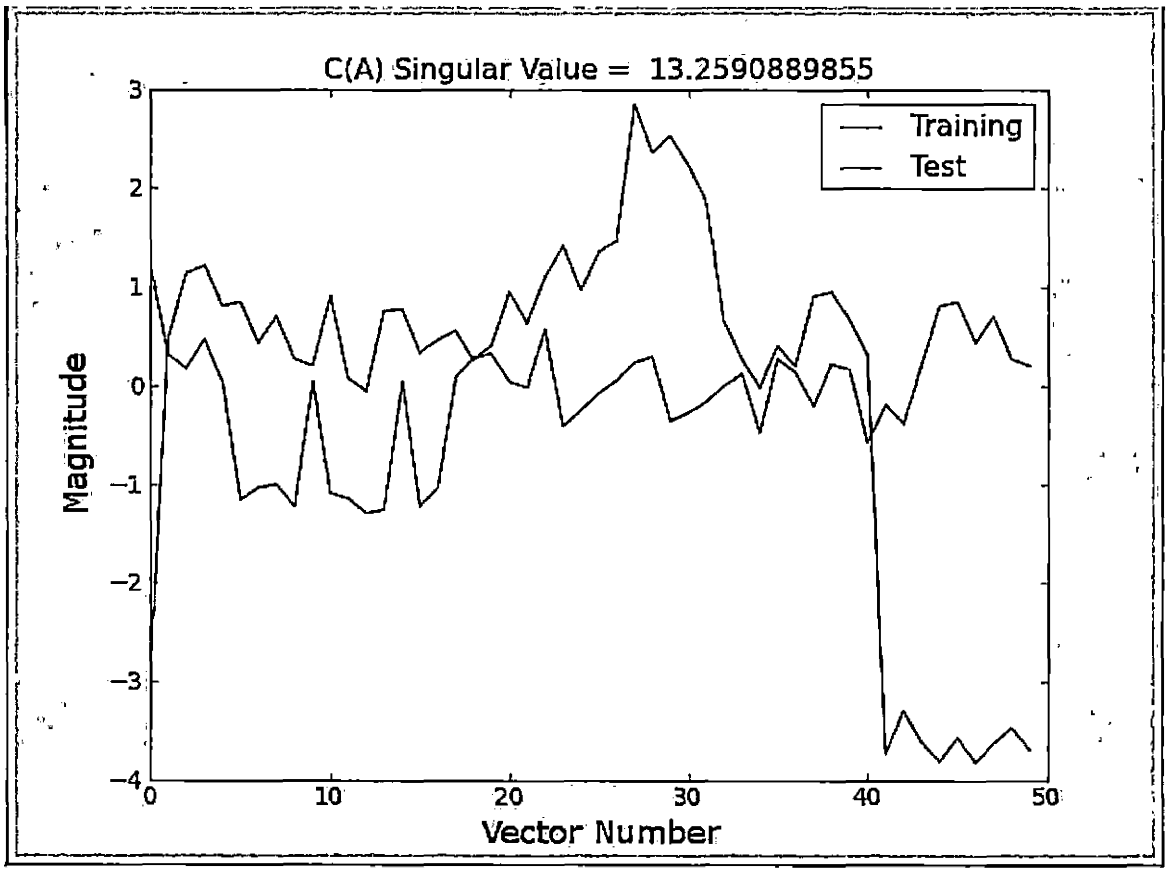

Fig. 3.31: 2nd largest singular vector - Reddit vs. Chinese

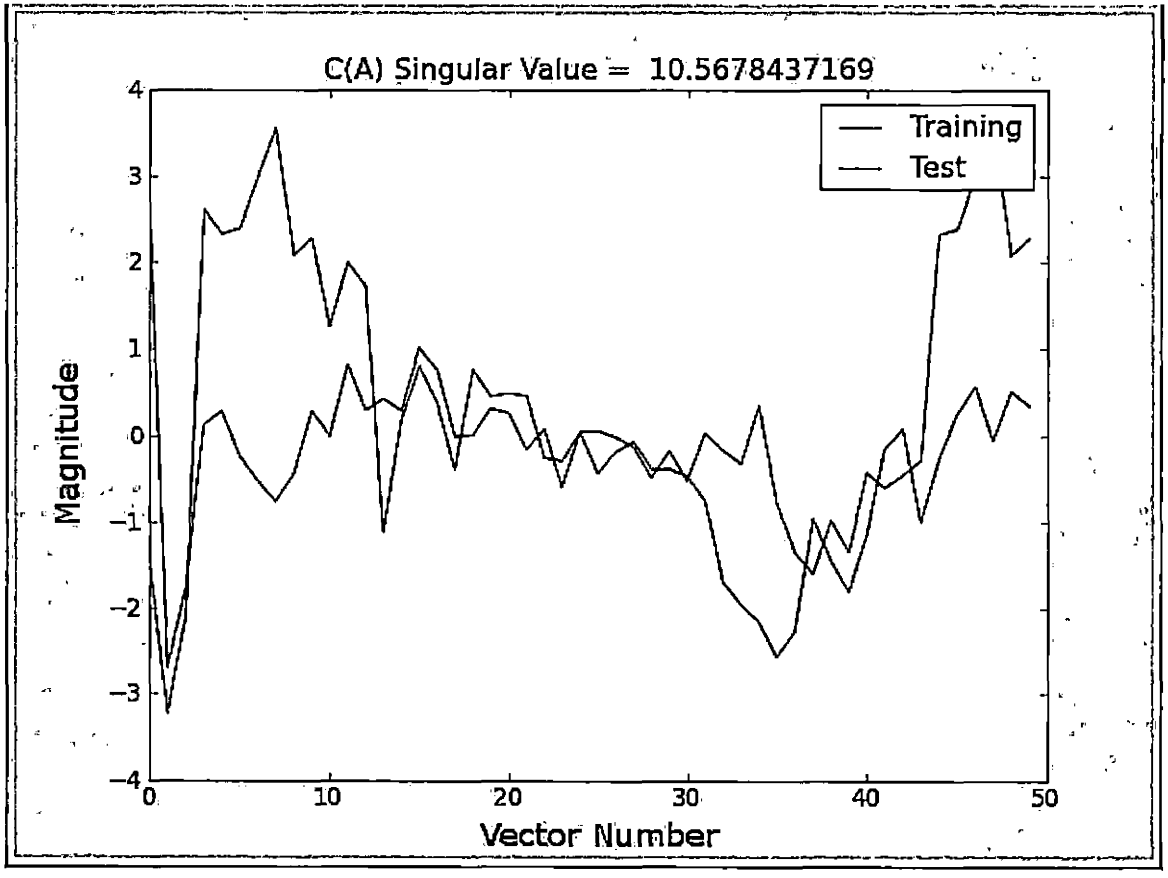

Fig. 3.32: 3rd largest singular vector - Reddit vs. Chinese

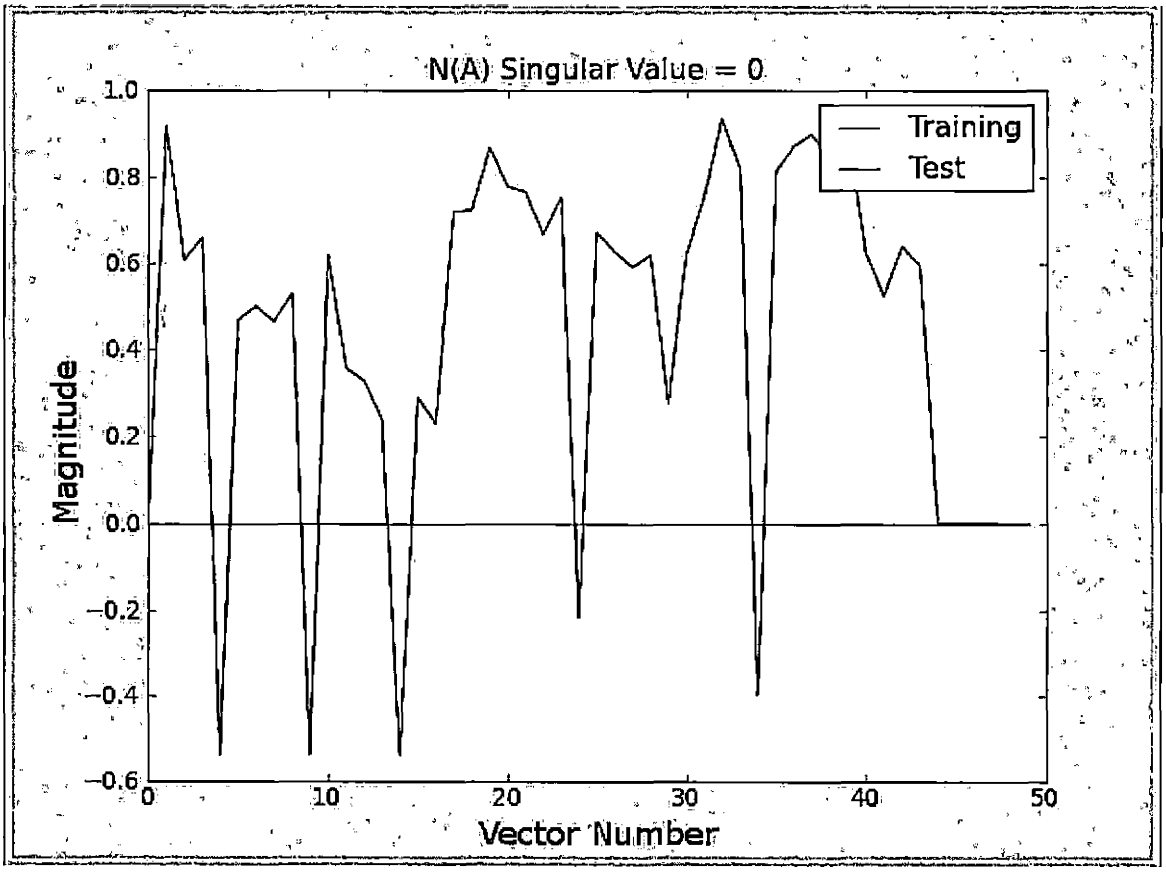

Fig. 3.33: 1st singular vector in the null space - Reddit vs. Chinese

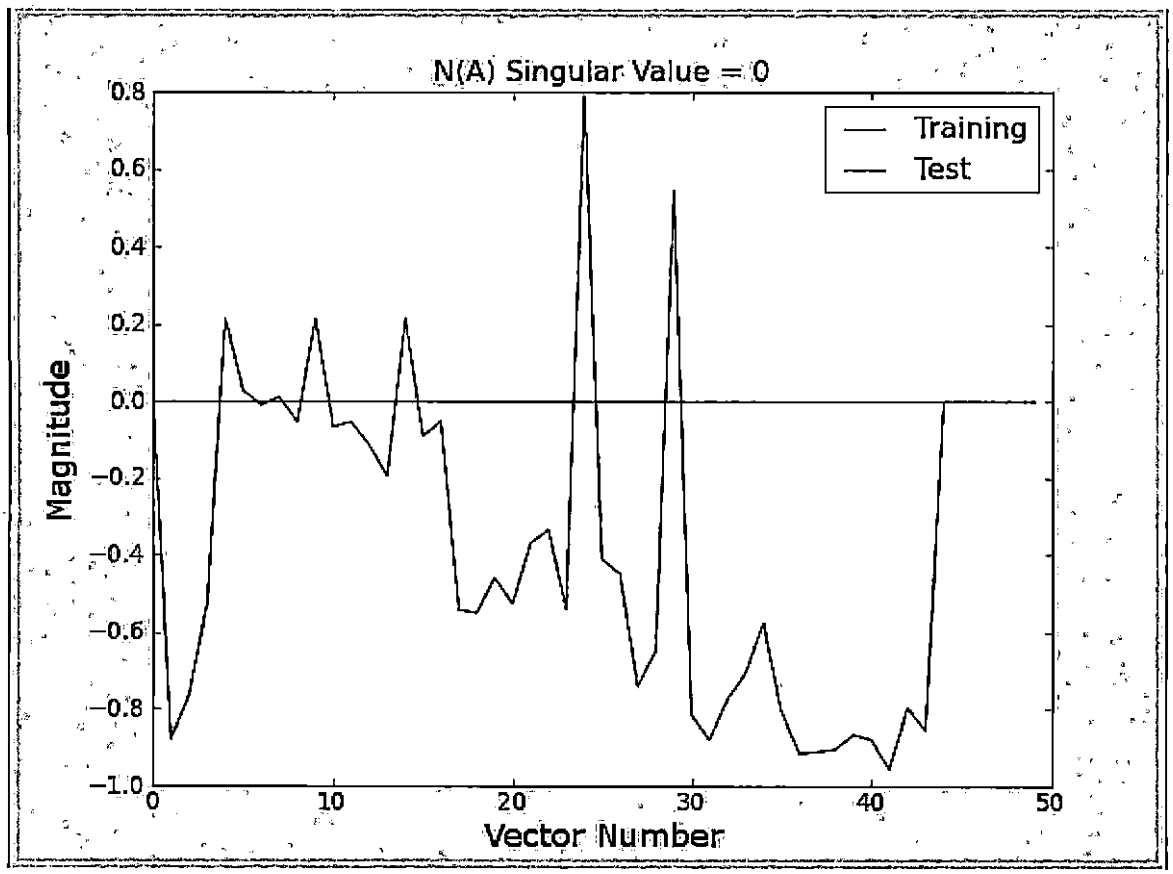

Fig. 3.34: 2nd singular vector in the null space - Reddit vs. Chinese

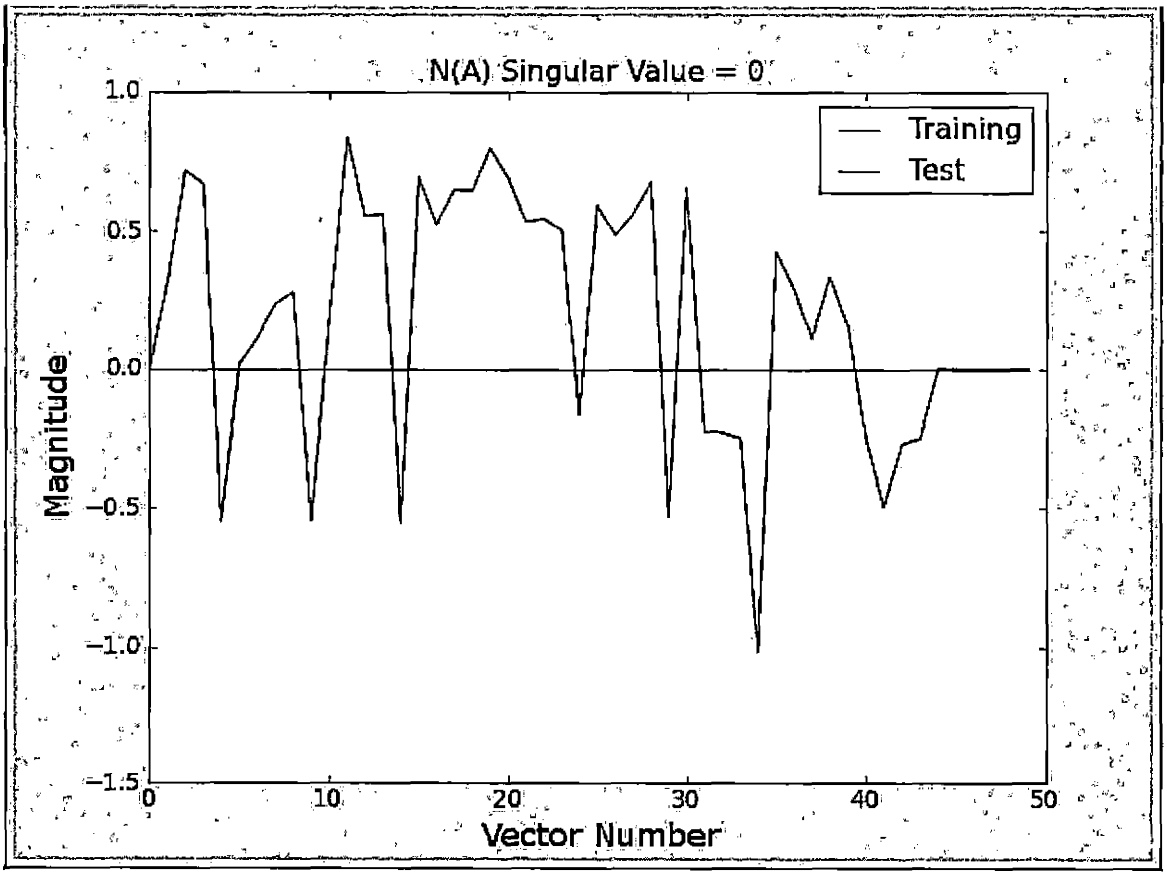

