



**Using an Inductive Learning Algorithm to Improve
Antibody Generation in a Single Packet
Computer Defense Immune System**

THESIS

Russell J. Aycok, First Lieutenant, USAF

AFIT/GIR/ENG/02M-01

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

**Using an Inductive Learning Algorithm to Improve
Antibody Generation in a Single Packet
Computer Defense Immune System**

THESIS

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science

Russell J. Aycock, B.S.
First Lieutenant, USAF

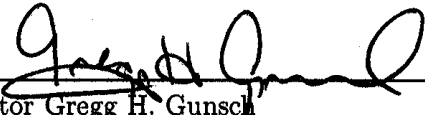
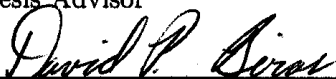
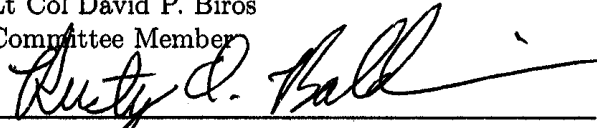
March, 2002

Using an Inductive Learning Algorithm to Improve
Antibody Generation in a Single Packet
Computer Defense Immune System

Russell J. Aycock, B.S.

First Lieutenant, USAF

Approved:

 _____ Doctor Gregg H. Gunsch Thesis Advisor	<u>6 MAR 2002</u> _____ Date
 _____ Lt Col David P. Biros Committee Member	<u>6 Mar 02</u> _____ Date
 _____ Major Rusty O. Baldwin Committee Member	<u>6 Mar 02</u> _____ Date

Acknowledgements

I would like to first thank Dr. Gunsch for helping me to find a research topic and for keeping me pointed in the right direction. I also sincerely appreciate the help of the other members of our research group. If we were not constantly bouncing ideas off of each other I am certain that this research effort would not have been nearly as successful as I feel it has been.

Though this experience has been satisfying, I regret the stress that it sometimes put on my family. I would like to give special thanks to my wife for acting interested for hours on end while I rambled on about the intricacies of programming, intrusion detection, and many other subjects that I'm sure she never intended to have to learn about. I shudder to think what kind of experience this might have been without her help and support.

Russell J. Aycock

Table of Contents

	Page
Acknowledgements	iii
List of Figures	vii
List of Tables	viii
Abstract	ix
 I. Introduction	 1-1
1.1 Intrusion Detection Systems	1-1
1.1.1 Misuse and Anomaly-Based Detection Systems	1-2
1.1.2 False Positives and False Negatives	1-3
1.1.3 Host-Based Intrusion Detection Systems	1-4
1.1.4 Network-Based Intrusion Detection Systems	1-4
1.1.5 Application-Based Intrusion Detection Systems	1-6
1.2 A Computer Immune System	1-6
1.2.1 CDIS Antibody Generation [Will01]	1-6
1.2.2 Attack Detection Using CDIS Antibodies	1-8
1.3 Research Focus	1-8
1.3.1 Problem Statement	1-8
1.3.2 Scope	1-8
1.4 Summary	1-9
 II. Literature Review	 2-1
2.1 Inductive Machine Learning Methods	2-1
2.2 Bayesian Classifiers	2-2
2.3 Decision Trees	2-3
2.4 Neural Networks	2-6

	Page
2.5 Genetic Algorithms	2-9
2.6 Computer Defense Immune System (CDIS) Antibodies	2-12
2.7 Choosing a Classification Method	2-15
2.8 Summary	2-16
III. Methodology	3-1
3.1 Antibody Generation	3-1
3.2 Initial Antibody Testing	3-2
3.2.1 Building the Decision Tree	3-3
3.2.2 Non-Self Detection	3-7
3.3 Initial Antibody Modifications	3-8
3.4 Further Antibody Analysis	3-9
3.5 Summary	3-9
IV. Analysis and Findings	4-1
4.1 Initial Decision Tree Analysis	4-1
4.2 Modified Data Set Analysis	4-4
4.2.1 Bad Antibody Comparison	4-4
4.2.2 Generation Time Comparison	4-4
4.2.3 Non-Self Detection Comparison	4-6
4.3 Further Antibody Modification	4-7
4.4 Final Data Set Analysis	4-8
4.4.1 Bad Antibody Comparison	4-9
4.4.2 Antibody Generation Time Comparison	4-9
4.4.3 Non-Self Detection Comparison	4-10
4.5 Final Analysis	4-10
4.6 Summary	4-12

	Page
V. Conclusions and Recommendations	5-1
5.1 Conclusions	5-1
5.2 Limitations	5-1
5.3 Recommendations for Future Research	5-2
Appendix A. Decision Tree from Good/Bad Antibodies	A-1
Appendix B. Decision Tree from Successful/Unsuccessful Antibodies	B-1
Bibliography	BIB-1
Vita	VITA-1

List of Figures

Figure		Page
2.1.	Decision Tree Example	2-4
2.2.	The Difficulty of Using Rectangles to Separate a Linear Decision Space. . . .	2-6
2.3.	A Simple Perceptron	2-7
3.1.	Weka Decision Tree Output	3-4
3.2.	Weka Decision Tree Error Output	3-6
4.1.	Original Antibody Generation Time z -Distribution	4-6

List of Tables

Table		Page
1.1.	Antibody Features	1-7
2.1.	Sample Classifier Data	2-2
2.2.	Sample Classifier Data	2-9
2.3.	Sample GA Classifier Data	2-10
2.4.	Packet Features	2-13
3.1.	Arff File Format	3-3
4.1.	Original Antibody Data	4-3
4.2.	Modified Antibody Data	4-4
4.3.	Non-Self Detection Information	4-6
4.4.	Successful Modified Antibody Training File	4-7
4.5.	Modified Packet Features	4-8
4.6.	Final Antibody Data	4-8
4.7.	Figures For All Data Sets	4-11

Abstract

The United States Air Force relies heavily on computer networks for many day-to-day activities. Many of these networks are affected by various types of attacks that can be launched from anywhere on the globe. The rising prominence of organizations such as the AFCERT and the MAJCOM NOSC is evidence of an increasing realization among the Air Force leadership that protecting our computer networks is vitally important.

A critical requirement for protecting our networks is the ability to detect attacks and intrusion attempts. This research is an effort to refine a portion of an AFIT-developed intrusion detection system known as the Computer Defense Immune System (CDIS). CDIS is based on the human immune system and uses antibodies to attempt to detect network intrusion attempts. The antibodies have various attributes of which a subset is randomly activated at generation time. This research attempts to determine which of the antibody attributes are most useful in helping to build successful antibodies.

Using an Inductive Learning Algorithm to Improve Antibody Generation in a Single Packet Computer Defense Immune System

I. Introduction

The protection of computer networks against attacks, misuse, or a multitude of other threats is becoming an increasing concern for many organizations. This is especially true in the Air Force where life or death decisions are sometimes made based on data that travels over these networks. A large amount of effort is being expended to develop many different types of systems to protect computer networks. One type of system that is becoming increasingly prevalent in many network defense taxonomies is an intrusion detection system (IDS).

This chapter begins with a brief discussion of intrusion detection systems and some of the current concerns related to them. Section 1.2 gives a description of the specific IDS that is the focus of this research. Finally, Section 1.3 provides a quick overview of this research, including its aims and scope.

1.1 Intrusion Detection Systems

There are many reasons that intrusion detection systems are needed. Oftentimes the very feature that makes a system most useful is the same feature that makes it vulnerable to attack. Denning gives the following reasons for the development and use of a real-time IDS: “technical and economical feasibility inhibit the correction of system flaws; replacing a flawed system with a more secure system may result in a loss of attractive features; it is difficult, if not impossible, to develop an absolutely secure system; and the threat of insider abuse makes even a highly secure system vulnerable” [Denn87].

An IDS complements firewalls and other types of network security devices. A firewall may block an attack from reaching the network that it is protecting but, in most cases, it will not be able to alert an analyst that an attack has taken place. A firewall simply filters traffic based on a predetermined set of rules. An IDS monitors all activity on a system, examining that activity for patterns of misuse or attack. In addition, an IDS may monitor traffic inside the firewall and possibly detect internal system misuse or insider attacks, which in many cases is traffic that would never be seen by the firewall.

1.1.1 Misuse and Anomaly-Based Detection Systems. In much the same way that anti-virus programs use signatures to detect viruses on machines, many intrusion detection systems use signatures to detect intrusions. Such a system would be classified as one that performs misuse detection. Misuse detection “is performed by looking for the exploitation of known weak points in the system, which can be described by a specific pattern or sequence of events or data (the ‘signature’ of the intrusion)” [Bala98].

An analyst can gather information to build signatures from many different sources. An analyst who knows the weak points of the system and the common attack tools that might be used against the system could have some success in proactively developing signatures. However, in many cases an attack must be detected before a signature can be developed. Once an analyst detects a new type of attack, she can determine a set of unique features that define the attack to serve as the signature. The signature can then be added to the IDS. Misuse detection systems have shown themselves to be very reliable in detecting known attacks [Ghos99]. However, signature-based systems are very reactive in nature. “[T]he key drawback of misuse detection approaches is that they cannot detect novel attacks against systems that leave different signatures” [Ghos99]. An intrusion may occur and the intruder might have already accomplished his task before the signature is even developed.

Anomaly-based detection systems avoid some of the pitfalls of signature-based systems. A successful anomaly-based system must first develop some sort of notion of what “normal” traffic is

for the system in question, whether that system is a single host or an entire network. Once the IDS has a sense of what is normal, it can then search for anomalous activity, or activity that is outside of the normal range of what might be expected for that system. Since anomaly-based detection systems “compare current activities against models of past behavior,” they have somewhat more success detecting novel attacks than misuse-based detection systems [Ghos99].

A downfall of anomaly-based detection systems is that users may have a legitimate need to do something that the system will view as abnormal. If a user installs new software or does anything out of the ordinary, the detection system may view that as an attack. Another drawback of anomaly-based detection systems is their “inability to identify the specific type of attack that is occurring” [Ghos99]. In signature-based systems the type of attack is easily determined by examining the signature that detects it. Anomaly-based systems simply give an analyst an indication that something abnormal is happening.

1.1.2 False Positives and False Negatives. When an IDS mistakes legitimate activity for illegitimate activity, it generates what is known as a false positive. A false positive generated by an IDS causes an analyst to have to waste time investigating legitimate activity. “If too many false positives are generated, the operators will come to ignore the output of the system over time, which may lead to an actual intrusion being detected but ignored by the users” [Cros95]. A signature-based detection system does not generate many false positives because activity that matches a signature must be detected to generate an alarm. In contrast to that, false positives are a big concern when dealing with anomaly-based systems. A subtle change in usage patterns by user might generate alarms, even though the activity is legitimate. Many intrusion detection systems have mechanisms in place to help to limit false positives. One way to accomplish this is to set a threshold level. For example, a rule may require that monitored traffic must differ by a certain amount from normal traffic before it is considered anomalous. One problem with setting thresholds is the possible introduction of false negatives.

A false negative, in intrusion detection terms, is an attack that goes unnoticed by the IDS. “False negative errors are more serious than false positive errors because they give a misleading sense of security” [Cros95]. False negatives are a problem in all intrusion detection systems. Signature-based systems generate false negatives when an attacker uses a novel attack for which there is no signature. One could liken this to the “I Love You” virus. The anti-virus programs did not detect the virus because it was different than all of the previous viruses. This allowed the virus to spread and wreak havoc on computer systems around the world. The signature-based intrusion detection systems face the same sort of problem. Anomaly-based systems might generate false negatives when an attack closely resembles normal network traffic. For example, if a network passes quite a bit of http traffic to various ports, an attack using http to a rarely used port number might not be detected because it closely resembles normal traffic.

1.1.3 Host-Based Intrusion Detection Systems. “Host-based [intrusion detection] systems base their decisions on information obtained from a single host” [Spaf00]. The information gained from a host may come from many sources, such as audit trails and system logs [Bace00]. Some signature-based detection systems search through log files trying to find patterns that match the signatures in the system. Some anomaly-based detection systems use statistical methods to determine normal usage patterns based on the log files. The system then uses subsequent log files to determine when abnormal usage, and possibly an attack, has occurred.

