

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-2002

Using Relational Schemata in a Computer Immune System to Detect Multiple-Packet Network Intrusions

John L. Bebo

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Information Security Commons](#)

Recommended Citation

Bebo, John L., "Using Relational Schemata in a Computer Immune System to Detect Multiple-Packet Network Intrusions" (2002). *Theses and Dissertations*. 4414.

<https://scholar.afit.edu/etd/4414>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.



**Using Relational Schemata
in a Computer Immune System
to Detect Multiple-Packet
Network Intrusions**

THESIS

John L. Bebo, First Lieutenant, USAF

AFIT/GCS/ENG/02M-02

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.

AFIT/GCS/ENG/02M-02

**Using Relational Schemata
in a Computer Immune System
to Detect Multiple-Packet
Network Intrusions**

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

John L. Bebo, B.G.S.
First Lieutenant, USAF

March, 2002

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

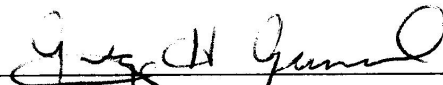
AFIT/GCS/ENG/02M-02

Using Relational Schemata
in a Computer Immune System
to Detect Multiple-Packet
Network Intrusions

John L. Bebo, B.G.S.

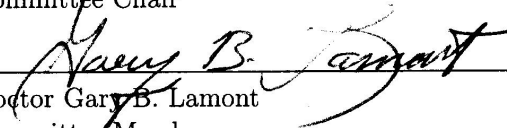
First Lieutenant, USAF

Approved:



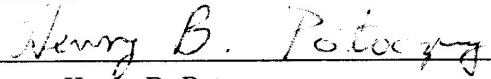
Doctor Gregg H. Gunsch
Committee Chair

8 MAR 2002
Date



Doctor Gary B. Lamont
Committee Member

8 MAR '02
Date



Doctor Henry B. Potoczny
Committee Member

8 MARCH 2002
Date

Acknowledgements

There are so many people who have helped me during these past eighteen months of study that naming them all would be impossible. Still, I'd like to mention a few individuals, to whom I'm particularly grateful.

First, I'd like to thank God, without whom, nothing is possible. I'd also like to thank the Air Force for believing in me enough to send me to such a prestigious academic institution. Until I got to AFIT, I thought I could hold my own with intellectuals. It wasn't until coming here that I saw how truly brilliant some people are.

Next, I'd like to thank my incredibly insightful and knowledgeable thesis advisor, Dr. Gunsch, who always gave me just enough rope to hang myself, but kept watch over me to ensure I didn't tie the knot too tight. More often than once, just when I thought I was plunging through the trap doors of the gallows, he would throw me a lifeline crafted from his pure genius. Without his guidance I would have surely failed in my research effort.

I'd also like to thank my sponsor, the Air Force Research Laboratory, as well as my AFIT colleagues, particularly Captains Paul Williams and Kevin Anchor, who patiently put up with my incessant questions, and propelled me along this research path.

Finally, and most importantly, I'd like to thank my sweetheart, confidant, counselor, companion, best friend, and the most beautiful (inside and out) personal trainer in Ohio. She meticulously proofread hundreds of pages of technically tedious material, patiently listened to hours of whining and complaining, happily pulled me away from the computers and books when I needed a break, and never failed with her belief in me. You helped me through much, much more than just school, honey, and I shall be eternally grateful.

John L. Bebo

Table of Contents

| | Page |
|------------------------------------------------|------|
| Acknowledgements | iii |
| List of Figures | ix |
| List of Tables | xi |
| List of Abbreviations | xii |
| Abstract | xiv |
| I. Introduction | 1-1 |
| 1.1 Motivation | 1-1 |
| 1.2 Background | 1-2 |
| 1.3 Research Problem | 1-6 |
| 1.4 Multiple-Packet Detection | 1-6 |
| 1.5 Computer Immune System | 1-9 |
| 1.6 Approach | 1-9 |
| 1.6.1 Develop Relational Genes | 1-10 |
| 1.6.2 Develop Relational Chromosomes | 1-10 |
| 1.6.3 Generate Antibodies | 1-10 |
| 1.6.4 Construct Training Sets | 1-11 |
| 1.6.5 Construct Testing Sets | 1-11 |
| 1.6.6 Detection Tests | 1-11 |
| 1.7 Scope and Limitations | 1-11 |
| 1.8 Roadmap | 1-13 |

| | Page |
|-----------------------------------------------------------------|------|
| II. Literature Review | 2-1 |
| 2.1 Intrusion Detection | 2-1 |
| 2.2 Intrusion Detection Systems | 2-1 |
| 2.3 Methodologies in Anomaly Detection | 2-3 |
| 2.3.1 Neural Networks | 2-3 |
| 2.3.2 Data Mining | 2-4 |
| 2.3.3 Genetic Algorithms | 2-4 |
| 2.4 Artificial Immune Systems | 2-6 |
| 2.4.1 Pathogen | 2-6 |
| 2.4.2 Immunity | 2-6 |
| 2.4.3 Gene | 2-7 |
| 2.4.4 Chromosome | 2-7 |
| 2.4.5 Antibody | 2-8 |
| 2.4.6 Negative Selection | 2-8 |
| 2.4.7 Affinity Maturation | 2-9 |
| 2.4.8 Costimulation | 2-9 |
| 2.5 Intrusion Detection Via a Computer Immune Systems | 2-9 |
| 2.6 Multiple-Packet Intrusion Detection | 2-10 |
| 2.7 Relational Schemata | 2-12 |
| III. High Level Design and Methodology | 3-1 |
| 3.1 Goals | 3-1 |
| 3.2 Relational Design | 3-2 |
| 3.2.1 Packet | 3-3 |
| 3.2.2 Field Value | 3-3 |
| 3.2.3 Gene | 3-4 |
| 3.2.4 Chromosome | 3-5 |
| 3.2.5 Antibody | 3-5 |

| | Page | |
|-------|---------------------------------------------------------------|------|
| 3.3 | Crosschecking Genes | 3-6 |
| 3.4 | Packet Values | 3-7 |
| 3.5 | Relational Space | 3-7 |
| 3.6 | Self/Non-Self Domains | 3-10 |
| 3.7 | CIS Usage | 3-12 |
| 3.8 | Attack Visualization | 3-13 |
| IV. | Low Level Design and Implementation | 4-1 |
| 4.1 | Version 0.0, Equal/Not-Equal Gene | 4-2 |
| 4.2 | Version 1.0, Less-Than, Greater-Than, Equal Gene | 4-5 |
| 4.3 | Version 2.0, Multiple Protocols, Additional Fields | 4-6 |
| 4.4 | Version 3.0, Three-Dimensional Chromosome Structure | 4-8 |
| 4.5 | Version 4.0, Four-Dimensional Chromosome Structure | 4-10 |
| 4.6 | Version 2.44, Beta Crosschecking Gene | 4-10 |
| 4.7 | Version 2.116, Full Crosschecking Gene | 4-12 |
| 4.8 | GOPHER | 4-13 |
| V. | Experimentation and Analysis | 5-1 |
| 5.1 | Testing Methodology | 5-1 |
| 5.2 | Version 0.0 Testing | 5-4 |
| 5.3 | Version 1.0 Testing | 5-5 |
| 5.4 | Version 2.44 Testing | 5-6 |
| 5.5 | Version 2.116 Testing | 5-10 |
| 5.6 | Versions 2.0, 3.0, and 4.0 Testing | 5-11 |
| 5.6.1 | Speed Testing | 5-12 |
| 5.6.2 | Effectiveness Testing | 5-14 |
| 5.7 | Analysis of Results | 5-19 |
| 5.7.1 | Low Hanging Fruit | 5-19 |

| | Page |
|-------------------------------------------------------------------|------|
| 5.7.2 Higher Hanging Fruit | 5-22 |
| 5.7.3 Unreachable Fruit | 5-24 |
| 5.7.4 Fruit With Worms | 5-24 |
| 5.7.5 Comparison To Snort | 5-24 |
| 5.7.6 Discussion of Attacks | 5-25 |
| 5.8 GOPHER Use | 5-27 |
| VI. Findings and Conclusions | 6-1 |
| 6.1 Benefits | 6-1 |
| 6.1.1 Relational Schemata | 6-1 |
| 6.1.2 More Difficult Detection Avoidance | 6-1 |
| 6.1.3 CIS Framework | 6-2 |
| 6.1.4 Low and Slow Attacks | 6-2 |
| 6.1.5 GOPHER | 6-2 |
| 6.2 Limitations | 6-3 |
| 6.2.1 Packet Payload | 6-3 |
| 6.2.2 Alert Confirmation | 6-3 |
| 6.3 Future Research Opportunities | 6-4 |
| 6.3.1 Expansion of the CIS Model | 6-4 |
| 6.3.2 Statistical ID | 6-4 |
| 6.3.3 Developing Relational Signatures | 6-5 |
| 6.3.4 Relational Host-Based ID | 6-5 |
| 6.3.5 Relational Schemata Changes | 6-6 |
| 6.3.6 Statistical Lack of Relationships as an Indicator | 6-6 |
| 6.3.7 Stochastic Packet or Field Selection | 6-6 |
| 6.4 Concluding Remarks | 6-7 |
| Appendix A. Matrix of Code Version 1.0 | A-1 |

| | Page |
|--------------------------------------------------------------------|--------|
| Appendix B. Matrix of Code Version 2.0 | B-1 |
| Appendix C. Matrix of Code Version 2.44 | C-1 |
| Appendix D. Matrix of Code Version 2.116 | D-1 |
| Appendix E. Matrix of Code Version 1.0 Plus MAC Address | E-1 |
| Appendix F. Run Time Data | F-1 |
| Appendix G. Lincoln Lab Attack Descriptions | G-1 |
| Appendix H. Screen Shots of Attacks Analyzed with GOPHER | H-1 |
| Appendix I. Source Code Availability | I-1 |
| Bibliography | BIB-1 |
| Vita | VITA-1 |

List of Figures

| Figure | | Page |
|--------|-------------------------------------------------------------------------------------------------|------|
| 1.1. | Man in the Middle Attack Scenario | 1-5 |
| 1.2. | Man in the Middle Attack Expanded | 1-8 |
| 2.1. | Example Intrusion Detection Signature | 2-2 |
| 2.2. | Multilayered Immune System [Hofm99] | 2-8 |
| 2.3. | Costimulation Process | 2-10 |
| 2.4. | CIS Antibody Lifecycle | 2-11 |
| 3.1. | Typical Packet Textual Display | 3-2 |
| 4.1. | Man in the Middle Attack | 4-3 |
| 4.2. | Man in the Middle Attack Equal/Not-Equal Relational Matrix . . . | 4-4 |
| 4.3. | Partial Less Than, Greater-Than, and Equal Relational Matrix . . | 4-6 |
| 4.4. | GOPHER Main Screen | 4-15 |
| 4.5. | GOPHER Field Breakouts | 4-16 |
| 4.6. | GOPHER Drill Down Screen | 4-17 |
| 5.1. | ARP Spoof Attack | 5-8 |
| 5.2. | DSniff Attack Data Files | 5-9 |
| 5.3. | Run Time Comparisons | 5-13 |
| 5.4. | SYN Flood Depicted by GOPHER | 5-28 |
| 5.5. | Attack 51.180445 Visualized With GOPHER | 5-29 |
| A.1. | Less-than, Greater-than, and Equal Relational Matrix and the Joncheray MitM Attack | A-1 |
| B.1. | Example 2-D Matrix | B-1 |
| C.1. | Example Beta Crosschecking Matrix | C-1 |

| Figure | | Page |
|--------|------------------------------------------------------------------|------|
| D.1. | Example Full Crosschecking Matrix | D-1 |
| E.1. | Less-than, Greater-than, Equal Matrix With MAC Addresses Added | E-1 |
| F.1. | Run Times of Code Versions | F-1 |
| F.2. | Standard Deviation | F-1 |
| G.1. | Attack Descriptions (adapted from Lincoln Lab Database [LLab99]) | G-1 |
| H.1. | Attack 45.192523 IP Sweep | H-1 |
| H.2. | Attack 54.145832 Satan Scan Main Screen | H-1 |
| H.3. | Attack 54.145832 Satan Scan UDP Drill Down | H-2 |
| H.4. | Attack 54.145832 Satan Scan TCP Drill Down | H-2 |
| H.5. | Attack 54.145832 Satan Scan TCP Slice | H-3 |
| H.6. | Attack 54.195951 Mscan | H-3 |
| H.7. | Attack 41.162715 Portsweep | H-4 |
| H.8. | Attack 52.130655 Ping of Death | H-4 |

List of Tables

| Table | | Page |
|-------|-----------------------------------------------------------------|------|
| 3.1. | Packet Fields Used in Gene Construction | 3-8 |
| 4.1. | Code Version Overview | 4-2 |
| 4.2. | Matrix Fields Used in Code Versions 2.0, 3.0, and 4.0 | 4-8 |
| 4.3. | Beta Crosschecking Field Categories | 4-12 |
| 4.4. | Full Crosschecking Field Categories | 4-13 |
| 5.1. | Lincoln Lab Attacks Test Results | 5-16 |
| 5.2. | False Positive Test Results | 5-17 |

List of Abbreviations

| Abbreviation | | Page |
|--------------|-----------------------------------------------------------------|------|
| AFIT | Air Force Institute of Technology | 2-9 |
| AIDS | Application-based IDS | 1-2 |
| AIS | Artificial Immune System | 1-4 |
| ARP | Address Resolution Protocol | 1-6 |
| BIS | Biological Immune System | 2-6 |
| CIS | Computer Immune System | 1-4 |
| DARPA | Defense Advanced Research Projects Agency | 1-11 |
| DoD | Department of Defense | 1-1 |
| DoS | Denial of Service | 1-2 |
| EAs | Evolutionary Algorithms | 2-4 |
| GAs | Genetic Algorithms | 2-3 |
| GAO | Government Accounting Office | 1-1 |
| GASSATA | Genetic Algorithm for Simplified Security Audit Trails Analysis | 2-6 |
| GOPHER | Graphically Oriented Pattern Honing and Evaluation Repository | 4-14 |
| GUI | Graphical User Interface | 1-3 |
| HIDS | Host-based IDS | 1-2 |
| ICMP | Internet Control Message Protocol | 1-12 |
| ID | Intrusion Detection | 1-3 |
| IDS | Intrusion Detection System | 1-1 |
| InfoCon | Information Condition | 4-10 |
| ISSA | Information Systems Security and Assurance | 5-28 |
| MAC | Media Access Control | 3-3 |
| MIT | Massachusetts Institute of Technology | 1-11 |
| MitM | Man in the Middle | 1-4 |
| MPA | Multiple-Packet Attack | 3-1 |

| Abbreviation | | Page |
|--------------|-----------------------------------------|------|
| NIC | Network Interface Card | 5-5 |
| NIDS | Network-based IDS | 1-2 |
| TCP | Transmission Control Protocol | 1-11 |
| TTL | Time To Live | 1-12 |
| UDP | User Datagram Protocol | 1-12 |

Abstract

Given the increasingly prominent cyber-based threat, there are substantial research and development efforts underway in network and host-based intrusion detection using single-packet traffic analysis. However, there is a noticeable lack of research and development in the intrusion detection realm with regard to attacks that span multiple packets. This leaves a conspicuous gap in intrusion detection capability because not all attacks can be found by examining single packets alone. Some attacks may only be detected by examining multiple network packets collectively, considering how they relate to the “big picture,” not how they are represented as individual packets.

This research demonstrates a multiple-packet relational sensor in the context of a Computer Immune System (CIS) model to search for attacks that might otherwise go unnoticed via single-packet detection methods. Using relational schemata, multiple-packet CIS sensors define “self” based on equal, less than, and greater than relationships between fields of routine network packet headers. Attacks are then detected by examining how the relationships among attack packets may lay outside of the previously defined “self.”

Furthermore, this research presents a graphical, user-interactive means of network packet inspection to assist in traffic analysis of suspected intrusions. The visualization techniques demonstrated here provide a valuable tool to assist the network analyst in discriminating between true network attacks and false positives, often a time-intensive, and laborious process.

Using Relational Schemata
in a Computer Immune System
to Detect Multiple-Packet
Network Intrusions

I. Introduction

1.1 Motivation

As we become more dependent on computers, the potential becomes greater for attacks on computer systems to cause more significant damage. In fact, the United States Government Accounting Office (GAO) reports an “ever-increasing number of cyber threats and attacks occurring over the Internet” and declared that, “Thousands of potential cyber attacks are launched against Department of Defense (DoD) systems and networks daily” [USGAO01]. We need to protect ourselves from these attacks. Regrettably, we are not protecting ourselves to an acceptable level. According to Internet Security Systems, Inc., “Current government systems are protected by a patchwork collection of tools of dubious quality” [Durs99]. To ensure that governmental systems are protected to the highest extent possible, those tools need to be improved.

There are many tools used for computer protection and security, including firewalls, anti-virus software, file integrity checkers, and effective user authentication policies. However, none of these methods of defense is a complete solution in and of itself. The best defense is defense-in-depth, or a layered defense [Mand01]. Even in a layered defense, however, each layer needs to be as robust as possible because the total system is only as secure as its most vulnerable layer. One such layer of protection is the Intrusion Detection System (IDS). IDSs do just what the name implies; they try to detect intrusions or unauthorized activity on the computers or networks that they are protecting. The goal of this thesis investigation is strengthening the current IDS technology to detect more attacks.

1.2 Background

In the past, IDSs have been categorized into two types: those that reside on a network, and those that reside on a host, or the computer under attack [Sans01, Bace01]. More recently, another type of IDS has begun to emerge as its own type of intrusion detection system. In this third type the intrusion detection sensors have been incorporated into system applications [Almg01]. Here they shall be referred to as simply network-based IDS (NIDS), host-based IDS (HIDS), and application-based IDS (AIDS). These three types of IDSs have their own strengths and weaknesses; the strengths are discussed, in a comparison fashion, alluding to the weaknesses.

Some benefits of NIDSs are that they typically cost less to manage than a series of HIDSs. Since NIDSs typically monitor real-time network traffic, they make it harder for an intruder to remove evidence of his attack. Another advantage of NIDSs is that they can be placed outside the firewall, detecting attacks that do not even penetrate the firewall. This is useful in identifying patterns of malicious activity, perhaps providing forewarning of an upcoming, potentially successful attack. Finally, NIDSs can detect attacks that host-based IDSs miss, such as many IP-based denial of service (DoS) attacks. [ISSI98]

Host-based IDSs, on the other hand, have their own benefits, which may sometimes make them more effective than network-based or application-based IDSs. For instance, HIDS can handle encrypted traffic, something NIDSs have difficulty with. HIDSs also allow for a stronger forensic analysis of the attack, as they are capable of monitoring host-specific system activities, such as file transfers, file permission changes, and application-centric tasks. Finally, HIDSs can detect attacks that NIDSs cannot possibly detect, such as those that occur on the host itself and not across the network [ISSI98].

Application-based ID used to be considered a subset of host-based ID [Bace01]. This is largely due to the fact that any application must reside on a some kind of computer host, and because AIDS and HIDS share some of the same strengths, such as being able to handle encrypted traffic. More recently, AIDS have been incorporated into system network applications. For instance, a data-collection module has been implemented in an Apache Web server application to serve as an intrusion detection sensor [Almg01]. This type of

detection shifts away from the traditional log file analysis of other AIDs to complement current network-based and host-based detection [Almg01]. Ideally, a system of protection would have NIDSs monitoring the network, several HIDSs monitoring individual hosts, and AIDs to monitor applications – again, a layered defense. This research focuses on one of those layers in an effort to improve the security of the overall system, that layer being the network IDS.

Just as there are different types of IDSs, there are two techniques of intrusion detection (ID): signature-based detection, also called misuse detection, and anomaly-based detection [Grah00]. Signature-based detection strives to match potential attacks against a previously defined set of attack “signatures,” while anomaly-based detection strives to detect patterns of activity that are departures from normal activity. [Ghos99]

An often-significant drawback of anomaly detection is determining if the IDS alert is based on an actual attack or is based merely on benign, though anomalous, activity. The latter is termed a “false positive.” Determining if an alert is a true positive or a false positive consumes much time, resources, and attention largely because the primary means of network packet inspection is textual, meaning, the analyst must wade through network packets, inspecting one packet at a time. To mitigate this burden this research introduces a user-driven, interactive, visual technique for network packet inspection and analysis. With the assistance of a Graphical User Interface (GUI) the network analyst can more quickly and efficiently swim through the sea of network packets, looking for an intrusion.

Most of the currently available network IDSs utilize packet signatures to detect known attacks. Typically a network IDS will have a database of attack signatures, and as network traffic is monitored, the IDS checks to see if any of this traffic matches the stored signatures. While this is a good method for detecting known attacks, it also means that these IDSs will likely not catch unknown attacks. This is because with unknown attacks, there is no previously defined signature with which to compare incoming network traffic. What may be more alarming is that even known attacks need to be modified only slightly to change the signature of the attack and slip past the IDS. Clearly, network-based intrusion detectors need a more reliable method to detect both known and unknown attacks.

Much work and study has been done to improve the present capability of network IDSs, including the use of data mining, state-based systems, and manual detection [LeeW99, Me98, Kemm01]. One area of research that holds much promise in detecting intrusions is the use of an Artificial Immune System (AIS) or Computer Immune System (CIS) [Will01, Soma97]. Computer immunology is a special brand of anomaly-based detection where the computer imitates the body's natural protection mechanisms, treating an invader to the body, such as a virus, as an attack. This is such a very interesting concept that it is explored further in this thesis.

Nearly all of the systems mentioned above presently target single-packet attacks, sometimes using a simple counter to detect numerous single-packet attacks. This counter mechanism is not sufficient, as discussed in Chapter II, because not all attacks are single-packet attacks. Thus, some attacks may go unnoticed with single-packet detection alone. For example, the well-known Man In The Middle (MitM) attack can be perpetrated by an attacker through desynchronizing communications between two other parties. In the MitM attack, an attacker has a sniffer on the communications path between the two parties and can monitor all traffic flowing back and forth. In this instance, if the attacker can desynchronize communications between both parties, he can establish his own communications with each, and reproduce the traffic that the other was sending. He may also modify this reproduced traffic to suit his needs. This is useful for overcoming one-time password security systems like SKEY, or ticketing authentication systems such as Kerberos. [Jonc95]

Figure 1.1 (a) shows a routine network session, with a three-way handshake and data being sent. The letters *S* and *A* represent Transmission Control Protocol (TCP) *SYN* and *ACK* fields set to one, respectively, while the *C* and *S* depict TCP client and server sequence numbers. This is discussed more in depth in Section 1.4. The right hand side, (b), shows how an attacker, sending the shaded packets, can desynchronize this same traffic and cause confusion between the parties, allowing him to inject his own data.

In this scenario the attacker sends a *reset* packet with the spoofed client's sequence number, causing the server to ignore any further packets from the true client with that sequence number. This desynchronizes the communications between the client and server, enabling the attacker to execute his assault. The attacker then initiates his own session

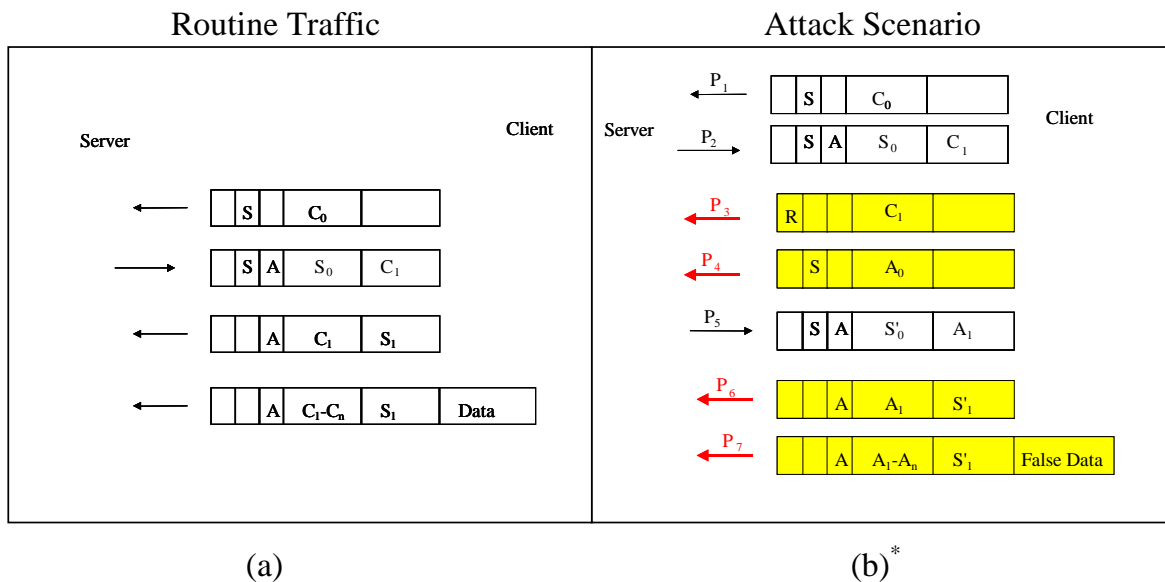


Figure 1.1 Man in the Middle Attack Scenario
 *This figure is a modified version of Figure 2 in [Jonc95]

with the server, depicted by packet P_{3a} , with the attacker's initial sequence number A_0 and the forged address of the client. At this point, the client will continue to send data which the server will ignore since the server's connection has been reset. This allows the attacker to copy, or "sniff" selected data from the client and present it to the server as data from the attacker. [Jonc95]

The devious part of this attack is that the intruder sends only packets that might very well be seen in actual network traffic, making the attack hard to detect by single-packet monitoring IDSs. Specifically, the packet that desynchronizes the communications between the client and the server is depicted in Figure 1.1 (b) as packet P_3 . This packet could very well be seen in normal network activity, making it difficult to notice an attack in progress. Only when viewed in the larger context, that of the other packets flowing through the network, can this type of attack be detected. This is what is meant by multiple-packet intrusion detection.

1.3 Research Problem

The MitM attack described previously is just one example of a multiple-packet attack. Other examples include Address Resolution Protocol (ARP) and IP spoofing, as will be discussed more in Chapter V. Still other examples of multiple-packet attacks include scans and probes. Currently scans and probes are typically detected via simple counter mechanisms. If a specified number of scans occur in a specified time, then it is deemed an attack [Stan00]. This is not necessarily effective, because if an attacker can guess the counter value, he has to only scan one less than that value to go undetected. He might then wait until the specified time has expired and resume his stealthy attack. This is an example of a “low and slow” attack.

Since there are attacks, such as the multiple-packet MitM attack, which will likely slip past a simple counter, a mechanism for detecting these multiple-packet attacks is needed. This research addresses developing that mechanism.

1.4 Multiple-Packet Detection

This thesis effort concentrates on improving current IDS capabilities by providing an enhanced venue for detecting not only known multiple-packet attacks, but also unknown multiple-packet attacks. As stated earlier, this research is geared toward NIDSs. This is because NIDSs, sitting at the entry point to a network, can detect an attack even before it penetrates the firewall. This makes a NIDS a valuable means of protection; one that should be as robust as possible. Although this research is tailored to network-based IDSs, it is certainly likely that the means of detecting multiple-packet attacks on networks discussed here might be extended to host-based IDSs as well.

In order to detect multiple-packet attacks, this document defines schemata based on the relationships among packets flowing through the network. These schemata are then used to define patterns of normal and abnormal network traffic, the heart of anomaly-based detection. To accomplish this feat, relationship schemata are addressed as simple equal, greater-than, or less-than relationships between the fields of two network packets. Further research might extend these relationships to include other nominal, ordinal, or

even quantitative relationships containing formulas that may be user-defined. Multiple-packet detection may also be enhanced to detect attacks that span three or more packets. As a minimum, the proof of concept that multiple-packet attacks can be detected via relationship schemata is demonstrated here via simple equal, greater-than, and less-than relationships between fields of two packets.

Section 1.2 discussed how the MitM attack could be perpetrated, but not how it could be detected with respect to packet relational patterns. To understand this, we broaden the view of the attack scenario to include one more packet, and its relationship to the other packets flowing through the network.