Fig. 3.35: 3rd singular vector in the null space - Reddit vs. Chinese

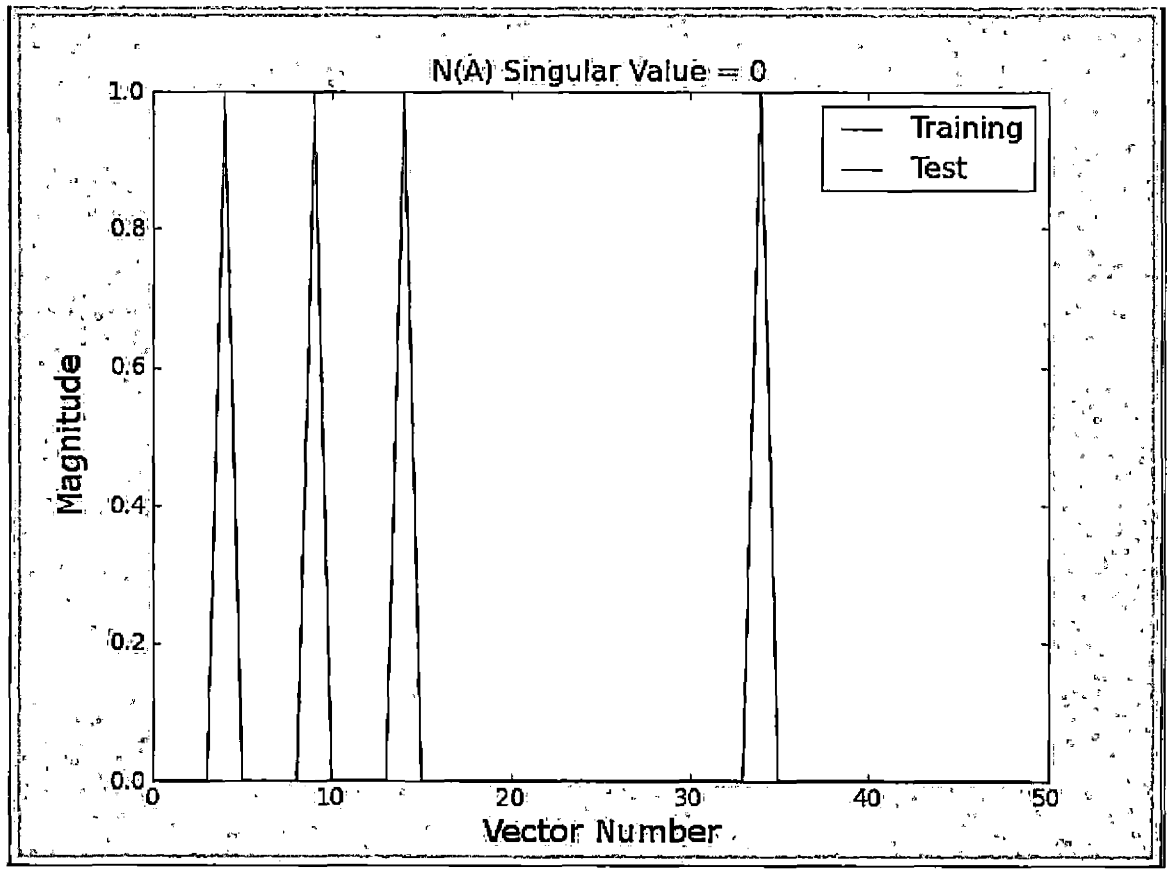

Fig. 3.36: Some "bad" vectors in the null space - Reddit vs. Chinese

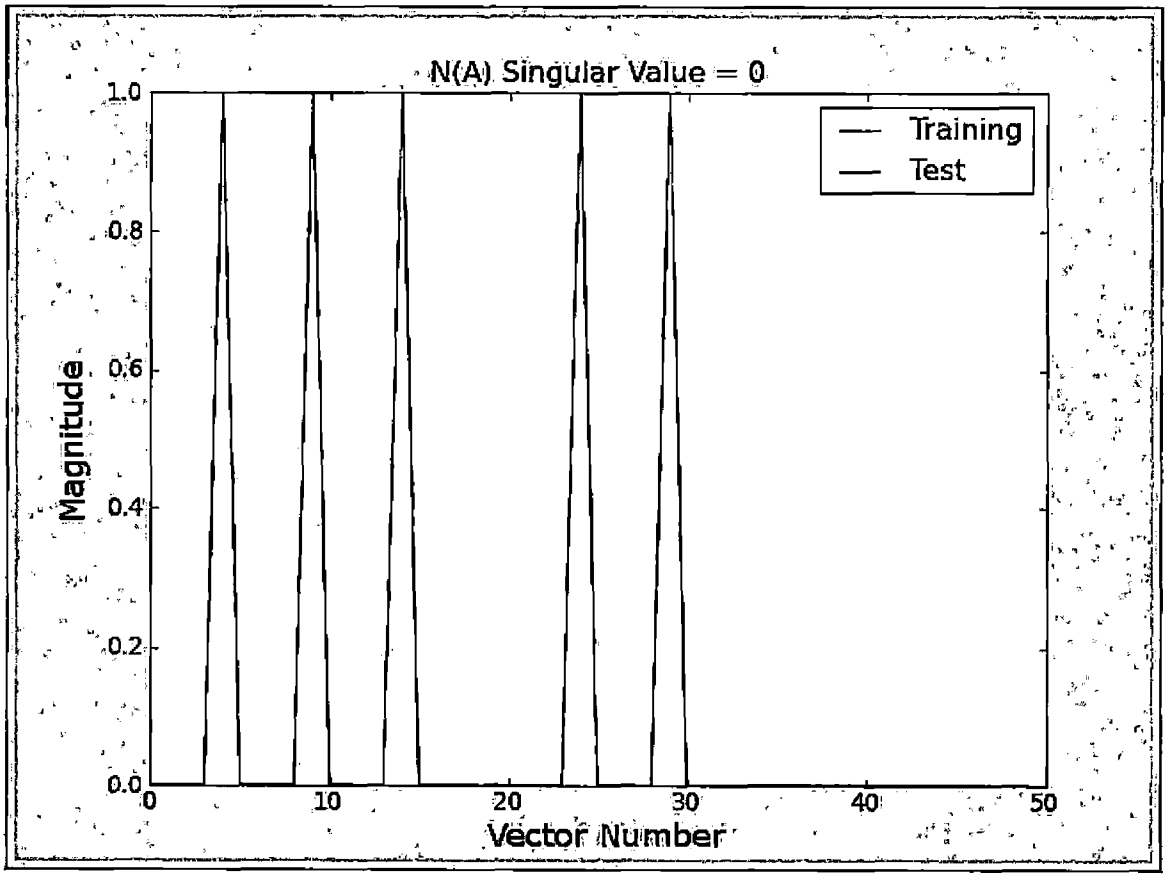

Fig. 3.37: Some "bad" vectors in the null space - Reddit vs. Chinese

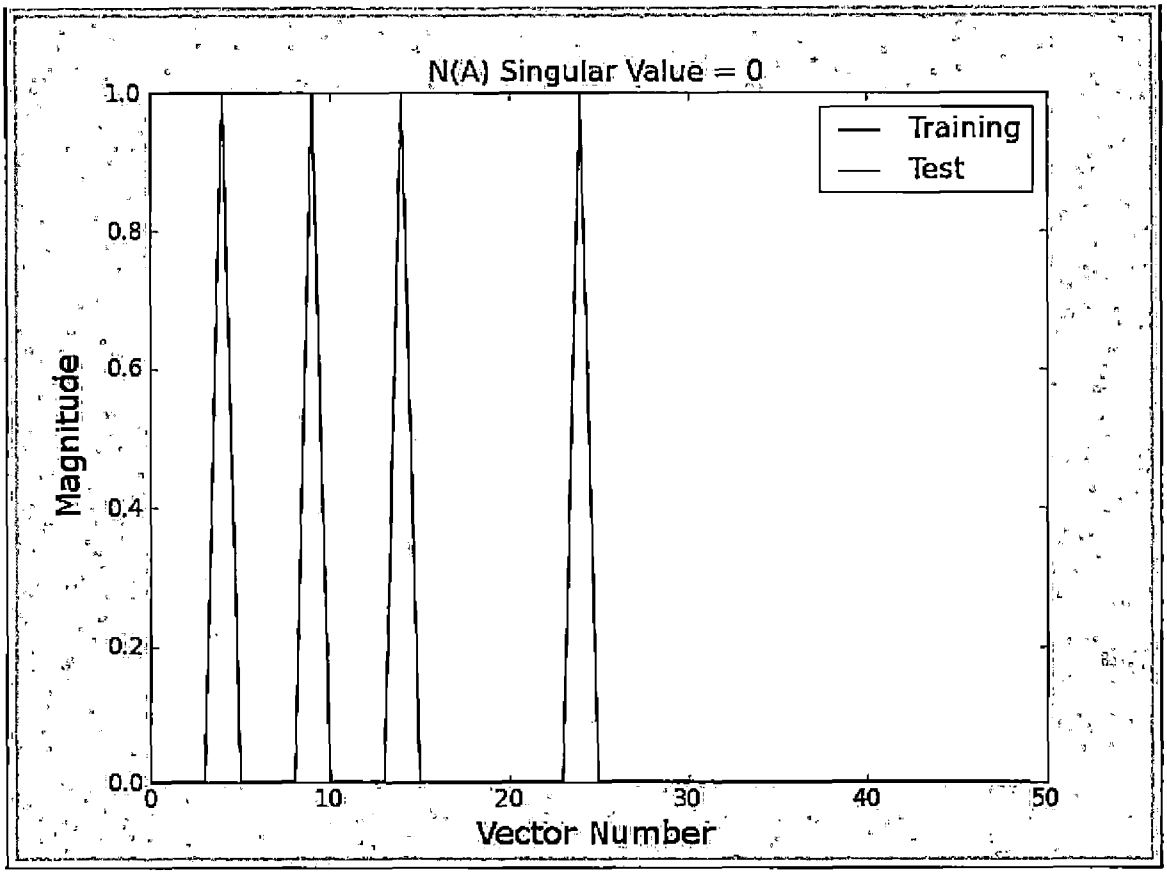

Fig. 3.38: Some "bad" vectors in the null space - Reddit vs. Chinese

3.2.6 *Set 6: Reddit.com vs. Pandora.com*

This was the one set of testing data that provided no conclusive results. The graphs from this test were unreadable noise both in the Column Space and in the Null Space. It is hypothesized that the reason for this was a lack of a training set of suitable size for the Pandora data. This hypothesis warrants future research to determine if training set size was indeed the limitation, or if the complexity of audio encoding makes it unsuitable for this type of analysis.

3.3 *Summary*

A test network was established as a virtual environment hosting multiple Linux, Windows, and Mac operating systems. Packet data was transmitted across this network to be collected in a central location for analysis. The data tested included the protocols ICMP and HTTP, as well as ASCII and Unicode. Known good traffic was sent across the network. The collected data was converted to binary and used to populate a set of training data that was composed of a large matrix whose column vectors were composed of the binary packet data. Using Singular Value Decomposition on this large matrix produced a set of singular values that were used to identify new "unknown" incoming traffic composed of both good and bad packets on the network by using the vector dot product on vectors from the training data multiplied against vectors created from the unknown data. By plotting the resulting values into charts, in most cases it was easy to identify the the bad traffic amongst the good traffic. Further analysis of the results could provide even greater refinement and allow for more granularity in pattern recognition.