1.1.4 Network-Based Intrusion Detection Systems. In a network-based IDS, “information is collected from the network traffic streams as it travels on the network segment” [Bace00]. To capture the traffic, an analyst need not disturb any of the systems on the network. A sniffer (detector in IDS terms) or other piece of equipment may be attached to the network to simply listen and record the traffic traversing it. In his 1989 book, *The Cuckoo’s Egg*, Clifford Stoll details a very rudimentary example of such a system [Stol89]. Stoll’s system allowed him to record an

attackers movement through the UC Berkeley computer network. Today's IDS employs the same methods that Stoll used, albeit much more automated and on a much larger scale.

The traffic captured by the detector can be analyzed through the use of signature or anomaly detection. For signature-based detection, an IDS will take compare the captured network traffic to the signatures in the system. In general, some sort of pattern matcher will search for a match between an attack signature and the data, returning an alert if a match is found. Some systems return an alert when only a partial match is found, while others may look for sequences of events that signify an attack [Bace00]. Another consideration for signature-based systems is the event horizon, or the "maximum time interval over which matching of an attack signature occurs" [Bace00]. Many analysts are concerned about network probing or reconnaissance, which can take place over hours, days, or even weeks. In many cases, such activity may be a precursor to an actual attack against a network, so the value of being able to detect activity over a large event horizon is immense.

As previously discussed, anomaly detection in a network-based IDS relies on a dependable definition of what "normal" network traffic is. Defining what is normal is a problem for which there appears to be no foolproof answer. To define what is normal, an IDS must monitor network traffic for a period of time. If any attacks occur during that period, then they are part of what the IDS considers to be normal traffic, and it would therefore be unlikely that the IDS would catch the attack in the future. One possible solution for this problem is to have an analyst screen network traffic, removing any attack data, so the system can train on a reliable set of data. This solution is hardly an optimal one.

The system is faced with the challenge of detecting attacks once it has established what is considered to be "normal" network activity. A network-based system will generally do a statistical comparison between normal traffic and a collection of recent traffic, with a significant statistical difference in the two representing possible anomalous activity. Setting the threshold, or significance level (α value), determines the rate for false positives and false negatives. A higher level of sig-

nificance should, in theory, raise the false positive rate as the system will be much more sensitive to subtle changes in network traffic. A lower significance level should reduce the sensitivity of the system and therefore would increase the likelihood of an attack slipping by unnoticed.

1.1.5 Application-Based Intrusion Detection Systems. The newest trend in intrusion detection systems is application-based systems [Almg01]. In an application-based system, an individual application is monitored for possible misuse. Although application-based systems offer the advantages of having access to otherwise encrypted information and true session reconstruction, the main disadvantage is that the numerous application types have distinct interfaces. The need for application-specific IDS modules ensures that an immense development effort would be required to build a truly robust application-based IDS [Almg01].

1.2 A Computer Immune System

Some researchers have come to look at the domain of intrusion detection as being very similar to the human immune system [Hofm99, Will01]. Hofmeyr states, “A computer security system should protect a machine or set of machines from intruders or foreign code, which is similar in functionality to the immune system protecting the body from invasion by inimical microbes” [Hofm99]. A recent AFIT graduate, Capt Paul Williams, worked to implement a computer defense immune system (CDIS) that is based on the human immune system. CDIS builds antibodies based on a training set of data (previously captured, cleansed network traffic) and then uses those antibodies to detect anomalous network traffic [Will01].

1.2.1 CDIS Antibody Generation [Will01]. A CDIS antibody is a 320-bit binary string that contains up to 28 possible fields, or axes. Williams illustrated the makeup of the binary string with Table 1.1, presented below. During antibody generation, the protocol type of the antibody is randomly chosen from the four possibilities of IP, TCP, UDP, and ICMP. Once the protocol type is established, a random number of fields within the antibody are populated with values and assigned

Table 1.1 Antibody Features

Field Name	Possible Values	Bits Used
IP Fields (Common to all Packets)		
Protocol Type	TCP, UDP, ICMP	0-1
IP Identification Number	0-65535	2-10
IP Time to live (TTL)	0-255	18-25
IP Flags	0-65535	26-41
IP Overall Packet Length	0-65535	42-57
IP Source Address (A.B.C.D)	Valid IP address	58-89
IP Destination Address (A.B.C.D)	Valid IP address	90-121
TCP-Only Fields		
TCP Source Port	0-65535	122-137
TCP Destination Port	0-65535	138-153
TCP Sequence Number	0-4294967295	154-185
TCP Next Sequence Number	0-4294967295	186-217
TCP Acknowledgement Number	0-4294967295	218-249
All TCP Flags	0-255	250-257
Individual TCP Flags (PCWR, Echo, Urgent, Ack, Push, Reset, Syn, Fin)	Boolean	258-265
TCP Packet Size	0-65535	266-280
UDP-Only Fields		
UDP Source Port	0-65535	122-137
UDP Destination Port	0-65535	138-153
UDP Data Length	0-65535	266-281
ICMP-Only Fields		
ICMP Type	0-255	122-129
ICMP Code	0-255	130-137
ICMP Data Length	0-65535	266-281
Validity Bits		
Specify activated fields for an antibody		292-319

ranges. For example, a TCP antibody could have only four fields with assigned values and ranges; IP Src Address A, TCP Src Port, TCPSyn, and TCP Packet Size. In a process known as negative selection, the antibody is compared to the training data (known as self). In this example, if the IP Src Address A value was 131 and the associated range was 3, then CDIS would look for packets in the training data with an IP Src Address A value of between 128 and 134. If any packets in the training data match the first query, then they are queried for matches with the TCP Src Port value and range. This process continues until either no matches exist or the antibody fields that were populated have all been tested. If matches exist after all of the antibody fields have been tested, the antibody is considered to have matched self and is thrown out. If no matches are found, then the antibody has been successfully generated and is kept for further use.

1.2.2 Attack Detection Using CDIS Antibodies. To detect attacks, CDIS uses a process that is very similar to negative selection. The network traffic to be scanned for attacks is stored as a packet database. Antibodies that were successfully generated using training data are compared to the packet database. Any packets that match an antibody are considered to be possible attacks [Will01]. In his research, Williams found that false positives could be a problem for CDIS. Recent research by Anchor [Will01a] explored an implementation of CDIS that uses a process known as costimulation to help reduce false positives. Costimulation is a process whereby a packet is only considered to be an attack if it is detected by more than one antibody.

1.3 Research Focus

This research is an effort to further refine CDIS, particularly in the area of antibody generation. The work done by Williams demonstrated the potential of CDIS, and it is hoped that this research effort, in conjunction with other efforts, will help to further develop CDIS.

1.3.1 Problem Statement. Williams' research revealed many possibilities for CDIS refinements, one of these being the improvement of CDIS antibodies. There has been no investigation into whether or not the CDIS antibodies are being efficiently generated. If any of the twenty-eight axes are not relevant factors in detecting attacks, then CDIS might be a more efficient yet just as potent system without them. The problem, then, is to determine what attributes successful antibodies (antibodies that detect attacks) share and if there are possible antibody improvements based on those shared attributes. Also, if there are fields that seem to encourage the development of bad antibodies, modifications to the antibody structure might prove to yield a more efficient generation process.

1.3.2 Scope. There are a few areas in which the focus of this research must be limited. For one, the axes of the antibodies are quite complex. Most attributes have many possible values. Since this research attempts to determine whether or not an attribute was helpful in detecting

attacks, the specific values associated with the attributes will not be considered. Future research may explore the possibility that there are certain ranges of values associated with each attribute that might be more helpful in detecting attacks than other ranges.

This research will require the examination of large amounts of antibodies. Though there may be other methods of examining the antibodies and discerning the dominant attributes, it would appear that some sort of inductive algorithm would best fit this purpose. It has been noted that inductive algorithms “could assist a user in detecting interesting conceptual patterns or in revealing structure in collections of observations” [Mich83]. Since this would appear to fall very much in line with what needs to be accomplished here, this research effort will focus on using an inductive algorithm to detect patterns in the antibodies being tested.

1.4 Summary

This chapter discussed the background of this research effort and included a discussion of the problem that this research will address. A discussion of intrusion detection systems was provided to give the reader an idea of the purpose of CDIS and to familiarize him with many of the terms that will be used throughout this text. A low level discussion of CDIS was meant to give a basic understanding of the CDIS antibodies and issues associated with them. Finally, the research problem and scope were provided to give the reader specific information on the focus of this research effort, and hence the following chapters.

The next chapter is a review of relevant literature concerning various types of inductive algorithms that may be useful for this type of research. Chapter III provides a discussion of the methodology that was used for this effort. Chapter IV presents the results and analysis yielded by this research. Finally, Chapter V gives final conclusions, limitations of this research effort, and recommendations for future research.

II. Literature Review

The focus of this chapter is to establish a sound theoretical background on which to build a methodology. Section 2.1 contains a discussion of various types of inductive machine learning methods, and examples of how those methods have been used in previous research. Section 2.2 examines CDIS antibodies in great detail. Section 2.3 is devoted to a discussion of which inductive machine learning method might be best suited to the current research.

2.1 Inductive Machine Learning Methods

The main purpose of this research is to search through a set of data, namely antibodies, and try to find a set of attributes that allows the data to be classified as either bad or good. In doing so, one might find certain attributes are more useful for producing good antibodies. There are several methods of inductive machine learning that might be used for such a task. The ones that seem to offer the most promise for this research are Bayesian learning systems, decision trees, neural networks, and genetic algorithms. Marmelstein [Marm99] points out that though the methods may vary greatly, they each have the following characteristics:

- Search Method(s) - As the name implies, this describes the type of search used by the induction algorithm. Examples of search methods include depth-first, evolutionary, and gradient descent.
- Search Heuristic - The search heuristic is the metric the search method seeks to optimize. For example, in a genetic algorithm, a heuristic is used by the objective function to rate the fitness of a given solution
- Data Related Assumptions - This category refers to assumptions the induction method makes about the structure of the data. For example, does the technique assume the data is discrete or continuous?

- Rule Related Assumptions - These are assumptions made about the structure of the induced rule set. For example, are the conditions in the rule based on single (univariate) or multiple (multivariate) data attributes?

A detailed examination of how these characteristics apply to each of the previously mentioned learning methods should be sufficient to allow the determination of an appropriate method for this research.

2.2 Bayesian Classifiers

A practical approach to the problem of classification involves using statistical methods to compute probabilities of any particular hypothesis being correct [Mitc97]. Bayesian classifiers are commonly used for such purposes. Bayesian classifiers require a training set of data to develop the probabilities for use in classification. If some probabilities are known initially, then they can be incorporated into the method [Mitc97].

Table 2.1 Sample Classifier Data

Classification	TCP_Syn_Flag	TCP_Ack_Flag	TCP_Reset_Flag
Good	0	0	0
Good	0	0	1
Good	1	1	0
Good	0	1	1
Bad	1	0	1
Bad	1	0	0
Bad	1	1	1
Bad	0	1	0

Table 2.1 gives sample data using three antibody fields and associated classes. The zeros and ones shown are not field values. Instead, they are used to represent whether or not a field was active for a given antibody, with a 0 signifying an inactive field and a 1 signifying an active field. If the data from Table 2.1 were entered into a Bayesian classification system, it would develop probabilities for various hypotheses. For example, one hypothesis might be that given an inactive TCP_Ack.Flag field and an active TCP_Reset.Flag field, the result is a good antibody. The data in Table 2.1

would suggest that this hypothesis would have a 50 percent probability as there are two instances in which it would apply and they are both classified differently (examples 2 and 5 in Table 2.1). For each new example added to the training set that match the hypothesis, its probability would shift. Given a large enough training set, the probability should become optimal [Weis91]. Once the Bayesian classifier is trained, new instances can be classified using the probabilities generated during training.

Bayesian methods provide a flexible approach to classification problems. For instance, a hypothesis that is incompatible with a single example need not be discarded. It would simply require an adjustment of the probability of said hypothesis. In addition, Bayesian methods allow for probabilistic predictions. That is to say, given set a training data, one could make predictions about the probabilities associated with future observations [Mitc97].

A problem with Bayesian learning methods is that many implementations assume conditional independence among the observations [Weis91]. Conditional independence is the notion that all examples in the training set (and any future observations) occur independently of one another. In reality, this is not always the case, and dependencies among the observations that have been disregarded can lead to classification anomalies [Weis91]. Another problem stems from possible relationships between the variables in a single observation. Marmelstein [Marm99] points out that “[p]erhaps the most dangerous assumption is that there exists a causal relationship between the recorded variables in the database.” This assumption can lead to further classification errors.