Figure 1.1 (a) showed normal network traffic and how, after receiving a *syn/ack* from the server, the client continued with its three-way handshake and sent its own *ack* back in response. If Figure 1.1 (b) is examined, it can be seen that the attacker sends the *reset* request to the server, desynchronizing this communication path. However, the *reset* is sent to the server, not to the client. Since the client does not know that the server has dropped its session, the client continues sending its traffic, just as it normally would if there were no attack taking place. This is shown in Figure 1.2 with the addition of packet P_{3a} . This additional packet is sent from the true client, just as was shown in Figure 1.1 (a). By examining this addition in Figure 1.2, a clue that something is amiss can be seen.

In this broader view of the attack, two network packets can be seen, both seemingly coming from the same client with the same sequence numbers, but saying two different things. In packet P_3 , the spoofed client's packet, the client is requesting a reset. However, in packet P_{3a} , the real client's packet, the client is sending an acknowledgement to the server. This would not normally happen in network traffic, as sending contradictory messages would quickly cause chaos in the network. This anomaly can be detected with relational schemata.

With relational schema it can be determined that these two packets are coming from the same client, as they have the same client sequence number (equal relationship), but they are saying two different things, as the reset, acknowledge, and server sequence number fields are different (not equal relationship). This can be shown mathematically with the

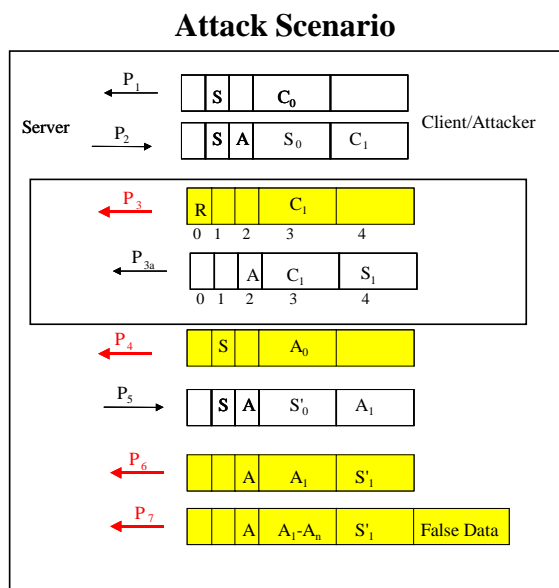


Figure 1.2 Man in the Middle Attack Expanded
The center box shows a spoofed, and a real client packet

addition of one more subscript to note the field number from left to right, as depicted in Figure 1.2. In this instance we have three different ways of determining that there is an inconsistency in two packet transmissions, which is to say three possibly different means of detecting this one attack from the two network packets under consideration. The relational patterns that represent abnormal behavior are:

$$(P_{3,3} = P_{3a,3}) \wedge (P_{3,0} \neq P_{3a,0})$$

or

$$(P_{3,3} = P_{3a,3}) \wedge (P_{3,2} \neq P_{3a,2})$$

or

$$(P_{3,3} = P_{3a,3}) \wedge (P_{3,4} \neq P_{3a,4})$$

In short, these relationships represent the client (identified by client sequence number) saying two different things in three different ways. A benefit of using this relationship mapping is that equal/not equal relationships are easily converted to a string of ones and zeros, with equal mapped to one, and not equal mapped to zero. These strings of ones and

zeros can be thought of as digital “chromosomes” that contain instructions which determine the makeup of the system much like human chromosomes contain instructions which determine the biological makeup of each of us. These chromosomes can then be used to define sensors to search for intrusions in a Computer Immune System.

1.5 Computer Immune System

A Computer Immune System is useful in detecting intrusions in a network by emulating the biological processes of the human body. In a CIS, sometimes called an Artificial Immune System (AIS), “self” may be defined by “building the normal behavior patterns of a monitored system” [Kim01a, Kim01]. This research does just that; it represents self as digital chromosomes, constructed from normal patterns of network activity. In the human body, a virus or infection constitutes non-self, and the body’s immune system strives to detect and repel such foreign entities [Harm01, Will01]. A CIS used in network ID strives to emulate this process in a digital, vice physical manner, detecting network intrusions instead of biological intrusions. As digital chromosomes in this research represent normal network activity, any pattern of activity that does not match a self-chromosome is considered malicious, possibly an intrusion. A central component of the CIS is the antibody, a mechanism for detecting non-self activity. Antibodies can be explicitly defined, as might be useful in signature-based detection, or implicitly defined, as merely activity not matching previously defined chromosomes. The latter method is used in this research, as within the testing conducted, the concept of self does not change after it is defined. Explicitly defining antibodies may be useful, however, as they might then be assigned a temporal life span and evolve with the changing concept of self. This is discussed more in Chapter II. A brief overview of how a CIS can be used in detecting multiple-packet network intrusions follows, as well as some possible limitations to this approach.

1.6 Approach

In using a CIS to detect multiple-packet network intrusions, a number of steps must be accomplished. These steps are:

1.6.1 Develop Relational Genes. Defining the genes that can be used as a vocabulary for representing self and non-self is the first step that needs to be accomplished in intrusion detection using a CIS. Since this research is focused on relational patterns, relational schemata are used in defining genes of network activity. As discussed in Section 1.4, relational patterns can be defined as equal/not-equal relationships between fields of different network packets. This is implemented in Section 4.1, where a 0 bit represents a not-equal gene, and a 1 bit represents an equal gene. This implementation is expanded upon to include less-than, and greater-than, and equal relationships, as discussed in Chapters III and IV. Here, multiple fields of packets (two packets, two fields, at first) are compared, with less-than, greater-than, and equal relationships between two fields represented by a 01, 10, and 11 bit strings in the gene.

1.6.2 Develop Relational Chromosomes. After genes can be built with the relationship mechanism mentioned above, the genes are used as cyber-genetic building blocks to form chromosomes that represent self. Creation of chromosomes is achieved by using a multi-dimensional matrix to hold all combinations of field relationships (genes), with each cell in the matrix housing multiple chromosomes. These chromosomes are generated deterministically by constructing the set of all gene combinations encountered during training (with respect to the number of fields that are selected to be compared). For the purposes of this research, relationship patterns between two, three, and four fields of the total n fields will be examined.

1.6.3 Generate Antibodies. If antibodies are explicitly defined, they might be generated in much the same way that chromosomes are generated. That is to say, antibodies are made up of the same relational genes as chromosomes, with the exception that antibodies represent non-self, whereas chromosomes represent self. Therefore, after self is defined via chromosomes, antibodies can easily be generated deterministically by examining all possible antibodies, and discarding those that match self (chromosomes). This process of discarding antibodies that match self is called “negative selection.” After negative selection, only antibodies that represent non-self (non-normal network activity, such

as intrusions) will remain. These antibodies can then be used in a CIS for the detection of network attacks.

1.6.4 Construct Training Sets. After the methods for defining genes, chromosomes, and antibodies have been defined, these methods can be used with actual data to instantiate the model. Sets of normal (self) chromosomes are constructed by training the chromosomes on attack-free network activity. In a perfect world, the totality of these chromosomes represent the totality of normal network activity. This is loosely analogous to the chromosomes in our body representing an individual organism (through the instructions that they carry). These self-chromosomes can then be used in reducing and refining the (non-self) antibodies during a negative selection process.

1.6.5 Construct Testing Sets. Testing sets are used to validate the function of the antibodies in detecting network intrusions. These sets consist of self (attack-free activity), non-self (network attack activity), and sets of intermixed self and non-self. Once chromosomes are generated with the use of training sets, the testing sets are inspected for patterns of activity that do not match those chromosomes. Since chromosomes represent self, packet relationships in the testing set that do not match these chromosomes will represent non-self, possibly an intrusion.

1.6.6 Detection Tests. Tests are run in a comparison fashion to determine how the number of fields compared affect speed and effectiveness of the relational multiple-packet sensors, as well as giving an indication of the level of false positive, true positives, false negatives, and true negatives encountered by this system. In order to obtain a diverse and repeatable testing set, some of the data used was obtained from the 1999 Defense Advanced Research Projects Agency (DARPA) intrusion detection evaluation conducted at the Massachusetts Institute of Technology (MIT) Lincoln Laboratories [LLab99].

1.7 Scope and Limitations

There is a plethora of protocols with which an attacker may choose to attack his victim. This study is limited to the Transmission Control Protocol (TCP), the User Data-

gram Protocol (UDP), and the Internet Control Message Protocol (ICMP), as this makes up the majority of all current network activity. As these protocols ride on top of the Internet Protocol (IP) and use MAC addresses in communication, some IP fields as well as MAC address fields are also used. Additionally, this research addresses only attacks that can be detected without regard to packet payload, or the data portion of the packet. While it is true that there are a vast number of attacks that cannot be detected without the examination of packet payload, this problem is currently beyond the scope of this detection strategy. Currently, signature-based ID is the primary means of detection with regard to attack payload, and it will likely remain so for some time.

Each network packet has a variety of fields. However, it is not practical or necessarily meaningful to compare a relationship between a given field of one packet, and a different field of another. For instance, it is very unlikely that there is an equal/not equal relationship between the source IP address field in one packet and the Time To Live (TTL) field in another that would provide any useful information. Hence, this research considers only relationships between similar fields of a network packet.

As the number of relationships between all possible fields of all possible packets is intractable, some smaller number needs to be addressed. Because this research is a proof of concept with respect to relational schemata in intrusion detection, it is limited to relationships between two packets at a time. With two packets comparing any two fields, this yields k^2 possible relationships, with k being the number of fields compared. However, since the combinations of the fields are an *AND* relationship, and since (A and B) yields the same relationship information as (B and A), this number can be reduced to $\frac{k^2}{2}$ relationships. It should be kept in mind that if this research is expanded beyond the simple intersection of equal, greater-than, or less-than relationships, the search space might not be reducible by half as is done here. For example; there are many types of relationships, from nominal, such as equal/not equal, to ordinal, such as greater-than/less-than or equal to, to quantitative, such as $x = y * 2$. The intersection of these relationships is not the only method of comparison. If unions and intersections are defined through formulas not to be bijective, more interesting results may be attainable. This research examines the

nominal equal, and ordinal greater-than and less-than relationships, as this conveniently maps to the binary representation used in the CIS chromosomes.

1.8 Roadmap

This chapter provided an introduction to intrusion detection. It addressed why ID is needed to combat the current threats to DoD computer systems and networks. Also discussed were the differences between host-based and network-based ID, and the differences between single-packet and multiple-packet attacks. How a Computer Immune System can be used in conjunction with relational schemata to detect network intrusions was also addressed. This information is covered in greater detail in the following chapters.

Specifically, Chapter II covers related work, detailing what has been done in the area of CIS use in network intrusion detection, and how this research differs from previously existing work. Chapter III deals with the top-level implementation of the research, elaborating on the approach that is used in the rest of the paper. Chapter IV delves into the implementation of this research in greater depth than in Chapter III, addressing how the actual relational CIS is coded in different versions of code and why. Chapter V discusses experiments, and provides statistical data along with results. Finally, Chapter VI provides conclusions to this research, along with presenting possible avenues of future work in this arena.

II. Literature Review

2.1 Intrusion Detection

A computer or network intrusion can be categorized as an attack that is considered to be successful from the opinion of the victim, meaning that the victim has suffered some loss or consequence [Alle00]. An intrusion differs from an attack, in that an attack may be unsuccessfully conducted. Therefore, by some definitions, an attack is not necessarily an intrusion, although an intrusion is certainly an attack. This research considers intrusion detection to be a mechanism that strives to detect not only intrusions, but also attacks, successful or not.

An example of this is detection of scans and probes. Scans are typically used in an attempt to map a network or search for victim hosts, such as scanning a network to find which addresses belong to actual computers [Stan00]. An attacker conducts probes to search for vulnerabilities in hosts that may be exploited. The lines of distinction are blurred, however, as probes and scans are often conducted simultaneously, so they are often referred to as one in the same. Even further clouding the distinction is that an attacker may use scans or probes simply to flood an intrusion detection system, not to actually map a network or find a vulnerability to exploit. Conversely, a scan may be conducted for legitimate reasons, such as a web search engine, or to discover publicly available resources [Bace01]. Ultimately, many events will need to be examined by a system analyst before the actual activity is classified.

2.2 Intrusion Detection Systems

The types and methods of intrusion detection (ID) mentioned in Chapter I are often combined into one product to provide more than one layer of defense. For example, RealSecure, an IDS developed by Internet Security Systems, provides both host-based and network-based ID. Its host-based ID sensor analyzes system logs, while the network-based sensor monitors network packets in search of an attack [ISS00]. Similarly, EMERALD, the Event Monitoring Enabling Responses to Anomalous Live Disturbances IDS developed by SRI International, uses both anomaly detection (in the form of statistical profiling) and

signature-based detection, striving to develop a scalable IDS that can detect attacks in real time [Porr97].

This “blurring of lines” makes comparison of ID technologies and the comparison of IDSs themselves a difficult matter. To further complicate matters, there are other methods of computer security that have not been covered, but are often called IDSs. These methods include honeypots, fishbowls, and padded cells, which strive to divert an attacker from a real system into a system where the attacker can be controlled or monitored, perhaps gaining insight to the methods and motivations of a hacker to avoid further hacking. Some would even argue that somewhat unconventional means such as micro-cameras constitute intrusion detection technologies [Glob01].

As the realm of ID techniques and systems span such a broad area of concern, this research will not attempt to cover all the differences between each of the efforts, rather it focuses on a subset of that area; that subset being network intrusion detection. This area is concentrated upon not only because network IDSs make up the majority of commercial intrusion detection systems, but also because network IDSs have several advantages over other IDSs, as discussed in Chapter I.

A few of the more well-know NIDSs that use signature detection, also called misuse detection, include Snort, Emerald, Bro, NetProwler, Cisco IDS (formerly known as NetRanger), and ASIM [Roes01, Porr97, Paxs98, Axen01, Cisc01, Alle00]. These systems match packets to previously defined signatures of known malicious activity. An example of the types of signature these IDSs use to detect an attack is shown in Figure 2.1.

```
alert tcp any any -> $HOME_NET 111 (content:"|00 01 86 a5|";  
\msg: "mountd access";)
```

Figure 2.1 Example Intrusion Detection Signature

The Snort signature shown in Figure 2.1 detects the *mountd* buffer overflow attack, which allows an entity to gain administrative access to an NFS server [Roes01a, Cert98]. Here, the Snort IDS will alert on any Transmission Control Protocol (TCP) packet bound for port 111, the portmapper port, with the contents “00 01 86 a5”.

This type of signature matching is typical of signature-based network IDSs, matching network packets against a database of known attacks. However, utilizing these canned signatures does not allow an IDS to detect unknown attacks, or even attacks which deviate slightly from the stored signature. This would leave a gaping hole in the area of intrusion detection, were it not for anomaly detection.

Despite the fact that anomaly-based ID is still in its infancy, many companies are beginning to incorporate it into their products. As the benefits of anomaly detection become more well known, and as its high number of false positives are mitigated with checks and balances and user-interactive components like the one addressed by this research, more and more security products will likely include anomaly detection engines. Current IDSs that contain both signature-based and anomaly detection modules include EMERALD, NetStat, CMDS, RealSecure and NFR [Porr97, Kemm01, Proc01, ISS00, NFRS01].

2.3 Methodologies in Anomaly Detection

There are several forms of network anomaly detection. By definition, an anomaly detector is any mechanism that strives to categorize normal network activity and detect intrusions as events that constitute abnormal activity. The primary components of anomaly detection are the methods used to define normal activity and the methods used to determine abnormal activity. Most often, IDSs strive to categorize an activity into one set, with the second set being the compliment of the first. Some of the methodologies used in anomaly detection are neural networks, data mining, data fusion, Genetic Algorithms (GAs), and immune system modeling.

2.3.1 Neural Networks. Neural networks offer several advantages in ID. Neural networks can be fast, allowing the intrusion to be detected before much damage has been done. Also, neural networks offer a predictive capability of attacks due to their probabilistic nature as natural generalizers. Perhaps most importantly, however, neural networks offer the ability to “learn” normal and non-normal network traffic, allowing them the possible potential to detect unknown attacks [Deba92, Forr98, Rhod00, Ghos99a].

Neural networks also have some significant disadvantages, however. First, they require vast amounts of data to train [Forr98]. To accurately detect intrusions, they have to be trained on large amounts of data, which is often difficult to obtain. Because of the vast amount of data required in training, training is not conducted quickly, as detection may be. Second, and most importantly, neural networks offer only a “black box” capability in detection; meaning that “the connection weights and transfer functions of various nodes are usually frozen after the network has achieved an acceptable level of success in identification of events” [Forr98]. Therefore, the accuracy of the system is often unknown [Forr98]. Not knowing the accuracy of the IDS is compounded by the problem that many organizations blindly trust their IDS. This may establish a false sense of security in the IDS, making detection of an intrusion even *less* likely.

2.3.2 Data Mining. The majority of work with using data mining in intrusion detection is conducted at Columbia University [LeeW99, LeeW99a, LeeW00, LeeW00a]. Their work uses predominantly *Classification*, *Link Analysis*, and *Sequence Analysis* as techniques for data mining, and incorporates both misuse and anomaly detection. *Classification* maps data into one of several pre-defined categories, *Link Analysis* determines relations between fields of the database, and *Sequence Analysis* models sequence patterns [LeeW98].

Like neural networks, data mining also suffers from several drawbacks. First, data mining of anomalous activity tends to produce large numbers of false positives. Second, it involves vast amounts of computing power, which can make data mining inefficient. And third, it involves vast amounts of training data and is significantly more complex than traditional means, making it somewhat unusable. In fact, Dr. Lee, a well-respected scientist now working at the Georgia Institute of Technology, stated that “successful data-mining techniques are themselves not enough to create deployable IDSs” [LeeW00b].

2.3.3 Genetic Algorithms. Genetic Algorithms (GAs) are the most widely used and well-known form of Evolutionary Algorithms (EAs), which are algorithms that are inspired by evolution [Mich00]. GAs encode a potential solution to a problem in a “chromosome-like” data structure, applying recombination and/or mutation operators to

these structures to preserve critical information. GAs are often used in an attempt to find maximum or minimum values in a search space that is too large to search deterministically [Whit94].

With GAs, variables that represent parameters are first discretized in an *a priori* fashion, with the range corresponding to some power of two in order to efficiently use a binary representation. In GAs, most chromosomes are generally larger than 30 bits, as anything smaller can normally be enumerated exhaustively. The GA process is as follows [Whit94]:

2.3.3.1 Initial Population Creation. An initial “population” of chromosomes is first chosen. This is typically performed in a random manner, although if information about the sought solution exists beforehand, this may be incorporated to bootstrap the initial population. After creation each chromosome is evaluated and assigned a fitness value.

2.3.3.2 Selection. Selection is applied to the current population to produce an intermediate population. Common types of selection include “tournament” selection and “roulette wheel” selection.

2.3.3.3 Recombination/Mutation. Recombination and/or mutation is then applied to the intermediate population. In recombination, sometimes called “crossover,” two members of the population are combined to produce a child. With mutation, one member is simply mutated to produce a child.

2.3.3.4 Next Population. Lastly, child or parent members of the intermediate population are chosen in a “with replacement” or “without replacement” manner. These members will make up the next population in the GA. This completes one *generation* of the GA. The algorithm then loops back to 2.3.3.2 until some termination criterion is reached, such as finding a certain value, or reaching a specific generation or time.

Since GAs have been shown to effectively search an innumerable search space, more and more research is being conducted to their applicability in intrusion detection [Me98,

Neri00, Will01, Brid00]. For example, in the Genetic Algorithm for Simplified Security Audit Trails Analysis, (GASSATA), Mé used GAs to search for 24 attacks in 28 events, making his search space a 24 by 28 matrix [Me98]. Neri’s GA intrusion detection mechanism, REGAL, took an input set of data and, after training with the GA, output a set of symbolic classification rules [Neri00]. Williams used GAs in his algorithm in the affinity maturation process, which is discussed more in Section 2.4.7 [Will01].

2.4 Artificial Immune Systems

An Artificial Immune System (AIS), sometimes referred to as a Computer Immune System (CIS) when applied to the computer field, models biological processes in a digital manner. One of the first uses of a AIS in computer security was in the virus detection arena [Keph97, Keph97a, Dasg98, Forr97, Lamo99]. Here, the body’s immune system was modeled to overcome the static virus detection capabilities of signature-based detection models. Just as the Biological Immune System (BIS) adaptively counters new viral threats to the body, the AIS may adaptively counter new threats to the computer. In order to understand many of the functions of an AIS and how they relate to a BIS, some biological terminology must be addressed, along with references of how it relates to the computer. Some excellent documents covering many aspects of an AIS can be found in [Hofm99, Hofm99a, Harm00] and [Will01], and those details shall not be covered again here. What is covered are the essentials required to understand the context of this thesis. However, the aforementioned documents are suggested reading for a more comprehensive understanding of the subject matter, and are well worth the time.

2.4.1 Pathogen. A pathogen, a term given to encompass disease-causing microorganisms such as viruses, bacteria, and fungi, are parasitic organisms [Jane97]. These type of infectious agents must be detected and eliminated from the body. The problem of detecting pathogens is often referred to as distinguishing “self” from “non-self,” which correlates to the body and pathogens, respectively [Hofm99].

2.4.2 Immunity. Immunity refers to the mechanisms the body uses to protect itself from foreign bodies (pathogens). The human immune system is indirectly capable

of recognizing cells and proteins that make up the body. When components not of the body are detected, defense mechanisms strive to protect the body from these components. Immunity can be divided into *innate* immunity, and *adaptive* immunity.

2.4.2.1 Innate Immunity. Innate immunity refers to those means of protection that are present from birth, such as skin and mucous membranes, coughing and sneezing reflexes, and a vast array of specialized cells that seek out and destroy invaders. Components of the innate immune system may affect pathogens directly, or enhance the effectiveness of other immune system reactions to them. Examples are a phagocyte cell engulfing and digesting an invading microbe, or a T-cell punching a hole in the cellular membrane of the invader, causing water to rush in and explode it. The innate immune system “provides a rapid first line of defense, to keep early infections in check, giving the adaptive immune system time to build up a more specific response” [Hofm99a].

2.4.2.2 Adaptive Immunity. Innate immunity is not enough to protect the body from invaders, as microbes evolve rapidly in an attempt to overcome the body’s originally programmed defense mechanisms. Adaptive immunity allows for the body to adaptively detect and destroy invading specific pathogens. This is exactly why the immunity model is emulated in cyber-defense. Just as the body can detect and respond to foreign and adaptive intrusions, so can the computer.

2.4.3 Gene. Gene segments carry information for the production of a unique protein or enzyme, and govern both the structure, and the metabolic functions of a cell [CEnc00]. In this way they govern the makeup of the entire organism at a very low level. Genes are similarly one of the lowest levels of abstraction of the CIS in this research. Much like genes are used by Kim and Bentley to construct antibodies [Kim01a], genes of this research are used as building blocks to construct both chromosomes and antibodies.

2.4.4 Chromosome. According to Columbia Electronic Encyclopedia, sixth edition, a chromosome is “the structural carrier of hereditary characteristics, found in the nucleus of every cell” Just as a BIS chromosome carries hereditary characteristics of the

species as a linear arrangement of genes, a cyber-chromosome can be considered to carry information about the characteristics of the network or computer system.

2.4.5 Antibody. In the body, antibodies, also called immunoglobulin, are proteins produced by plasma cells, or B-cells (a type of lymphocyte, white blood cell). Upon binding to an antigen via Y-shaped pathogen binding sites, antibodies stimulate the B-cell to produce copies of the antibody. These antibodies, which are all designed to detect the antigen, mark the antigen for destruction by a phagocyte and the complement system. The antibody is shown in Figure 2.2 along with a visual representation of the immune system, taken from [Hofm99]. Just as an antibody in the BIS is designed to detect a pathogen, an antibody in the ID AIS is designed to detect an attack.

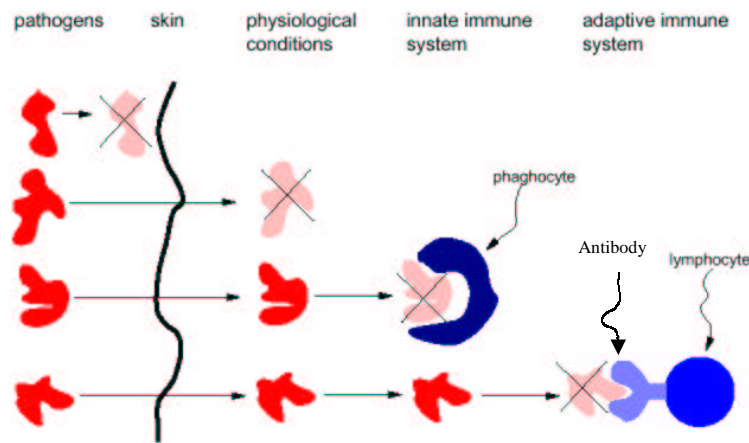


Figure 2.2 Multilayered Immune System [Hofm99]

2.4.6 Negative Selection. The antibodies that mark pathogens for destruction must be tolerant of self, lest they contribute in the destruction of the host body vice the pathogen. However, B-cells will often match self after they have undergone hypermutation. These hypermutated cells have somewhat different binding criteria than the original B-cell, and may bind to self, which is undesirable. Negative selection, sometimes called clonal deletion, is the process of censoring (killing) an antibody that binds to self. In intrusion detection, negative selection has come to mean the activity that kills the entities that detect self. Just as biological antibodies that have undergone negative selection should

only detect pathogens (and not self), digital antibodies used in intrusion detection, should only detect non-self (normal network or computer activity).

2.4.7 Affinity Maturation. The strength of the bond between the receptor of the antibody and the epitope (locations on the surface of the pathogen) is termed *affinity*. In the body, “both receptors and epitopes have complicated three-dimensional structures that are electrically charged” [Hofm99]. The more complementary the structure and charge of the receptor and the epitope, the higher the affinity, and the more likely binding will occur. Affinity maturation is the term for the Darwinian process of selecting those antibodies with higher affinity to be replicated more often.

2.4.8 Costimulation. The costimulation process in the body is acutely complex, with many details beyond the scope or intent of this research. However, a somewhat general definition may be sufficient to understand its function in the body, and its usefulness in ID. After B-cells are hypermutated in the body, they may bind to self. To block this action, a B-cell must receive two distinct costimulation signals to be activated. One signal is received when binding to the pathogen. This can be thought of as the *bind to something* signal. Another signal is from a Th-cell, which must also bind to the same pathogen. Since Th-cells are trained not to match self in the thymus, they should not bind to self. When a Th-cell binds to a pathogen, it will send out a signal for any B-cells close to it. This signal can be thought of as the *pathogen* signal. Therefore, an active B-cell will have received both a *bound to something* signal, and a *pathogen* signal, telling the B-cell that it is bound to a pathogen and not self. This process is shown in Figure 2.3 below.

2.5 Intrusion Detection Via a Computer Immune Systems

Computer immune systems have been demonstrated as successful in the area of intrusion detection [Dasg98, Soma97, Kim01, Kim01a]. Much of this research is continuing at the Air Force Institute of Technology (AFIT), with a large portion centering around the antibody as this is the central detection mechanism. This is why the antibody is discussed in some detail here, even though it is only defined implicitly with this research.

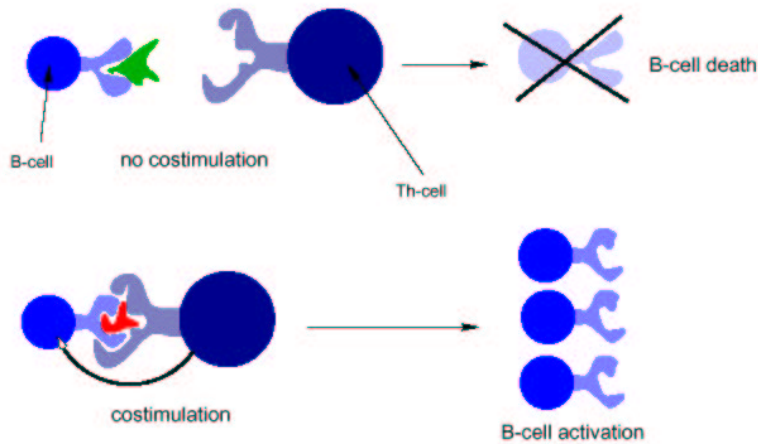


Figure 2.3 Costimulation Process

B-cells that are not costimulated die. Those that are costimulated reproduce. [Hofm99]

The antibody lifecycle is shown in Figure 2.4, with antibodies being generated to not match self, then going through affinity maturation to define non-self as thoroughly as possible. They then set about trying to detect attacks, and if they do not detect anything in a specified lifetime, they die and are replaced with new antibodies [Will01a]. This model will be utilized to the extent that the relational multiple-packet antibodies are generated deterministically and implicitly as the complement to all chromosomes. As a complement set, the antibodies, by definition of complement, undergo the negative selection process. Affinity maturation is not done because the list of antibodies is complete (since they are generated deterministically). The detection process is an off-line process, simulating how the actual IDS would work had it actually been on line. Due to time constraints, explicitly defining antibodies and the lifetime criterion of those antibodies were not addressed in this research, but may be explored at a later date.