4. CONCLUSIONS

4.1 Review of Contributions

The research has demonstrated conclusively that by using Singular Value Decomposition within the proper context it is possible to detect anomalous data in several contexts. Using ICMP as a proof of concept, the algorithm was developed and refined to overcome early mistakes in analysis. The more comprehensive approach that arose from these mistakes provided a strong framework that proved effective at identifying anomalies across nearly every set of testing data, including ICMP traffic, ASCII vs Unicode language, HTTP text-based traffic, HTTP ASCII encoding vs. Unicode, as well as to some degree HTTP streaming audio traffic. The characteristic magnitude spikes contained within the results clearly identified the location of "bad" packets within the testing data.

The implications of this research are many fold. The type of calculations being utilized by the algorithm lend themselves readily to parallelization, and thus would be ideal candidates for use with stream processors. This may enable a significant increase in the speeds typically required to analyze packet data. Furthermore, this research was largely a proof of concept endeavor, and further refinement of the algorithm may allow for even finer distinctions across a broad range of protocols, thereby opening up an entirely new way to conduct deep packet inspections.

4.2 Future Directions

As mentioned previously, future experimentation in a variety of other protocols is warranted. Furthermore, larger samples of data might be tested to increase the success rate of the streaming audio analysis, which was the only non-conclusive protocol with regard to the testing contained herein. Furthermore, application of this algorithm within the framework of Cuda, and utilizing the inherent parallelizable properties of the data being analyzed could prove exceedingly fruitful in providing a solution that can do autonomous packet filtering at line speed on large 10Gb networks. There are a number of permutations of the algorithm that was uncovered herein that might also prove more effective. For example, manipulating the size and shape of the matrices might allow for a greater variety of operations as the size of the Null Space is increased or decreased This may open the door to Eigenvalue/Eigenvector analysis. Further comparison with existing signal processing algorithms might also prove to be a valuable tangent. Though there was some precedents, this research was fairly novel in it's approach; as such, it's nascent abilities are very likely to be only scratching the surface of its full potential.

5. APPENDIX

5.1 Software and Hardware Descriptions

In the course of the work the following pieces of software: Virtual Box, Hping, Python, SciPy, Tcpdump, Wireshark, Nmap, Windows XP, Windows 2003 R2, Windows Server 2008, Windows 7, Ubuntu, CentOS, Debian, BSD, Fedora, and Mac OSX.

5.1.1 Operating System Software

Windows: - Windows is the dominant operating system in the market place today, as such it was imperative to include Windows operating systems in testing. Three different variants of Windows were tested: The latest release for workstations, Windows 7 acted as both a testing unit, as well as a host system for three virtual environments. The latest server implementation of Windows, Server 2008, was used as a virtual test unit. This is a prominent operating system used by many businesses running server-based applications in today's marketplace. An older implementation of Windows workstation solution, Windows XP, was also included as a virtual testing unit, as was the previous implementation of Windows server product line, Windows Server 2003 R2.

Linux: - Linux is an extremely popular operating system in enterprise environments, and a large percentage of the world's web servers are housed on Linux boxes. There

are hundreds of different variants of the Linux kernel available; however, there are a few that dominate popular usage, and these were the ones selected for testing. Ubuntu Linux is probably the most popular workstation variant in the Linux community today, three separate virtual instances were used in testing. CentOS is the free version of Red Hat Enterprise Linux (RHEL), another extremely popular variant of Linux within the business community. Three separate virtual instances of CentOS were included in the testing group. There were also single virtual instances of Debian Linux, the distribution that Ubuntu was based off of, as well as Fedora, the workstation solution developed based on the Red Hat platform.

Unix: - A single virtual instance of BSD unix was included as a test unit.

Mac OSX: - A single physical MacBook Pro running Mac OSX 10.6 (Snow Leopard) was included as a test unit.

Virtual Box is a piece of software developed by Sun that enables a user to load a "virtual environment" onto an existing "host" operating system which allows the user to then install and run different (or the same) "guest" instances of operating systems within that "host" operating system. These "guest" operating systems use a virtualization layer to partition off segments of the "host" operating systems hardware resources (CPU, Hard Disk, and RAM) and treat them as if they are distinct and separate pieces of hardware that the "guest" can then use. The only limitation for how many "guest" operating systems can be run at once is the amount of physical resources that are available to the "host" operating system. For this research project, three "host" systems and 12 "guest" systems were used. The architecture was as follows:

- Host 1 - Dell Workstation running Windows 7

    1. 8 CPU cores
    2. 8 GB of RAM
    3. 500 GB of Hard Disk Space

    - Guest 1 - Windows Server 2008

        1. 2 Virtual CPUs
        2. 2 GB Virtual RAM
        3. 60 GB Virtual Hard Drive

    - Guest 2 - Ubuntu Linux

        1. 1 Virtual CPU
        2. 1 GB Virtual RAM
        3. 20 GB Virtual Hard Drive

    - Guest 3 - CentOS Linux

        1. 1 Virtual CPU
        2. 1 GB Virtual RAM
        3. 20 GB Virtual Hard Drive

    - Guest 4 - BSD Unix

        1. 1 Virtual CPU
        2. 1 GB Virtual RAM
        3. 20 GB Virtual Hard Drive

HOST 2 - HP Workstation running Windows 7

1. 8 CPU cores
2. 16 GB of RAM
3. 1 TB of Hard Disk Space

- Guest 1 - Windows Server 2003R2
    1. 2 Virtual CPUs
    2. 4 GB Virtual RAM
    3. 60 GB Virtual Hard Drive
- Guest 2 - Ubuntu Linux
    1. 1 Virtual CPU
    2. 1 GB Virtual RAM
    3. 20 GB Virtual Hard Drive
- Guest 3 - CentOS Linux
    1. 1 Virtual CPU
    2. 1 GB Virtual RAM
    3. 20 GB Virtual Hard Drive
- Guest 4 - Windows XP
    1. 1 Virtual CPU
    2. 1 GB Virtual RAM
    3. 40 GB Virtual Hard Drive