2.3 Decision Trees

Decision trees are data structures that are used to divide a set of data. Figure 2.1 shows an example of a decision tree that is based on the information in Table 2.1. A decision tree is made up of nodes, branches, and leaves. The first node of a decision tree is labelled as the root node. At the root node, the first test is made to divide the data set. In Figure 2.1, the test at the root

node involves the TCP_Syn_Flag. Two branches leave the root node. Depending on which branch is followed from the root node, the test at the next node might involve different attributes. At the bottom of Figure 2.1 are the leaves of the tree, which is where one can determine the class of the particular record(s) that terminates at that point. For example, the leftmost leaf represents the first two records from Table 2.1, and the class that record belongs to is “Good.” The corresponding hypothesis would be: If a record has an inactive TCP_Syn_Flag field and an inactive TCP_Ack_Flag field, then it belongs to the “Good” class.

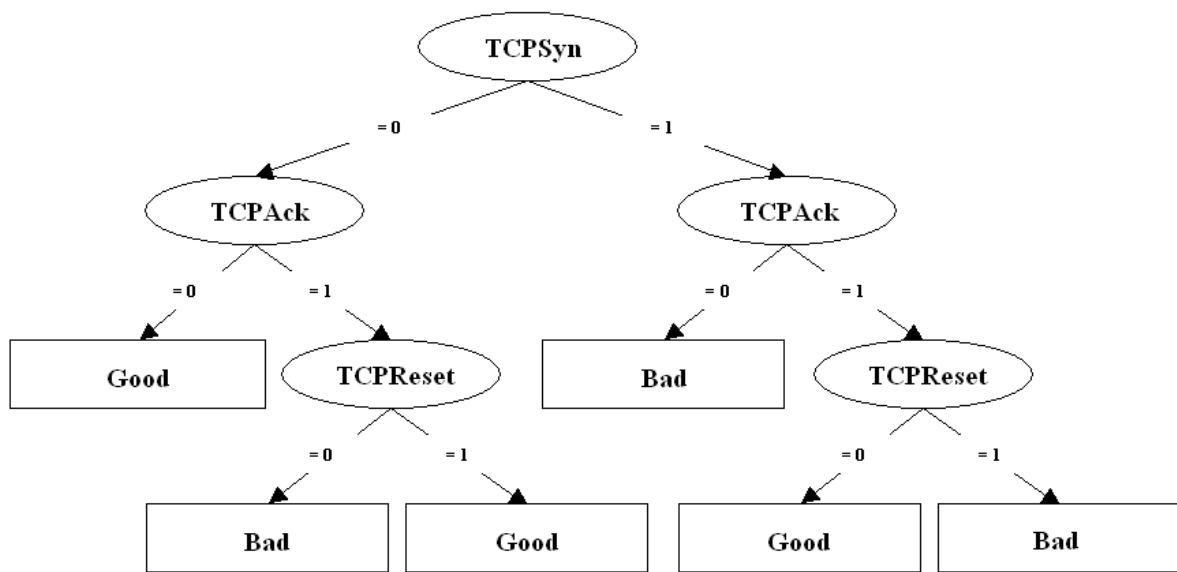


Figure 2.1 Decision Tree Example

To inductively generate a decision tree, most algorithms use a greedy search algorithm [Marm99]. In plain terms, at every decision node, the algorithm splits the data set on the attribute (or set of attributes) that appears to offer the best chance of enhancing the decision tree’s performance [Weis91].

A problem with the induction of decision trees is the difficulty of knowing when to stop the induction process. Some implementations have used “look ahead” algorithms in an attempt to determine when further splitting would be statistically insignificant [Weis91]. This method allows

for the possibility of “under-fitting” the decision tree, which is caused by stopping the induction algorithm prematurely [Marm99]. If the look-ahead algorithm does not look ahead far enough, it may not detect a split of some significance at a much lower level of the tree, which can introduce errors.

A widely used method of determining a more optimal decision tree involves the concept of “pruning.” Before a decision tree is pruned, the inductive algorithm is allowed to over-fit the data. An over-fit of the data implies that the tree has induced more structure than is actually present in the population that the training cases are drawn from [Marm99]. Because of this, the weakest branches of the tree are pruned, until no weak branches remain [Weis91]. When pruning, some of the sub-trees in a decision tree are replaced with a leaf node. To determine the class of the new leaf node, the leaves of the cases that are covered by the new leaf are examined and the most frequently occurring class is chosen as the class of the new leaf [Quin93]. The problem of knowing when to stop pruning is dealt with by examining the predicted error rates of the tree and the effect further pruning might have. When more pruning would cause predicted error rates to rise to an unacceptable level, pruning is stopped.

Marmelstein [Marm99] gives the following explanation for the popularity of decision trees:

- They produce rules that people can understand. This is primarily because the rules have logical conditions and the data they cover are tightly clustered together.
- Rules can be generated relatively quickly due (in part) to the use of greedy search algorithms.
- The generated rules tend to exhibit good classification accuracy and generalization (due to pruning of the tree).

There are some limitations when using decision tree induction. One limitation mentioned by Weiss and Kulikowski [Weis91] is that tree induction methods are not optimal. The lack of a backtracking mechanism in tree induction methods results in the inability to change a tree in hindsight, which might prove useful if an algorithm could look back to see that splitting a particular

node on a different attribute would have been more beneficial in the overall performance of the tree.

Another limitation of decision trees involves the shape of the decision regions that are generated within the data space. Both Weiss [Weis91] and Quinlan [Quin93] point out that decision trees produce hyper-rectangles to represent decision regions within the data space. These hyper-rectangles are not able to represent linear relationships very well. Figure 2.2 gives a representation of the difficulties of using rectangular regions to model linear relationships. In short, the rectangular regions will incorrectly classify some of the instances associated with a linear relationship.

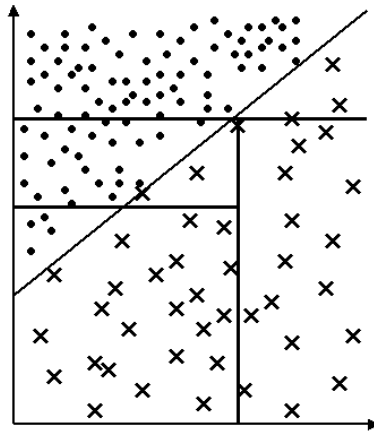


Figure 2.2 The Difficulty of Using Rectangles to Separate a Linear Decision Space.

2.4 Neural Networks

Weiss and Kulikowski [Weis91] define neural networks as “relatively simple mathematical constructs that are often thought to loosely model biological nervous systems.” It is currently held that the human brain has roughly 100 billion neurons (nerve cells) each having at least 1000 connections to other neurons [Pand96]. As a neuron processes inputs to produce outputs, biochemical processes are thought to cause changes in the strength of the connections to other neurons. It is believed that learning and memory are associated with the strength of the connections between

various neurons. As such, when a human processes new input to the nervous system, the strength of the connections between the neurons are changing and learning is taking place [Pand96].

At a very basic level, a neural network works much like the human brain. It receives an input, processes it, and produces an output based on the received input. Neural networks are often used in attempts to develop a classification system for a particular type of input (e.g., an antibody), which is why they might be of use in the current research.

Frank Rosenblatt developed the perceptron, the simplest of neural network devices, in the 1950s [Pand96]. A simple perceptron can be used for pattern classification involving two classes [Weis91]. Figure 2.3 illustrates the operation of a simple perceptron. The perceptron accepts the various attributes of a pattern as inputs.

Each input has an assigned weight. The sum of the products of the inputs and their respective weights is compared to a predetermined threshold. If the sum falls under the threshold, the pattern is identified as belonging to Class 0, with the opposite being true for Class 1 [Weis91].

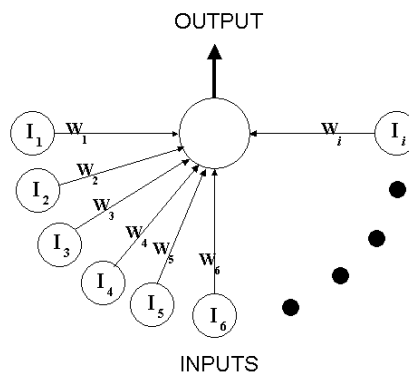


Figure 2.3 A Simple Perceptron

Obviously, it is vital for the perceptron to have the appropriate weights assigned to the various inputs. The method used for doing so involves a set of training cases that are evaluated by the perceptron. The perceptron initially randomly assigns a weight to each input. As each training case is processed, the perceptron determines the class to which the pattern belongs. Since the class of

the training cases is previously known, the perceptron can compare its classification to the correct classification. If the perceptron identified the class of the pattern correctly, nothing is done. If the pattern was incorrectly identified, the perceptron adjusts the weights of the inputs accordingly [Weis91].

As previously stated, the simple perceptron is useful for classifying some patterns with only two classes. By connecting the outputs of perceptrons to the inputs of other perceptrons, more complex neural networks can be implemented. Weiss and Kulikowski [Weis91] assert that a “three-layer network, with two layers of hidden units, can implement any separating decision surface.” To train a multi-layer neural network, a method known as back-propagation is used. In this method, an error is propagated backwards through the hidden layers of the network and the weights of the connections are adjusted.

Pandya and Macy [Pand96] list the following advantages of using neural networks:

1. Adaptiveness—Powerful learning algorithms and self-organizing rules allow [them] to self-adapt as per the requirements in a continually changing environment.
2. Nonlinear processing—Ability to perform tasks involving nonlinear relationships and noise-immunity make [them good candidates] for classification and prediction.
3. Parallel processing—Architectures with a large number of processing units enhanced by extensive interconnectivity provide for concurrent processing as well as parallel distributed information storage.

A disadvantage to using neural networks involves the concept of overtraining. Overtraining in a neural network is very similar to the overfitting of a decision tree. The “network begins memorizing the minutiae of the training set and loses its ability to generalize” [Pand96]. Another issue is that the shape of the decision space can cause problems for a neural network. When the network is being trained and is attempting to minimize the total error through a gradient descent

search, the network may accept a local minimum as the best it can do, even though a global minimum may exist that would allow the network to perform as a much better classifier.

Another disadvantage of neural networks is that it is difficult to know what is going on within the hidden layers of the network [Weis91]. This can be especially troublesome when trying to understand the effects of the input with respect to output. Finally, back-propagation can be extremely costly in computing time. Mitchell points out that “it is not uncommon for back-propagation to require tens of thousands of iterations through the weight update loop when training complex networks” [Mitc97].

2.5 Genetic Algorithms

According to Booker et al [Book89], the use of a genetic algorithm (GA) in a classification method involves building classifiers, determining their utility, and replacing the weak classifiers with offspring (random combinations) of the strong classifiers. The randomness associated with a GA allows for a wide-ranging coverage of the search space.

Table 2.2 Sample Classifier Data

	TCP_Syn_Flag	TCP_Ack_Flag	TCP_Reset_Flag
Good	0	0	0
Good	0	0	1
Good	1	1	0
Good	0	1	1
Bad	1	0	1
Bad	1	0	0
Bad	1	1	1
Bad	0	1	0

Mitchell remarks that a GA classifier is often represented as a bit string [Mitc97]. The number of bits in the string depends on the number of inputs into the classifier. The training examples from Table 2.2 (Table 2.1 repeated for convenience) would require a 7-bit classifier. Each input can have either one of two values, so the classifier must be able to represent both values. For instance,

the `TCP_Syn_Flag` input to the classifier can be a zero or a one. The classifier might use a string of 01 to represent that a zero is expected for this input.

Conversely, a classifier might use a string of 10 to represent that a value of one is expected from the input. A string of 11 would indicate that this particular classifier does not care what value is input for the variable [Mitc97]. Since there are three variables in Table 2.2, six of the bits in the classifier would be able to represent the expected input values from those variables. The other bit in the 7-bit classifier would be used to denote the class associated with each specific classifier, with a 0 representing good and a 1 representing bad [Quin93].