2.6 Multiple-Packet Intrusion Detection

Most of the current intrusion detection mechanisms rely on single-packet signatures to detect attacks. Even those signatures that detect scans or probes rely solely on a simple counter mechanism to detect the scan or probe. Experts recognize this shortfall, as Stephen Northcutt, a well-respected author on computer security stated, “until some

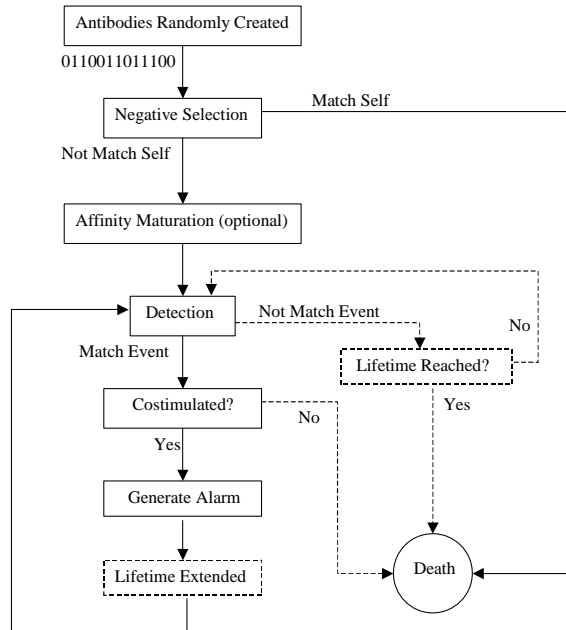


Figure 2.4 CIS Antibody Lifecycle

brilliant researcher comes up with a better technique, scan detection will boil down to testing for x events of interest across a y -sized time window” [Nort00].

Not all multiple packet attacks are scans or probes, however. Amoroso describes a “temporal” intrusion model, and defined an intrusion as “a sequence of related actions by a malicious adversary that results in the occurrence of unauthorized security threats” [Amor99]. This definition highlights that an intrusion is a *sequence* of actions, meaning more than one action, and likely, more than one packet. These multiple-packet intrusions, whether they be reconnaissance techniques such as scans and probes, or full-fledged intrusions such as Amoroso was describing need to be detected.

ARP and IP spoofing are other forms of multiple-packet attacks, as mentioned in Chapter I. In fact, these types of attacks have become so dangerous that SANS Institute has published the following in a recent alert [SANS01a].

Once upon a time, administrators didn’t have to worry about ARP cache poisoning and redirection because such assaults required attackers to gain direct access to the network segment that hosted the target systems. That segment was protected by physical boundaries. With the surge of wireless network installations on the rise, however, the door to ARP redirection and connection

hijacking has been opened to anyone with an off-the-shelf wireless network card. Essentially, a remote attacker who can get within range of an access point could potentially poison client ARP caches into believing they are the default gateway. The result? All traffic routes through the attacker.

This form of attack is demonstrated in Chapter V to be detectable with the relational CIS structure proposed via this research.

2.7 Relational Schemata

Examining how packets of a network relate to one another as a form of intrusion detection is nothing new. In fact, it happens thousands of times a day when network analysts worldwide sift through network data while monitoring their networks. Often these analysts will reconstruct entire sessions of network activity and analyze each packet in the session to find out what the suspect was up to. In effect, they take each packet and compare it to another to determine the “big picture” view of the network activity.

This research attempts to somewhat automate that process of multiple-packet inspection by using relational schemata to examine how network packets interrelate. In this context, relational schemata are a means of detecting the above mentioned ARP and IP spoof attacks, as well as other attacks that are detectable when viewed in a “big picture” context.

The MitM attack shown in Chapter I has been shown to be detectable with equal/not-equal relationships. These relationships can assist in detecting anomalous activity, where a transmitting node appears to be sending conflicting packets. This may go unnoticed in network traffic by a single-packet IDS, as each packet examined individually may represent a perfectly normal network packet. Only when examined in the broader context, in the context of the other packets traversing the network, may this type of attack be detected. Relational schemata are perfect for this function, as by their very definition they incorporate more than one packet.

The relational representations described in Chapter III form relational multiple-packet antibodies for multiple-packet ID. Using the CIS paradigm, this thesis shows that relational patterns can be used to detect some multiple-packet attacks. This may increase

the security capability of current IDSs, and strengthen an important layer of the defensive structure.

III. High Level Design and Methodology

The real problem is in the hearts and minds of men. It is easier to denature plutonium than to denature the evil spirit of man.

- Albert Einstein

The previous chapters gave an overview of why a multiple-packet attack (MPA) detection tool is necessary, and what has been done in the recent past with regard to intrusion detection in this area. This chapter explains the methodology used in developing an MPA detection tool using a computer immune system. Section 3.1 discusses the goals of this research as exploring a new intrusion detection model, and developing a multiple-packet visualization tool. The remaining sections outline, at a conceptual level, how these goals are pursued. In pursuing a relational intrusion detection model, more than one method of relational mapping is explored. Section 3.2 addresses the definition of packet relational attributes used in the main thrust of this research, while Section 3.3 discusses another method of developing relational attributes which may be more costly, yet more effective. Section 3.4 outlines the packet protocols and values used in this research, and Section 3.5 discusses the size of the relational space and its intractability. Finally, Section 3.6 discusses self and non-self domains, Section 3.7 discusses the use of a CIS with relational schemata, and Section 3.8 discusses one potentially instrumental method for multiple-packet visualization.

3.1 Goals

The main goal of this research is to detect network MPAs, or attacks that may be detected when viewed in the context of more than a single network packet. To that end, relationships are drawn between values of similar packet fields. These relationships are then used to represent self and non-self domains. An important aspect of this research is that the actual values themselves are not kept when constructing a representation, only the relationships between those values are kept. A value is only inspected when determining the relationship between it and another value. Finally, these self and non-self representations are used to detect potential network MPAs.

A secondary goal of this research is to develop a user-interactive, graphical method of multiple-packet visualization. The current method of packet analysis relies heavily on programs such as TCPDump and Ethereal, which present data in a textual format, as shown in Figure 3.1. While these types of tools are extremely valuable in examining each packet individually, they do not perform well at providing the “big picture” view that is often necessary when working with multiple packets. This research strives to demonstrate a method of visually inspecting the values of multiple packets simultaneously, allowing the analyst to determine packet relationships at a glance.

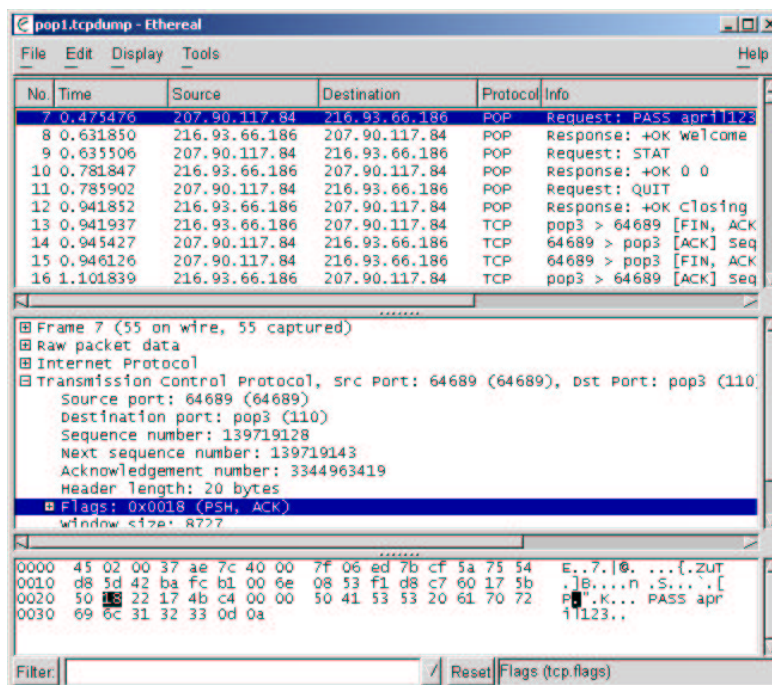


Figure 3.1 Typical Packet Textual Display

3.2 Relational Design

As stated in Chapter I, there are several types of relationships that may occur between packet fields, from nominal to ordinal to quantitative. This research began with examining simple nominal Boolean equal/not-equal relationships between packet fields as a method of ID. When this method of detection was tested on the MitM attack described on page 1-5, it worked only initially. That is, the MitM attack in question was detected

until it was discovered that there is a circumstance where the relationships mentioned on page 1-9 may actually occur. These relationships occur normally if a client sends its data and then decides to reset the connection—a legitimate action. This is contrasted to the attack, where the spoofed client reset the connection and then the real client sent its data. Therefore, packet order matters.

Since the order of the packets matters, the relationships between packet fields were expanded beyond simple nominal equal/not-equal relationships, to include ordinal greater-than and less-than relationships in place of not-equal relationships. In a stroke of genius my Committee Chair pointed out that this provides a means of temporal packet ordering. While the addition of ordinal relationships does increase the complexity of the problem slightly, it is well worth the cost of determining packet order.

With this information in mind, the language that represents the relational schemata can be defined. If a packet, P , has n fields, each of which has a value labeled V_i , then the following definitions can be stated:

3.2.1 Packet. A network packet, P , is defined by the values it contains. This is given by the n -tuple:

$$P = (V_0, V_1, V_2, \dots, V_{n-1})$$

3.2.2 Field Value. Considering the IP version 4 and Ethernet fields of this research, the values of any packet field (not considering the packet payload) can be any number from zero to 2^{48} [Stev99]. However, network packet values range from 0 to 1 for the TCP flags, to 2^8 for the IP TTL, to 2^{48} , for Media Access Control (MAC) address fields. If γ is defined as the number of bits necessary to define the packet field under considerations, the numerical value of any field, V_i , is represented by:

$$V_i \in \{0, 1, \dots, 2^\gamma\}$$

For example, TCP flag values range from zero to one, so γ would be zero. However, in a MAC address, the values range from zero to 2^{48} , so γ would be 48. Therefore, the size of γ_i determines the size of the value V_i .

3.2.3 *Gene.* The relational gene primarily used in this research is defined by the equal, less-than, or greater-than relationships drawn between similar fields of two packets. Since the gene has three possible relationships, it takes two bits to represent this gene. Thus, for this research, a binary 01 represents a less-than relationship, 10 represents a greater-than relationship, and 11 represents an equal relationship. Using a relationship, R, with n fields per packet, the relational genes are defined in the following manner. Given that $0 \leq i < n$ and $1 \leq j < k$, and V_i^j is the i th element of packet P_j and V_i^k is the i th element of packet P_k , then if

$$R(V_i^j, V_i^k) \quad \text{is} < \quad \text{then} \quad G_i = 01$$

$$R(V_i^j, V_i^k) \quad \text{is} > \quad \text{then} \quad G_i = 10$$

$$R(V_i^j, V_i^k) \quad \text{is} = \quad \text{then} \quad G_i = 11$$

Example

Given:

Packet 1, $P_1 = (1,0,0,345645,0)$

Packet 2, $P_2 = (0,0,1,345645,23423)$

and

$$G_i = R(V_i^1, V_i^2)$$

where R is a less-than, greater-than, or equal relationship mapping, this gives

$$G_0 = 10 (1 > 0)$$

$$G_1 = 11 (0 = 0)$$

$$G_2 = 01 (0 < 1)$$

$$G_3 = 11 (345645 = 345645)$$

$$G_4 = 01 (0 < 23423)$$

3.2.4 *Chromosome.* A relational chromosome, C, is defined by the m-tuple of the genes it contains, where m is the number of fields compared. This is given by

$$C = (G^1, G^2, \dots, G^m)$$

If m were assigned a value of two, then each chromosome would be the concatenation of two genes. If the genes on the bottom of page 3-4 are used to define the chromosomes, the possible two-gene-chromosomes generated are

$$(G_o, G_1)(G_o, G_2)(G_o, G_3)(G_o, G_4)$$

$$(G_1, G_2)(G_1, G_3)(G_1, G_4)$$

$$(G_2, G_3)(G_2, G_4)$$

$$(G_3, G_4)$$

It can be easily seen how these cyber-chromosomes might be stored in a matrix data structure, and indeed, this research does just that with m equal to two, three and four fields, as is discussed in Section 3.6.

3.2.5 *Antibody.* Whereas relational chromosomes serve to define self, relational antibodies serve to define non-self. Since there is a fixed number of packet fields, there is also a fixed amount of relational chromosome and antibody space, with the antibody space constituting the complement of the relational chromosome space. Hence, antibodies can be generated deterministically to fill all space not occupied by chromosomes. Because the chromosomes are stored in the matrix structure mentioned above, generating antibodies is a simple matter of checking each cell of the matrix, and generating an antibody for each missing chromosome in that cell. In this research, chromosomes are generated during training, and the complement of this set, the conceptual antibody set, is used for intrusion detection.

3.3 Crosschecking Genes

This research also delves into another form of relational gene construction, named “crosschecking.” In this gene structure the subscript of the packet values compared do not remain the same when performing the relational comparison. Instead, using a relationship, R , with n fields per packet, the crosschecking relational genes are defined in the following manner. Given that $0 \leq i < n$, $0 \leq h < n$, $1 \leq j < k$, V_i^j is the i th element of packet P_j , and V_h^k is the h th element of packet P_k , then if

$$R(V_i^j, V_h^k) \quad \text{is} < \quad \text{then} \quad G_{(i,h)} = 01$$

$$R(V_i^j, V_h^k) \quad \text{is} > \quad \text{then} \quad G_{(i,h)} = 10$$

$$R(V_i^j, V_h^k) \quad \text{is} = \quad \text{then} \quad G_{(i,h)} = 11$$

Here the packet fields that are compared are not necessarily the same fields. This is where the distinction mentioned earlier of *similar* packet fields is important. Since comparing the TTL field to the SYN flag field will likely provide no useful information, the acceptable values of (i, h) over $G_{(i,h)}$ are limited to *similar* fields. For example, IP source and IP destination fields being similar would provide only IP source/IP source, IP source/IP destination, IP destination/IP source, and IP destination/IP destination relationships and ignore any other packet field comparisons.

This research explores crosschecking in an attempt to determine its potential use and value in catching such attacks as the DSniff MitM attack addressed in Chapter V, and the “Land” attack conducted in the 1999 DARPA intrusion detection evaluation. This is because some attacks may not be detectable with the direct field-to-field gene used in the majority of this research. In the “Land” attack, for example, the packet source address is forged to be the same as the packet destination address in an attempt to conduct a denial of service attack against the victim [LLab99]. Since the most obvious way of detecting this attack would be to compare the source address to the destination address, crosschecking is implemented to study its usefulness.

It should be noted, however, that since the gene structure changes in the crosschecking model, so too does the chromosome makeup, making it much more computationally complex. This is discussed more in Section 5.5 on page 5-10.

3.4 *Packet Values*

A question that needs to be addressed is which packet fields are valuable in detecting an intrusion when building a CIS sensor. The packet fields utilized by Williams seemed promising [Will01]. Therefore, those fields are used here with only slight modifications to improve the efficiency of relational comparisons, reduce redundancy, or include additional fields. For instance, Williams split the source and destination IP address into its separate quads, but an IP address is represented as merely a single field here, as comparing the relationship of a single address seems more useful than comparing the relationships between four separate fields that make up a single address. Additionally, the IP fragment fields were split up into three separate fields, as the comparison of the *unused IP flag* field (not specified in the protocol, but an attacker may intentionally or unintentionally set the bit), the *do not fragment* field, and *more fragment* field should be examined separately. The reason for examining these fields separately is that an attacker might set all of the IP fragment fields at once, similar to a Christmas Tree attack, in order to take advantage of how different operating systems may respond differently to this abnormal situation. This sort of packet manipulation has been shown to be capable of conducting insertion and evasion attacks [Ptac98]. Also, the MAC address fields were added to detect ARP and IP spoofing attacks. Table 3.1 shows the fields utilized in this research, along with the number of bits necessary to represent these fields.

3.5 *Relational Space*

Each network packet has a given number of fields, and the relationship among these fields is what makes up the genes in this research. However, as discussed above, it is not practical to compare relationships between a given type of field of one packet, and a *different* type of field of another. Hence, this research shall consider only relationships between similar types of fields of network packets. The word “similar” takes on different

Table 3.1 Packet Fields Used in Gene Construction

| Field Name | Field Range |
|----------------------------------------------------------|--------------|
| Fields Common to all packets | |
| IP Identification Number | $0 - 2^{16}$ |
| IP Time To Live | $0 - 2^8$ |
| IP Don't Fragment | $0 - 2^0$ |
| IP More Fragment | $0 - 2^0$ |
| Unused IP Flag (Not Used in IP Specification) | $0 - 2^0$ |
| IP Fragment Offset | $0 - 2^{13}$ |
| IP Header Length | $0 - 2^4$ |
| IP Packet Length | $0 - 2^{16}$ |
| IP Source Address | $0 - 2^{32}$ |
| Source MAC Address | $0 - 2^{48}$ |
| IP Destination Address | $0 - 2^{32}$ |
| Destination MAC Address | $0 - 2^{48}$ |
| Total Packet Length (IP Packet Length + Ethernet Header) | $0 - 2^{17}$ |
| TCP-Only Fields | |
| TCP Source Port | $0 - 2^{16}$ |
| TCP Destination Port | $0 - 2^{16}$ |
| TCP Sequence Number | $0 - 2^{32}$ |
| TCP Next Sequence Number | $0 - 2^{32}$ |
| TCP CWR (Congestion Window Reduced Flag) | $0 - 2^0$ |
| TCP ECHO (Echo Flag) | $0 - 2^0$ |
| TCP URG (Urgent Flag) | $0 - 2^0$ |
| TCP ACK (Acknowledge Flag) | $0 - 2^0$ |
| TCP PUSH (Push Flag) | $0 - 2^0$ |
| TCP RST (Reset Flag) | $0 - 2^0$ |
| TCP SYN (Synchronize Flag) | $0 - 2^0$ |
| TCP FIN (Final Flag) | $0 - 2^0$ |
| TCP Header Length | $0 - 2^4$ |
| UDP-Only Fields | |
| UDP Source Port | $0 - 2^{16}$ |
| UDP Destination Port | $0 - 2^{16}$ |
| UDP Data Length | $0 - 2^{16}$ |
| ICMP-Only Fields | |
| ICMP Code | $0 - 2^8$ |
| ICMP Type | $0 - 2^8$ |
| ICMP Data Length | $0 - 2^{16}$ |

meaning with the normal relational gene than with the crosschecking gene. In this research only the same fields are compared when constructing relational genes, with the exception of the method used in the crosschecking gene structure mentioned above. This means that the space of possible crosschecking genes is much larger than the space of possible normal genes, because the normal genes are actually a subset of the crosschecking genes. The space of the normal gene is discussed below, as the normal relational gene comprises the majority of this research, and as the space of the crosschecking genes depends largely upon

the implementation of which fields are deemed “similar” and compared when crosschecking. It is demonstrated in Section 5.5, however, that the space of the crosschecking gene can quickly become unwieldy.

Since the number of fields of each packet is fixed, the number of fields compared in a normal relational gene does not increase the order of the problem with respect to the number of packets that are compared by more than some constant, k . Therefore, in discussing the order of complexity of the problem this constant is ignored for now to focus on the more important variable, the number of packets. Even ignoring the number of fields, the number of possible relationships between all packets can be calculated to be intractable. For instance, if three packets (labeled 1, 2, and 3) are compared, the relationship patterns that could be compared are:

No relationships: compare single packets: (1), (2), (3)

Two relationships: compare two packets at a time: (1,2), (1,3), (2,1), (2,3), (3,1), (3,2)

Three relationships: (1,2,3), (1,3,2), (2,3,1), (2,1,3), (3,1,2), (3,2,1)

With some thought, it is clear that the total number of relationships possible is the permutations of the number of total packets, n , and the number of packets being considered, b . In the above three-packet scenario, this is the sum of ${}_3P_1, {}_3P_2, {}_3P_3$ or 3, 6, and 6 relationships respectively, for a total of 15 possible relationship patterns. This sum is given by the formula

$$\sum_{b=1}^n {}_n P_b$$

where the relationships of n packets are permuted b at a time. This equals

$$\sum_{b=1}^n \frac{n!}{(n-b)!}$$

which equals

$$n! \sum_{b=1}^n \frac{1}{(n-b)!}$$

As the number of packets compared, b , grows large, the denominator of the summation approaches 1, and since the summation is multiplied by $n!$, the order of comparisons of relationships possible is $O(n!)$. This is definitely intractable. This is the size of all possible relationships between network packets.

3.6 Self/Non-Self Domains

While the number of possible relationships between network packets is intractable, not all or even most of these relationships need to be considered in the search for an attack. For instance, if there are a million packets per day flowing through the network, it is unlikely that the relationships need to be examined for all million packets, as an attack would likely be only a very small subset of these packets. This research considers only the smaller case, that of only two-packet relationships. With n packets, $\frac{n^2-n}{2}$ relationships can be considered (a packet need not be compared to itself, and packets are not compared to previous packets). This is referred to as simply $O(n^2)$ relationships, as it is still of the order n^2 . Examining even $O(n^2)$ packets is considered non-optimal, however, as in peak traffic times the network may be saturated. This can make $O(n^2)$ too large to examine “on the fly” and packets may have to be dropped. Therefore, a “sliding window” concept was used to reduce this space.

Many attack packets are likely not separated by vast numbers of packets, and in fact, are often likely nearby one another in MPAs. Therefore, as a starting point this research searches for attacks by not examining all $O(n^2)$ two-packet relationships, but by using a sliding window of packets, comparing only packets within the window. In limited testing of window sizes from 3 to 30 packets, there was not a large difference noticed in the relationships obtained in large amounts of training data. Therefore, a window size of six packets was selected as the baseline of this research, and one packet is compared to the next five packets throughout the training and testing processes. A six-packet window was selected because this size allows for determining relationships between packets during the sessions of the MitM attack discussed, while being a small enough window to provide efficient implementation. Future research may delve into changing this window size to detect specific attacks which fall outside this window, such as “low and slow” attacks.

The use of a sliding window reduces the number of packet relationships from $O(n^2)$ to approximately $5n$ two-packet relationships, which still $O(n)$, and is much more manageable.

Using the packet relationships constructed above, patterns of “self” and “non-self” can be drawn. Assuming that an attack-free set of n packets is obtainable for training, self can be defined as the $O(n)$ relationships drawn from these packets. Once self is defined, the compliment of this set would be non-attack-free network traffic, or “non-self.” Thus, relationships drawn among this non-self that do not match the relationships among attack-free traffic are considered abnormal. The data set that is used to define self is called training data, whereas the data set that is examined for attacks is called *testing* data.

One problem is that since each packet contains a number of fields of varying sizes, attack-free self cannot be completely defined. First, the data portion of the IP packet, termed the “payload” consists of 256^{65536} possible values [Will01], a number that is clearly intractable and cannot be defined deterministically. As the packet payload is not considered in this research, this problem is ignored. However, even discarding the packet payload, the other fields in the packet are still too numerous to define completely. Using the 32 fields discussed in Section 3.4 the search space can be calculated as follows:

No packet contains all TCP, UDP, and ICMP fields simultaneously, so they need not be considered together. Additionally, the TCP fields dominate the search space with regard to the UDP and ICMP fields. This can be seen in Figure 3.1, where there are a greater number, as well as larger TCP fields. Therefore only the common fields and the TCP fields are considered in the space calculations.

Eleven of these fields have a range of 0 to 1, two have a range of 0 to 15, one has a range of 0 to 255, one has a range of 0 to 8191, four have a range of 0 to 65535, one has a range of 0 to 131071, four have a range of 0 to 4294967295, and two have a range of 0 to 281474976710655. This yields a possible $2^{11} * 16^2 * 256 * 8192 * 65536^4 * 131072 * 4294967296^4 * 281474976710656^2$ or $7.167 * 10^{103}$ possible strings. Even calculating a string every nanosecond (of which no computer is currently capable), it would take $2.27 * 10^{87}$ years to cover all possible packet strings, and this is even before relationships between fields are considered. This amount of time is clearly not practically efficient.

In addition to being unable to completely define self in the first place, self actually changes as new hardware or user activity is encountered, and new protocols and applications are developed. Therefore, even if self could be defined, that definition would be only a temporary definition.

Since self cannot be completely and permanently defined, all possible attacks cannot be detected with any manner of intrusion detection. At best, it is hoped that enough varying layers of detection are put in place so that one layer may catch an attack missed by another. This is the purpose of this research, which is to detect MPAs that go undetected by the current intrusion detection corpus, providing another layer which the attacker must avoid.

3.7 CIS Usage

The current corpus of research in intrusion detection at AFIT focuses on the usage of a CIS in defining self and non-self, as discussed in Chapter II. There have been two CIS implementations used at AFIT in the ID arena. The first of these is Warthog, an implementation conceived by Williams in an effort to detect “low and slow” network attacks [Will01]. The second is Ferret, which is designed to be faster and more efficient than Warthog, yet still perform the CIS functions shown in Figure 2.4 [Will01a]. This research diverts from the Warthog and Ferret implementation to utilize packet relationships, vice simple single packets. This requires a transformation away from the single-packet input and detector parameters of Warthog and Ferret to multiple-packet *relational* detectors, called relational antibodies.

The conceptual antibodies of this research could be used in a larger CIS structure and have life cycles like depicted in Figure 2.4. Here, they are simply static, and do not change once defined. One benefit of this research is that the chromosomes are neatly stored in a logical matrix structure, which could similarly be done with antibodies. This can make examining, understanding, and modifying these antibodies a somewhat easier task. Additionally, storing the relational sensors in x -dimensional matrices adds an aspect of determinism to the stochastic implementation currently underway at AFIT as another layer of defense. This is beneficial for this small-scale implementation, because the number

of possible two-packet, limit-field relationships of this research is small enough to allow for a deterministic search. This allows for a more complete coverage of possible antibodies. When the dimensionality of the sensors gets too great, however, deterministically traversing the search space in real-time may be impractical, as stated in Section 4.5. In higher-dimensional sensors, if on-line processing is a necessity, the completeness of a deterministic algorithm may have to be traded for a faster stochastic algorithm.

3.8 Attack Visualization

One problem with MPA ID is that vast amounts of data must be drudged through in the search for attacks. This is often beyond the capability of the network analyst to do with simple textual-based programs such as Ethereal and TCPDump, which display network traffic a packet at a time. What is needed is a means of visualizing numerous packets simultaneously, allowing for dynamic filtering of packets as appropriate.

Parallel coordinates are one means that allow for the simultaneous comparison of multivariate data, and in fact, Inselberg states that “the special strength of parallel coordinates is in modeling relations” [Inse97]. With parallel coordinates, multiple dimensions are shown, with each dimension depicted on its own axis. As stated, a secondary goal of this research is to develop a visual tool to aid the analyst in examining events that are labeled “attacks” by the IDS to determine if it is a true attack, or a false positive. Therefore, a parallel coordinates system with the same fields used in packet detection in the IDS is developed to assist the network administrator in post-mortem analysis. This is discussed more in Chapter IV.

IV. Low Level Design and Implementation

The implementation of this research progressed through several stages in its evolution. All stages of this research focus on the relationships of multiple fields of two packets at a time. While it is true that there are likely some attacks that must examine relationships between more than two packets to be detected, this research is focused on the number of fields and the method for comparing those fields, and leaves more-than-two-packet detection for future work. As stated in Chapter III, the two-packet comparisons are inspected in a sliding window of six packets in an effort to detect intrusion packets which lie close to one another.