HOST 3 - ASUS Laptop running Windows 7

1. 8 CPU cores

2. 8 GB of RAM

3. 500 GB of Hard Disk Space

- Guest 1 - Fedora Linux

    1. 1 Virtual CPU

    2. 1 GB Virtual RAM

    3. 20 GB Virtual Hard Drive

- Guest 2 - Ubuntu Linux

    1. 1 Virtual CPU

    2. 1 GB Virtual RAM

    3. 20 GB Virtual Hard Drive

- Guest 3 - CentOS Linux

    1. 1 Virtual CPU

    2. 1 GB Virtual RAM

    3. 20 GB Virtual Hard Drive

- Guest 4 - Debian Linux

    1. 1 Virtual CPU

    2. 1 GB Virtual RAM

    3. 20 GB Virtual Hard Drive

Figure 1 details the physical/virtual topology of the lab environment where all testing was conducted.

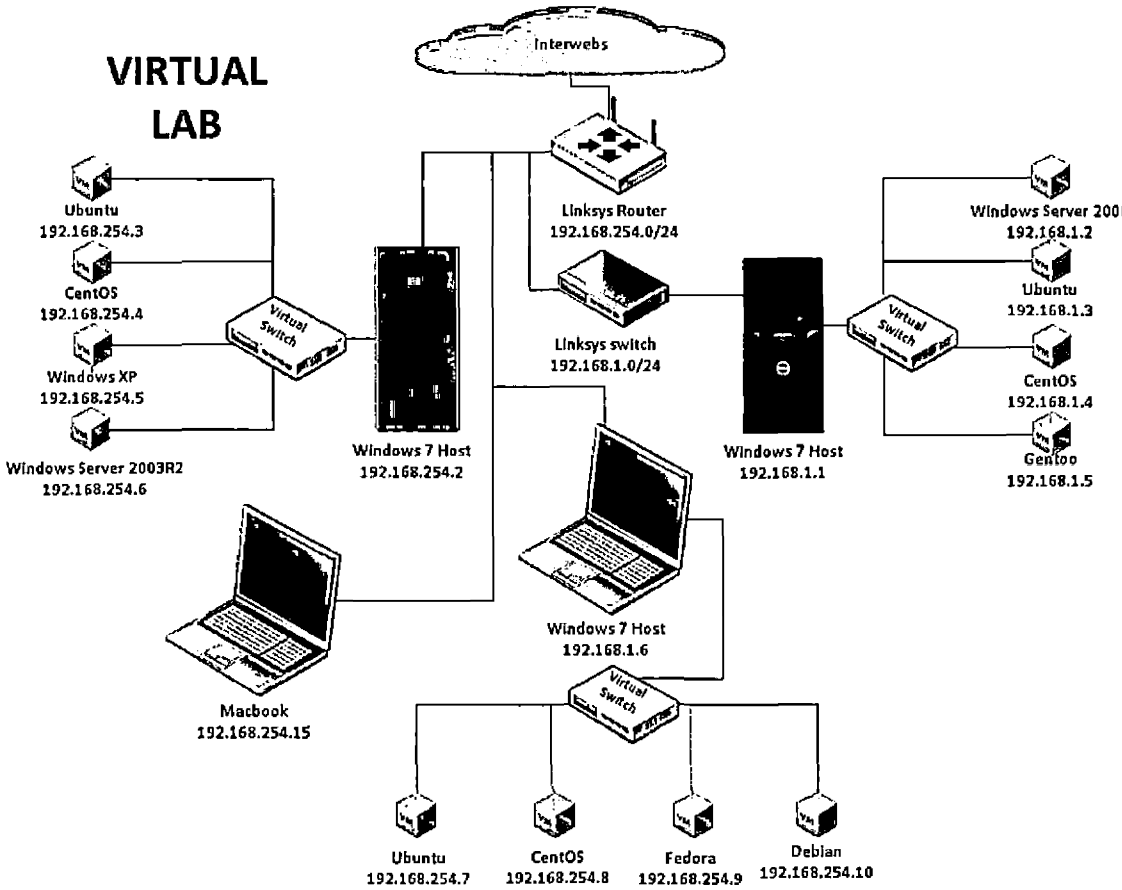

Fig. 5.1: Virtual Lab Topology

5.1.2 Network Tools

Tcpdump: - Tcpdump is a common network tool that comes pre-installed on most unix/linux systems that enables a user to capture all network traffic in packet form on a designated network interface. Once captured the data can be translated in a few different ways and stored to a file. This is the primary tool that was used to both capture all test packet data, as well as translate it to hex so that it could be further translated to binary for use in subsequent matrix analysis. Tcpdump was utilized on all Linux boxes in the lab environment. An analogue to Tcpdump, Windump, was used on the Windows boxes to similarly capture packet data from that location.

Hping: - Hping is piece of packet crafting software written by Salvatore Sanfilippo that can be used for a variety of purposes related to network analysis and security including Firewall testing, Advanced Port Scanning, Network testing with a variety of protocols, manual path MTU discovery, advanced traceroute, remote OS fingerprinting, remote uptime guessing, and TCP/IP stack auditing. (source: www.hping.org). The research utilized Hping to create simulated malicious packets for testing the ability of the algorithm to detect malformed ICMP, UDP, and TCP packets.

Wireshark: - Wireshark is another packet capture/analysis tool similar to Tcpdump, but with a more intuitive and user-friendly Graphical User Interface. Wireshark was used as a secondary tool to do individual packet-level analysis of testing data sets.

Nmap: - "Nmap ("Network Mapper") is a free and open source utility for network discovery and security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap uses raw IP packets in novel ways to

determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics." (source www.nmap.org)

5.1.3 Programming Tools

Python: - "Python is a remarkably powerful dynamic programming language that is used in a wide variety of application domains." (source: www.python.org) Python was used as the primary language for all programming done for this thesis.