Table 2.3 Sample GA Classifier Data

Classifier	Class	TCP_Syn_Flag	TCP_Ack_Flag	TCP_Reset_Flag
0011110	0	01	11	10
1110110	1	11	01	10
1100110	1	10	01	10
0101011	0	10	10	11

As a simple example, the first classifier in Table 2.3 is expecting a zero from the `TCP_Syn_Flag` input, which would be provided by four of the training examples in Table 2.2. A “do not care” condition exists for the classifier’s expectant `TCP_Ack_Flag` value, so it still matches the same four records in Table 2.2. A value of 1 is expected from the `TCP_Reset_Flag` input, so the classifier ends up matching only training instances two and four. Since the two training instances belong to the same class, this particular classifier will correctly classify both of them. A larger number of training examples that are representative of the population would allow for a more precise determination of whether or not this classifier is good enough to be allowed to produce offspring.

Classifiers are trained against a data set with known classes to determine their fitness. GAs use a fitness function to evaluate the performance of the classifiers. The best performers are allowed to reproduce and carry on. The poor performers, like the second classifier in Table 2.3, which misclassifies training example 2, are replaced by the offspring of the better performers. The fitness function in a GA may take more into account than just the number of training examples

that were correctly classified. Marmelstein [Marm99] proposes that classifier fitness is associated with “coverage” and “purity.” The notion of coverage involves the number of examples the classifier matches, while the concept of purity deals with how many of the matches were correctly classified. Obviously, a classifier with a wide coverage that is very pure would be most useful [Marm99].

The concept behind genetic algorithms clearly stems from evolutionary ideas, with the propagation of the best classifiers being a sort of machine learning “survival of the fittest.” Crossover and mutation are two methods that are used to produce offspring from the best performing classifiers [Mitc97]. The use of a bit string as the classifier makes these operations fairly simple. In crossover, portions of the parent classifiers are swapped to produce offspring. Mutation involves changing random bits in a classifier, often after crossover has already been performed [Mitc97].

The manner in which classifiers evolve may produce various effects. Since a child classifier can be drastically different than its parent, the GA search is able to abruptly move around the search space. This could help to avoid the local minima problem that might affect gradient descent methods [Mitc97]. On the other hand, if a highly fit classifier reproduces too quickly, a crowding of the search space can occur that might slow the progress of the GA [Mitc97].

Mitchell [Mitc97] lists the following as reasons for the popularity of genetic algorithms:

- GAs can search spaces of hypotheses containing complex interacting parts, where the impact of each part on overall hypothesis fitness may be difficult to model.
- Genetic algorithms are easily parallelized and can take advantage of the decreasing costs of powerful computer hardware.

A disadvantage of using a GA is the previously mentioned crowding problem. In addition, Marmelstein [Marm99] points out that since evolved classifiers “[are] analogous to a path in a decision tree...classifier systems suffer from the same disadvantage—the way the data is separated may be inconsistent with a natural structure of the data.”

2.6 Computer Defense Immune System (CDIS) Antibodies

A CDIS antibody is a 320-bit binary string that contains up to 28 possible fields, or axes. Williams [Will01] illustrated the makeup of the binary string with Table 2.4, presented below. During antibody generation, the protocol type of the antibody is randomly generated from the four possibilities of IP, TCP, UDP, and ICMP. This is not to be confused with the types of packets that CDIS uses for self and non-self, namely TCP, UDP, and ICMP only. The IP antibody consists of only the IP fields, while the other three types are made up of the IP fields and the additional fields associated with the respective type, as described in Table 2.4.

All of the antibody fields are assigned random values, and then a random number of fields are activated. For example, a TCP antibody could have only four fields activated; IP Src Address A, TCP Src Port, TCPSyn, and TCP Dest Port. In a process known as negative selection, the antibody would be compared to the training data (known as self). In this example, if the IP Src Address A value was 131 and the associated range was 6, then CDIS would look for packets in the training data with an IP Src Address A value of between 125 and 137. If any packets in the training data match the first query, then they are queried for matches with the TCP Src Port value and range. This process continues until either no matches exist or the antibody fields that were activated have all been tested. If matches exist after all of the antibody fields have been tested, the antibody is considered to have matched self and is thrown out. If no matches are found, then the antibody has been successfully generated and is kept for further use [Will01].

Since the TCP antibody has a significantly greater number of fields available than the other types of antibodies, the average number of fields randomly selected to be used in a TCP antibody will be higher as well. This distinction is important because antibodies with a large number of activated fields are less likely to match self (or non-self) traffic. If an antibody has only one activated field, it is highly likely that at least one packet in the self data will have a value for that field that falls within the antibody's range. With each additional field that gets activated, the

likelihood of finding a packet that has a field value that falls within the antibody's range for all fields becomes less.

Table 2.4 Packet Features

Field Name	Possible Values	Bits Used
IP Fields (Common to all Packets)		
Protocol Type	TCP, UDP, ICMP	0-1
IP Identification Number	0-65535	2-10
IP Time to live (TTL)	0-255	18-25
IP Flags	0-65535	26-41
IP Overall Packet Length	0-65535	42-57
IP Source Address (A.B.C.D)	Valid IP address	58-89
IP Destination Address (A.B.C.D)	Valid IP address	90-121
TCP-Only Fields		
TCP Source Port	0-65535	122-137
TCP Destination Port	0-65535	138-153
TCP Sequence Number	0-4294967295	154-185
TCP Next Sequence Number	0-4294967295	186-217
TCP Acknowledgement Number	0-4294967295	218-249
All TCP Flags	0-255	250-257
Individual TCP Flags (PCWR, Echo, Urgent, Ack, Push, Reset, Syn, Fin)	Boolean	258-265
TCP Packet Size	0-65535	266-280
UDP-Only Fields		
UDP Source Port	0-65535	122-137
UDP Destination Port	0-65535	138-153
UDP Data Length	0-65535	266-281
ICMP-Only Fields		
ICMP Type	0-255	122-129
ICMP Code	0-255	130-137
ICMP Data Length	0-65535	266-281
Validity Bits		
Specify activated fields for an antibody		292-319

It is worth repeating that the number of fields that get activated in each antibody is random. That being the case, an antibody could theoretically have anywhere from 1 to 28 activated fields. Antibodies with a limited number of activated fields are very general in nature and will likely match self. It is therefore improbable that an antibody would make it through negative selection if it contained only a single activated field. With each additional field that gets activated, the more general the antibody becomes, and the less likely it is that the antibody will match self.

Once an antibody has completed negative selection it is considered to be “good” and can be compared to new samples of network traffic. A “bad” antibody is one that is discarded during negative selection because it matched self. There is a capacity within CDIS to encourage the growth of good antibodies through a process known as affinity maturation [Will01]. The general premise behind affinity maturation is that it may be possible for an antibody to cover larger ranges within its activated fields without impinging upon self. In a computationally costly process, Williams employed a genetic algorithm to adjust an antibody’s range parameters to make the antibodies cover the largest area possible [Will01]. This idea for the current research was generated, in part, by the desire to improve antibody performance while foregoing the computational cost of affinity maturation. Therefore, affinity maturation is not used in this research. Also, due to changes in the implementation of CDIS that have come about since Williams’ research, it is impossible (at this time) to make any meaningful comparison between the results yielded through affinity maturation and the current research. Hence, a thorough examination of the differences in the two approaches will be left for a future research effort.

When testing new samples of network traffic, if an incoming packet matches all of the activated fields of a good antibody, then CDIS will consider that packet to be non-self and possibly an attack. A “successful” antibody is one that has correctly identified an incoming packet as non-self. “Unsuccessful” antibodies are ones that do not detect non-self packets.

This research involves building antibodies using network traffic from a laboratory network. The initial focus of the research will be to try to determine if there are any antibody fields that, when used, are more likely to cause an antibody to be thrown out during the negative selection process. A classification system will be used in this phase of research to aid in determining the relevant antibody fields, using the good and bad antibodies.

The good antibodies will then be tested against non-self traffic generated from the previously mentioned laboratory network. The antibodies will be divided into two groups based on their ability

to detect attacks. The successful and unsuccessful antibodies will be entered into a classification system to try and determine if there are any antibody fields that are more likely to produce [un]successful antibodies.

2.7 Choosing a Classification Method

Since this research aims at gaining a better understanding of the importance of the various antibody fields in producing good/successful antibodies, the classification method must be one that provides feedback regarding the inputs to the classifier. Each of the methods discussed in Sections 2.2 through 2.5 might easily be trained to classify antibodies. Of utmost importance in this effort is the need to understand the means of classification, in particular, which fields of the antibody are most influential in determining the antibody's class.

Neural networks seem to be ill-suited for this problem. The hidden layers within the network can make it difficult to comprehend the full effects of the inputs to the classifier with respect to the output. Also, the computing expense involved with back-propagation might prove to be very costly, and is unnecessary when viable alternatives exist.

Among the three remaining classification methods, Bayesian learning methods appear to be the weakest. This research is aimed at learning about the possible relationships between variables. The assumption of conditional independence among antibodies and the notion of any causality among the variables are unfounded. They could ultimately prove to be detrimental to efforts to gain an understanding of the relationship between attributes and success or failure of an antibody.

Since GAs produce hypotheses that are essentially branches of a decision tree, it seems imprudent to prefer them over decision trees. Marmelstein [Marm99] asserts that GAs and decision trees share the weakness of possibly separating the data in a manner inconsistent with the natural data structure. That weakness aside, decision trees seem to offer exactly what is needed for this

research. After a decision tree is trained on sample data, the structure of the tree can provide information concerning the utility of the various attributes.

2.8 Summary

The focus of this chapter was to develop a theoretical background on which to build a methodology for accomplishing the current research. The chapter started with an examination of various machine-learning methods, giving advantages and disadvantages of each type. The next discussion involved the CDIS antibodies that are the focal point for this research. The final section of this chapter contained a discussion of how the specific machine learning methods examined earlier might be applied in this research and ended with a determination that decision trees appear to be the appropriate machine learning method to employ .

III. Methodology

This chapter presents the methodology used to conduct this research. The first step, discussed in Section 3.1, involved the use of the Computer Defense Immune System (CDIS) to generate antibodies. Section 3.2 is an examination of the initial antibody testing that was carried out to determine the antibody fields that might be helpful in distinguishing between good and bad antibodies. The initial implementation of a modified set of antibodies is discussed in Section 3.3. Finally, Section 3.4 provides a summation of the process involved in attempting to distinguish between successful and unsuccessful antibodies.

3.1 Antibody Generation

In order to generate antibodies, CDIS requires a sample of “normal” network traffic, also referred to as “self.” The normal network traffic for this research was derived from training data developed by Lincoln Laboratory (LL) at the Massachusetts Institute of Technology (MIT) [Lipp99]. This training data was created in 1999 and is based on network traffic from a fictitious military base. The training data was intended for use in evaluating various intrusion detection systems, but has also been used to aid in the development of intrusion detection systems, most recently by Williams [Will01]. The training data consists of weeks worth of traffic, with some days having only normal traffic and other days having both normal and attack traffic. The first week of training data is attack free and the traffic from Friday of that week was used as self for this research. The non-self database that was used to test the antibodies in this research was developed by filtering out attacks from the fourth and fifth weeks of LL data.

The LL data was stored in tcpdump format. Since CDIS did not have the capability to read data directly from a tcpdump file, the data had to be transformed into a CDIS-readable format. Ethereal, an open source network analyzer software package, was used to convert the tcpdump

file into a text file that CDIS could import. In addition, Ethereal allowed the filtering out of any packets that did not match the protocols currently supported by CDIS (TCP, UDP, and ICMP).

Once the LL network data was inserted into CDIS, antibody generation commenced (Section 2.6 contains details on antibody generation). As the antibodies were generated, some of them were thrown out because they matched self. For initial evaluation, antibodies that were successfully generated were considered to be good and antibodies that were thrown out were considered to be bad. The initial evaluation involved the generation of 5 groups of antibodies, each containing 30 sets of 256 antibodies, which resulted in a total of 38,400 good antibodies.

The antibody generation session was carefully logged via the CDIS logging utility. The log file specified the fields that were tested for each antibody, the antibodies that were thrown out, and the total generation time for all antibodies. The logs were saved for use in comparing antibodies generated with the original axes set and any antibodies generated with a reduced set of axes.

The end result of the antibody generation phase was 38,400 antibodies generated against the normal LL network traffic. There were also over 2,400 antibodies that were thrown out during the negative selection process. The first step in antibody modification was a comparison between the successfully generated antibodies and the ones that were thrown out. An inductive algorithm was used with both types of antibodies to attempt to determine the fields that were useful in classifying an antibody as good or bad.