Even considering only relationships between two packets at a time, each packet has a number of fields, and the method of comparison utilized between these fields evolved with this research. As this evolution facilitates understanding of the final product and constitutes a large portion of the research, the versions leading up to the final implementation are also discussed. The low-level implementation details of these versions are discussed in this chapter, and the testing conducted on each version is discussed in Chapter V.

Here is a brief overview of the versions that were implemented; the first version was a simple beta version intended to detect the MitM attack discussed in Chapter I. This version, version 0.0, implemented strictly equal/not-equal relationship genes, and initially detected the MitM attack it was developed to find. However, after extensive testing, it was noticed that the order of the packets flowing through the network mattered, so the equal/not-equal relationship gene was changed to the equal/greater-than/less-than gene discussed on page 3-4. This became version 1.0.

Then, to make the IDS more rigorous and useful, the protocols and fields were expanded to include all 32 fields of Table 4.2. This version, version 2.0, seemed to work very well. In fact, it was so promising that this version became a mainstay component in this research. Version 2.0 seemed to be fast and efficient, but it is likely that there are attacks that require the examination of more than two fields of the two packets in order to be detected. Therefore, this version was expanded beyond the two-gene chromosomes (simple

two-dimensional matrix) to include three- and four-gene chromosomes (three-dimensional and four-dimensional matrices) with versions 3.0 and 4.0.

Finally, during testing concern arose that attacks such as the DSniff MitM attack discussed in Chapter V might be missed by these versions as they simply compare the same fields when creating a gene schemata. Therefore, the crosschecking gene was investigated with versions 2.44 and 2.116 (named after the length of one side of the matrix). Table 4.1 summarizes the main differences between the versions.

| Version | Gene Schemata | Schemata Construction | Dimensionality | # of Fields | Matrix Breakout |
|----------------|----------------------|------------------------------|-----------------------|--------------------|------------------------|
| V 0.0 | ≠ 0 = 1 | Same Fields | 2 | 11 | Fig 4.2 |
| V 1.0 | < 01 > 10 = 11 | Same Fields | 2 | 11 | Appendix A |
| V 2.0 | < 01 > 10 = 11 | Same Fields | 2 | 32 | Appendix B |
| V 3.0 | < 01 > 10 = 11 | Same Fields | 3 | 32 | Table 4.2 |
| V 4.0 | < 01 > 10 = 11 | Same Fields | 4 | 32 | Table 4.2 |
| V 2.44 | < 01 > 10 = 11 | Cross Checked Fields | 2 | 44 | Appendix C |
| V 2.116 | < 01 > 10 = 11 | Cross Checked Fields | 2 | 116 | Appendix D |

Table 4.1 Code Version Overview

4.1 Version 0.0, Equal/Not-Equal Gene

This research started out by attempting to detect the MitM attack discussed in Chapter I. In order to detect this attack, the attack had to be reconstructed to obtain data for testing. Figure 4.1 shows the MitM attack constructed in a telnet session to emulate the attack mentioned in [Jonc95]. This data, extracted from a sniffed TCPCDump file, is limited to the fields that facilitate understanding of the attack in order to reduce distractions. Since the MitM attack discussed in Figure 1.1 addressed only IP and TCP fields, the initial implementation was limited to eleven IP and TCP fields (excluding the packet number, which was used in selecting packets, but not in field to field comparisons).

| # | IPSRC | IPDST | TCPsrcPort | TCPDestPort | TCPSeq# | TCPNxtSeq# | Ack | Push | Reset | Syn | Fin |
|----|-------------|-------------|------------|-------------|------------|------------|-----|------|-------|-----|-----|
| 1 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753888 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 101.10.10.2 | 101.10.10.4 | 23 | 1761 | 212397464 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753889 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753889 | 2057753892 | 1 | 1 | 0 | 0 | 0 |
| 5 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753892 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 601928704 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 101.10.10.2 | 101.10.10.4 | 23 | 1761 | 212403865 | 0 | 1 | 0 | 0 | 1 | 0 |
| 8 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753892 | 0 | 1 | 0 | 0 | 0 | 0 |
| 9 | 101.10.10.2 | 101.10.10.4 | 23 | 1761 | 212403865 | 0 | 1 | 0 | 0 | 1 | 0 |
| 10 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753892 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11 | 101.10.10.2 | 101.10.10.4 | 23 | 1761 | 212403865 | 0 | 1 | 0 | 0 | 1 | 0 |
| 12 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 2057753892 | 0 | 1 | 0 | 0 | 0 | 0 |
| 13 | 101.10.10.2 | 101.10.10.4 | 23 | 1761 | 212403865 | 0 | 1 | 0 | 0 | 1 | 0 |
| 14 | 101.10.10.4 | 101.10.10.2 | 1761 | 23 | 601928705 | 601928725 | 1 | 1 | 0 | 0 | 0 |
| 15 | 101.10.10.2 | 101.10.10.4 | 1761 | 23 | 601928725 | 0 | 1 | 0 | 0 | 0 | 1 |
| 16 | 101.10.10.4 | 101.10.10.2 | 23 | 1761 | 212403866 | 0 | 1 | 0 | 0 | 0 | 0 |
| 17 | 101.10.10.4 | 101.10.10.2 | 23 | 1761 | 212403866 | 0 | 1 | 0 | 0 | 0 | 1 |
| 18 | 101.10.10.2 | 101.10.10.4 | 1761 | 23 | 601928726 | 0 | 1 | 0 | 0 | 0 | 0 |

Figure 4.1 Man in the Middle Attack

In Figure 4.1 packet number five is the attacker’s reset packet with the spoofed client IP address, designed to desynchronize communications between the client (101.10.10.4) and the server (101.10.10.2). This is the portion of the attack that was intended to be caught with relational schemata. After the attacker sends the reset and initiates his own spoofed telnet session, there is a short ACK storm between the true client and the server in packets 7 through 13. This ACK storm is described in ???. Finally, the attacker sends some malicious data and a termination command (in the data portion of the packet, which is not used in this research) in packet 14, and the session is closed in packets 15 through 18.

Each packet in the data stream is compared, one packet at a time, to the five packets which follow it. An efficient method of examining all $O(n^2)$ packet field relationships, where n is the number of fields, is with the use of a matrix, as was suggested in Section 3.2.4. Here the fields of two packets are compared in a row/column manner that provides two genes, which when concatenated together produced a relational chromosome. First the field designated by the row is compared between the two packets, with an equal/not-equal relationship making up a one or zero relational gene, respectively. Then the field designated by the column is compared, making a second relational gene. These two genes concatenated together produce one relational chromosome for that row and column cell of

the matrix. Then the field corresponding to the next column is examined, looping through all $O(n^2)$ row and column fields. Using a six-packet sliding window, and the data in Figure 4.1, this produced the relationship matrix shown in Figure 4.2.

| | IPSrc(0) | IPDst(1) | TCPsrcPort(2) | TCPdstPort(3) | TCPSeq#(4) | TCPNextSeq#(5) | ACK(6) | PUSH(7) | RST(8) | SYN(9) | FIN(10) |
|-----------------|----------|----------|---------------|---------------|------------|----------------|-----------|-----------|-----------|-----------|-----------|
| IPSrc (0) | 0 | 3,0 | [3,0,2,1] | [3,0,2,1] | [3,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,0,1,2] | [3,1,2,0] |
| IPDst (1) | 0 | 0 | [3,0,2,1] | [3,0,2,1] | [3,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,0,1,2] | [3,1,2,0] |
| TCPsrcPort (2) | 0 | 0 | 0 | [3,0] | [3,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,0,1,2] | [3,1,0,2] |
| TCPdstPort (3) | 0 | 0 | 0 | 0 | [3,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,0,1,2] | [3,1,0,2] |
| TCPSeq# (4) | 0 | 0 | 0 | 0 | 0 | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,1,0,2] | [3,0,1] | [3,1,0,2] |
| TCPNextSeq# (5) | 0 | 0 | 0 | 0 | 0 | 0 | [3,1,2,0] | [3,0,1] | [3,1,2,0] | [3,2,0,1] | [3,1,2,0] |
| ACK (6) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | [3,2,1,0] | [3,0,1,2] | [3,2,0,1] | [3,1,2,0] |
| PUSH (7) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | [3,1,2,0] | [3,2,0,1] | [3,1,2,0] |
| RST (8) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | [3,2,0,1] | [3,1,2,0] |
| SYN (9) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | [3,1,0,2] |
| FIN (10) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

0 = 00 (not equal/not equal) (4,6) 2 equates to TCPSeq# equal, ACK not equal
1 = 01 (not equal/equal) (4,8) 2 equates to TCPSeq# equal, RST not equal
2 = 10 (equal/not equal)
3 = 11 (equal/equal)

Figure 4.2 Man in the Middle Attack Equal/Not-Equal Relational Matrix

In this matrix the relational chromosomes of each cell are determined by the relationships between the packets that were examined. For example, in cell (0,1) the relational chromosomes [3,0] are shown. Since cell (0,1) represents the IP source and IP destination cell, this cell gives all encountered relationships between the IP source fields of any two packets concatenated with the relationships between the IP destination fields of those packets (within the six-packet sliding window). Because there were only two IP addresses in the data used, and since they only communicated with one another, either the relationships were both equal (3), or they were both not-equal (0). For instance, in Figure 4.1, The IP source of packet number one and the IP source of packet number two are not-equal (a 0 gene), and IP destinations are similarly not equal (another 0 gene). However, if packet number one is compared to packet number three, both the IP source and IP destinations are equal (a 11 chromosome, which is binary for a decimal 3).

This method of intrusion detection worked at the outset in identifying this attack, as the relationships of cells (4,6) and (4,8) in Figure 4.2, might be detectors, as discussed on page 1-8. To test the effectiveness of this version, the sensors were trained against 200,000 packets of MIT Lincoln Laboratories data used in the 1999 Intrusion Detection Evaluation [LLab99]. This data was utilized in training because it is known to be attack free, and

because it was created to model the traffic that might be seen in a typical Air Force network. When the sensors were trained on the first 200,000 packets of TCP data, the relational antibodies performed well. In fact, the IDS detected the attack in only two cells—cells (4,6), and (4,8) of Figure 4.2, where the circled twos indicate relationships which were not detected during training with the Lincoln Lab’s data. The two in cell (4,6) represents two packets having equal TCP sequence numbers, but unequal ACK fields. The two in cell (4,8) represents two packets having equal TCP sequence numbers, but unequal RST fields. This is noteworthy, because these cells match two of the three abnormal relational patterns shown on page 1-8. This is also noteworthy because it did not detect the attack in any other cells—alleviating the concern of false positive detectors in those cells. Thus, the relational IDS seemed to work initially.

However, when a very large amount of data was utilized in training, it was noticed that packet P_3 and packet P_{3a} of Figure 1.2 may actually happen in reverse order. Namely, a user may send packet P_{3a} , then reset the connection—a legitimate activity. When this sequence of packets was detected during training, the equal/not-equal CIS gene did not detect the MitM attack, as it was encountered in training and eliminated as a relational antibody. Therefore, since packet order matters, the not-equal relationship was split up into greater-than and less-than relationships. The relational gene discussed in Section 3.2.3, with the 01, 10, and 11 schemata was used to represent less-than, greater-than, and equal relationships. This was incorporated into version 1.0.

4.2 *Version 1.0, Less-Than, Greater-Than, Equal Gene*

When two equal/not-equal (one/zero) genes were concatenated together there were only four possible chromosome values—00, 01, 10, and 11. However, in this version, using the gene schemata discussed in Section 3.2.3, each gene takes on three possible values—01, 10 and 11. Therefore, there are nine possible chromosome values in each cell using this gene structure: the two-gene combinations of the three values being 0101, 0110, 0111, 1001, 1010, 1011, 1101, 1110, 1111. This equates to the decimal numbers 5, 6, 7, 9, 10, 11, 13, 14, and 15, seen in Figure 4.3 and the appendices.

Using the decimal representation of the above chromosome values, a matrix such as the sample in Figure 4.3 is constructed from the data in Figure 4.1. Again, a field is never compared to itself in both the row and column (as it provides no additional data to be compared in both the row and the column). This produces a matrix similar to the one shown below, with only the upper right $\frac{m^2-m}{2}$ cells of the matrix having values.

| | IPSrc (0) | IPDst (1) | TCPsrcPort (2) | TCPdstPort (3) |
|-----------------|-----------|-----------|--------------------------------|--------------------------------|
| IPSrc (0) | | 6, 15, 9 | 5, 15, 10, 11, 13, 7, 14, 6, 9 | 6, 15, 9, 11, 14, 7, 13, 5, 10 |
| IPDst (1) | | | 9, 15, 6, 7, 13, 11, 14, 10, 5 | 10, 15, 5, 7, 14, 11, 13, 9, 6 |
| TCPsrcPort (2) | | | | 6, 15, 9 |
| TCPdstPort (3) | | | | |
| TCPSeq# (4) | | | | |
| TCPNextSeq# (5) | | | | |
| ACK (6) | | | | |
| PUSH (7) | | | | |
| RST (8) | | | | |
| SYN (9) | | | | |
| FIN (10) | | | | |

Figure 4.3 Partial Less Than, Greater-Than, and Equal Relational Matrix
 *Full Matrix is found in Appendix A

The entire matrix produced by the data in Figure 4.1, as well as the matrix derived from the training data are shown in Appendix A. Here, even when the IDS was trained on the same data that caused the attack to be missed in version 0.0, it caught the MitM attack in the suspect cells (4,6) and (4,8). This is because this version, unlike version 0.0, took into account packet order.

This system seemed to work well for one specific attack, but more fields needed to be added to the matrix if the IDS was to be of any practical use to catch a number of diverse attacks. To address this concern, more fields and more protocols were added to the IDS sensor.

4.3 Version 2.0, Multiple Protocols, Additional Fields

The fields of version 1.0 needed to be expanded beyond the simple eleven Transmission Control Protocol (TCP) fields addressed in the first two code versions in order to detect attacks involving other protocols and other fields. The three protocols and 28 fields used by Williams seemed to be an appropriate starting place, and these fields were used

with only slight modifications, as discussed on page 3-7. Thus, version 1.0 was expanded to include the fields of Table 3.1.

With these protocol and field modifications to the code, the number of fields of any packet may differ, as a TCP packet has more fields than a UDP or ICMP packet. Since it is likely of little use to compare TCP fields to UDP fields, the packets were compared according to protocol categories. When two packets were selected to be compared from the six-packet sliding window, the first thing checked was each packet's protocol. If the two packets were of the same protocol type, then all fields were compared when obtaining gene schemata. When the protocols of the two packets were different protocols, only common fields were compared, and the remaining protocol-dependent fields were skipped. The IP and Ethernet fields of all packets could be compared because TCP, UDP, and ICMP all use these fields.

Because protocol-dependent fields were not compared across protocols, the matrix of this IDS does not fill $\frac{m^2-m}{2}$ cells, but has blank areas. Additionally, this matrix, consisting of approximately 32^2 fields is somewhat larger than the previous matrices. Because of this increase in size, this matrix will not be shown here, but an example is shown in Appendix B. What is shown here instead, and presented as a reference for code versions 3.0 and 4.0, are the numerical breakouts of each field as they are represented in the IDS implementation. Via Table 4.2, if a cell is referenced, the meaning of that cell can be decoded.

The matrix that displays the concatenation of two relational gene schemata is a two-dimensional matrix. Therefore, this code version is referred to as a 2-D sensor throughout the rest of this document. Similarly, 3-D and 4-D matrices are built by examining all three-gene and four-gene combinations of these fields.

Version 2.0 worked well in detecting some attacks, and it was fast enough to easily use in a real time IDS "on the wire." In fact, 200,000 packets, which compromised approximately four hours of traffic of the Lincoln Laboratory data, could be run through the IDS in about 35 minutes. How well this 2-D sensor performed is covered more in Chapter V.

Table 4.2 Matrix Fields Used in Code Versions 2.0, 3.0, and 4.0

| | |
|----|--------------------------|
| 0 | IPID |
| 1 | IP TTL |
| 2 | Don't Fragment |
| 3 | More Fragment |
| 4 | Unused Flag |
| 5 | Fragment Offset |
| 6 | IP Header Length |
| 7 | IP Packet Length |
| 8 | IP Source |
| 9 | Source MAC |
| 10 | IP Destination |
| 11 | Destination MAC |
| 12 | Total Packet Length |
| 13 | TCP Source Port |
| 14 | TCP Destination Port |
| 15 | TCP Sequence Number |
| 16 | TCP Next Sequence Number |
| 17 | CWR |
| 18 | ECHO |
| 19 | URG |
| 20 | ACK |
| 21 | PUSH |
| 22 | RST |
| 23 | SYN |
| 24 | FIN |
| 25 | TCP Header Length |
| 26 | UDP Source Port |
| 27 | UDP Destination Port |
| 28 | UDP Length |
| 29 | ICMP Code |
| 30 | ICMP Type |
| 31 | ICMP Length |

Because examining the relationships of only two fields at a time may fail to detect some intrusions, the 2-D chromosome was expanded to make another chromosome, the 3-D chromosome.

4.4 *Version 3.0, Three-Dimensional Chromosome Structure*

The 3-D chromosome is constructed in the same way as the 2-D chromosome, with the exception that each 2-D chromosome is concatenated to another gene. Thus, the nine 2-D chromosomes mentioned earlier (0101, 0110, 0111, 1001, 1010, 1011, 1101, 1110, and 1111) can each be preceded by a 01 (less-than gene schemata), 10 (greater-than gene

schemata), or 11 (equal gene schemata). This means that the possible chromosome values for any cell increase from 9 values to 27 values.

Additionally, when the relational matrix is changed from a square to a cube the number of cells also increases. Whereas a 2-D matrix with 32 fields spanned three or four pages, the 3-D matrix is 32 times as large to print. This is the reason that 3-D and 4-D matrices are never printed in this document. Increasing the possible number of chromosomes per cell, as well as increasing the number of cells, significantly decreased the rate at which the IDS can function.

Considering the worst case of TCP as was done earlier, the following calculations can be made. With 26 fields compared in a square matrix using m fields, like the one in Appendix B, this gives $\frac{m^2-m}{2}$, or 325 cells in version 2.0. Each cell may have any of 9 chromosomes, for a maximum possible 2,925 comparisons for any two packets chosen. The higher the dimensionality, the more cells there are in the entire matrix, and the more chromosomes there are in each cell. If m is the number of fields, the number of cells in a d -dimensional matrix is given by the formula

$$\frac{\prod_{i=0}^{d-1} (m - i)}{d!}$$

In this case, where chromosomes that are the concatenation of three genes, the number of possible chromosomes per cell is 3^d . With a 3-D matrix vice a 2-D matrix, there are 2,600 cells checked for every two packets. Additionally, each cell has a possible 27 chromosome values. This yields a maximum possible 70,200 comparisons for any two packets. This means that it may be twenty-four times more costly to compare two packets with the 3-D sensor than with the 2-D sensor.

The increase in cost may be worth it, however, as the three-gene chromosome provides a greater resolution in intrusion detection than the two-gene chromosome. This is because during training the antibody that represents an attack may not be eliminated completely from the 3-D antibody matrix, where it might be from the 2-D version. This means that the 3-D sensor, while being more costly, may detect attacks that go unnoticed by the 2-D

sensor since negative selection is much more indiscriminant in the 2-D sensor. This indeed turned out to be the case, as is discussed in Chapter V.

4.5 Version 4.0, Four-Dimensional Chromosome Structure

The 4-D sensor is even more costly than the 3-D sensor. With the 4-D relational matrix, there are 14,950 cells compared for every two packets. Each cell has a possible 81 values, giving a maximum of 1,210,950 comparisons for any two packets selected! Because of this, the 4-D sensor is probably not useful in a “real time” IDS, as is shown in Figure 5.3. Rather, this sensor could be used in an off-line scan of stored data, searching for attacks that went undetected by the other versions.

With this in mind, the three versions are tested in a comparison fashion in Chapter V. Each of these sensors is examined in this research because they each have their own advantages in speed or resolution. They could be implemented in one IDS, with a “switch” that allows it to search for an intrusion in normal, medium, and high resolution. This might be implemented according to the Information Conditions (InfoCons) put in place at many Air Force institutions. Here, when a higher InfoCon is issued due to suspected malicious activity, the IDS could be switched from a 2-D to a 3-D or 4-D sensor, for a slower, but more thorough search for intrusions.

The Land attack discussed in Section 3.3 may go unnoticed by a direct field-to-field comparison, however, and comparing fields in a crosschecking manner may serve to provide additional insight into network activity to detect this attack. To test the intrusion detection ability of the crosschecking construct, the 2-D crosschecking gene was researched in versions 2.44 and 2.116.

4.6 Version 2.44, Beta Crosschecking Gene

The crosschecking gene was conceptualized to detect attacks that might be missed when comparing only the same fields when constructing chromosomes. The high-level language used in defining this gene is discussed in Chapter III on page 3-6. However, the low-level details, such as which fields are compared to which fields, also must be addressed.

In order to detect an attack such as the Land attack or other such attacks involving the IP source and destination address, only these two fields might need to be crosschecked against one another. But then that begs the question—if the crosschecking method works with the IP source and destination address fields, is it needed in other fields to detect similar attacks?

During testing, an attack was conducted which resembled the Land attack in that it involved the source and destination IP addresses. However, this attack, utilizing the hacker program DSniff, required a little more sophistication for detection than merely checking the two addresses of one packet. This attack also involved ARP Spoofing, which can be a major problem, as discussed in Section 2.6. The specifics of this attack, and how well the crosschecking gene performed in detecting it are addressed in Chapter V, but addressing the attack here provides a sound reason for using more than the IP source and destination addresses in the crosschecking mechanism, as detecting this attack also involved crosschecking in the MAC address fields.

Since the DSniff attack utilized TCP and ICMP fields, the beta version of the crosschecking gene was first implemented with the TCP fields of version 1.0, along with the ICMP fields of version 2.0. Here, the fields were split into categories of similar fields, and only those fields that were similar were compared. For example, the source MAC address and the destination MAC address were compared, but these two fields were not compared to any other field, as any MAC address would be larger than, say, a TCP Flag. The fields that were grouped into like categories are shown in Table 4.3.

Checking all possible combinations of the fields of each category produced c^2 possible combinations of the fields of any category, where c is the number of fields in that category. Therefore, the 16 fields above produced a 44 by 44 element matrix, the first page of which is shown in Appendix C. The remaining seven pages of this matrix were left out to conserve space, but the first page is shown to convey the concept of how the fields were crosschecked. Only the first three columns of Appendix C are really needed to understand cell breakouts.

This crosschecking gene proved promising, so it was expanded upon to include all 32 fields used in versions 2.0, 3.0, and 4.0 in an effort to make comparisons between these

Table 4.3 Beta Crosschecking Field Categories

| Crosschecking Field Name | Category |
|-------------------------------------------------|-----------------|
| IP Source IP Destination | Category 1 |
| MAC Source MAC Destination | Category 2 |
| TCP Source Port TCP Destination Port | Category 3 |
| TCP Sequence Number TCP Next Sequence Number | Category 4 |
| ACK PUSH RST SYN FIN | Category 5 |
| Non-Crosschecking Field Name | Category |
| ICMP Code | Category 6 |
| ICMP Type | Category 7 |
| ICMP Data Length | Category 8 |

versions and their crosschecking gene counterpart. This code is described in version 2.116 below.

4.7 Version 2.116, Full Crosschecking Gene

The initial crosschecking code included only 16 fields, and to compare the crosschecking gene's function to the gene used in versions 2.0, 3.0, and 4.0, those same fields were added. Those fields that seemed to be comparable were grouped into like categories for thoroughness. Those fields that did not seem to have any relationship to another field were grouped in their own categories. The 32 fields used previously were grouped into the categories shown in Table 4.4.

As stated previously, a category with c fields produces c^2 values which can be compared. From Category 1 to Category 15, this is $1^2 + 1^2 + 4^2 + 3^2 + 2^2 + 2^2 + 2^2 + 2^2 + 8^2 + 1^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2$ which equals 116 fields. This produces a 116 by 116 matrix, the first two pages of which are shown in Appendix D. This matrix proved very costly computationally, as in a 116 by 116 matrix, where each cell may take on one of 9 values, there are a maximum of 52,002 comparisons for every two packets checked (again, taking

Table 4.4 Full Crosschecking Field Categories

| Field Name | Category |
|-------------------------------------------------------------------------------|-------------|
| IPID | Category 1 |
| IP TTL | Category 2 |
| Don't Fragment More Fragment Unused IP Fragment Flag Fragment Offset | Category 3 |
| IP Header Length IP Packet Length Total Packet Length | Category 4 |
| IP Source IP Destination | Category 5 |
| Source MAC Destination MAC | Category 6 |
| TCP Source Port TCP Destination Port | Category 7 |
| TCP Sequence Number TCP Next Sequence Number | Category 8 |
| CWR ECHO URG ACK PUSH RST SYN FIN | Category 9 |
| TCP Header Length | Category 10 |
| UDP Source Port UDP Destination Port | Category 11 |
| UDP Length | Category 12 |
| ICMP Code | Category 13 |
| ICMP Type | Category 14 |
| ICMP Length | Category 15 |

TCP as the worst case). The computational cost of this version is comparable to the 3-D sensor with version 3.0, and this is only a 2-D crosschecking gene! The 3-D version of this crosschecking matrix would quickly require more time than was available for testing in this research, and the benefits of this method were unclear. Therefore, no additional testing was conducted on this version of code with the exception of speed testing, which was done to determine how computationally expensive this code is compared to the other versions with 32 fields.

4.8 GOPHER

The previous sections discussed the code versions implemented in this research. How these versions actually performed in testing is covered in more detail in Chapter V. What remains to be covered is the implementation of a visualization tool that might be used to

sort through suspected attack packets after the IDS issues an alert. The problem is how to visualize the field values of multiple packets at once. This is where parallel coordinates are useful.

Parallel coordinates allow for the transformation of multidimensional data, which is difficult to understand, into two-dimensional data, which is much easier to understand. This is necessary to compare large numbers of packets, each of which has multidimensional data (fields) associated with it. With parallel coordinates, multiple fields of multiple packets can be simultaneously analyzed so that comparisons can be made, and patterns can be found. In fact, parallel coordinates are one of the very best ways of modeling relationships between large dimensional data [Inse97]. This was the inspiration behind GOPHER.

In order to visualize network packets, a graphical representation of these packets needed to be developed. Also, in order to visualize patterns of nefarious network activity among these packets, a multidimensional to few-dimensional concept, such as parallel coordinates, needed to be developed. Therefore, the Graphically Oriented Pattern Honing and Evaluation Repository (GOPHER) was implemented. GOPHER allows for the inspection, filtering, and comparison of multiple network packets, with the goal of understanding network activity.

Since this research focused on TCP, UDP, and ICMP packets, GOPHER was designed to display all three of these protocols either simultaneously, or separately, allowing the user to filter out packets by means of Java sliders. The main display of GOPHER is shown in Figure 4.4.

Each field of a packet makes up one value on the x-axis of the display. The y-axis, however, had to have some sort of coordinate system to depict the values of the packet fields. Since the values of the specific fields under consideration differ quite significantly, depicting each field under the same coordinate system would be impossible. For instance, the IP Fragment field can only be a zero or a one, while the Client Sequence Number field can range from zero to one less than 2^{32} . This is quite a discrepancy, and if the y-axis, depicting field value, ranged from zero to 2^{32} to accommodate the Client Sequence

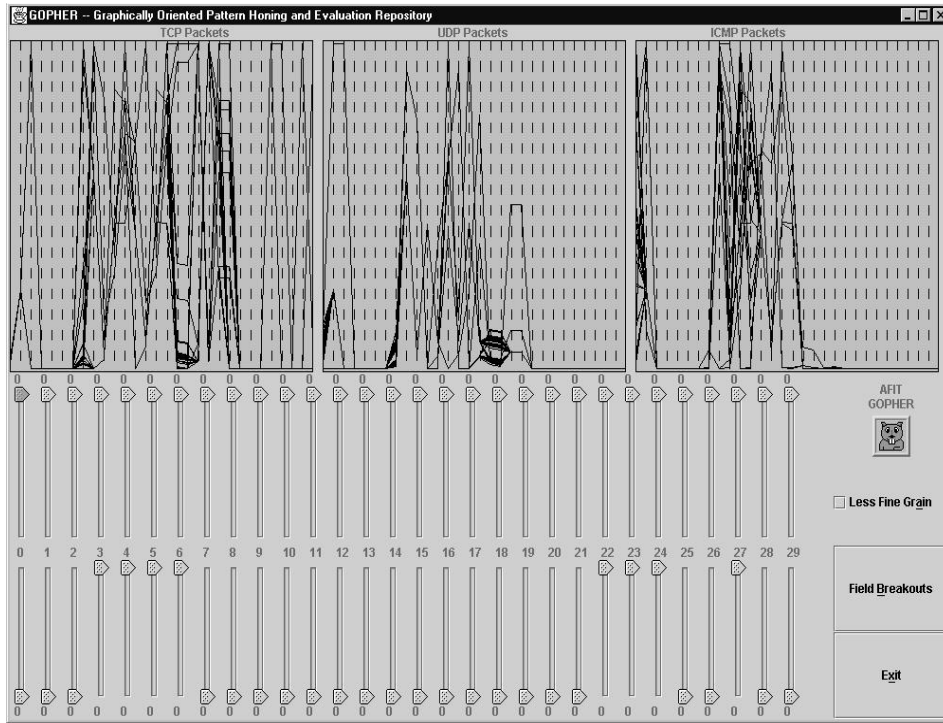


Figure 4.4 GOPHER Main Screen

Number, it would be impossible to visually inspect whether the IP Fragment field was either a zero or a one. Therefore, each field was depicted in a normalized fashion in its own y-axis, as is typical in a parallel coordinates system. This allows for the IP Fragment field to be visually inspected adjacent to the Client Sequence Number field, even though both are of vastly diverse scales.