SciPy: - "SciPy is open-source software for mathematics, science, and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization." (source: www.scipy.org) SciPy and NumPy were the basis for all linear algebraic operations implemented by the algorithms used in the scope of the data analysis stage of this research.

R: - "R is a language and environment for statistical computing and graphics." "R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible." (source: www.r-project.org). R was used as a data analysis tool in the early stages of the research process to determine the feasibility of certain numerical approaches. When it was discovered that a more iterative, programmatic approach was desired, R was discarded in favor of SciPy, which enabled the researchers

to accomplish the same tasks with direct and seamless integration with the Python programming language.

MatPlotLib: - "MatPlotLib is a python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms." (source: matplotlib.sourceforge.net) MatPlotLib was the library that was used in order to create graphical representations of the results of numerical analysis using SciPy. It was through the use of these graphs that researchers were able to translate the results into a format that demonstrated that the algorithm did indeed produce valuable, concrete, results.

REFERENCES

[1] Florin Baboescu and George Varghese. Scalable packet classification. In *ACM SIGCOMM'01*, pages 199–210, 2001.

[2] Liana B. Baker. Hackers hit sony sites raising more security issues. "http://www.reuters.com/article/2011/05/20/us-sonyhacker-idUSTRE74J3Z820110520", 2011.

[3] Sergey Brin and Larry Page. The anatomy of a large-scale hypertextual search engine. *A Paper from the Computer Science Department at Stanford University*, 1998.

[4] US Strategic Command. Us cyber command fact sheet. "http://www.stratcom.mil/factsheets/cyber_command/", 2009.

[5] P. Gupta and N. McKeown. Packet classification on multiple fields. In *ACM SIGCOMM'99*, pages 147–160, 1999.

[6] Pankaj Gupta and Nick McKeown. Algorithms for packet classification. *Network, IEEE*, 15(2), 2001.

[7] SANS Institute. Top cyber security risks. "http://www.sans.org/top-cybersecurity-risks/summary.php", 2009.

[8] Yaakov Katz. Stuxnet may have destroyed 1,000 centrifuges at natanz. "http://www.jpost.com/Defense/Article.aspx\?id=200843", 2010.

[9] Wenke Lee, Salvatore J. Stolfo, and Kui W Mok. Adaptive intrusion detection: A data mining approach. *Artificial Intelligence Review*, 14(6):533–567, 2000.

[10] Duo Liu, Bei Hua, Xianghui Hu, and Xinan Tang. High-performance packet classification algorithm for many-core and multithreaded network processor. In *ACM CASES'06*, 2006.

[11] Todd C. Lopez. 8th air force to become new cyber command. "http://www.af.mil/news/story.asp?storyID=123030505", 2006.

[12] John Markoff. Before the gunfire cyber attacks. "http://www.nytimes.com/2008/08/13/technology/13cyber.html", 2008.

[13] Samuel Patton, William Yurcik, and David Doss. An achilles heel in signature-based ids: Squealing false positives in snort. *A Paper from the Department of Applied Computer Science at Illinois State University*, 2001.

[14] David Sanger and Elizabeth Bumiller. Pentagon to consider cyberattcks acts of war. "http://www.nytimes.com/2011/06/01/us/politics/01cyber.htm", 2011.

[15] Mei-Ling Shyu, Thiago Quirino, and Xie XongXing. Network intrusion detection through adaptive sub-eigenspace modeling in multiagent system. *ACM Transactions on Autonomous and Adaptive Systems*, 1(3), 2007.

[16] Summet Sing, Florin Baboescu, George Varghese, and Jia Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM'03*, 2003.

[17] Haoyu Song and John W Lockwood. Efficient packet classification for network intrusion detection using fpga. In *ACM/SIGDA 13th International Symposium on Field Programmable Arrays*, 2005.

[18] Srinivasan T., Balakrishnan R., Gangadharan S.A., and Hayawardh V. Supervised grid-of-tries: A novel framework for classifier management. In *ICDCN'06 Proceedings of the 8th international conference on Distributed Computing and Networking*, pages 373–378, 2006.

[19] Sandeep A. Thorat, Amit K. Khandelwal, Bezawada Bruhadeshar, and Kishore K. Anomalous packet detection using partitioned payload. *Journal of Information Assurance and Security*, 3:195–202, 2008.

[20] Marina Thottan and Ji Chuanyi. Anomaly detection in ip networks. *IEEE Transactions on Signal Processing*, 51(8), 2003.

[21] Brett Tjaden, Lonnie Welch, Shawn Ostermann, David Chelberg, Ravindra Balupari, Marina Bykova, Aaron Mitchell, Denis Lissitsyn, Lu Tong, Michael Masters, Paul Werme, David Marlow, Brett Chapell, and Philip Irey Iv. Inbounds: The integrated network-based ohio university network detective service. *A Joint Paper from the The Naval Surface Warfare Center at Dahlgren in Virginia and the School Of Electrical Engineering and Computer Science at Ohio University*, 2000.

[22] Vimal Vaidya. Dynamic signature inspection-based network intrusion detection. *US PATENT*, (6,279,113), 2001.

[23] Giovanni Vigna, Bryan Cassell, and Dave Fayram. An intrusion dectection system for aglet. pages 64–77, 2000.

[24] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. *Recent Advances in Intrusion Detection*, 3224:203–222, 2004.

[25] Junbi Xiao and Hao Song. A novel intrusion detection method based on adaptive resonance theory and principal component analysis. In *2009 International Conference on Communications and Mobile Computing*, pages 133–139, 2009.

[26] Kim Zetter. Hacker spies hit security firm rsa. "`http://www.wired.com/threatlevel/2011/03/rsa-hacked/`", 2011.

[27] Denis Zuev and Andrew W. Moore. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05 Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.