3.2 Initial Antibody Testing

In order to build a decision tree using the antibodies, they had to be converted to the correct format. An output utility was added to CDIS to allow the antibodies to be written out as a comma-delimited file. The good and bad antibodies were output to separate files. Having the antibodies in comma-delimited format allowed for a smooth transition into the file format required by the induction algorithm software.

3.2.1 Building the Decision Tree. The University of Waikato’s Weka software (freely available at <http://www.cs.waikato.ac.nz/~ml/weka>) has an implementation of Quinlan’s [Quin93] C4.5 decision tree algorithm that was well suited to this research. The Weka decision tree implementation of C4.5 implements C4.5 Revision 8 and is known as j48 [Witt00]. Weka has an exceptionally user-friendly graphical user interface, but the data set must be converted to an arff file format of the type shown in Table 3.1 for it to be useful.

Table 3.1 Arff File Format

```
@relation Test

@attribute TCP_Syn_Flag {0,1}
@attribute TCP_Ack_Flag {0,1}
@attribute TCP_Reset_Flag {0,1}
@attribute Status {Good,Bad}

0,0,0,Good
0,0,1,Good
1,0,0,Good
1,1,0,Good
0,1,1,Good
0,0,1,Bad
0,1,1,Bad
1,0,1,Bad
1,1,0,Bad
0,1,0,Bad
```

The Weka software builds a decision tree from data held in a training file, such as the one shown in Table 3.1. With the exception of two data entries, the data in Table 3.1 was taken from Table 2.1 and converted to the correct arff file format. The additional data entries were added to provide a more robust example. The Weka software uses the data contained in the file to produce a decision tree. Figure 3.1 is the Weka-built decision tree for the arff file shown in Table 3.1.

Section 2.3 introduced the notion that a decision tree algorithm chooses the best attribute on which to split the tree at each node. The method implemented in j48 uses an entropy function (given below) to help to determine the information gain when splitting on a particular attribute [Witt00]. Witten and Franke point out that the logarithms are normally used with a base of 2 and the arguments of the entropy formula are fractions that add up to 1. The number of arguments

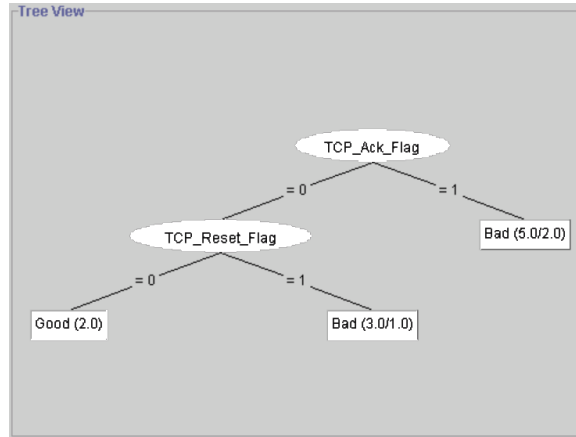


Figure 3.1 Weka Decision Tree Output

for each attribute is equal to the number of branches that will stem from that attribute. There are two branches associated with each of the antibody attributes; one represents the use of the field within the antibody and the other represents the exclusion of the field.

$$entropy(p_1, p_2, \dots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \dots - p_n \log p_n.$$

Consider the data in Table 3.1. To determine the information gain associated with any of the attributes, one must first determine how much information can be gained from the given examples. There are 10 examples: 5 of the examples are good and 5 are bad. The arguments for the entropy function would therefore be 5/10 and 5/10, representing the ratios of good and bad examples with respect to the total. The resulting function is presented below.

$$entropy(5/10, 5/10) = -(5/10) * \log(5/10) - (5/10) \log(5/10)$$

$$entropy(5/10, 5/10) = 1.0$$

The information value of 1.0 gives a numerical description of the amount of information that *can* be gained from the examples. Since the number of good examples is equal to the number of

bad examples, the information value is at its maximum. To find the information gain of a specific attribute one must first find the information value associated with it. For example, in Table 3.1 the TCP_Ack_Flag is not used (value of 0) in 3 good examples and 2 bad. The TCP_Ack_Flag is used in 2 good examples and 3 bad. This results in the two equations below that yield the information value for each branch, which, when added together, give the information value associated with the TCP_Ack_Flag attribute. Each result is a product of the entropy for a branch multiplied by the proportion of examples that belong to that branch with respect to the total number of examples present at the decision node. This ensures that proportionality is considered when determining the information values for the branches.

$$(5/10) * entropy(3/5, 2/5) = 5/10 * [-(3/5) * log(3/5) - (2/5) * log(2/5)]$$

$$(5/10) * entropy(3/5, 2/5) = 0.485$$

$$(5/10) * entropy(2/5, 3/5) = 5/10 * [-(2/5) * log(2/5) - (3/5) * log(3/5)]$$

$$(5/10) * entropy(2/5, 3/5) = 0.485$$

The equation results above represent the information value of the two branches associated with the TCP_Ack_Flag attribute. The sum of the values brings the total information value for the entire attribute to 0.971. When the total value for the attribute (0.971) is subtracted from the information value present in all of the examples (1.0), one gets the information gain offered by splitting the examples on this particular attribute. The information gain for the TCP_Ack_Flag attribute is 0.029.

Further analysis would show that neither of the other available attributes have a better information gain than the TCP_Ack_Flag. The TCP_Reset_Flag attribute does offer an equal information gain, but splitting on the TCP_Ack_Flag offers better gain in the lower levels of the tree. Frank and Witten [Witt00] do not specify the method used by j48 to determine the attribute

to choose when two or more attributes have the same information gain. It may be that j48 uses a look-ahead algorithm to determine the attribute that offers better information gain in the lower levels of the tree, but the decision could also simply rely on using the first attribute that yielded the shared information gain value.

The information gain of an attribute is dependent upon the number of examples being tested at the node being considered. For example, after splitting on the TCP_Ack_Flag, each of the branches of the root node have five corresponding examples from the training set. Therefore, the five examples that follow the 0 branch to the next decision node are never considered for any decision nodes that fall under the 1 branch, and vice versa. At each decision node the algorithm either splits on an attribute that has not been previously used in the branch or, if none of the attributes offer any information gain, leaves that node as an impure leaf.

Once the tree has been built, Weka tests it against the original training file (or a separate file consisting of a similar data set drawn from the same population) to determine the utility of the tree as a classifier,. The testing yields error information for the decision tree such as the information shown in Figure 3.2, which is the result of testing the decision tree in Figure 3.1 against the training file of Table 3.1.

<pre> === Run information === Scheme: weka.classifiers.J48.J48 -U -M 2 Relation: Test Instances: 10 Attributes: 4 TCP_Syn_Flag TCP_Ack_Flag TCP_Reset_Flag Status Test mode: evaluate on training data === Classifier model (full training set) === J48 unpruned tree ----- TCP_Ack_Flag = 0 TCP_Reset_Flag = 0: Good (2/0) TCP_Reset_Flag = 1: Bad (3/0/1.0) TCP_Ack_Flag = 1: Bad (5/0/2.0) Number of Leaves : 3 Size of the tree : 5 </pre>	<pre> Time taken to build model: 0.02 seconds === Evaluation on training set === === Summary === Correctly Classified Instances 7 70 % Incorrectly Classified Instances 3 30 % Kappa statistic 0.4 Mean absolute error 0.3733 Root mean squared error 0.432 Relative absolute error 74.6667 % Root relative squared error 86.4099 % Total Number of Instances 10 === Detailed Accuracy By Class === TP Rate FP Rate Precision Recall F-Measure Class 0.4 0 1 0.4 0.571 Good 1 0.6 0.625 1 0.769 Bad </pre>
--	--

Figure 3.2 Weka Decision Tree Error Output

The tree in Figure 3.2 shows that Weka chose to split the sample data set from Table 3.1 on the TCP_Ack_Flag attribute. There were 5 instances where the TCP_Ack_Flag was used (value of 1). Of those 5 instances, 3 were bad and 2 were good; hence, the label of “Bad” on the corresponding leaf. Of the 3 instances in which the TCP_Ack_Flag was not used and the TCP_Reset_Flag was used, 2 were bad and 1 was good, resulting in the leaf being labelled as “Bad.”

Once the antibodies were generated and written out to a comma-delimited file, they were reformatted for entry into Weka. For the purpose of the current research, the decision tree algorithm merely requires the ability to distinguish between a field that was used and one that was not. Therefore, each antibody was reformatted into a 33 bit string, with each bit representing an antibody field. If a field was used in an antibody, its corresponding bit was a 1; if a field was not used, its bit was a 0. The status of the antibody (good or bad) was appended to the bit string. The Weka output was useful in determining modifications that might help to improve the antibodies.

The antibodies needed to be tested against non-self traffic to determine the initial detection rates. These initial detection rates were to become the standard by which any future modified antibodies with reduced axes would be measured.

3.2.2 Non-Self Detection. Non-self traffic was generated on the same LL network as the normal traffic. The non-self traffic consisted of several types of attacks carried out against the various machines on the network. In gathering the non-self traffic from the LL data, large chunks of network traffic were filtered from the fourth and fifth weeks of LL data based on known attacks and their parameters. Hence, the 50,000 packet non-self database that was used to test the antibodies was made up of non-self packets that would naturally resemble self traffic, in addition to the actual attack packets. This causes some problems with evaluating the quality of the antibodies, because it makes the derivation of false positive and false negative rates extremely difficult. However, this research is not dependent upon false positive or false negative rates, but rather on how modified antibodies compare to their predecessors. These comparisons can take place in many ways; to

include average antibody generation time, the number of bad antibodies yielded per group of antibodies produced, and even non-self detection rates.

The 38,400 antibodies that had been successfully generated were tested against the non-self data. In this process, each antibody was compared to every single non-self packet. If an antibody matched any packet, it was marked as successful: an antibody that did not match any of the non-self packets was tagged as unsuccessful. The time to scan (per antibody) and the success rate of the antibodies (number of non-self packets detected) were logged for future analysis. Also, the specific packets that were detected were logged so that a comparison could be made with the packets detected by future antibody data sets.

3.3 Initial Antibody Modifications

Once j48 had built the decision tree, a careful analysis of the decision tree showed how an adjustment to the antibody structure could help to improve the production of good antibodies. Modifying the antibodies required some minor rework of the CDIS code. As shown in Table 2.4, the final 28 bits of the antibody bit string are validity bits. These bits determine whether or not each of the 28 possible fields will be used for a particular antibody. When the validity bit is set to the value of one, the corresponding field is used for that particular antibody. Explicitly setting a validity bit to zero ensures that the field is not valid for that antibody.

With the antibodies modified within the CDIS code, new antibodies were generated, again using the LL data. The generation time and discard rate of these antibodies was compared to the original antibodies. The analysis of these findings is presented in Chapter IV.

When the modified antibodies completed negative selection, they were tested against the previously discussed non-self traffic. The successful and unsuccessful antibodies were written out to separate comma-delimited files for use in the next phase of the research. The antibody time

to scan and detection rate information was collected to be compared with the previously gathered original antibody data.

The number of non-self packets that the reduced axes antibodies detected is very important. One major indicator of a successful reduced axes set would be little to no significant decrease in the number of packets detected by the new antibodies. Another indicator would be an increase in the number of successful antibodies over the original data set. The analysis of these findings is also presented in Chapter IV.

3.4 Further Antibody Analysis

Having adjusted the antibodies to discourage the generation of bad antibodies, the next goal was to determine if there were any fields that could affect whether or not an antibody was successful or unsuccessful in detecting attacks. This process involved formatting the antibodies in the same manner as mentioned in Section 3.2.1, with a 1 representing a field that was used and a 0 representing a field that was not. The antibodies were labelled as successful or unsuccessful and input into j48. The resulting decision tree provided insight as to which of the fields could be useful in classifying an antibody as successful or unsuccessful.

Some of the antibody fields were eliminated based on analysis of the Weka output. New antibodies were generated using this final set of reduced axes. Finally, those antibodies were tested against the non-self data. The analysis of the discard rate and the rates of detection is given in Chapter IV.

3.5 Summary

This chapter describes the methodology used to complete this research. A discussion of antibody generation is given in Section 3.1. Section 3.2 presented the method used to gather normal and attack traffic from the LL data. There is also an examination of how the j48 algorithm

was used to build a decision tree. Section 3.3 discusses the process used to initially modify the CDIS antibodies. Finally Section 3.4 provides a quick summation of the method used to further refine the axes set by examining both successful and unsuccessful antibodies.