With the exception of the MAC address, GOPHER examines all fields used in Table 4.2. However, a few of the fields were modified as necessary. For instance, the IP address fields were split into their separate octets to match the work conducted by Williams [Will01]. Also, to speed calculations, the TCP sequence numbers were reduced by $2^{31} + 1$. Figure 4.5 displays the output of the “Field Breakouts” button on the GUI interface. From this figure it can be noticed that each field may represent a different value, depending on the screen viewed.

One of the most powerful aspects of GOPHER is that it allows for a drill down look of specific protocols. This also allows for a more spatially consistent understanding

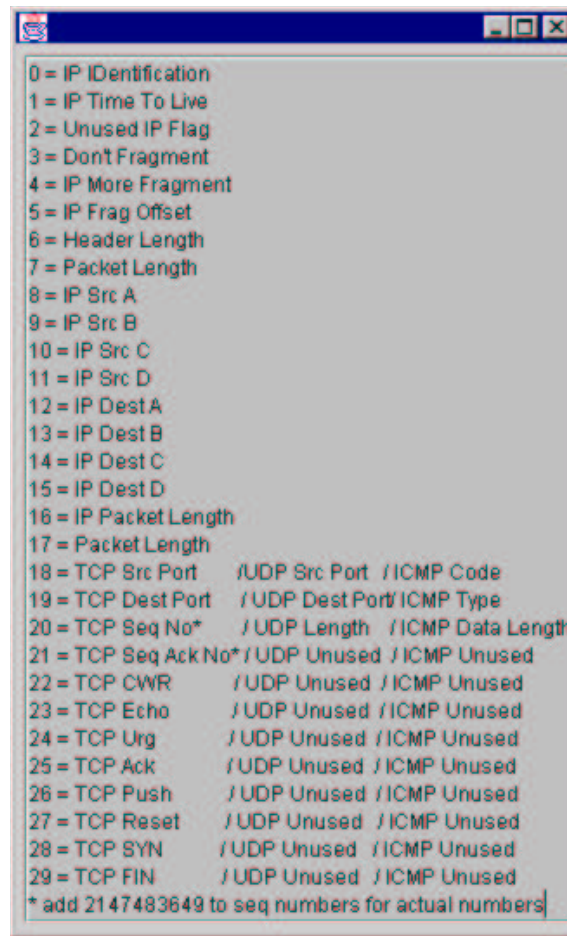


Figure 4.5 GOPHER Field Breakouts

of the tool, because in the drill down screen each slider is placed immediately under the field it is meant to filter. This could not be done in the initial window because all three protocols were depicted. Hence, in the initial window there tended to be some confusion as to which field a slider is filtering. However, in the drill down window, since the sliders are immediately under their respective field, understanding which field a slider is associated with becomes intuitive.

The GOPHER drill down window also allows for highlighting a specific packet amongst the plethora of packets. This is accomplished via a scrollable list in the lower right hand corner of the display. In this way, if a packet needs to be distinguished from other packets,

it is simply selected, and that packet is depicted in red (whereas all the other packets are displayed in black). The drill down display of the TCP protocol is shown in Figure 4.6.

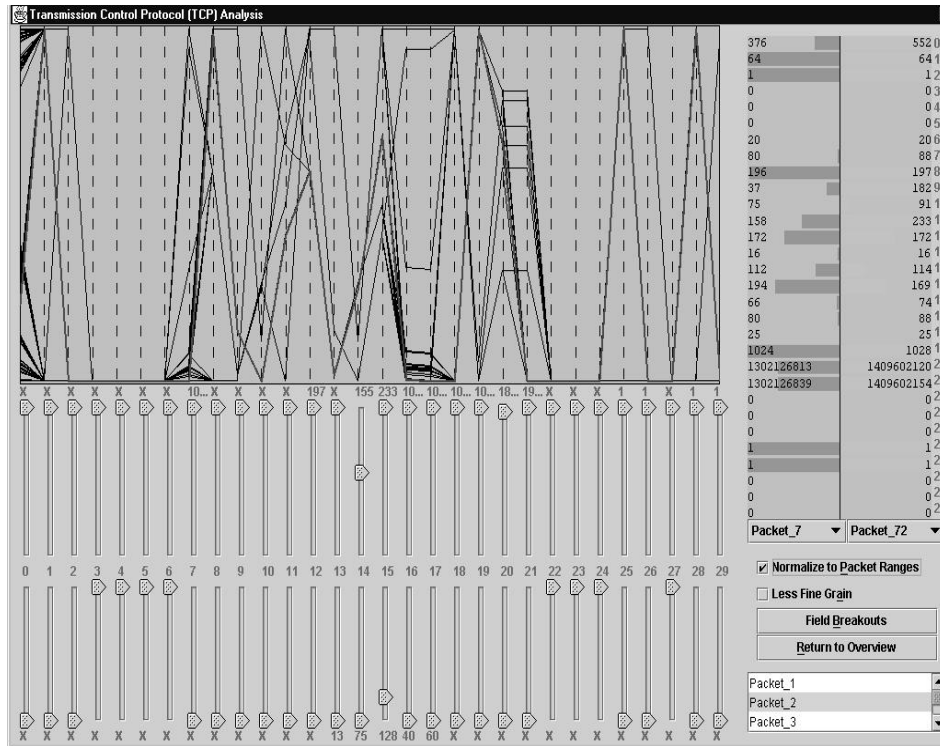


Figure 4.6 GOPHER Drill Down Screen

Along with viewing the packet values as a they relate to one another in the parallel coordinates pane, the analyst may wish to view individual packet values separately. To accomplish this another panel was added. This panel, shown in the upper right-hand portion of Figure 4.6, allows for the values of individual packets to be shown. From a drop-down menu, packets may be selected, and the fields of these packets will be displayed both graphically and numerically. Often, a single packet may need to be compared to another packet, to examine anomalies between the two. Therefore, two packet windows are depicted next to one another so that relationship patterns (such as equal, less-than, or greater-than) can be examined.

With the overview capability, the drill-down capability, the individual packet inspection and highlighting capability, and the two-packet comparison capability, GOPHER is armed and ready to be tested against actual network attacks, as is done in Chapter V.

V. *Experimentation and Analysis*

The testing of this research effort was conducted in phases, beginning with small-scale tests and progressing to larger-scale tests as the coding progressed. As versions 2.0, 3.0, and 4.0 are the primary focus of this research, the majority of the testing was conducted on these versions of code. However, each of the other versions are important in its own way for conveying an awareness of the overall topic and providing a foundation for understanding how the larger scale tests worked without the additional details. Therefore some of the small-scale tests that were conducted on the earlier versions of code are discussed first, leading up to the main testing section.

5.1 *Testing Methodology*

As stated in Section 1.6, the IDS must first be trained on normal data in order to develop a representation of “self.” In order to make the tests reproducible, publicly available attack-free data was utilized in the training process. Therefore, the first week of the 1999 Lincoln Laboratory’s Intrusion Detection Evaluation data is used throughout much of the training of this research, as this data is constructed to emulate real-world network traffic, is known to be attack-free, and is public domain. Similarly, many of the attacks with which this research was tested were Lincoln Lab’s attacks, for much the same reasons. While it is true that this data has some limitations [McHu00], it is still some of the best available attack-free or attack-ridden network data.

The Lincoln Labs data is stored in a TCPDump file format, which is the most common format for storing network packet information. The TCPDump files were first converted to human readable form with the program Ethereal to obtain the necessary packet information. Then the field values were extracted and stored in a comma separated value (*.csv) file.

To train the IDS on normal data, data was read in from a *.csv file, and a matrix of relational chromosomes was constructed and stored in a relational matrix object (*.rmo) file. The *.rmo file stored only the relational matrix, similar to the ones shown in the appendices. The matrix representation of self was stored so that it could be used repeatedly

in multiple tests without having to train the system every time it was tested. This is much like what would be done in an enterprise-level IDS, as the system could be initialized, read in the *.rmo “self” file, and begin scanning network activity.

There was also an additional training module that could read in a *.rmo file and add to it the relationships found in a user-specified *.csv file. This was used to train the sensors on large amounts of training data that spanned multiple files.

Once the IDS is trained on self, it can read in *.csv files (with or without attacks) and check the relationships between the packets of these files to the relationships of the *.rmo (self) file. In scanning for attacks among the testing data, there are three ways in which the IDS can scan these packets and report alerts to the user. Each of these methods is discussed briefly below.

One way of looking for anomalous relationships among the testing data is to examine every two packets in real time and look for a relationship that do not match any chromosomes in the training (self) matrix structure (or match antibodies in a non-self matrix structure). With this process, as soon as an anomalous relationship is detected, an alert could be generated specifying the two packets, the fields, and the relationships that generated the alert. This may prove useful in assisting the network analyst to more readily identify those alerts that are true positives, and those that are false positives. A drawback to this approach, however, is that the same antibody and/or same matrix cell may be repeatedly seen in multiple packets. If every one of these instances generates an alert, the IDS may quickly overwhelm the analyst with alerts generated on the same type of data, just spread across different packets. For example, in the IP spoofing attack discussed earlier, packets with different IP and MAC addresses may generate an alert. If many packets are sent in a single telnet session, and each time two packets are encountered with this relationship an alert is generated, one telnet session from one IP address may generate literally hundreds or thousands of alerts.

A second method of scanning packets and generating alerts is to scan the testing data and build a “test data” relational matrix with the relationships encountered by x number of packets. This matrix could be built in the same way that the training data

(self) matrix is built. After this test matrix is constructed, the chromosomes in each cell of the test data matrix may be compared to the antibodies of the training data matrix, and alerts could be generated. In this way, alerts are generated as collections of x packets, not as alerts between any two packets within those x packets. In the above IP spoofing example, this may provide the same information per cell, at the cost of specifying which packets generated which alerts. This is the testing methodology of this research, as the IDS reads in the entire testing *.csv file, generates a testing matrix (which is later discarded), and compares the chromosomes of this matrix to the antibodies of the training matrix. Alerts are then generated by cell, which specifies the fields of the packets that generated the attacks. Which packets, as well as which chromosomes generated the alerts might have been stored, but this required bookkeeping mechanisms that were not implemented in the IDS. When this information is presented to the reader, it is merely to provide understanding of how the IDS detected the attack.

In the case where only two packets are examined in testing, both of the above methods generate the same alert. In fact, both methods will detect exactly the same types of attacks, because they construct the same relationships from the training data, and compare them to the same relationships of the testing data. What differs is the method with which they report alerts to the user. The first way is by packet and the second way is by cell. A third way of scanning packets and generating alerts is with the use of some portions of both of the above methods. This may provide some details of alerts from the perspective of the packets, and some from the perspective of the matrix, and is likely the most useful form of alert reporting. For instance, an IDS may scan packets for anomalous relationships, and after encountering anomalies, may store the packets that generated these relationships in a database. After a small period of time the IDS may present the anomalous cells to the analyst with a link to the packets that caused them. Conversely, an IDS may report a list of anomalous packets, and link the analyst to the cells which caused them. In either way, the IDS may need to collect like alerts, and present them to the user in one neat packaged alert, instead of as a series of individual alerts.

As stated above, this research used the second method of intrusion detection and alert generation, where x represented the number of packets in the entire attack *.csv file.

Thus, alerts reported the relational matrix cells with abnormal relationships among all packets of the test file. Using this methodology, the following versions were tested.

5.2 *Version 0.0 Testing*

Version 0.0 was tested to determine if its equal/not-equal gene structure could detect the Joncheray MitM attack discussed earlier. When the IDS was trained on the first 200,000 packets of Monday, week one of the Lincoln Lab's data, it detected the attack in cells (4,6) and (4,8) as mentioned in Section 4.1. This proved promising, and more extensive training of the system was conducted to determine if these fields would be eliminated in the negative selection process of training. During more training the system still caught the attack, even when trained on all 807,201 packets of Monday data. When Tuesday's data was added to the training, however, the attack was no longer detected.

With closer inspection it was noticed that the relationships of packets P_3 and P_{3a} of Figure 1.2 occurred in reverse order in the Tuesday training; namely, a normal packet was sent, and then a reset followed it. This is in contrast to the MitM attack shown, where the attacker sent a spoofed reset, and then the client sent its own data. Therefore, since packet order matters, no further testing was done on this version, and coding of version 1.0 began. It should be noted, however, that even though this version was scrapped to pursue code that could detect more attacks, the equal/not-equal relationships of this version still might be useful in intrusion detection. For example, this version may detect the ARP spoof and IP spoof attacks mentioned later, as these attacks can be detected with simple equal/not-equal relationships and packet order does not matter in the detection process. Additionally, since this version does not have the additional overhead of the greater-than and less-than relationships, it is faster than the following versions.

Despite the potential usefulness of this code and the fact that it is faster than the following versions, this research sought to develop a mechanism to detect unknown attacks. Therefore completeness was put before speed and this version, along with its strictly equal/not-equal gene, was deprecated.

5.3 *Version 1.0 Testing*

When the relational gene schema was changed from a 0/1 mapping to a 01, 10, or 11 mapping, this new mapping detected the Joncheray MitM attack where the 0/1 mapping did not. This is because since version 1.0 takes into account packet order, the antibodies for detection were not eliminated during the negative selection process of training. When trained on the same 100,000 packets of Tuesday week one of the Lincoln Lab's data, it detected the attack in only the cells (4,6) and (4,8). As stated in Section 4.1, since the attack was only detected in expected cells, much of the concern for false positives was alleviated in this instance.

To test this version a little more extensively, a spoofed IP address attack was constructed. Here, in another MitM scenario, the attacker was hypothetically sniffing the line between the client and server. Then, after seeing the client conduct a telnet session with the server, the attacker performs a like telnet session using the client's spoofed IP address. Not sniffed was a DoS attack perpetrated on the client (committed to prohibit the true client from responding to the server), which could easily be done, by say, cutting a wire or turning the true client's machine off. Basically, from the server's perspective, and from the perspective of the IDS, which resides on the same subnet, there were two telnet sessions with the same IP address, but coming from different MAC addresses. The attacker can spoof his IP address, but since packets are routed at the Network Interface Card (NIC) level in the Internet Protocol, the attacker cannot spoof his MAC address, lest he lose all communications.

Since this attack involved the MAC address, the source MAC address and the destination MAC address had to be added to the matrix to detect this attack. These additional two fields did not change the code significantly, so the code version number was not increased. The additional two fields did, however, change the matrix slightly. The matrix of version 1.0 with MAC addresses is shown in Appendix E for completeness.

It was expected that the system would detect the attack, as two packets coming from the same IP address but from different MAC addresses could be thought of as abnormal. However, when trained on the Lincoln Lab's data used previously, the attack was missed.

Upon analysis it was determined that this was because the training data was collected from a Lincoln Lab's sniffer that was located outside the firewall. In such a case, where there were multiple routers communicating with one another, it is common for packets to have like IP addresses and different MAC addresses.

This brings home an important point, which is that in order to be most effective, the IDS must be trained in the same location in the network topology that it is placed for detection. Therefore, the IDS was retrained on 100,000 packets of Tuesday, week one data collected from inside the firewall. With this training the attack was detected in cells (0,1) and (2,3) of the matrix in Appendix E. Here, the attack was detected in the two cells expected – namely, the IP and MAC addresses of the first and second telnet sessions. From Appendix E it can be seen that they were detected by the value 14 in both cases. In cell (0,1) this states that the source IP addresses of two packets were equal, while the source MAC address of the first packet was greater than the source MAC address of the second. In cell (2,3) this states that the destination IP addresses of two packets were equal, while the destination MAC address of the first packet was greater than the destination MAC address of the second.

This version seemed to work well with limited fields and one protocol, so coding began to expand the number of fields and protocols utilized by this system. This became version 2.0. As mentioned earlier, the crosschecking gene was examined in somewhat of a tangent to the actual focus of the research. This method of gene construction was so interesting, however, that it could not be left out of the thesis altogether. Therefore, before going on to the discussion of the main code tests, the crosschecking version tests are first discussed.

5.4 Version 2.44 Testing

Version 2.44 addressed the crosschecking gene schema discussed in Section 4.6 on page 4-10. As stated in that section, this version of the code was produced to incorporate the fields of a MitM attack committed with the use of the hacker program DSniff. While the crosschecking gene should detect attacks such as the Land attack, where the Source IP address is the same as the Destination address, the DSniff MitM attack was the “baseline”

attack for three reasons. First, detecting the Land attack may easily be done with a signature-based IDS (although it is not done with Snort). Second, detecting the Land attack with this crosschecking gene is a trivial matter if the packet is going to a known IP address, which would likely be the case, lest the malicious packets never reach the target. Therefore, an attack that is more difficult to detect was sought. And third, the DSNIFF MitM attack included an ARP spoofing attack, which is particularly threatening in today's wireless network [SANS01a]. Hence, a method for detecting this attack needs to be addressed. The DSNIFF MitM attack is described in somewhat lengthy detail here, as understanding the attack allows for a more complete understanding of the detection process.

With the use of the DSNIFF program, which contains an ARP Spoof utility, an attacker can easily sniff traffic destined for any address within even a switched network, as long as the attacker is located within the same subnet. In the example attack, the attacking computer was located between two computers, which are called the client and the server. Since the traffic of a switched network is routed at the NIC level, if an attacker can make both parties think that the attacker's NIC is the NIC of the other party, then they will both send their traffic to the attacker's NIC address. The attacker then forwards the packets to the intended recipient, and neither party is the wiser.

The attacker fools each party into thinking that his NIC address is the intended recipient's NIC address by means of ARP spoofing. Understanding ARP spoofing requires a bit of background in Address Resolution Protocol, which is provided here. While Internet Protocol packets are bound for IP addresses at the IP layer, they are actually sent at the MAC layer. This means that IP packets are really sent from NIC address to NIC address, not from IP address to IP address. The NIC address of the next hop in the path toward the destination IP address is stored in an ARP table. When the NIC address of a destination IP address is not known, as is often the case since the ARP tables purge old data every thirty seconds or so, the first thing that needs to be done is to find the NIC address of the destination. Therefore, the first packet in a session where NIC address of a destined IP address is not known is an ARP request packet, which is broadcast to the subnet. The ARP request basically asks "which NIC address has this IP address?" The destination

address replies with an ARP reply, saying “I do. Here is my NIC address,” and the ARP table is updated. Then the source can send its traffic toward the destination IP address. [RFC826]

The problem with the Address Resolution Protocol is that to improve efficiency ARP tables are updated whenever an ARP reply is heard. This means that the attacker can send a reply to a request that was never sent, and the ARP table will be updated accordingly. This is the heart of ARP spoofing—the attacker continually sending out ARP replies stating that his NIC address belongs to whomever he chooses to spoof.

Figure 5.1 shows a graphical depiction of this attack as it was perpetrated in testing. Here, the attacker is located at IP address 101.10.10.5, and spoofs the client and server addresses with bogus ARP replies. As this research did not include the ARP packets, they were filtered out of the data used to detect this attack. While it is true that the ARP spoof attack might be detected using the ARP packets themselves, this research focused on a method to detect this attack without the Address Resolution Protocol, and therefore uses only TCP and ICMP packets.

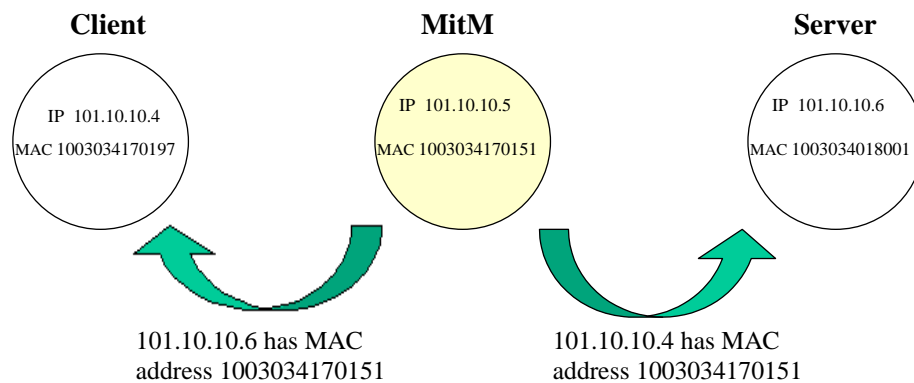


Figure 5.1 ARP Spoof Attack

An interesting thing to note about this attack is that since neither the client nor the server ever communicate directly with one another (because the attacker is forwarding their packets), an IDS might not detect the attack except for one thing. The NIC of any machine examines packets that are sent to it to see if there is a better method of routing those packets. If there is, it sends an ICMP redirect packet to the sending party (while still

forwarding on the sent packet). This is where the ICMP packets of this attack come from, and why it can be detected without examining the ARP packets. The data packets of a complete telnet session between the client and server, which was forwarded through the attacker, is shown in Figure 5.2. In this figure the packet number constitutes the packet ordering, but the ICMP packets were extracted to the bottom because their fields do not match those of the TCP packets. Again, only those fields that were implemented in the genes of this version are shown.

| Num | Protocol | SrcIP | SrcMac | DstIP | DstMac | SrcTCPPort | DstTCPPort | Seq# | NextSeq# | A | P | R | S | F |
|-----|----------|--------------|---------------|--------------|---------------|------------|------------|------------|-----------|---|---|---|---|---|
| 1 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179403 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72653 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179404 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72654 | 72703 | 1 | 1 | 0 | 0 | 0 |
| 5 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179404 | 0 | 1 | 0 | 0 | 0 | 0 |
| 7 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179404 | 180179424 | 1 | 1 | 0 | 0 | 0 |
| 9 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72703 | 72745 | 1 | 1 | 0 | 0 | 0 |
| 10 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179424 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179424 | 180179431 | 1 | 1 | 0 | 0 | 0 |
| 13 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72745 | 72780 | 1 | 1 | 0 | 0 | 0 |
| 14 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179431 | 0 | 1 | 0 | 0 | 0 | 0 |
| 15 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179431 | 180179437 | 1 | 1 | 0 | 0 | 0 |
| 17 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72780 | 72787 | 1 | 1 | 0 | 0 | 0 |
| 18 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72787 | 0 | 1 | 0 | 0 | 0 | 1 |
| 19 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179437 | 0 | 1 | 0 | 0 | 0 | 0 |
| 20 | TCP | 101010010004 | 1003034170197 | 101010010006 | 1003034170151 | 1717 | 21 | 180179437 | 0 | 1 | 0 | 0 | 0 | 1 |
| 21 | TCP | 101010010006 | 1003034018001 | 101010010004 | 1003034170197 | 21 | 1717 | 72788 | 0 | 1 | 0 | 0 | 0 | 0 |
| Num | Protocol | SrcIP | SrcMac | DstIP | DstMac | ICMPCode | ICMPType | DataLength | | | | | | |
| 6 | ICMP | 101010010005 | 1003034170151 | 101010010004 | 1003034170197 | 1 | 5 | 40 | | | | | | |
| 8 | ICMP | 101010010005 | 1003034170151 | 101010010004 | 1003034170197 | 1 | 5 | 60 | | | | | | |
| 12 | ICMP | 101010010005 | 1003034170151 | 101010010004 | 1003034170197 | 1 | 5 | 47 | | | | | | |
| 16 | ICMP | 101010010005 | 1003034170151 | 101010010004 | 1003034170197 | 1 | 5 | 46 | | | | | | |

Figure 5.2 DSniff Attack Data Files

Close examination of this data shows why the crosschecking gene was examined. Looking at packet number one and packet number six clearly show a discrepancy between the source IP and MAC addresses of packet six, and the destination IP and MAC addresses of packet one—they have the same MAC address, but different IP addresses. However, this discrepancy will not be detected comparing strictly source address to source address, or destination address to destination address. To detect the attack in this figure, a source address needs to be compared to a destination address—exactly what crosschecking does.

The results of this testing were very positive indeed. Trained on the first 100,000 packets of Tuesday, week one of the Lincoln Lab’s data, the above attack was detected in cells (0,43) (1,5) (1,43) (2,6) (2,43) (3,43) (4,43) (5,43) (6,43) (7,43) (41,43) and (42,43). When examined with the example matrix in Appendix C, it can be determined that fields

(1,5) and (2,6) contain IP and MAC source and destination addresses. These are the intended antibody cells. The detections in other cells are due to the fact that ICMP redirect packets are not common, and were not seen in the 100,000 packet training session. Therefore, the packet length of those redirect packets was also uncommon and was flagged as an alert.

The Joncheray MitM attack was also tested with this sensor, and was detected in cells (0,4) (0,14) (1,5) (1,14) (2,6) (2,14) (3,7) (3,14) (4,12) (4,14) (5,14) (6,14) (7,12) (7,14) (12,16) (12,28). Cells (0,4), (1,5), (2,6), and (3,7) are cells where the attack was detected with source or destination IP and MAC address discrepancies. The other cells all involved TCP sequence numbers, which were also anomalous (hence the ACK storm).

The sensor was also tested on the IP Spoof attack mentioned in Section 5.3, and detected the attack in only cells (0,4), (1,5), (2,6), and (3,7), which are expected. Additionally, a Land attack (a telnet session, where one packet had the same IP Source and IP Destination field) was run through this IDS sensor to see if it would detect this attack. It detected the Land attack in cells (0,1), (0,2), (1,3), (1,4), (1,5), (1,7), (1,14), (2,3), (2,4), (2,6), (2,7), (2,14), (3,5), (3,6), (3,7), and (3,14).

The fact that these four attacks were detected in the predicted cells and that no false positive cells were alerted was tremendously encouraging, and further coding was conducted to expand the crosschecking gene to include the protocols and fields mentioned earlier. This became version 2.116.

5.5 Version 2.116 Testing

Version 2.116, as stated in Section 4.7, included crosschecking between 32 fields, for a 116 by 116 cell matrix. This much larger matrix caused a significant increase in the amount of time required for processing. In fact, where version 2.0 (also with 32 fields) only took 35 minutes to train on 200,000 packets, this version took nearly six hours. While it is true that this time requirement is about equal to the time it took to train version 3.0, it must be kept in mind that this is only a 2-D version. Expanding all 116 fields in a 3-D

manner, as was done with version 2.0 would be vastly more costly than expanding the dimensionality of the non-crosschecking genes.

With the same sort of calculation that was conducted in Section 4.4, again considering that TCP dominates the other protocols, it can be determined that the 2-D crosschecking matrix of version 2.116 has a maximum of 52,002 comparisons for every two packets. However, in testing, the additional overhead of the crosschecking code caused this version to perform about the same as version 3.0 did with respect to time, even though version 3.0 made a maximum of 70,200 comparisons for every packet. That means that with large amounts of data the crosschecking code performed about 1.3 times slower as the non-crosschecking gene.

With a little more calculation, it can be determined that the 3-D crosschecking code would make a maximum of 5,512,212 comparisons for every two packets, and the 4-D crosschecking code could make 434,086,695 comparisons for every two packets! The latter is about 360 times as many comparisons as version 4.0. Since version 4.0 took about 22 hours to inspect 200,000 packets, and assuming that crosschecking is 1.3 times slower, this means that the 4-D crosschecking CIS would take over a year to examine 200,000 packets!