IV. Analysis and Findings

The focus of this chapter is the analysis of the data produced during this research. Section 4.1 covers the initial decision tree built from antibodies gathered from a normal data set. The first round of modification yielded a new data set of antibodies. The results of the comparison between the new and old data sets is discussed in Section 4.2. The testing of the modified data set yielded some successful and unsuccessful antibodies (antibodies that did or did not detect non-self). These antibodies were then used to create a second decision tree for further evaluation and antibody modification. The examination of the second decision tree is covered in Section 4.3. The analysis of the data set yielded through a final round of modification is presented in Section 4.4. Finally, Section 4.5 provides some final insight into the research results.

4.1 Initial Decision Tree Analysis

As mentioned in Chapter III, the original data set for this research consisted of 38,400 “good” antibodies and 2,402 “bad” antibodies. A random sample of 2,402 good antibodies was drawn to be combined with the bad antibodies for decision tree analysis. Both samples were split into halves. One half of the good antibodies and half of the bad were combined to make up a training file for the decision tree software. The other halves were used to build a test file.

The decision tree in Appendix A was generated using the original antibody training file. It is one of five decision trees that was built using the bad antibodies with five different sets of good antibodies. Each decision tree was built using a training file and tested on a completely different file to determine its utility as a classifier. The decision tree in Appendix A correctly classified 80.8 percent of the cases in its test file, and was the worst performing tree of the five in that respect. The trees shared many similarities in respect to the fields that were deemed to be most useful. Among four of the five trees, the first five fields were identical, and the remaining tree only had

slight differences. Based on these observations, the decision tree presented in Appendix A was chosen to carry out the initial analysis.

It was hoped that the decision tree(s) would show that some of the antibody fields were blatantly useless. If a certain field or group of fields could be shown to have a cause and effect relationship in producing bad antibodies, the next phase of the research was destined to go more smoothly. However, it turned out that the use of certain fields was easier to link with the production of good antibodies, as opposed to bad.

A good example of this phenomenon is the UDPLength field. This field can be found at the first node of the decision tree (Appendix A). Every antibody that used the UDPLength field (value of 1 in the tree) turned out to be good. The UDPLength field alone could be used to explain roughly 13 percent of all of the good antibodies in the test set.

Further evidence supporting the notion that field inclusion was to be more important than field exclusion can be found in the first five nodes of the decision tree. These nodes represent the UDPLength, TCPFlags, ICMPCCode, UDPSrcPort, and TCPECNEcho fields. There were 615 antibodies that used one of these fields. Of those antibodies, 603 were good, accounting for roughly 51 percent of all of the good antibodies in the test set.

Another noteworthy and unexpected feature of the decision tree was that the exclusion of certain fields seemed to make bad antibody generation more likely. The decision tree in Appendix A shows that 683 of the bad antibodies share a common characteristic; none of them use any of the fields represented by the first 13 nodes of the decision tree. This alone accounts for approximately 57 percent of the bad antibodies in the test set. If one were to include the use of the IPFlags field, and exclude the use of the IPDestA field, another 142 bad antibodies are accounted for, a total of 68 percent of all of the bad antibodies.

Analysis of the decision tree suggested that rather than having fields of little importance that could be done away with, there existed fields of great importance that should be more highly

emphasized. A quick scan of the decision tree made it apparent that the TCP, UDP, and ICMP fields were more important when it came to producing good antibodies. These fields were typically near the root of the tree, which implied that they had a greater significance in determining whether an antibody was good or bad. Also clearly evident in the decision tree is the lack of any prominent role by any of the IP fields. The IP fields tend to turn up at the very low levels of the tree where their significance in regards to classification is greatly reduced.

Table 4.1 Original Antibody Data

	Total	IP	TCP	UDP	ICMP
All Antibodies					
Number Good	38,400	9,636	9,607	9,719	9,438
Number Bad	2,402	1,803	80	253	266
Training File					
Number Good	1,201	316	307	312	266
Number Bad	1,201	901	40	127	133
Testing File					
Number Good	1,201	322	333	270	276
Number Bad	1,201	902	40	126	133

Further evidence of the importance of the TCP, UDP, and ICMP fields is shown in Table 4.1. Of the 2,402 bad antibodies, 1,803 (75 percent) of them were IP antibodies. A reasonable explanation can be found for this phenomenon. Every packet in the self data contains IP fields, whether it is a TCP, UDP, or ICMP packet. Therefore, when an IP antibody undergoes negative selection, it is compared to every single packet in the self data. Conversely, the TCP, UDP, and ICMP antibodies are only compared to TCP, UDP, and ICMP packets respectively, thus reducing the likelihood that they will match any of the self data. Because they are compared to a fewer number of packets, TCP, UDP, and ICMP antibodies have a lower mortality rate during negative selection.

With that in mind, it seemed the sensible thing to do was to rid CDIS of the IP antibody. The IP fields were to be left intact because the other antibody types would still use them. However, the generic IP antibody was no longer available for production.

4.2 Modified Data Set Analysis

With the first round of modification complete, the focus was to build a new data set of antibodies. As discussed in Chapter III, 5 groups of 30 sets of antibodies were generated, with each set containing 256 antibodies.

4.2.1 Bad Antibody Comparison. The breakout of good and bad antibodies in the modified data set is shown in Table 4.2. The number of bad antibodies, 831, represents a 75 percent decrease from the 2,402 in the original data set. As in the original data set, the number of bad TCP antibodies is significantly lower than the number of bad UDP or ICMP antibodies. As discussed in Section 2.6, a likely explanation for this is that the greater number of fields available for use in the TCP antibody allows it to be more specialized and therefore less likely to match self.

Table 4.2 Modified Antibody Data

	Total	TCP	UDP	ICMP
Number Good	38,400	12,658	13,016	12,726
Number Bad	831	128	331	372

4.2.2 Generation Time Comparison. The antibodies had shown improvement, at least with respect to the number of bad antibodies produced during negative selection. However, another area of concern was antibody generation time. One possible method to help determine whether or not there is a difference in average antibody generation times was to statistically compare the generation times of the two groups of antibodies.

Antibodies were generated in sets of 256, with each set taking roughly an hour to complete the generation process. The generation time data for the original antibodies were gathered from 60 sets of antibodies that were generated on the same machine under identical circumstances. The original antibodies had a mean (μ_o) generation time of 12.48 seconds per good antibody, with a standard deviation (s_o) of 2.70. Using the z statistic with the mean and standard deviation known,

a hypothesis test can be carried out to determine if the antibody generation times of the two groups of antibodies are significantly different.

The time data for the modified antibodies were gathered from 60 sets of antibodies that were generated on the same machine and under identical circumstances as previously discussed. The mean generation time (μ_a) per antibody of the modified antibodies was 11.45 seconds and the standard deviation (s_a) was 2.27.

The null hypothesis (H_o) for this test was that there was no difference in generation times for the two groups of antibodies, or that $\mu_o - \mu_a = 0$. The alternate hypothesis (H_a) was that the generation time of the modified antibodies was less than the generation time of the original, or that $\mu_o - \mu_a > 0$. Given a value for α of 0.05, the upper critical bound for the z statistic is shown by Devore to be 1.645 [Devo00]. Using the equation below [Devo00] and the variable values listed above one could easily find the z value to be tested. Note that m and n represent the number of trials for the two sets and are both equal to 60.

$$z = \frac{\mu_o - \mu_a}{\sqrt{\frac{\sigma_o^2}{m} + \frac{\sigma_a^2}{n}}}$$

Solving the equation with the variable values listed yields 2.306 as the result. Since the z statistic test value is greater than the upper critical bound, the null hypothesis could be disregarded in favor of the alternate hypothesis. The shaded region of the distribution in Figure 4.1 represents the area below the upper critical bound. As is readily apparent, the z statistic test value falls well above the critical value, implying that the modified antibodies took a statistically significantly shorter amount of time to generate at a significance level of $\alpha = 0.05$. Hence, the modified antibodies had proven to be improved on two fronts; they yielded fewer bad antibodies and were generated in a statistically significantly shorter time.

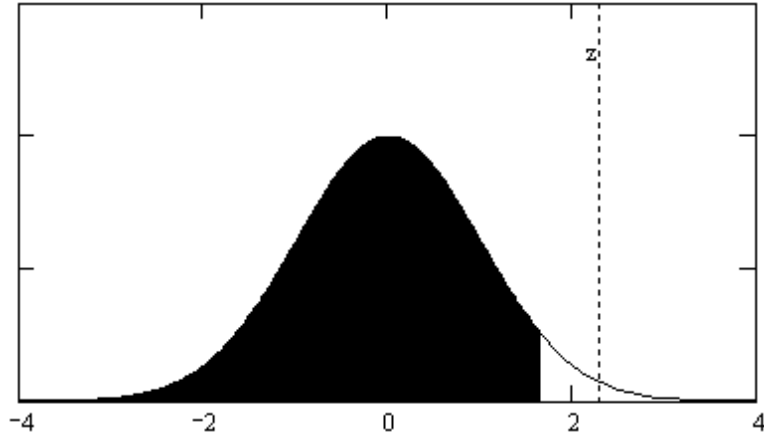


Figure 4.1 Original Antibody Generation Time z -Distribution

4.2.3 Non-Self Detection Comparison. Another comparison that could be made between the two sets of antibodies is their ability to detect non-self packets. The original 150 sets of antibodies were tested against a non-self database of 50,000 packets (cf., Section 3.2.2). The original 38,400 antibodies detected 17,725 non-self packets in the attack database. Of the 38,400 antibodies, only 76 actually detected non-self.

Table 4.3 Non-Self Detection Information

	Total	IP	TCP	UDP	ICMP
Good Antibody Data					
Original	38,400	9,636	9,607	9,719	9,438
Modified	38,400	0	12,658	13,016	12,726
Original Detection Data					
Successful	76	7	2	25	42
Unsuccessful	38,324	0	9,603	9,644	9,436
Modified Detection Data					
Successful	101	0	8	31	62
Unsuccessful	38,299	0	12,650	12,985	12,664

The 38,400 modified antibodies were tested against the same non-self database as the original antibodies. They detected 18,294 packets; 569 packets more than the original antibodies. Of these 18,294 non-self packets, 17,553 were also detected by the original set of antibodies. This means that the modified antibodies found 99 percent of the packets detected by the original antibodies.

In addition, a total of 101 different modified antibodies detected non-self, an increase of 33 percent over the original antibodies.

4.3 Further Antibody Modification

A final attempt at antibody improvement involved building five decision trees using the successful antibodies from the modified antibody testing. Because of the small number of successful antibodies (see Table 4.3), the decision trees were built and tested using only a training file. The makeup of the training file is presented below in Table 4.4. The ratio of the unsuccessful antibodies is representative of the ratio found in the good modified antibodies.

Table 4.4 Successful Modified Antibody Training File

	Total	TCP	UDP	ICMP
Successful	101	8	31	62
Unsuccessful	101	31	36	34

One of the decision trees built from the successful and unsuccessful modified antibodies is shown in Appendix B. It correctly classified 84 percent of the training instances and appeared to suggest a few possible modifications. First, the use of the TCPECNEcho or UDPSrcPort fields accounted for approximately 32 percent of the unsuccessful antibodies. Additionally, a couple of the IPSrc fields present in the tree offered evidence that encourage their exclusion. For example, the IPSrcB field (in concert with the IPFlags field) accounted for 23 unsuccessful antibodies. The IPSrcC field, when *not* used in conjunction with a few of the other suspect fields, contributed to 58 of the 101 successful antibodies. Also, the use of the IPPacketLen field (in concert with the IPFlags field) accounted for 11 unsuccessful antibodies. Finally, the TCPSeqNum field, when used, accounted for 4 unsuccessful antibodies. The fields mentioned here were prominent in at least three of the five decision trees. Therefore, the decision to remove the fields from the antibody structure was based on analysis of all five of the trees.

Table 4.5 Modified Packet Features

	Fields Removed	Fields Remaining
IP Fields (Common)	IP Flags IP Source Address (B.C)	Protocol Type IP Identification Number IP Time to live (TTL) IP Overall Packet Length IP Source Address (A.D) IP Destination Address (A.B.C.D)
TCP-Only Fields	TCP Sequence Number TCPECNEcho	TCP Source Port TCP Destination Port TCP Next Sequence Number TCP Acknowledgement Number All TCP Flags TCPPCWR TCPUrgent TCPAck TCPPush TCPReset TCPSyn TCPFin TCP Packet Size
UDP-Only Fields	UDP Source Port	UDP Data Length
ICMP-Only Fields		ICMP Type ICMP Code ICMP Data Length

4.4 Final Data Set Analysis

The antibodies were modified once again based on the successful/unsuccessful decision tree analysis. Table 4.5 shows the antibody fields that were kept and those that were removed. Next came a final round of antibody generation and testing, again resulting in 5 groups of 30 sets of antibodies, with each set containing 256 antibodies. The data from the final group of antibodies is presented in Table 4.6.