Because this sort of time was not available for this thesis, and the benefits of pursuing this line of research were unknown, the crosschecking mechanism was not explored further, in order to more closely examine non-crosschecking genes. The crosschecking gene mechanism was not left without regret, however, as the testing conducted on version 2.44 clearly showed its potential use in intrusion detection. Perhaps some bright researcher will shoulder the task of examining this method of intrusion detection in the not too distant future.

5.6 Versions 2.0, 3.0, and 4.0 Testing

Versions 2.0, 3.0, and 4.0, each use non-crosschecking genes with less-than, greater-than, and equal schema of 01, 10, and 11, respectively, and all use the same fields. They are basically carbon copies of one another with the exception that the number of genes concatenated together ranges from two to four according to version number. This was

intentional, in order to determine how dimensionality plays a role in the effectiveness and speed of relational intrusion detection. Therefore, these versions are tested in a comparison fashion, with each being subjected to the same attacks and training packets.

There are two important aspects of these versions which require testing—speed and effectiveness. Since higher dimensionality equates to higher resolution, a higher dimensional detector would likely detect more attacks than a lower dimensional one. This is because negative selection removes much of the detection potential of the lower-dimensional chromosomes. Higher dimensionality allows greater selectivity in negative selection, possibly allowing more of the detection (non-self) space to remain intact. However, the additional comparison overhead of the higher dimension makes it more computationally expensive. These two important issues are addressed in the testing below.

5.6.1 Speed Testing. The graph in Figure 5.3 shows the time each version took to inspect a number of packets. A complete breakdown of the data, as well as the speed of the other versions of code is shown in Appendix F. These tests were conducted on the same 750Mhz Intel Pentium-III machine with 128 MB of RAM running Windows 98SE. The code was run from within the Java program Visual Age 3.5. Modifications could easily be made to improve the efficiency of these programs, such as running the code on faster machines, from within the Sun Java SDK environment, or even by coding the program in a lower-level language such as C or ADA. Some optimization methods were tested, and with the use of the Java Jet program, which compiles the Java code and runs it without the use of a virtual machine (much like C does), the run times were cut by more than half. However, as maximizing the speed of any one version was less of a priority than modular, easily changeable code, the tests were run with the code as-is. Therefore the times shown in Figure 5.3 and Appendix F are shown with the caveat that these times can easily be decreased. The figure should, however, provide a good gauge of how well each version performs with respect to one another.

Although only versions 2.0, 3.0, and 4.0 were compared with respect to efficiency, the speed of version 2.116 is also depicted in the figure, as this version also had 32 fields, and had a similar packet-checking rate as version 3.0. When examining the graph, however, it

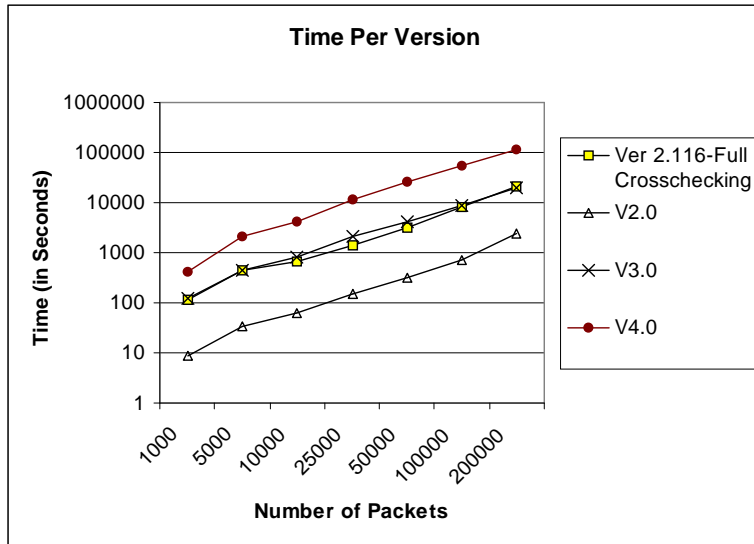


Figure 5.3 Run Time Comparisons

should be kept in mind that only versions 2.0, 3.0, 4.0, have the same non-crosschecking chromosome makeup. Figure 5.3 clearly shows that the increase of packets seems to cause a nearly linear increase in time in all cases. This is valuable because it allows for the coarse prediction of times per version, per x number of packets. The graph also demonstrates that higher dimensionality and crosschecking increases time significantly, as was mentioned earlier. The variation of these times is small, as is shown in the standard deviations graph in Appendix F.

As mentioned in Section 4.5 on page 4-10, the 4-D sensor may not be efficient enough to perform intrusion detection in real time. Therefore, either packets may need to be dropped, or the 4-D sensor might need to perform a scan of stored packets off-line. Even if the 4-D was made to run efficiently in a real time environment, still higher dimensional sensors would at some point become too expensive to run “on the wire.” They may still have their place, however, because the best defense is defense-in-depth, even if parts of the defense are run off-line. Additionally, all of the cells in the larger-dimensional matrices need not always be examined to provide a layer of ID. Some specific cells may be targeted in a signature-based approach, as discussed in Section 6.3.3.

5.6.2 Effectiveness Testing. The more difficult aspect of comparing these versions is demonstrating how well one version detected an attack compared to another. Using the same fields, higher-dimensional detectors will detect anything that the lower dimensional detectors find, and (potentially) more. This is because any cell in the 2-D matrix is contained approximately m times in the 3-D matrix, where m is the number of fields. A problem with the higher dimensional detectors, besides their speed, is that they require more training to eliminate false positives since they have so many more chromosomes to train. For example, version 2.0 has 2,929 possible chromosomes and version 3.0 may have 70,200 chromosomes, as stated in Section 4.4. Therefore, if x percent of this space represents self, then there are more self chromosomes which need to be identified during the training of the higher-dimensional detector. This was indeed born out in testing, as the results showed a higher false positive rate with the higher dimensionality.

To test these versions against one another, the 1999 Lincoln Lab's data was selected for the training and testing sets, for reasons discussed earlier. These systems were trained on all (1.9 million) TCP, UDP and ICMP packets of Monday and Tuesday data from week one of the Intrusion Detection Evaluation archive.

After training, tests were run to determine the true positive and false positive rates of these systems—namely, how many attacks they caught, and how often they alerted (wrongly) on normal data. A hint to the true negative and false negative rates (how much normal data was not alerted on, and how many attacks were missed) is also attainable from the tests posed, as tests were run on both normal and malicious traffic.

The attacks, as they were taken from the Lincoln Lab's data, are described briefly in Appendix G [LLab99]. Describing each attack and how it was or was not detected in detail is beyond the scope of this thesis, as it would fill volumes of text. Many of the Lincoln Lab's attacks were not attempted to be detected, and those attacks are not shown. For example, as some attacks involved host-based attacks that were not visible from the network, such as a user inserting a boot disk into the victim computer to download malicious code. Other attacks were present only in the packet payload of the network data. These attacks were not examined, since this thesis effort focused on a network-based IDS sensor that does not

examine the packet payload. The remaining attacks, which were thought to be present in the data sniffed from the Lincoln Lab's IDS located outside the firewall, were used.

To test the system as thoroughly as possible, tests were run on fifteen different types of attacks, often with multiple attack files within each type. This made for a total of 24 attacks. In order to make these tests reproducible, an issue that must be addressed is how the attacks were pulled from the Lincoln Lab's archive to be tested. This is because the files of the Lincoln Lab's Intrusion Detection Evaluation are stored as entire days of network traffic, with multiple attacks in each day. To test each attack separately, each attack needed to be pulled from the rest of the data.

In order to extract the attacks, the TCPDump program was used to filter out the attacks by attack time and source and destination IP addresses. Simply filtering the files by time was done where possible, but often this was not practical, as some attacks spanned thousands of packets of other parties communicating. When this was the case, determining if an alert by the IDS was a true positive (an alert due to the attack) or a false positive (an alert due to extraneous network activity) might come into question. Therefore, the filters kept the traffic to as little extraneous activity as possible.

Conversely, attacks 42.145441 and 51.200037 consisted of only one attack packet each. In order to detect any relational anomalies with these attacks, these packets need to be compared to something. As the IDS sitting on the wire would observe all packets flowing through the network, these attacks were filtered to include a few of the normal network packets that occurred around them. This provided a means of testing the IDS on these single-packet attacks.

Each sensor was trained on all Monday and Tuesday, week one of the Lincoln Lab's data collected from a sniffer outside the firewall, which represented 1.9 million packets worth of TCP, UDP, and ICMP data. Table 5.1 shows the results from these attack tests with all three versions of code. The attack names and descriptions are taken directly from the Lincoln Laboratory database of attacks for repeatability [LLab99]. Again, for a more in-depth explanation of the attacks, see Appendix G.

The attacks were also tested against version 1.8.1 of the increasingly popular open-source IDS Snort. The signatures used to test Snort were the default signatures, downloaded on 2 March 2002. This testing was done to give an indication of how well the IDS of this research performs when benchmarked against a commonly used, well established IDS. A comparison between Snort and the relational IDS of this research is addressed in Section 5.7.5.

| Attack | Description | # Packets | 2D Caught | 3D Caught | 4D Caught | Snort Caught |
|-----------------|---------------|-----------|-----------|-----------|-----------|--------------|
| 41.162715 | PortswEEP | 10 | Yes | Yes | Yes | Yes |
| 42.145441 | Land | 31 | No | Yes | Yes | No |
| 43.080401 | Satan | 172 | Yes | Yes | Yes | Yes |
| 43.093814 | Imap | 1371 | No | No | Yes | Yes |
| 43.164334 | PortswEEP | 4 | No | Yes | Yes | No |
| 44.080000 | NT Infoscan | 690 | Yes | Yes | Yes | Yes |
| 44.110000 | DosNuke | 53 | No | No | No | No |
| 45.181011 | PortswEEP | 9 | No | Yes | Yes | No |
| 45.192523 | IPSweep | 260 | Yes | Yes | Yes | No |
| 51.083800 | Ping of Death | 5 | Yes | Yes | Yes | Yes |
| 51.085000 | Ping of Death | 3 | Yes | Yes | Yes | Yes |
| 51.102700 | Apache 2 | 59751 | No | No | Yes | Yes |
| 51.142100 | Ping of Death | 3 | Yes | Yes | Yes | Yes |
| 51.180445 | SynFlood | 20480 | No | No | No | Yes |
| 51.194715 | DosNuke | 10 | Yes | Yes | Yes | Yes |
| 51.200037 | UDPStorm | 13 | No | Yes | Yes | No |
| 51.201715 | Self Ping | 137 | No | No | No | No |
| 52.083236 | Teardrop | 25 | No | No | No | Yes |
| 52.094514 | Self Ping | 196 | No | No | No | No |
| 52.130655 | Ping of Death | 892 | Yes | Yes | Yes | Yes |
| 52.165435 | Queso | 10 | Yes | Yes | Yes | Yes |
| 53.110516 | Queso | 10 | Yes | Yes | Yes | Yes |
| 54.145832 | Satan | 10959 | No | Yes | Yes | Yes |
| 54.195951 | Mscan | 173745 | No | Yes | Yes | Yes |
| Detection Rate: | | | 45.8% | 70.8% | 79.2% | 66.7% |

Table 5.1 Lincoln Lab Attacks Test Results

It can be seen from the figure that the higher-dimensional sensors alerted on every attack that was detected by the lower-dimensional sensors, as expected. Additionally, the higher-dimensional sensors detected more attacks than the lower-dimensional ones did, as is shown by the highlighted cells. Some attacks, such as the queso attack, were detected in so many cells of each matrix (not shown) that there seemed no doubt that it was a true positive alert. Other attacks had alerts in so few cells, that there was some question as to if these alerts might be eliminated with more training (potential false positive cells).

Therefore some testing was accomplished to determine the false positive rate of these detectors.

To test the false positive rate of the sensors, they were trained on the same data that was used in the training of the true positive tests—all of Monday and Tuesday of week one of the Lincoln Lab’s data. The sensors were tested against the first 201,000 packets of Wednesday, week one, as in this way they could be tested against attack-free data on which they were not trained. Since each of the attacks shown in Table 5.1 are shown as a detect/not-detect alert for a group of packets, the false positives were tested in the same manner. Thus, the 201,000 attack-free packets were divided into groups of packets, and each group was tested to see if there were any false positives within it. The average size of an attack was about a thousand packets, excluding the three attacks with the most and least number of packets to avoid skewing the data. Therefore, a test size of a thousand packets of normal traffic seemed appropriate for testing the number of false positive alerts in order to make a comparison to the detection rates of Table 5.1. This may give an abnormally high indication of the actual false positive rate of the IDS, however, as the number of anomalous relationships within any 1000-packet group may be small. Therefore, further testing was conducted to split each 1000-packet group into even smaller sections, and the false positive ratios of these sections were also tested. In all cases, the same packets were tested. The only difference was the size of the groups. These results are shown in Table 5.2.

| Version 2.0 | Numb FP | Numb TN | % FP | % TN |
|--------------------------------|---------|---------|--------|---------|
| FP on 201 runs of 1000 packets | 0 | 201 | 0.00% | 100.00% |
| FP on 402 runs of 500 packets | 0 | 402 | 0.00% | 100.00% |
| FP on 2010 runs of 100 packets | 0 | 2010 | 0.00% | 100.00% |
| FP on 20100 runs of 10 packets | 0 | 20100 | 0.00% | 100.00% |
| Version 3.0 | | | | |
| FP on 201 runs of 1000 packets | 12 | 189 | 5.97% | 94.03% |
| FP on 402 runs of 500 packets | 12 | 390 | 2.99% | 97.01% |
| FP on 2010 runs of 100 packets | 11 | 1999 | 0.55% | 99.45% |
| FP on 20100 runs of 10 packets | 9 | 20091 | 0.04% | 99.96% |
| Version 4.0 | | | | |
| FP on 201 runs of 1000 packets | 51 | 150 | 25.37% | 74.63% |
| FP on 402 runs of 500 packets | 60 | 342 | 14.93% | 85.07% |
| FP on 2010 runs of 100 packets | 70 | 1940 | 3.48% | 96.52% |
| FP on 20100 runs of 10 packets | 76 | 20024 | 0.38% | 99.62% |

Table 5.2 False Positive Test Results

Version 2.0 performed spectacularly, with no false positives at all. This was remarkable, as that same version detected 45 percent of the Lincoln Laboratory attacks. Version 3.0 did not perform quite as well, however, as it had from 0.04 to 5.97 percent false positives. Even worse, version 4.0 had from a 0.38 to a 25.37 percent false positive ratio. It can be seen that the larger the group of packets that were tested, the more likely the chance of obtaining a false positive alert. This is because there is a small chance of encountering a false positive with each packet, and the larger groups of packets compound this chance with each packet of the group. Also, the higher the dimensionality of the sensor, the more likely a false positive may be issued. This is likely due to the fact that higher dimensional sensors require more training to eliminate false positives, as discussed on page 5-14. With additional training, using appropriately diverse self-data, the number of false positives would certainly decrease. However, it is also possible that the number of true positives would decrease as well, since the antibodies that detected those attacks might be eliminated during more extensive training.

Nevertheless, as the false positive ratios in Table 5.2 were much smaller than the true positive ratios in Table 5.1 in all cases, this seems to indicate that the number of detections is expected to exceed the number of false positives. Therefore there seems to be some value to intrusion detection via relational schemata.

Another way of analyzing the false positive rate may have been to simply count the number of false positives encountered between every two-packet comparison within all 201,000 packets. This may give a better indication of the true number of false positives encountered, even though a false positive caused by one packet may cause up to five false positive alerts in the six-packet sliding window. However, testing the false positives in this manner involved changing the code used in the rest of the testing. Since the code could be used as-is to conduct the tests of Table 5.2, and since this would still give some indication to the number of false positives in the system, the amount of false positives were tested in the same manner as the rest of the tests with this unmodified code.

5.7 Analysis of Results

Table 5.2 shows a much smaller ratio of false positives than the ratio of true positives in Table 5.1. Therefore the attacks were assumed to have some relationships not commonly encountered in normal network traffic. To determine why the attacks were detected, some analysis was done regarding the attack packets and the cells in which the attack was detected.

5.7.1 Low Hanging Fruit. Many of the attacks that were alerted upon were immediately found to be anomalous upon inspection. Furthermore, the cells in which the IDS alerted these attacks pinpointed the fields in which the anomalies occurred, which might assist the analyst in conducting a post-alert forensic analysis. Those attacks are described in this section.

Attack 41.162715 was a portsweep attack, and was detected in 2D cell (14,15) with the antibody 11. This attack was detected because the TCP sequence numbers (field 15) were zero in all packets of the attack, as is shown in GOPHER in Figure H.7. The TCP sequence numbers of all packets being zero is likely a direct result of the hacker tool used in the attack, and is abnormal as TCP sequence numbers should continually increment because they identify the byte in the stream of data from the sending party to the receiving party [Stev99].

The other two portsweep attacks (attacks 43.164334 and 45.181011) were not the same as attack 41.162715, even though they were still labeled as portsweeps in the Lincoln Lab's Data. They were different because they were targeted at different services, and the TCP sequence numbers of these attacks incremented normally. Thus, these portsweeps were not detected with the 2D sensor, as portsweep 41.162715 was. However, both of these attacks probed the chargen port of the victim with a reset packet, and this is where they were detected. Attack 43.164334 was detected in the 3D cells (7,22,23) and (22,23,25), and attack 45.181011 was detected in cells (0,8,22), (7,22,23), and (22,23,25). This is noteworthy because Mansfield states that "Intrusions are in general characterized by some noise or indication of the intruder groping for a door, trying (unsuccessfully) a key, etc. In the network context these signals may be seen in the TCP reset packets and the ICMP

echo-response or destination/port unreachable packets [Mans00]” That noise was captured here with an alert due to the TCP reset packets. Each of these cells was alerted on due to the reset packet (field 22).

The queso attacks 53.110516 and 52.165435 were alerted on for multiple reasons. Amazingly, they were both alerted in the exact same 55 cells, or the 2D cells (0,17), (0,18), (1,17), (1,18), (2,17), (2,18), (3,17), (3,18), (4,17), (4,18), (5,17), (5,18), (6,17), (6,18), (7,17), (7,18), (8,17), (8,18), (9,17), (9,18), (10,17), (10,18), (11,17), (11,18), (12,17), (12,18), (13,17), (13,18), (14,17), (14,18), (15,17), (15,18), (15,20), (15,23), (16,17), (16,18), (16,21), (17,18), (17,19), (17,20), (17,21), (17,22), (17,23), (17,24), (17,25), (18,19), (18,20), (18,21), (18,22), (18,23), (18,24), (18,25), (20,21), (20,24), and (23,24). It is assumed that both of these attacks were conducted with the same hacker tool, and therefore they had the same type of packets. All of the packets of these attacks either had the same TCP sequence number, or were TCP resets (which also had the same TCP sequence number, but one that was different from the non-reset packets). Additionally, both of these attacks had a packet with the TCP CWR, ECHO, and SYN fields set simultaneously (fields 17, 18, and 23, respectively), and included a simple TCP FIN scan. The TCP sequence number and reset fields have already been shown to be a possible indication of anomalous activity. The CWR and ECHO fields being set simultaneously is a possible indication of the queso attack [Sans99] and a these two fields being set generated alerts in multiple fields. A TCP FIN scan is possibly identified by the TCP FIN and ACK fields being set simultaneously (fields 20 and 24, respectively) [Chap01]. The FIN scan was also not commonly found in training (since it was alerted on), and was picked up in all cells with field 24. The multiple alerts of these two attacks clearly demonstrate the usefulness of this type of intrusion detection, as the alerts that were generated pointed to several packet fields which arose suspicion regarding the attacker’s activities.

The Ping of Death (PoD) attacks, or attacks 51.083800, 51.08500, 51.142100, and 52.130655 were all alerted on with antibody 15 in cell (0,30). This was because the IPID field of these attack packets was static (did not increment). This is abnormal because the IPID normally increments by one each time a datagram is sent [Stev99]. What is interesting about this cell is that the attack was alerted on in not only field 0 (the IPID

field), but also field 30, which is the ICMP Type field. Upon examination of this field, the ICMP type was ECHO, which is a possible indication of a ping sweep [McCl99]. Thus, with one cell, this alert pointed to two different anomalies of this attack. In addition to being detected in cell (0,30), the PoD attacks 51.085000 and 52.130655 also generated alerts in cells (0,3), (1,3), (2,3), (3,4), (3,5), (3,6), (3,7), (3,8), (3,9), (3,10), (3,11), and (3,12). It can be seen that the common denominator of these cells is field 3, or the TCP more fragment field. The PoD attack is a Denial of Service attack where the attacker sends very large ICMP packets in an attempt to crash the victim machine [McCl99]. The packets that were alerted on had an ICMP data length of 1472 bytes, which is rather large, and the more fragment bit set. This might help to identify the traffic as a possible PoD attack.

Attack 51.194715 was a dosnuke attack and was alerted on in cells (12,16), (12,19), (15,19), (19,20), (19,23), (19,24), and (19,25). This is because this attack had a packet with the TCP URG, ACK, PUSH, and FIN set simultaneously (fields 19, 20, 21, and 24, respectively). This is obviously abnormal, and is a possible indication of an attack similar to the TCP XMAS scan [Chap01].

Attack 54.145832 was a Satan scan. This attack was alerted on in the 3D cells (7,10,27), (7,26,27), (10,12,27), (10,27,28), (12,26,27), and (26,27,28). Field 27 is the UDP destination port, and upon inspection of these packets it is immediately recognizable that this is a scan to the UDP source ports. This can also be seen with GOPHER, as is shown in Figure H.3.

The Mscan attack 54.195951 was alerted in the 3D cells (1,9,15), (1,11,15), (1,15,20), (1,15,22), (1,15,24), (2,15,20), (2,15,22), (13,14,22), and (13,15,24). Much like the portsweep attacks discussed previously, the TCP sequence numbers of this attack (field 15) were a cause of these alerts. Strangely enough, unlike the portsweep attacks in which the TCP sequence numbers remained static throughout the attack, the TCP sequence numbers of this attack seemed as likely to increment as to stay the same. Nevertheless, there were enough static sequence numbers to indicate a clear anomaly. Furthermore, upon inspection of these attack packets, it is immediately apparent that a scan is being conducted, as is shown in Figure H.6.

Attack 45.192523 was an IP sweep attack, and was alerted in cells (1,30), (2,29), (2,30), (2,31), and (10,30). The TTL was normal, but the ICMP types were echo requests and replies, as have been alerted on previously. Additionally, the IP do not fragment bit was alerted on when the victims replied to the sweep. Curiously enough, out of the six victims that replied to this attack, only two set the IP do not fragment bit (and generated alerts with the attacker's traffic). Still, since these two replies were compared in a sliding window of six packets, each generated an alert on the five packets above, and the five packets below their reply, generating alerts on 20 packet relationships. This might be enough for an analyst to detect the IP sweep attack, as the sweeps were conducted consecutively. It is immediately clear upon inspection of the attack traffic that this is an IP sweep, as is shown in Figure H.1.

5.7.2 Higher Hanging Fruit. Some of the attacks were alerted in fewer cells or fewer packet relationships, and the post-alert analysis of these attacks did not seem to provide quite as good an indication to the type of attack that was being conducted as the alerts previously seen. While these alerts still indicate some level of anomalous activity (since they were not found in training), the alerts themselves often provided no solid indication of an attack when the packets and cells were inspected. These alerts are discussed in this section.

The Land attack, or attack 42.145441 was alerted on in the 3D cells (1,23,25), (9,23,25), and (11,23,25). Fields 23 and 25 were the TCP SYN and header length fields, respectively, and there seemed nothing anomalous with these fields. However, the single packet of this attack had a Time To Live (TTL) (field 1) about four times greater than the non-attack packets that followed it. This is anomalous because the packets from one node to another are likely to travel the same route [Stev99], and therefore they would likely have the same TTL value. The 3D antibodies picked up this anomaly in cell (1,23,25). This attack had an IP source address which was spoofed to be the same as the IP destination address, and it is interesting to note that the attack was detected in the source MAC (field 9) and destination MAC (field 11) addresses. These fields point out the source and destination MAC address of the attack packet, and thus, indirectly, point to the IP

source and destination addresses. Upon inspection of the alert and this packet it is likely that a network analyst would notice that the IP source and destinations are the same—the definition of a Land attack [McCl99]. However, the cells of this alert do not, in themselves, provide a concrete indication of the attack.

The NT Infoscan attack, or attack 44.080000, was alerted on in cells (0,2), (0,20), and (0,22). The common denominator of these cells was field 0, or the IPID field. However, upon inspection of the attack packets, it was noticed that nearly all of the IPIDs incremented normally. One instance where the IPID field did not increment was picked up by the IDS in these three separate fields. This instance occurred between packets number 6 and 10. Packet 6 had the do not fragment bit (field 2) set, and was an ACK (field 20). Packet 10 had the same IPID number, and was a reset reply (field 22) to packet 6. These two packets, and these fields did not seem to provide any clear indication of the attack. It should be noted that the IPID does not always increment in normal traffic. Indeed, this can be demonstrated because when the same IPID was encountered between packets 6 and 10, only 3 cells were alerted on, not the rest of the cells in the zero row. While the IPID not incrementing, and the TCP reset and IP do not fragment bits being set are possible indications of an attack, it was difficult to notice that an attack was being conducted solely by these alerts.

The first packet of the Satan attack 43.080401 was an ICMP echo request packet sent to the victim. The second packet was an ICMP echo reply from the victim to the attacker. This attack was alerted in only cell (1,30) from only these two packets. These ICMP types are possible indications of ping sweeps [McCl99], and are possible indications of anomalous activity in general [Mans00], so this alert might provide some indication of an attack. However this one alert based only on two packets of traffic did not seem to provide as clear an indication of the attack as some of the other alerts previously discussed.

The rest of the attacks that were detected were alerted in cells that provided some indication of an attack, but only a slight indication. These types of alerts might easily be overlooked as false positives. These alerts were generated by ICMP echo request/replies, TCP resets, IP do not fragment bits, and like TCP sequence numbers similar to the alerts of the previously discussed attacks. However, often this was only slightly anomalous and

was picked up only by 3D or even 4D sensors in very few fields by a very few packets. With these remaining attacks, no clear-cut means of identifying that an attack was in progress could be found simply by looking at the alert cells and the packets that generated them. While these alerts were not false positives, they may cause the analyst headaches trying to figure out why an alert was issued.

5.7.3 Unreachable Fruit. Some of the attacks went undetected. These attacks were the dosnuke attack 44.10000, the SYN flood attack 51.180445, the self ping attacks, and the teardrop attack. Upon inspection of the attack packets, both with GOPHER and with Ethereal, no anomalous traffic stood out. Therefore these packets were overlooked by the IDS. The fact that these attacks were ignored by the IDS demonstrates that this method of relational intrusion detection will not detect all attacks. It is merely another layer in the defense to detect attacks that may go unnoticed by other means of intrusion detection.

5.7.4 Fruit With Worms. One attack, the UDPStorm attack 51.200037, was alerted upon in the 3D sensors in cells (1,8,26), (1,8,27), (1,8,28), (8,9,11), (8,9,26), (8,9,27), and (8,9,28). Inspection of the packets of this alert showed that indeed, the alert was generated from the attack packet and a normal UDP packet close to it. However, this alert was also generated on that same normal UDP packet and another (normal) UDP packet within the 13 packets of the attack file. This means that the “anomalies” picked up in those fields are seen in normal network traffic, they were just not detected during training. Therefore, although this alert pointed to the attack packet and might assist an analyst in detecting this attack, it would also issue false positives that the analyst would have to put up with. With more training the antibodies that generated this alert would likely be eliminated, but then this attack would go undetected. Thus, the choice of the analyst is to endure false positives to detect this attack, or risk missing the attack to reduce the false positive rate of the sensor.

5.7.5 Comparison To Snort. The relational IDS of this research performed somewhat comparable to Snort version 1.8.1 when the 2D, 3D, and 4D sensors are considered.

It did detect some attacks that went undetected by Snort. For example, the Land attack was detected due to an abnormal TTL, and MAC addresses, as discussed previously. This attack, however, was not detected by SNORT. Similarly, the reset to the CHARGEN port in attacks 43.164334 and 45.181011 and the IP Sweep attack 45.192523 was alerted upon by the relational sensors of this research, but went undetected by Snort. This demonstrates that this system can detect some attacks that are missed by other IDSs.

Still, Snort detected 66.7 percent of the attacks, and it must be kept in mind that when Snort detected these attacks, it provided a description of the possible attack to the analyst, which is very beneficial. Snort even detected an attack that went undetected by all the relational sensors of this research—the Teardrop attack (and Snort identified it as a Teardrop attack). Additionally, when attacks were not detected by the 2D sensors, and were only detected by the 3D or 4D sensors of this research, there was much less of a confidence in the alerts. Therefore, although the 4D sensors surpassed Snort in detection capability, the confidence level of these detections was not as high as the confidence level of the Snort alerts. Thus, additional work is likely required before the IDS posed in this research would be adopted by most network defenders.

5.7.6 Discussion of Attacks. Table 5.1 shows that the LAND attack was not detected in version 2.0, but was detected in versions 3.0 and 4.0. Much like the ARP Spoof attack described in Section 5.3, it is likely that this attack was not detected with the two-dimensional sensors because the IDS was trained with traffic from *outside* the firewall. Therefore, the 2D antibodies that detect the IP addresses of two packets being equal, but the MAC addresses not being equal were eliminated during training. It is certainly possible that this attack was alerted in antibodies in version 3.0 which might be eliminated during more extensive training, and the attack would be missed even by version 3.0. Conversely, it was assumed that the attack would be detected even in the 2-D version when the IDS was trained and tested *inside* the firewall. This turned out to be the case, as this attack was detected in the 2D cell (8,9) when the IDS was trained on 200,000 packets from Monday week one of the Lincoln Data taken inside the firewall. Again, the placement of the IDS may affect its performance.

Some of the attack packets were excluded from the filtered attack data during the filtering process, either due to limitations of the TCPDump filter, or erroneous data in the Lincoln Labs database of attacks. During filtering TCPDump sometimes gave “Bad Packet” errors, and dropped packets from the data. It is clear that the actual network did not drop these attack packets, as they were present in the data file. Therefore, these packets should have been included in the attack file data sent through the relational IDS sensor. Example of this situation is the Land attack 55.163447, the TCP Reset attacks 53.092039 and 52.081109 and the Reset Scan 54.170132 of the Lincoln Lab’s data. Multiple attempts were made to filter these attacks from the data, but every time the TCPDUMP filter dropped packets until there was no attack traffic left to test. Moreover, the same packet problems that were detected by TCPDump (and caused packets to be dropped) may have caused alerts with the IDS. Therefore, some attacks that were tested and missed might have actually not been missed, had all of the attack packets been present. The limitation of the TCPDump filtering may have actually caused the tests to perform more poorly than they would have in a real network environment.

A point worth mentioning is that the antibodies of version 2.0 detected some attacks that contained only a few attack packets after filtering. For instance, the Ping of Death attacks 51.14210 and 51.085000 had only three packets each after filtering. Therefore, there were no normal packets with which to compare these packets, yet they were still detected. On the other hand, attack 54.195951 had 173,745 packets, and it was not detected with the same 2D sensors. This shows that the relationships between the packets is much more important than the number of packets in the attack. Since the relationships, not the number of packets, are what matters, the relational detection schemata of this research might be one way to detect some “low and slow” attacks.

Indeed, one of the most common methods of “low and slow” attacking is to spread out a scan or send multiple probes over an extended period of time. Interestingly, many of the attacks that were detected by these code versions were scans or probes. Therefore, this research may be one step closer to providing an additional solution to the “low and slow” problem beyond the typical counter mechanism.

To that end, although this research poses a relational IDS sensor, and it is therefore a multiple-packet sensor, that does not mean that this sensor cannot detect single-packet attacks. For example, the UDP Storm, attack number 51.20037 of the Lincoln Lab's data, included only one attack packet. However, when it was compared to the normal network packets around it, an alert was generated. This is one example of how a multiple-packet sensor can detect even single-packet attacks.

The multiple-packet relational sensors of this research might even be useful in other areas of network activity analysis. For instance, they may help to point out unknown mechanical failures, or poorly configured equipment, as the network traffic generated in these instances may lie outside the previously defined (healthy) self.

5.8 *GOPHER Use*

A secondary goal of this research was to develop a graphical tool to assist the operator in determining if IDS alerts are truly attacks, or merely false positives. GOPHER became that tool. In fact, during this research GOPHER was used to visualize several attacks in order to better understand them and the network traffic around them. In effect, GOPHER was demonstrated to be useful as the tool it was designed to become. This is because GOPHER often allowed for a quicker understanding of vast numbers of network packets than is possible with textual-based programs. In fact, when a certain aspect of network packets was sought, such as packets with a field value in a certain range, GOPHER often presented this information in near-real time. This is because the slider filtering allowed for vast volumes of packets to be waded through almost instantaneously. For instance, while analyzing the MitM attack discussed in Section 4.1 to determine why it was missed in training, GOPHER was used to quickly sort through mounds of networks packets. With the use of a single slider, thousands of packets were whittled down to a handful of packets with the reset field set, which could then be easily examined to determine why the attack was missed. While the fields of GOPHER did not evolve to match the code as the code evolved, GOPHER still visualized a few attacks very well; so much so in fact, that the visualization techniques of GOPHER might be of significant value if incorporated into

an IDS itself, as well as in network packet examination programs such as Ethereal or TCPDump.

For example, many research-related attacks, which were conducted by other colleagues in the AFIT Information Systems Security and Assurance (ISSA) track, were examined during this research. When viewed with GOPHER, the network data in some of these attack files could quickly and easily be seen to be abnormal. As an example, Figure 5.4 shows one of these attacks, sniffed from a network during a Syn Flood. Even without using GOPHER’s sliders to filter the data, it is immediately apparent that the visualized network data is abnormal—there are three large spikes present. This is because the attacker sent the same SYN probe over and over again from multiple ports of his machine to one port of the victim machine. The three spikes depict IP identification number, multiple TCP source ports, and a continually increasing TCP sequence number. These three spikes of attack data would be prominent, even intermixed among more chaotically represented normal network activity, which would be similar to Figure 4.6 of Chapter IV.

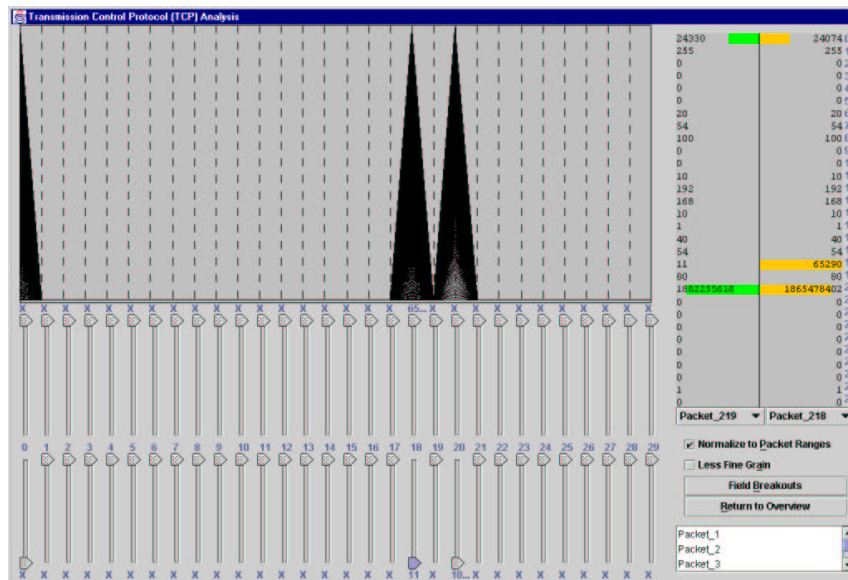


Figure 5.4 SYN Flood Depicted by GOPHER

Another example, this time with a Lincoln Lab’s attack from Figure 5.1, is depicted in Figure 5.5. This figure shows the first 238 packets of attack 51.180445 visualized with GOPHER. Textually, it may be difficult to discern that there was an attack taking place,

or what the attack consisted of. But graphically, it is instantly apparent that there are many TCP source ports all probing one of twelve TCP destination ports. In the rest of the attack this pattern of activity was repeated over and over again, cycling through the TCP destination ports. Therefore, when all 20,480 packets of the attack are loaded, GOPHER shows a sea of black in the TCP source and destination ports, revealing that thousands of ports are being scanned.

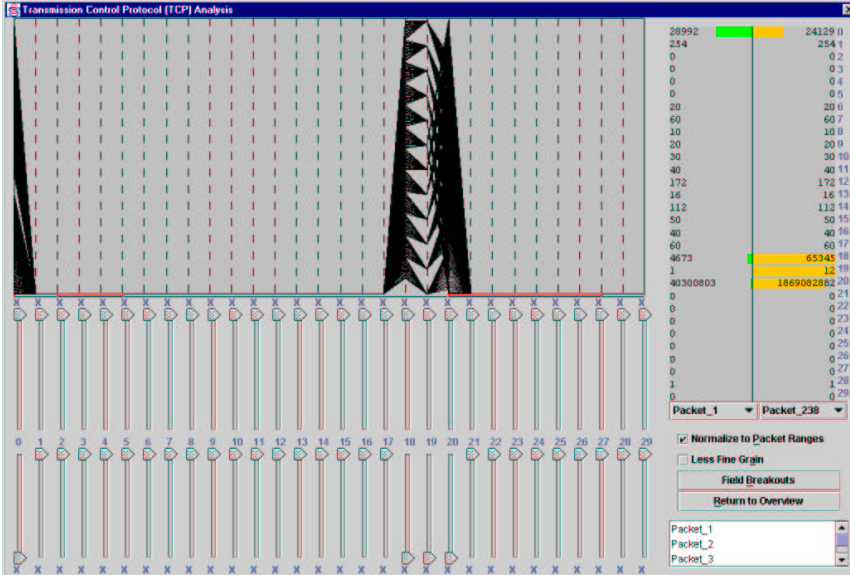


Figure 5.5 Attack 51.180445 Visualized With GOPHER

Visually depicting scans was one of GOPHER's strengths, as it shows divergence from and convergence to a point very well. Other scans which were immediately apparent when viewed in GOPHER were the IP Sweep in Figure H.1, the Satan scan in Figure H.2 and the Mscan in Figure H.6. With these attacks GOPHER presented the attack to the analyst in a visual manner, allowing unwanted packets to be filtered if desired. An example of this is depicted in Figure H.5. Here, GOPHER's sliders were used to filter traffic coming only from a few TCP source ports. This provides a visual slice of the scan, and shows how this Satan scan swept through groups of six TCP destination ports from a smaller subset of TCP source ports. This type of visualization and filtering may assist in better understanding the attack.

Two of the other capabilities of GOPHER are concentrating on a specific protocol, and showing when packets were similar or different. For example, in the Ping of Death attack in Figure H.8 there were 892 packets in the filtered data file. The Ping of Death is an ICMP attack, and the non-ICMP packets were simply normal traffic included by the filter. However, there were only 17 ICMP packets, intermixed among 875 TCP and UDP packets. The IDS alerted on the ICMP attack packets, so this was what was desired to inspect. With a simple click GOPHER drilled down to view only the 17 ICMP packets. Additionally, among these 17 ICMP packets, one packet was repeated 13 times, and another was repeated twice. Therefore these 17 packets looked like only two packets. This might indicate an anomaly—to the analyst that knows that the data consists of 17 packets. The problem with this is that if the analyst does not know that one line is depicting more than one packet, he may not notice anything anomalous. Therefore, the parallel coordinates system of GOPHER can be a double-edged sword. It can serve to point out repeated packets (often an indication of an attack), by showing fewer lines than the number of packets loaded, and it can also obscure repeated packets from consideration. One way of preserving the former while mitigating the latter might be by encoding an intensity factor into each line. For example, a line may grow darker each time it is repeated.

While analyzing the Lincoln Lab's attacks, GOPHER was used to visualize the attacks to better understand them or to look for anomalies. In a few instances GOPHER pointed out that the data files of the "attacks" that were being tested contained no attack packets at all. For instance, the Reset Scan attack 54.170132 of the Lincoln Lab's data was originally filtered from the Thursday week 5 data based on the time and IP source and destination addresses provided in the Lincoln Lab's database of attacks [LLab99]. After filtering, the data file contained 2037 packets. When tested along with the other Lincoln Lab's attacks, no alerts were generated with the 2D or 3D sensors. This was unexpected since many of the other attacks were detected because of TCP reset packets. When viewed by GOPHER it was noticed that there were no TCP resets in the 2037 packets of the data file—unusual for a Reset Scan. Upon examination it was noticed that this file contained only the victim communicating, and all of the attack packets were eliminated from the data by the TCPDUMP filter. Therefore this file was removed from consideration in the

performance of the relational IDS sensors, as there were no attack packets to test. This did, however demonstrate the potential effectiveness of GOPHER's slider filtering once again.

One additional slider that should be added to GOPHER to assist in network packet analysis is a time slider. This slider could be located on the bottom of the screen, and could filter out packets temporally. This may prove very useful if the user wishes to view only a specific time segments of the traffic, which is often the case in network packet analysis.

The figures in this section and Appendix H demonstrate that anomalies in network traffic can be visualized well via a parallel coordinate systems such as GOPHER. If GOPHER were to maintain a history of network packets, then when an IDS alerted on a potential attack, the network analyst might use GOPHER to dig through the network traffic more quickly. This may make his job easier as well as keep the network more secure.

VI. Findings and Conclusions

The previous chapters discussed how schemata representing multiple-packet relationships can be used in the context of a computer immune system to detect both known and unknown network intrusions. They also discussed how different types of schemata may be used to create different genes, changing the speed and effectiveness of the detection mechanism. The benefits and limitations of this type of intrusion detection are discussed in this chapter, along with a brief tête-à-tête on GOPHER as a method for multiple packet visualization. Finally, the chapter concludes with some suggestions for future study.

6.1 Benefits

6.1.1 Relational Schemata. Discussed in this research were two different relational schema, and two methods of mapping these schemata to genes. The 0/1 schemata of equal/not-equal relationships have the advantage of speed and ease of understanding, however they may miss some attacks. Two-bit schemata were used to represent equal, greater-than, and less-than relationships to provide a packet ordering mechanism. It was demonstrated how packet schemata could be mapped in a direct, and in a crosschecking manner to produce different intrusion detectors. Each of these detection mechanisms varied in their makeup, and therefore they varied in their intrusion detection capability as well. Each may someday provide its own layer in the defense-in-depth paradigm, closing the gap in finding malicious network activity.

6.1.2 More Difficult Detection Avoidance. Intrusion detection and intrusion avoidance are never-ending battles. The good guys put up massive walls of defense, only for the bad guys to search for and exploit any crack in that defense. In turn, the good guys patch that hole (or put up another wall), only to have the bad guys find another flaw to take advantage of. One problem with signature-based detection is that known attacks may be modified only slightly to slip past the sensor. Although anomaly detection makes sidestepping detection harder, it can still be done with single-packet anomaly detectors. The complexity of avoiding detection becomes much harder, however, when the attacker is trying to avoid multiple-packet *relational* sensors. This is because the attacker must

now consider not only the packets of his attack, but also how those packets relate to one another, as well as how they relate to all possible surrounding normal network traffic.

6.1.3 CIS Framework. The CIS framework addressed in this research provides a foundation for understanding and expressing both the concepts of self and of non-self. By expressing self in terms of genes and chromosomes, and non self in terms of genes and antibodies, these structures may fit into preexisting immune system models of intrusion detection. This may serve to bring relational antibodies into the traditionally non-relational immune system intrusion detection paradigm, and thereby increase the intrusion detection effectiveness of the overall immune system. Along with detecting attacks, as an anomaly detector, this framework may assist in detecting other network disturbances as well, such as mechanical failure or poorly configured equipment.

6.1.4 Low and Slow Attacks. This research provided a mechanism for detecting multiple-packet attacks beyond a simple counter mechanism, which is currently the primary engine in detecting attacks such as scans and probes. Since a counter depends on checking x packets in a t time window, it is easily defeated by “low and slow” attacks. Many scans and probes were detected in the testing done in Chapter V, and this suggests that the packets of scans and probes may have anomalous relationships. Since the relationships between packets depends more on the values of the fields than the number of packets, this research may assist in the development of a more robust “low and slow” detection apparatus. On the other hand, some “low and slow” attacks may only be detected by relationships strictly between attack packets (and not by relationships between attack packets and normal network traffic). If these attacks are perpetrated such that no attack packets occur close to one another temporally, then the window size of this research may significantly limit their detection.

6.1.5 GOPHER. The parallel coordinates system utilized by GOPHER provided a uniquely useful way to view multiple packets simultaneously, while also allowing for the filtering of traffic according to user-defined criteria. This may greatly increase the speed and depth at which a user can understand network activity beyond what may be possible

with current, textual-based programs. While the visualization methods used in GOPHER took a backseat to the relational CIS used in intrusion detection, they may actually turn out to be just as beneficial. If these techniques were incorporated into existing network analysis tools such as TCPDump or Ethereal, an analysts might be able to sort through alerts much quicker, weeding the true positives from the false positives. GOPHER may itself play a significant role in providing more secure networks.

6.2 *Limitations*

6.2.1 Packet Payload. One limitation to the research posed is that it is limited to non-payload attacks. By not considering packet payload, a vast number of attacks are overlooked, and therefore will likely go unnoticed by this means of intrusion detection. Therefore, this research is not designed as a stand-alone effort in ID, but rather as another layer in an enterprise-level defense-in-depth effort. This is not to say that this method of ID cannot detect attacks that occur in the payload portion of an attack. For instance, a victim's response to an attack may create abnormal relationships in normal network traffic (such as an ACK storm). It is more that detecting attacks that involve strictly the payload portion of the packet is much less likely.

6.2.2 Alert Confirmation. Unlike signature-based detection, one of the most common limitations to anomaly-based detection, such as the relational antibodies in this research, is determining what caused an alert. Since the antibodies of the CIS only show abnormal relationships between packets, a network forensics effort must be undertaken to determine why the alert was issued. Even considering that this research can provide two packets, cell information (in the form of fields compared), and the relationships between these fields, alert confirmation might still be a timely and grueling task, as it sometimes was in this research. This is why the GOPHER project was undertaken. With the use of GOPHER, determining why an alert has been issued in the CIS model of ID might be a less costly in terms of time and effort, because GOPHER provides another tool to the network data analysis toolbox.

6.3 *Future Research Opportunities*

This research opened a door to using the relationships between packets, not necessarily the packets themselves, as a means of intrusion detection. In doing so, it also opened the door to a multitude of future research possibilities. Some of the more lucrative areas of future research are discussed here. This is by no means a complete list. It does, however, present some areas that may be of benefit in intrusion detection.

6.3.1 Expansion of the CIS Model. Many of the aspects of the CIS ID model mentioned in Chapter II have not been implemented in this proof-of-concept research. For example, the antibodies of this research were only conceptually implemented, and therefore had a permanent lifetime. Since the representation of self changes with time, antibodies should also be able to change in order to protect the environment. Therefore, antibodies should have a limited lifespan and should reproduce and die, changing along with self. Another, perhaps very lucrative method of expansion, is the addition of costimulation to the research mentioned here. This is perhaps very lucrative because nearly all attacks were detected in multiple cells at once. The queso attack, for instance, was detected in nearly 60 cells in the 2-D version, in 800 cells in the 3-D version, and in about 17,000 cells in the 4-D version. The number of cells with which an alert is issued might be a very easy, and potentially useful way of eliminating false positives through costimulation.

6.3.2 Statistical ID. In training a CIS on normal data, the relationships between fields which might occur very infrequently, yet were still seen in training, were eliminated as detectors. Therefore, when these rare relationships were seen during testing, they were overlooked as possible alerts. Using relational statistics may be one very important method of ID. For example, if a statistical history of packet relationships was collected, then a sensor could be built that could detect attacks because of an abnormal number of these rare cell relationships. What may prove very beneficial with this sensor is that it may easily incorporate a “dial” mechanism, where the network analyst could tune the dial to make the IDS more, or less sensitive. This is advantageous because many of today’s IDSs are an “all or nothing” effort, and are not easily adjustable according to the network threat level.

A sensor using statistical data of relational patterns can overcome this problem, providing more, or fewer alerts as specified by the network analyst.

6.3.3 Developing Relational Signatures. The relational sensor mechanism posed does not have to be part of an anomaly detector, as was done here. Once inappropriate relationships (those that occur only in known attacks) are found, they could be incorporated into existing signature-based detection systems. For example, the source IP address equals the destination IP address relationship, which is a possible indicator of the Land attack, might be fed into existing signature-based IDSs.

Even as the higher-dimensional matrices become too computationally expensive to examine all x -dimensional relationships, a few select cells from this matrix may be useful for incorporation into signature-based detection. In this way, without examining every relationship in an x -dimension matrix, some of the benefit of high-dimensional relational detection can be achieved without the added cost.

Researching how anomalous relationships of known attacks could be converted to signatures for existing IDSs was not done here, as this research posed a method of detecting unknown attacks, not pinpointing the anomalous relationships among all known attacks. However, as more study is conducted in this area, the anomalous relationships posed by many attacks may be collected into a database of relational signatures, which could be one more means of detecting known attacks. In this way, even if the attacks were modified slightly to slip past the other signatures, they may still be picked up by the relational signatures. This not only provides an additional layer of security, it also supplies the potential for detecting and identifying new, as well as modified, attacks.

6.3.4 Relational Host-Based ID. The majority of research presented here dealt with network-based ID, not data specific to any one host. That is not to say that the network-centric ID posed cannot be extended to host-based ID as well. For instance, the genes of this research could be modified to examine relationships between host-specific system processes, vice examining relationships between network packets. One example of this might be to have the equal gene be mapped to system processes or file accesses that

occur concurrently, the greater-than gene could be mapped to one action occurring before another, and the less-than gene could be mapped to one action occurring after another. In this way the relational genes could become temporal vice numerical sensors.

6.3.5 Relational Schemata Changes. This proof-of-concept research included only equal, not-equal, greater-than, and less-than relational schemata between two packets at a time. Further research might extend these relationships to include other nominal, ordinal, or even quantitative relationships. Additionally, relationships may be user-defined or span more than two packets, and other crosschecking field schemata. The type of relationships used in a CIS is only limited to the language with which the packets and fields are represented, and the imagination of the programmer.

6.3.6 Statistical Lack of Relationships as an Indicator. Another method of ID is to use the statistically abnormal *lack* of relationships. For example, in normal network TCP activity there are x handshakes per y packets of network traffic. If t is some statistical threshold (for example 2 percent) and the relationships that would normally occur during these handshakes do not happen in $\frac{x}{y} - t$ percent of time, then there may be something wrong with the network (such as a denial of service flood). Just as an abnormally high amount of relationships may be an indicator of something amiss, so might an abnormally low amount of relationships.

6.3.7 Stochastic Packet or Field Selection. This research deterministically examined all relationships between packets in a sliding window of six packets. This was just a foundational starting point for this research, and the size of this sliding window may be adjusted to more quickly or effectively detect intrusions. Additionally, this deterministic method of field and packet selection might be traded, instead, for a stochastic packet-picking, or field-picking method. If packets, or fields, could be selected in a stochastic, vice deterministic manner, they might better be able to address the “low and slow” problem without being as limited to time or memory constraints.

6.4 Concluding Remarks

This chapter gave some of the benefits and limitations of using relational-schemata, in the context of a Computer Immune System, to detect multiple-packet network intrusions. It also discussed the use of parallel coordinates in an interactive manner, to visualize network traffic. This may allow the user to more quickly and efficiently identify the source of an IDS alert, as well as provide for a more comprehensive understanding of network activity as a whole. The research as a whole demonstrates, via in-depth examples and descriptions, the potential benefits of both relational schemata and GOPHER in intrusion detection.

Appendix A. Matrix of Code Version 1.0

| Training Data is Tuesday, Week One 100000 packets | | | | | |
|---------------------------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| IPSc (0) | IPDst (1) | TCPSrcPort (2) | TCPDstPort (3) | TCPSeq# (4) | TCPNextSeq# (5) |
| IPSc (0) | 9, 15, 6, 14, 11, 7, 13, 5, 10 | 9, 15, 6, 5, 13, 14, 11, 7, 10 | 10, 15, 5, 9, 13, 14, 6, 7, 11 | 10, 13, 5, 15, 6, 14, 9 | 11, 13, 9, 5, 15, 10, 14, 6, 7 |
| IPDst (1) | | 5, 15, 10, 9, 13, 14, 6, 7, 11 | 6, 15, 9, 5, 13, 14, 11, 7, 10 | 6, 13, 9, 15, 10, 14, 5 | 7, 13, 5, 9, 15, 6, 14, 10, 11 |
| TCPSrcPort (2) | | | 6, 15, 9, 5, 7, 13, 11, 14, 10 | 6, 13, 9, 15, 5, 14, 10, 7, 11 | 7, 13, 5, 9, 15, 6, 14, 10, 11 |
| TCPDstPort (3) | | | | 10, 13, 5, 15, 6, 14, 9, 7 | 11, 13, 9, 5, 15, 10, 14, 6, 7 |
| TCPSeq# (4) | | | | | 11, 5, 9, 15, 13, 10, 6, 7, 14 |
| TCPNextSeq# (5) | | | | | |
| ACK (6) | | | | | |
| PUSH (7) | | | | | |
| RST (8) | | | | | |
| SYN (9) | | | | | |
| FIN (10) | | | | | |

| Test Data Joncheray's Attack | | | | | |
|------------------------------|-----------|--------------------------------|--------------------------------|-------------------------|----------------------------|
| IPSc (0) | IPDst (1) | TCPSrcPort (2) | TCPDstPort (3) | TCPSeq# (4) | TCPNextSeq# (5) |
| IPSc (0) | 6, 15, 9 | 5, 15, 10, 11, 13, 7, 14, 6, 9 | 6, 15, 9, 11, 14, 7, 13, 5, 10 | 5, 13, 15, 14, 10, 9, 6 | 7, 5, 15, 13, 11, 14, 10 |
| IPDst (1) | | 9, 15, 6, 7, 13, 11, 14, 10, 5 | 10, 15, 5, 7, 14, 11, 13, 9, 6 | 9, 13, 15, 14, 6, 5, 10 | 11, 9, 15, 13, 7, 14, 6 |
| TCPSrcPort (2) | | | 6, 15, 9 | 5, 13, 15, 14, 10 | 7, 5, 15, 13, 11, 14, 10 |
| TCPDstPort (3) | | | | 9, 13, 15, 14, 6 | 11, 9, 15, 13, 7, 14, 6 |
| TCPSeq# (4) | | | | | 7, 5, 13, 11, 6, 10, 15, 9 |
| TCPNextSeq# (5) | | | | | |
| ACK (6) | | | | | |
| PUSH (7) | | | | | |
| RST (8) | | | | | |
| SYN (9) | | | | | |
| FIN (10) | | | | | |

| | | | |
|--------------------------|---------|--------|----------------------|
| not equal at [4,6] [4,8] | Row | Column | Nine Possible Values |
| | 01 = 4 | 01 = 1 | 0101 = 5 |
| | 10 = 8 | 10 = 2 | 0110 = 6 |
| | 11 = 12 | 11 = 3 | 0111 = 7 |
| | | | 1001 = 9 |
| | | | 1010 = 10 |
| | | | 1011 = 11 |
| | | | 1101 = 13 |
| | | | 1110 = 14 |
| | | | 1111 = 15 |

| | |
|---------------------------------------------------------------|--|
| Caught Packets: | |
| 5 1010100100004 1010100100002 1761 23 2057753892 0 0 0 1 0 0 | |
| 8 1010100100004 1010100100002 1761 23 2057753892 0 1 0 0 0 0 | |
| 5 1010100100004 1010100100002 1761 23 2057753892 0 0 1 0 0 0 | |
| 10 1010100100004 1010100100002 1761 23 2057753892 0 1 0 0 0 0 | |