Table 4.6 Final Antibody Data

	Total	TCP	UDP	ICMP
Number Good	38,400	12,740	12,808	12,852
Number Bad	6,166	794	2598	2774
Successful	642	55	279	308
Unsuccessful	37,758	12,685	12,529	12,544

4.4.1 Bad Antibody Comparison. The final data set had a noticeable increase in the number of bad antibodies. The 6,166 bad antibodies represented a 642 percent increase from the previous data set. It is likely no coincidence that there was a corresponding change in the average number of terms, which dropped from 9.00 in the first modified set down to 6.29 in the final set. One logical explanation for this phenomenon is that antibodies with a small number of fields activated are much more general in nature. For example, an antibody with only one field activated is not likely to survive the negative selection process. The reason is that the only comparison made between the antibody and all of the self packets is for a specific range of the field that is activated. All other fields are essentially irrelevant. This is a problem because it is likely that within the self packets there will be at least one with a value for the activated field that falls within the range that the antibody is covering. If there is at least one self packet that falls within the range, the antibody is discarded.

As the number of fields activated for an antibody increases, the likelihood that it will be discarded decreases. As discussed in Chapter II, for each additional field that is activated another logical AND operation is introduced to the comparison of the antibody and the self packet. For example, if an antibody has three fields activated, the self packet value for the first field AND the second field AND the third field must fall within the respective ranges that the antibody has for those fields. Each additional activated field makes an antibody more precise in nature, and therefore less likely to find matching packets within self. With that in mind, the connection between the drop in the average number of terms and the significant rise in the number of bad antibodies becomes much clearer.

4.4.2 Antibody Generation Time Comparison. Unlike the previous improvement attempt, this one resulted in a significant increase in antibody generation time. The average antibody generation time for the final set of antibodies was 18.4 seconds per antibody with a standard deviation of 2.52. This increase of 6.97 seconds represents a significant increase at any level of α .

At least some of this increase in generation time can be attributed to the increase in the number of bad antibodies, as the time taken to generate each antibody includes the time that was wasted on checking any bad antibodies.

4.4.3 Non-Self Detection Comparison. The number of packets detected by the final set of antibodies rose by 2,363 over the previous data set. The breakdown of the successful and unsuccessful antibodies is given in Table 4.6. In addition to detecting more packets, the final set of antibodies had a significantly higher number of successful antibodies. The 642 antibodies that detected packets in this data set represented a more than 500 percent increase over the previous data set.

There are two possible explanations for the increase in successful antibodies in the final data set. First, as the discussion in Section 4.4.1 makes clear, these antibodies averaged a fewer number of activated fields than their predecessors. For reasons already discussed, the smaller number of fields made it more likely that the antibodies would match non-self packets. Another possible explanation is that the antibodies were more highly concentrated in respect to the important fields for detecting the non-self packets. The elimination of the fields that appeared to be contributing to the generation of unsuccessful antibodies made it more likely that this set of antibodies would activate the fields that were common in the successful antibodies from the previous data set.

4.5 Final Analysis

When analyzing the results of this research effort, one must keep in mind its purpose; namely, to explore ways to build more effective, efficient antibodies through modification of the fields that may be used within them. The two rounds of antibody modification had very different outcomes. The first round resulted in an obvious improvement over the initial set of antibodies. The reduction in the number of bad antibodies coupled with a shorter generation time and a rise in the number of non-self packets detected appeared to vindicate the elimination of the IP antibody.

Table 4.7 Figures For All Data Sets

Data Set	Average Number Discarded per 256 Antibodies	Average Generation Time	Average Number of Terms	Number of Successful Antibodies
Original	16.1	12.48	8.13	76
First Modification	5.54	11.43	9	101
Final Modification	41.1	18.39	6.29	642

The second round of modification had more mixed results than the first. A significant rise in average generation time per good antibody and a much larger number of bad antibodies almost overshadows the gains made in the number of successful antibodies. The rise in antibody generation time and the rise in the number of bad antibodies are closely tied together as the greater the number of bad antibodies, the greater the amount of time wasted during antibody generation.

The results of the second round of modification raise the question as to what exactly is the most important aspect of antibody generation to improve. Reducing generation time is important, but will likely eventually become less so with advances in hardware and processor speed. At any rate, it is not the area of greatest importance when generating antibodies.

The number of bad antibodies produced during antibody generation is a concern, but only in that it increases average generation times. In truth, producing a large number of bad antibodies implies that the remaining antibodies may be very similar to the self data and is actually desirable. The reason for this is that one wants the areas surrounding self to be covered as completely as possible.

Finally, there is the need for the development of successful antibodies. Growing the number of successful antibodies was of paramount importance in this research effort. Although one must be satisfied when an antibody is successful in detecting an attack, a greater level of confidence is gained when multiple antibodies detect the same event. In reducing the number of fields available to the CDIS antibodies, they may have been concentrated in the areas where the non-self traffic is more likely to be found and are therefore more likely to detect non-self.

4.6 *Summary*

This chapter discussed the analysis of the research data. Section 4.1 was an examination of the decision tree that was built using the original data set of antibodies. The decision tree analysis yielded possible modifications to be implemented. Section 4.2 presented a discussion of the antibodies that were built after the adjustment was made to the CDIS code. The modified antibodies were compared to the original data set via generation times, bad antibody generation, and non-self packet detection. Section 4.3 covered the decision tree built from both successful and unsuccessful antibodies gathered from the modified data set. Further possible modifications were recommended as a result of the successful/unsuccessful decision tree analysis. Section 4.4 examines the differences between the final data set of antibodies and the previous sets within the previously discussed areas. Finally, Section 4.5 provided final insight into the research results.

V. Conclusions and Recommendations

This chapter consists of conclusions, limitations, and recommendations. Section 5.1 presents some general conclusions regarding the entire research effort. The limitations of the research and its applicability are discussed in Section 5.2. Finally, Section 5.3 gives some recommendations for future research.

5.1 Conclusions

The main goal of this research was to explore a process to create better computer defense immune system (CDIS) antibodies. While it is true that it was quite a broad goal, the research method ensured that the scope of the effort was limited to improvements in the antibody generation process, specifically focusing on the antibody fields. The improvements that were realized through this research came at a cost. The increased number of successful antibodies was paid for by the greater amount of time that was required for generation. Nevertheless, the main goal was satisfied and improvements were made.

This research also provides a methodology used to determine possible antibody improvements using different data sets. Since self and non-self data will likely be site-specific, the improvements developed through this research will likely not apply in all situations. Still, the use of decision trees and their inductive power has proven to be a valid method for determining modifications that can lead to the production of better antibodies.

5.2 Limitations

A major limiting factor in this research was the non-self data. As discussed in Chapter III, there was no way to derive false positive and false negative rates with the given data set. Definitive false positive and false negative rates would have lent more credibility to the results found here. As it stands, questions concerning the abilities of the modified antibodies to detect attacks remain. The

antibodies were successful in detecting at least some actual attack packets. Still, a better defined non-self data set with attack packets singled out would have allowed for a better determination of the success of the modified antibodies.

Another limitation associated with this research is the limited applicability. As previously mentioned, self and non-self data will most likely be site-specific. This fact ensures that the antibody configuration at one site may not be as effective at other sites. Hence, the field preferences found in this research will likely not apply at a real-world site. However, the methodology used to modify these antibodies could be implemented anywhere to help to improve antibody generation and performance.

A final limitation is that this research only considered whether or not a particular field was active within each antibody. It could be that there are ranges within the fields that are important. It could be that there was a very important small range of values within any of the fields that were discarded. If that is the case, then the methodology used here makes it impossible to detect that range of important values, and thus some possibly very important information is lost.

5.3 Recommendations for Future Research

One possible avenue for future research involves a modification of the methodology used in this research. The modification involves using the ranges of values associated with antibody fields as opposed to just binary values representing whether a field was active or not. It seems possible that certain ranges of values within each field might be important in generating successful antibodies. Some of the fields discarded in the latter stages of this research could prove to have important ranges within them. The use of range values as inputs to the decision tree would likely make it more complex to decipher, but it could also lead to the development of more effective antibodies.

Another possible area for future research would be an examination and refinement of the antibody fields themselves. For example, the IP source and destination fields do not seem to add

any real value in detecting attacks in their current form. A thorough study should be conducted to determine the features of packets that are important when scanning for attacks. Such information could enable improvements in CDIS antibody generation by enabling a higher concentration of antibodies in the areas of the search space that matter most.

A final recommendation for future research involves the use of the methodology presented here on a much better defined non-self data set. The ability to single out attack packets would allow for definitive false positive and false negative rates, and hence allow a researcher to better verify the levels of success of modified antibodies.

Appendix A. Decision Tree from Good/Bad Antibodies

Number of Leaves: 81

Size of the tree: 161

Time taken to build model: 0.95 seconds

=== Evaluation on test set ===

=== Summary ===

Correctly Classified Instances	1941	80.8077%
Incorrectly Classified Instances	461	19.1923%
Kappa statistic		0.6162
Mean absolute error		0.2295
Root mean squared error		0.3876
Relative absolute error		4538981%
Root relative squared error		77.5299%
Total Number of Instances		2402

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.798	0.182	0.815	0.798	0.806	Good
0.818	0.202	0.802	0.818	0.81	Bad

=== Confusion Matrix ===

<i>a</i>	<i>b</i>	– classified as
958	243	a = Good
218	983	b = Bad

=== Run information ===

Scheme: weka.classifiers.j48.J48 -C 0.05 -M 2

Relation: GBAntibodies

Instances: 2402

Attributes: 34

IPID
IPTTL
IPFlags
IPPacketLength
IPSrcA

IPSrcB
IPSrcC
IPSrcD
IPDestA
IPDestB
IPDestC
IPDestD
TCPSrcPort
TCPDestPort
TCPSeqNum
TCPNextSeqNum
TCPAckNum
TCPFlags
TCPCWR
TCPECNEcho
TCPUrgent
TCPAck
TCPPush
TCPReset
TCPSyn
TCPFin
TCPPacketSize
UDPSrcPort
UDPDestPort
UDPLength
ICMPType
ICMPCode
ICMPDataLength
Status

Test mode: user supplied test set: 2402 instances

=== Classifier model (full training set) ===

J48 pruned tree

UDPLength = 0
| TCPFlags = 0
| | ICMPCode = 0
| | | UDPSrcPort = 0
| | | | TCPECNEcho = 0
| | | | TCPReset = 0
| | | | | ICMPType = 0

```

| | | | | TCPNextSeqNum = 0
| | | | | UDPDestPort = 0
| | | | | ICMPDataLength = 0
| | | | | IPPacketLength = 0
| | | | | TCPCWR = 0
| | | | | IPFlags = 0: Bad (752.0/69.0)
| | | | | IPFlags = 1
| | | | | IPDestA = 0: Bad (188.0/46.0)
| | | | | IPDestA = 1
| | | | | IPTTL = 0
| | | | | IPSrcB = 0: Bad (57.0/16.0)
| | | | | IPSrcB = 1
| | | | | IPSrcA = 0
| | | | | IPDestD = 0
| | | | | IPSrcC = 0: Bad (11.0/1.0)
| | | | | IPSrcC = 1: Good (2.0)
| | | | | IPDestD = 1
| | | | | IPDestC = 0
| | | | | IPSrcC = 0: Good (2.0)
| | | | | IPSrcC = 1: Bad (6.0)
| | | | | IPDestC = 1: Good (6.0)
| | | | | IPSrcA = 1: Good (8.0)
| | | | | IPTTL = 1
| | | | | IPDestB = 0
| | | | | IPSrcC = 0: Bad (10.0/3.0)
| | | | | IPSrcC = 1: Good (7.0/2.0)
| | | | | IPDestB = 1: Good (15.0)
| | | | | TCPCWR = 1
| | | | | IPSrcB = 0
| | | | | IPID = 0: Good (2.0)
| | | | | IPID = 1: Bad (7.0/2.0)
| | | | | IPSrcB = 1: Good (5.0)
| | | | | IPPacketLength = 1
| | | | | TCPDestPort = 0
| | | | | IPFlags = 0: Bad (289.0/103.0)
| | | | | IPFlags = 1: Good (158.0/54.0)
| | | | | TCPDestPort = 1: Good (3.0)
| | | | | ICMPDataLength = 1
| | | | | IPSrcA = 0
| | | | | IPFlags = 0: Bad (12.0/3.0)
| | | | | IPFlags = 1
| | | | | IPPacketLength = 0
| | | | | IPSrcC = 0: Good (3.0)
| | | | | IPSrcC = 1: Bad (3.0)
| | | | | IPPacketLength = 1: Good (7.0)
| | | | | IPSrcA = 1: Good (25.0/2.0)
| | | | | UDPDestPort = 1
| | | | | IPDestA = 0
| | | | | IPFlags = 0
| | | | | IPPacketLength = 0: Bad (16.0/4.0)
| | | | | IPPacketLength = 1: Good (2.0)
| | | | | IPFlags = 1: Good (8.0)
| | | | | IPDestA = 1: Good (20.0)
| | | | | TCPNextSeqNum = 1: Good (17.0/2.0)
| | | | | ICMPType = 1

```

```

| | | | | IPFlags = 0
| | | | | IPSrcA = 0
| | | | | IPDestD = 0: Bad (11.0/1.0)
| | | | | IPDestD = 1: Good (11.0/2.0)
| | | | | IPSrcA = 1: Good (28.0/4.0)
| | | | | IPFlags = 1: Good (34.0)
| | | | | TCPReset = 1
| | | | | IPDestA = 0
| | | | | TCPPacketSize = 0
| | | | | IPFlags = 0: Bad (7.0/1.0)
| | | | | IPFlags = 1
| | | | | IPSrcC = 0: Good (4.0)
| | | | | IPSrcC = 1: Bad (2.0)
| | | | | TCPPacketSize = 1: Good (19.0/1.0)
| | | | | IPDestA = 1: Good (30.0)
| | | | | TCPECNEcho = 1: Good (80.0/6.0)
| | | | | UDPSrcPort = 1: Good (95.0/2.0)
| | | | | ICMPCode = 1: Good (134.0/2.0)
| | | | | TCPFlags = 1: Good (151.0/2.0)
| | | | | UDPLength = 1: Good (155.0)

```

Number of Leaves : 42

Size of the tree : 83

Time taken to build model: 1.48 seconds

=== Evaluation on test set ===

=== Summary ===

Correctly Classified Instances	2006	83.5137 %
Incorrectly Classified Instances	396	16.4863 %
Kappa statistic	0.6703	
Mean absolute error	0.2302	
Root mean squared error	0.3597	
Relative absolute error	46.0453 %	
Root relative squared error	71.9472 %	
Total Number of Instances	2402	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.767	0.097	0.888	0.767	0.823	Good
0.903	0.233	0.795	0.903	0.846	Bad

=== Confusion Matrix ===

```

a b <-- classified as
921 280 | a = Good
116 1085 | b = Bad

```

Appendix B. Decision Tree from Successful/Unsuccessful Antibodies

Number of Leaves: 13

Size of the tree: 25

Time taken to build model: 0.17 seconds

=== Evaluation on test set ===

=== Summary ===

Correctly Classified Instances	170	84.1584%
Incorrectly Classified Instances	32	15.8416%
Kappa statistic		0.6832
Mean absolute error		0.2523
Root mean squared error		0.3552
Relative absolute error		50.4679%
Root relative squared error		71.0408%
Total Number of Instances		202

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.911	0.228	0.8	0.911	0.852	Good
0.772	0.089	0.897	0.772	0.83	Bad

=== Confusion Matrix ===

<i>a</i>	<i>b</i>	– classified as
92	9	a = Good
23	78	b = Bad

=== Run information ===

Scheme: weka.classifiers.j48.J48 -C 0.25 -M 2

Relation: GBAntibodies

Instances: 202

Attributes: 34

- IPID
- IP TTL
- IPFlags
- IPPacketLength
- IPSrcA
- IPSrcB
- IPSrcC
- IPSrcD
- IPDestA
- IPDestB
- IPDestC
- IPDestD
- TCPSrcPort
- TCPDestPort
- TCPSeqNum
- TCPNextSeqNum
- TCPAckNum
- TCPFlags
- TCPCWR
- TCPECNEcho
- TCPUrgent
- TCPAck
- TCPPush
- TCPReset
- TCPSyn
- TCPFin
- TCPPacketSize
- UDPSrcPort
- UDPDestPort
- UDPLength
- ICMPType
- ICMPCode
- ICMPDataLength
- Status

Test mode: evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree

TCPECNEcho = 0

| UDPSrcPort = 0

| | IPFlags = 0

| | | TCPSeqNum = 0

| | | | IPSrcC = 0: Successful (68.0/10.0)

| | | | IPSrcC = 1

| | | | | IPTTL = 0: Successful (21.0/6.0)

| | | | | IPTTL = 1

| | | | | | ICMPDataLength = 0

```

| | | | | IPSrcB = 0
| | | | |   IPDestB = 0
| | | | |     ICMPCode = 0: Successful (5.0/1.0)
| | | | |     ICMPCode = 1: Unsuccessful (2.0)
| | | | |       IPDestB = 1: Unsuccessful (4.0)
| | | | |       IPSrcB = 1: Successful (6.0/1.0)
| | | | |         ICMPDataLength = 1: Unsuccessful (2.0)
| | | | |         TCPSeqNum = 1: Unsuccessful (5.0/1.0)
| | | | |   IPFlags = 1
| | | | |     IPSrcB = 0
| | | | |       IPPacketLength = 0: Successful (15.0/5.0)
| | | | |       IPPacketLength = 1: Unsuccessful (12.0/1.0)
| | | | |       IPSrcB = 1: Unsuccessful (28.0/5.0)
| | | | |     UDPsrcPort = 1: Unsuccessful (15.0/1.0)
| | | | |     TCPECNEcho = 1: Unsuccessful (19.0/1.0)

```

Bibliography

- [Almg01] Almgren, Magnus and Ulf Lindqvist. "Application-Integrated Data Collection for Security Monitoring." *Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection, RAID 2001*. 22–36. Davis, CA: Springer-Verlag, October 2001.
- [Bace00] Bace, Rebecca G. *Intrusion Detection*. Indianapolis: Macmillan Technical Publishing, 2000.
- [Bala98] Balasubramanian, Jai Sundar, et al. *An Architecture for Intrusion Detection Using Autonomous Agents*. Technical Report 98/05, Purdue University, 1998.
- [Book89] Booker, L.B., et al. "Classifier Systems and Genetic Algorithms." *Machine Learning Paradigms and Methods*. 235–282. Cambridge, MA: The MIT Press, 1989.
- [Cros95] Crosbie, Mark and Eugene H. Spafford. *Active Defense of a Computer System using Autonomous Agents*. Technical Report 95-0008, Purdue University, 1995.
- [Denn87] Denning, Dorothy E. "An Intrusion Detection Model," *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.
- [Devo00] Devore, Jay. *Probability and Statistics for Engineering and the Sciences*. Pacific Grove, CA: Brooks/Cole Publishing Company, 2000.
- [Ghos99] Ghosh, Anup, et al., "Learning Program Behavior Profiles for Intrusion Detection." World Wide Web, 1999. World Wide Web Page. URL http://www.usenix.org/publications/library/proceedings/detection99/full%_papers/ghosh/ghosh_html.
- [Hofm99] Hofmeyr, Steven and Stephanie Forrest. "Architecture for an Artificial Immune System," *Evolutionary Computation*, 7(1):1289–1296, 1999.
- [Marm99] Marmelstein, Robert E. *Evolving Compact Decision Rule Sets*. PhD dissertation, AFIT/DS/ENG/99-05, School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, September 1999.
- [Mich83] Michalski, Ryszard S. "A Theory and Methodology of Inductive Learning." *Readings in Machine Learning*. 70–95. San Mateo: Morgan Kaufmann Publishers, 1983.
- [Mitc97] Mitchell, Tom M. *Machine Learning*. Boston, MA: McGraw-Hill, 1997.
- [Pand96] Pandya, A. and R. Macy. *Pattern Recognition with Neural Networks in C*. Boca Raton, FL: CRC Press, 1996.
- [Quin93] Quinlan, J. *C45: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
- [Spaf00] Spafford, Eugene H. and Diego Zamboni. *Data Collection Mechanisms for Intrusion Detection Systems*. Technical Report 2000-08, Purdue University, 2000.
- [Stol89] Stoll, Clifford. *The Cuckoo's Egg*. New York: Doubleday, 1989.
- [Weis91] Weiss, S. and C. Kulikowski. *Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. San Mateo, CA: Morgan Kaufmann, 1991.
- [Will01] Williams, Paul D. *Warthog: Towards a Computer Immune System for Detecting "Low and Slow" Information System Attacks*. MS thesis, AFIT/GCS/ENG/01M-15, School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2001.

- [Will01a] Williams, Paul D., et al. “CDIS: Towards a Computer Immune System for Detecting Network Intrusions.” *Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection, RAID 2001*. 117–133. Davis, CA: Springer-Verlag, October 2001.
- [Witt00] Witten, Ian H. and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, CA: Morgan Kaufmann, 2000.

Vita

1st Lt Russell J. Aycock enlisted in the United States Air Force in June of 1991 as an F-15 Avionics Apprentice. After attending technical training at Lowry AFB, CO, his first assignment was with the 60th Fighter Squadron at Eglin AFB, FL. In January of 1994 Lt Aycock was reassigned to the 493rd Fighter Squadron at RAF Lakenheath, UK. During his four year tour in England he attended night classes with the University of Maryland's European division and earned his Bachelors degree in Computer Science. Lt Aycock was selected to attend Officer Training School in August of 1997 and was commissioned as a 2nd Lt in the Air Force in April, 1998.

Lt Aycock's first assignment as an officer was with the 355th Communications Squadron at Davis-Monthan AFB, AZ. He spent a majority of his time as a network projects engineer and served as the chief of the Network Control Center. Lt Aycock was selected to attend the Air Force Institute of Technology (AFIT) in the Information Resource Management program in January of 2000, and reported to AFIT in August of that year.

1st Lt Aycock was a distinguished graduate from Airman Leadership School and the Basic Communication Officer Training course. In his 10 years of service he has been awarded two Air Force Commendation medals and one Air Force Achievement medal. Upon graduation, 1st Lt Aycock will be assigned to the Air Force Communications Agency at Scott AFB, IL.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 26-03-2002		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From - To) Jun 2001 - Mar 2002		
4. TITLE AND SUBTITLE USING AN INDUCTIVE LEARNING ALGORITHM TO IMPROVE ANTIBODY GENERATION IN A SINGLE PACKET COMPUTER DEFENSE IMMUNE SYSTEM				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Russell J. Aycock, 1st Lt, USAF				5d. PROJECT NUMBER 01-179		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P. Street, Building 640 WPAFB, OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GIR/ENG/02M-01		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFGB Attn: Mr. John Feldman 525 Brooks Rd. Rome, NY 13441-4505 Commercial: (315) 330-2664				10. SPONSOR/MONITOR'S ACRONYM(S)		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT The United States Air Force relies heavily on computer networks for many day-to-day activities. Many of these networks are affected by various types of attacks that can be launched from anywhere on the globe. The rising prominence of organizations such as the AFCERT and the MAJCOM NOSC's is evidence of an increasing realization among the Air Force leadership that protecting our computer networks is vitally important. A critical requirement for protecting our networks is the ability to detect attacks and intrusion attempts. This research is an effort to refine a portion of an AFIT-developed intrusion detection system known as the Computer Defense Immune System (CDIS). CDIS is based on the human immune system and uses antibodies to attempt to detect network intrusion attempts. The antibodies have various attributes of which a subset is randomly activated at generation time. This research attempts to determine which of the antibody attributes are most useful in helping to build successful antibodies.						
15. SUBJECT TERMS intrusion detection, inductive algorithm, computer security, artificial immune systems, computer immunology, machine learning						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			Dr. Gregg H. Gunsch, Ph.D., ENG	
			UU	94	19b. TELEPHONE NUMBER (Include area code) (937) 255-3636, ext 4281	