Figure A.1 Less-than, Greater-than, and Equal Relational Matrix and the Joncheray MitM Attack

| | ACK (6) | PUSH (7) | RST (8) | SYN (9) | FIN (10) |
|-----------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| IPSrc (0) | 11, 15, 7, 10, 5, 13, 14, 6, 9 | 11, 13, 9, 5, 15, 10, 7, 14, 6 | 11, 15, 7, 13, 9, 14, 6, 5, 10 | 10, 14, 7, 15, 11, 9, 5, 6, 13 | 11, 15, 7, 13, 5, 9, 10, 14, 6 |
| IPDst (1) | 7, 15, 11, 6, 9, 13, 14, 10, 5 | 7, 13, 5, 9, 15, 6, 11, 14, 10 | 7, 15, 11, 13, 5, 10, 9, 6, 14 | 6, 14, 11, 15, 7, 5, 9, 10, 13 | 7, 15, 11, 13, 9, 5, 6, 14, 10 |
| TCPSrcPort (2) | 7, 15, 11, 6, 9, 13, 5, 10, 14 | 7, 13, 5, 9, 15, 6, 11, 14, 10 | 7, 15, 11, 13, 5, 10, 9, 14 | 6, 14, 11, 15, 7, 5, 9, 10, 13 | 7, 15, 11, 13, 9, 5, 6, 14, 10 |
| TCPDstPort (3) | 11, 15, 7, 10, 6, 5, 13, 14, 9 | 11, 13, 9, 5, 15, 10, 7, 14, 6 | 11, 15, 7, 13, 9, 14, 6, 5, 10 | 10, 14, 7, 15, 11, 9, 5, 6, 13 | 11, 15, 7, 13, 5, 9, 10, 14, 6 |
| TCPSeq# (4) | 11, 7, 15, 10, 5, 6, 9, 14 | 11, 5, 9, 15, 13, 10, 7, 6, 14 | 11, 7, 15, 5, 9, 13, 10, 6 | 10, 6, 7, 15, 11, 9, 5 | 11, 7, 15, 5, 9, 10, 6, 13, 14 |
| TCPNextSeq# (5) | 15, 7, 11, 10, 14, 13, 5 | 15, 5, 10, 11, 7, 6, 9 | 15, 7, 11, 9, 13, 14, 6 | 14, 6, 7, 15, 11, 9, 13 | 15, 7, 11, 9, 14, 13, 6, 5, 10 |
| ACK (6) | | 15, 13, 14, 10, 11, 7, 5 | 15, 11, 7, 9, 14, 6, 13 | 14, 15, 9, 13, 7, 6, 11, 5, 10 | 15, 13, 11, 14, 10, 7, 5 |
| PUSH (7) | | | 15, 7, 11, 9, 13, 14, 6 | 14, 6, 7, 15, 11, 9, 13 | 15, 7, 11, 9, 14, 13, 6, 5, 10 |
| RST (8) | | | | 14, 15, 13, 7, 9, 11, 6 | 15, 13, 14, 7, 6, 11, 9 |
| SYN (9) | | | | | 11, 15, 13, 7, 14, 6, 9 |
| FIN (10) | | | | | |

| | ACK (6) | PUSH (7) | RST (8) | SYN (9) | FIN (10) |
|-----------------|----------------------------|----------------------------|--------------------------|--------------------------------|--------------------------------|
| IPSrc (0) | 7, 6, 15, 14, 11, 9, 13 | 7, 5, 15, 13, 11, 14, 10 | 7, 5, 15, 13, 11, 14, 10 | 6, 7, 15, 13, 9, 11, 14 | 7, 15, 11, 9, 13, 5, 6, 14, 10 |
| IPDst (1) | 11, 10, 15, 14, 7, 5, 13 | 11, 9, 15, 13, 7, 14, 6 | 11, 9, 15, 13, 7, 14, 6 | 10, 11, 15, 13, 5, 7, 14 | 11, 15, 7, 5, 13, 9, 10, 14, 6 |
| TCPSrcPort (2) | 7, 6, 15, 14, 11, 9, 13 | 7, 5, 15, 13, 11, 14, 10 | 7, 5, 15, 13, 11, 14, 10 | 6, 7, 15, 13, 9, 11, 14 | 7, 15, 11, 13, 5, 9, 10, 14, 6 |
| TCPDstPort (3) | 11, 10, 15, 14, 7, 5, 13 | 11, 9, 15, 13, 7, 14, 6 | 11, 9, 15, 13, 7, 14, 6 | 10, 11, 15, 13, 5, 7, 14 | 11, 15, 7, 13, 9, 5, 6, 14, 10 |
| TCPSeq# (4) | 7, 6, 15, 10, 11, 9, 13, 5 | 7, 5, 13, 11, 6, 10, 15, 9 | 7, 5, 15, 11, 10, 14 | 6, 7, 15, 9, 11 | 7, 15, 11, 9, 5, 10, 6, 13 |
| TCPNextSeq# (5) | 15, 7, 14, 10, 11, 13 | 15, 5, 10 | 15, 7, 13, 9, 11, 14 | 14, 6, 15, 7, 13, 11, 9 | 15, 7, 11, 13, 9, 14 |
| ACK (6) | | 15, 13, 11, 10, 14, 7 | 15, 9, 11, 14, 6, 7 | 14, 10, 11, 15, 9, 13, 5, 7, 6 | 15, 11, 7, 13, 14 |
| PUSH (7) | | | 15, 7, 13, 9, 11, 14 | 14, 6, 15, 7, 13, 11, 9 | 15, 7, 11, 13, 9, 14 |
| RST (8) | | | | 14, 6, 15, 7, 13, 9, 11 | 15, 7, 11, 13, 14 |
| SYN (9) | | | | | 11, 15, 7, 13, 9, 14 |
| FIN (10) | | | | | |

(4,6) 13 = TCPSeq# equal, ACK less-than
(4,8) 14 = TCPSeq# equal, RST greater-than

Appendix B. Matrix of Code Version 2.0

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|----|-------------------|------------------------------|----------------------------------|-------------|-------------|-------------|-------------|----------------------------------|
| | IPID | IP TTL | DontFrag | MoreFrag | UnusedFlag | FragOffset | IPHrdLength | IPPacketLength |
| 0 | IPID | [9, 11, 6, 7, 5, 10, 14, 13] | [11, 9, 7, 5, 6, 10, 15] | [11, 7, 15] | [11, 7, 15] | [11, 7, 15] | [11, 7, 15] | [10, 11, 5, 6, 9, 7, 14, 13, 15] |
| 1 | IP TTL | 0 | [7, 15, 5, 11, 13, 10, 14, 9, 6] | [7, 15, 11] | [7, 15, 11] | [7, 15, 11] | [7, 15, 11] | [6, 15, 14, 9, 10, 5, 11, 13, 7] |
| 2 | DontFrag | 0 | 0 | [15, 7, 11] | [15, 7, 11] | [15, 7, 11] | [15, 7, 11] | [14, 15, 6, 13, 9, 5, 7, 10, 11] |
| 3 | MoreFrag | 0 | 0 | 0 | [15] | [15] | [15] | [14, 15, 13] |
| 4 | UnusedFlag | 0 | 0 | 0 | 0 | [15] | [15] | [14, 15, 13] |
| 5 | FragOffset | 0 | 0 | 0 | 0 | 0 | [15] | [14, 15, 13] |
| 6 | IPHrdLength | 0 | 0 | 0 | 0 | 0 | 0 | [14, 15, 13] |
| 7 | IPPacketLength | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | IP Src | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | Src MAC | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | IP Dst | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | DST MAC | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | TotalPacketLength | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | TCP SrcPort | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | TCP DstPort | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | TCP Seq# | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | TCP NextSeq# | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | CWR | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | ECHO | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | URG | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | ACK | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | PUSH | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | RST | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | SYN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | FIN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | TCP HrdLength | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | UDP SrcPort | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | UDP DstPort | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | UDP Length | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | ICMP Code | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | ICMP Type | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 31 | ICMP Length | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

200000 packets took 2434 seconds to finish
 Trained on Tuesday inside 200K packets of TCP, UDP, and ICMP traffic

Figure B.1 Example 2-D Matrix
 Trained on 200K TCP, UDP, and ICMP packets from Tuesday inside data

| | 8 | 9 | 10 | 11 | 12 |
|----|---------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| | IPSrc | SrcMAC | IPDst | DSTMAC | TotalPacketLength |
| 0 | IPID | [10, 9, 5, 7, 6, 11, 13, 14] | [9, 11, 6, 5, 7, 10, 14, 13] | [10, 11, 5, 7, 9, 6, 13, 14] | [10, 11, 5, 6, 9, 7, 14, 15] |
| 1 | IP TTL | [6, 14, 13, 9, 15, 5, 10, 7, 11] | [5, 15, 10, 7, 9, 11, 6, 14, 13] | [5, 15, 13, 10, 14, 6, 11, 9, 7] | [6, 15, 14, 9, 10, 5, 11, 13, 7] |
| 2 | Don't Frag | [14, 13, 6, 5, 15, 7, 9, 10, 11] | [13, 15, 5, 14, 7, 10, 11, 6, 9] | [13, 15, 5, 14, 7, 10, 6, 11, 9] | [14, 15, 6, 13, 7, 9, 5, 10, 11] |
| 3 | More Frag | [14, 13, 15] | [13, 15, 14] | [13, 15, 14] | [14, 15, 13] |
| 4 | Unused Flag | [14, 13, 15] | [13, 15, 14] | [13, 15, 14] | [14, 15, 13] |
| 5 | Frag Offset | [14, 13, 15] | [13, 15, 14] | [13, 15, 14] | [14, 15, 13] |
| 6 | IP Hdr Length | [14, 13, 15] | [13, 15, 14] | [13, 15, 14] | [14, 15, 13] |
| 7 | IP Packet Length | [10, 14, 9, 5, 11, 6, 13, 7, 15] | [9, 15, 11, 6, 10, 5, 14, 7, 13] | [9, 15, 6, 10, 11, 5, 14, 13, 7] | [10, 15, 5, 11, 7] |
| 8 | IP Src | 0 | 0 | 0 | 0 |
| 9 | Src MAC | 0 | 0 | 0 | 0 |
| 10 | IP Dst | 0 | 0 | 0 | 0 |
| 11 | DST MAC | 0 | 0 | 0 | 0 |
| 12 | Total Packet Length | 0 | 0 | 0 | 0 |
| 13 | TCP Src Port | 0 | 0 | 0 | 0 |
| 14 | TCP Dst Port | 0 | 0 | 0 | 0 |
| 15 | TCP Seq# | 0 | 0 | 0 | 0 |
| 16 | TCP Next Seq# | 0 | 0 | 0 | 0 |
| 17 | CWR | 0 | 0 | 0 | 0 |
| 18 | ECHO | 0 | 0 | 0 | 0 |
| 19 | URG | 0 | 0 | 0 | 0 |
| 20 | ACK | 0 | 0 | 0 | 0 |
| 21 | PUSH | 0 | 0 | 0 | 0 |
| 22 | RST | 0 | 0 | 0 | 0 |
| 23 | SYN | 0 | 0 | 0 | 0 |
| 24 | FIN | 0 | 0 | 0 | 0 |
| 25 | TCP Hdr Length | 0 | 0 | 0 | 0 |
| 26 | UDP Src Port | 0 | 0 | 0 | 0 |
| 27 | UDP Dst Port | 0 | 0 | 0 | 0 |
| 28 | UDP Length | 0 | 0 | 0 | 0 |
| 29 | ICMP Code | 0 | 0 | 0 | 0 |
| 30 | ICMP Type | 0 | 0 | 0 | 0 |
| 31 | ICMP Length | 0 | 0 | 0 | 0 |
| | | | | | |
| | | | | | |

| | 13 | 14 | 15 | 16 | 17 | 18 |
|----|----------------------------------|----------------------------------|----------------------------------|----------------------------------|-------------|-------------|
| | TCPSrcPort | TCPDstPort | TCPSeq# | TCPNextSeq# | CWR | ECHO |
| 0 | [6, 7, 9, 5, 10, 11, 13, 14] | [5, 7, 10, 6, 9, 11, 14, 13] | [5, 10, 7, 6, 9, 11, 13, 14] | [7, 11, 5, 10, 6, 9, 14, 13] | [7, 11, 15] | [7, 11, 15] |
| 1 | [10, 15, 5, 13, 14, 6, 9, 11, 7] | [9, 15, 6, 13, 14, 11, 7, 10, 5] | [9, 13, 6, 15, 14, 10, 5] | [1, 15, 7, 9, 6, 13, 14, 10, 5] | [1, 15, 7] | [1, 15, 7] |
| 2 | [14, 7, 5, 13, 15, 11, 10, 9, 6] | [13, 7, 6, 14, 15, 11, 9, 10, 5] | [13, 5, 6, 14, 15, 9, 10, 11, 7] | [15, 7, 13, 14, 10, 11, 5, 9, 6] | [15, 7, 11] | [15, 7, 11] |
| 3 | [14, 15, 13] | [13, 15, 14] | [13, 14, 15] | [15, 13, 14] | [15] | [15] |
| 4 | [14, 15, 13] | [13, 15, 14] | [13, 14, 15] | [15, 13, 14] | [15] | [15] |
| 5 | [14, 15, 13] | [13, 15, 14] | [13, 14, 15] | [15, 13, 14] | [15] | [15] |
| 6 | [14, 15, 13] | [13, 15, 14] | [13, 14, 15] | [15, 13, 14] | [15] | [15] |
| 7 | [14, 11, 9, 6, 7, 10, 15, 5, 13] | [13, 11, 10, 5, 7, 9, 15, 6, 14] | [13, 9, 10, 5, 7, 6, 14, 15, 11] | [15, 11, 5, 10, 9, 13, 6, 7, 14] | [15, 11, 7] | [15, 11, 7] |
| 8 | [6, 15, 9, 13, 14, 11, 7, 10, 5] | [5, 15, 10, 13, 14, 9, 6, 7, 11] | [5, 13, 10, 15, 14, 9, 6] | [7, 15, 11, 5, 10, 13, 14, 6, 9] | [7, 15, 11] | [7, 15, 11] |
| 9 | [10, 15, 5, 6, 9, 13, 14, 7, 11] | [9, 15, 6, 5, 10, 13, 14, 7, 11] | [9, 13, 6, 15, 5, 10, 14] | [1, 15, 7, 9, 6, 13, 14, 10, 5] | [1, 15, 7] | [1, 15, 7] |
| 10 | [10, 15, 5, 13, 14, 9, 6, 7, 11] | [9, 15, 6, 13, 14, 7, 11, 10, 5] | [9, 13, 6, 15, 14, 5, 10] | [1, 15, 7, 9, 6, 13, 14, 10, 5] | [1, 15, 7] | [1, 15, 7] |
| 11 | [6, 15, 9, 10, 5, 13, 14, 7, 11] | [5, 15, 10, 9, 6, 13, 14, 7, 11] | [5, 13, 10, 15, 9, 6, 14] | [7, 15, 11, 5, 10, 13, 14, 6, 9] | [7, 15, 11] | [7, 15, 11] |
| 12 | [14, 15, 13, 6, 9, 11, 7, 10, 5] | [13, 15, 14, 5, 10, 11, 7, 9, 6] | [13, 14, 5, 10, 9, 7, 6, 15, 11] | [15, 5, 10, 9, 13, 6, 14] | [15, 7, 11] | [15, 7, 11] |
| 13 | 0 | [9, 15, 6, 7, 13, 11, 14, 10, 5] | [9, 13, 6, 15, 5, 14, 10, 7, 11] | [1, 15, 7, 9, 6, 13, 14, 10, 5] | [1, 15, 7] | [1, 15, 7] |
| 14 | 0 | 0 | 0 | [7, 11, 5, 10, 13, 6, 15, 9, 14] | [7, 11, 15] | [7, 11, 15] |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | 0 | 0 | 0 | 0 | 0 | 0 |
| 31 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 19 | 20 | 21 | 22 | 23 |
|----|---------------------|------------------------------|----------------------------------|----------------------------------|----------------------------------|
| | URG | ACK | PUSH | RST | SYN |
| 0 | IPID | [5, 11, 7, 6, 10, 9, 15] | [7, 11, 5, 10, 6, 9, 14, 15, 13] | [7, 11, 5, 6, 10, 15, 9] | [7, 6, 10, 11, 5, 9, 15] |
| 1 | IP TTL | [1, 15, 7, 13, 9, 14, 6] | [1, 15, 7, 9, 6, 13, 14, 10, 5] | [1, 15, 7, 13, 5, 14, 10, 9, 6] | [1, 14, 6, 7, 15, 10, 13, 5, 9] |
| 2 | Don't Frag | [15, 7, 11, 13, 14] | [13, 5, 7, 15, 11, 9, 14, 6, 10] | [15, 7, 11, 13, 5, 14, 10, 9, 6] | [15, 6, 11, 7, 14, 13, 9, 5, 10] |
| 3 | More Frag | [15, 13, 14] | [15, 13, 14] | [15, 13, 14] | [15, 14, 13] |
| 4 | Unused Flag | [15, 13, 14] | [15, 13, 14] | [15, 13, 14] | [15, 14, 13] |
| 5 | Frag Offset | [15, 13, 14] | [15, 13, 14] | [15, 13, 14] | [15, 14, 13] |
| 6 | IP Hdr Length | [15, 13, 14] | [15, 13, 14] | [15, 13, 14] | [15, 14, 13] |
| 7 | IP Packet Length | [15, 11, 7, 5, 13, 9, 10, 6] | [15, 11, 5, 10, 7, 6, 13, 9, 14] | [15, 11, 7, 9, 13, 6, 14] | [15, 10, 7, 11, 6, 9, 5, 14, 13] |
| 8 | IP Src | [7, 15, 11, 5, 10, 14] | [7, 15, 11, 5, 10, 13, 14, 6, 9] | [7, 15, 11, 13, 9, 14, 6, 5, 10] | [7, 14, 10, 11, 15, 6, 13, 9, 5] |
| 9 | Src MAC | [11, 15, 7, 13, 9, 14, 6] | [9, 13, 7, 11, 15, 5, 14, 10, 6] | [11, 15, 7, 13, 9, 14, 6, 5, 10] | [11, 14, 6, 7, 15, 10, 13, 9, 5] |
| 10 | IP Dst | [11, 15, 7, 5, 9, 10, 6, 14] | [9, 13, 7, 11, 15, 14, 6, 10, 5] | [11, 15, 7, 13, 5, 10, 9, 6, 14] | [11, 14, 6, 7, 15, 10, 13, 5, 9] |
| 11 | DST MAC | [7, 15, 11, 13, 5, 14, 10] | [5, 13, 11, 7, 15, 9, 14, 6, 10] | [7, 15, 11, 5, 10, 13, 14, 6, 9] | [7, 14, 10, 11, 15, 6, 13, 5, 9] |
| 12 | Total Packet Length | [15, 7, 11, 13, 14, 6] | [13, 15, 7, 11, 5, 10, 14] | [15, 5, 10, 11, 7, 14, 6, 13, 9] | [15, 14, 7, 11, 6, 9, 13] |
| 13 | TCP Src Port | [11, 15, 7, 9, 13, 6, 14] | [9, 13, 7, 11, 15, 6, 5, 10, 14] | [11, 15, 7, 9, 6, 13, 14, 10, 5] | [11, 14, 6, 7, 15, 10, 5, 13, 9] |
| 14 | TCP Dst Port | [7, 15, 11, 5, 10, 14] | [5, 13, 11, 7, 15, 14, 10, 6, 9] | [7, 15, 11, 5, 10, 13, 14, 6, 9] | [7, 14, 10, 11, 15, 6, 13, 9, 5] |
| 15 | TCP Seq# | [7, 11, 15, 5, 9, 10, 6] | [5, 11, 7, 15, 6, 10, 9, 14] | [7, 11, 5, 10, 13, 14, 6, 9] | [7, 6, 10, 11, 15, 5, 9] |
| 16 | TCP Next Seq# | [15, 7, 11, 5, 9, 10, 6] | [13, 15, 7, 11, 5, 10, 14] | [15, 5, 10, 11, 7, 6, 9, 14] | [15, 14, 7, 11, 6, 9, 13] |
| 17 | CWR | [15, 13, 14] | [13, 15, 14] | [15, 13, 14] | [15, 14, 13] |
| 18 | ECHO | [15, 13, 14] | [13, 15, 14] | [15, 13, 14] | [15, 14, 13] |
| 19 | URG | 0 | [13, 15, 14, 7, 11] | [15, 13, 14, 7, 11] | [15, 14, 13, 7, 11] |
| 20 | ACK | 0 | 0 | [7, 15, 11, 9, 14, 6, 13] | [7, 6, 14, 15, 9, 13, 11, 5, 10] |
| 21 | PUSH | 0 | 0 | [15, 7, 11, 9, 13, 14, 6] | [15, 14, 7, 11, 6, 9, 13] |
| 22 | RST | 0 | 0 | 0 | [15, 14, 13, 7, 9, 11, 6] |
| 23 | SYN | 0 | 0 | 0 | 0 |
| 24 | FIN | 0 | 0 | 0 | 0 |
| 25 | TCP Hdr Length | 0 | 0 | 0 | 0 |
| 26 | UDP Src Port | 0 | 0 | 0 | 0 |
| 27 | UDP Dst Port | 0 | 0 | 0 | 0 |
| 28 | UDP Length | 0 | 0 | 0 | 0 |
| 29 | ICMP Code | 0 | 0 | 0 | 0 |
| 30 | ICMP Type | 0 | 0 | 0 | 0 |
| 31 | ICMP Length | 0 | 0 | 0 | 0 |

| | | 24 | 25 | 26 | 27 | 28 |
|----|---------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----|
| | FIN | TCPHrdLength | UDPSrcPort | UDPDstPort | UDPLength | |
| 0 | IPID | [7, 11, 5, 9, 10, 6, 15] | [1, 7, 5, 9, 10, 6, 13] | [1, 7, 5, 9, 10, 6, 14] | [10, 11, 5, 6, 9, 7, 14] | |
| 1 | IP TTL | [1, 15, 7, 13, 9, 5, 6, 14, 10] | [1, 14, 6, 7, 15, 10, 13, 5, 9] | [7, 15, 11, 5, 9, 10, 6, 14, 13] | [6, 15, 9, 14, 5, 10, 7, 11, 13] | |
| 2 | Don't Frag | [15, 7, 9, 6, 11, 13, 14, 5, 10] | [15, 6, 11, 7, 14, 13, 9, 5, 10] | [15, 7, 13, 11, 9, 14, 6, 10, 5] | [14, 15, 13, 7, 11, 9, 6, 10, 5] | |
| 3 | More Frag | [15, 13, 14] | [15, 14, 13] | [15, 13, 14] | [14, 15, 13] | |
| 4 | Unused Flag | [15, 13, 14] | [15, 14, 13] | [15, 13, 14] | [14, 15, 13] | |
| 5 | Frag Offset | [15, 13, 14] | [15, 14, 13] | [15, 13, 14] | [14, 15, 13] | |
| 6 | IP Hrd Length | [15, 13, 14] | [15, 14, 13] | [15, 13, 14] | [14, 15, 13] | |
| 7 | IP Packet Length | [15, 11, 7, 9, 14, 13, 6, 5, 10] | [15, 10, 7, 11, 6, 9, 5, 14, 13] | [11, 15, 7, 5, 10, 9, 6, 13, 14] | [10, 15, 5] | |
| 8 | IP Src | [7, 15, 11, 13, 5, 9, 10, 14, 6] | [7, 14, 10, 11, 15, 6, 13, 9, 5] | [11, 7, 15, 5, 9, 10, 6, 14, 13] | [10, 11, 5, 14, 9, 6, 7, 15, 13] | |
| 9 | Src MAC | [11, 15, 7, 13, 9, 5, 6, 14, 10] | [11, 14, 6, 7, 15, 10, 13, 9, 5] | [7, 15, 11, 13, 9, 5, 6, 10, 14] | [6, 15, 9, 14, 5, 10, 7, 11, 13] | |
| 10 | IP Dst | [11, 15, 7, 13, 9, 5, 6, 14, 10] | [11, 14, 6, 7, 15, 10, 13, 5, 9] | [7, 15, 11, 5, 10, 6, 9, 13, 14] | [6, 15, 9, 14, 5, 10, 7, 11, 13] | |
| 11 | DST MAC | [7, 15, 11, 13, 5, 9, 10, 14, 6] | [7, 14, 10, 11, 15, 6, 13, 5, 9] | [11, 15, 7, 9, 6, 5, 10, 13, 14] | [10, 15, 5, 14, 9, 6, 11, 7, 13] | |
| 12 | Total Packet Length | [15, 7, 11, 9, 13, 14, 6, 5, 10] | [15, 14, 7, 11, 6, 9, 13] | [11, 15, 7, 5, 10, 9, 6, 13, 14] | [10, 15, 5] | |
| 13 | TCP Src Port | [11, 15, 7, 13, 9, 5, 6, 14, 10] | [11, 14, 6, 7, 15, 10, 5, 13, 9] | | | |
| 14 | TCP Dst Port | [7, 15, 11, 13, 5, 9, 10, 14, 6] | [7, 14, 10, 11, 15, 6, 13, 9, 5] | | | |
| 15 | TCP Seq# | [7, 11, 15, 5, 9, 10, 6, 13] | [7, 6, 10, 11, 15, 5, 9] | | | |
| 16 | TCP Next Seq# | [15, 7, 11, 9, 14, 13, 6, 5, 10] | [15, 14, 7, 11, 6, 9, 13] | | | |
| 17 | CWR | [15, 13, 14] | [15, 14, 13] | | | |
| 18 | ECHO | [15, 13, 14] | [15, 14, 13] | | | |
| 19 | URG | [15, 13, 14, 7, 11] | [15, 14, 13, 7, 11] | | | |
| 20 | ACK | [7, 15, 13, 14, 11, 10, 5] | [7, 6, 14, 15, 9, 13, 11, 5, 10] | | | |
| 21 | PUSH | [15, 7, 11, 9, 14, 13, 6, 5, 10] | [15, 14, 7, 11, 6, 9, 13] | | | |
| 22 | RST | [15, 13, 14, 7, 6, 11, 9] | [15, 14, 13, 7, 9, 11, 6] | | | |
| 23 | SYN | [15, 11, 13, 14, 9, 7, 6] | [15, 10, 5, 14, 11, 7, 13] | | | |
| 24 | FIN | | [15, 14, 7, 11, 6, 13, 9] | | | |
| 25 | TCP Hrd Length | | | | | |
| 26 | UDP Src Port | | | | | |
| 27 | UDP Dst Port | | | | | |
| 28 | UDP Length | | | | | |
| 29 | ICMP Code | | | | | |
| 30 | ICMP Type | | | | | |
| 31 | ICMP Length | | | | | |
| | | | | | | |
| | | | | | | |

| | | 29 | 30 | 31 |
|----|---------------------|--------------------|----------------------|--------------------|
| | | ICMPCode | ICMPType | ICMPLength |
| 0 | IPID | [11, 7, 15, 6] | [10, 7, 5, 14, 6, 9] | [11, 7, 15, 6] |
| 1 | IP TTL | [11, 15, 7, 10, 6] | [10, 15, 5, 6, 9] | [11, 15, 7, 10, 6] |
| 2 | Don't Frag | [15, 14] | [14, 15, 13] | [15, 14] |
| 3 | More Frag | [15, 14] | [14, 15, 13] | [15, 14] |
| 4 | Unused Flag | [15, 14] | [14, 15, 13] | [15, 14] |
| 5 | Frag Offset | [15, 14] | [14, 15, 13] | [15, 14] |
| 6 | IP Hrd Length | [15, 14] | [14, 15, 13] | [15, 14] |
| 7 | IP Packet Length | [15, 10] | [14, 15, 13, 9, 10] | [15, 10] |
| 8 | IP Src | [11, 15, 7, 10] | [10, 15, 5, 6, 9] | [11, 15, 7, 10] |
| 9 | Src MAC | [7, 15, 11, 6, 10] | [6, 15, 9, 10, 5] | [7, 15, 11, 6, 10] |
| 10 | IP Dst | [7, 15, 11, 10] | [6, 15, 9, 10, 5] | [7, 15, 11, 10] |
| 11 | DST MAC | [11, 15, 7, 14, 6] | [10, 15, 5, 6, 13] | [11, 15, 7, 14, 6] |
| 12 | Total Packet Length | [15, 10] | [14, 15, 13, 9, 10] | [15, 10] |
| 13 | TCP Src Port | 0 | 0 | 0 |
| 14 | TCP Dst Port | 0 | 0 | 0 |
| 15 | TCP Seq# | 0 | 0 | 0 |
| 16 | TCP Next Seq# | 0 | 0 | 0 |
| 17 | CWR | 0 | 0 | 0 |
| 18 | ECHO | 0 | 0 | 0 |
| 19 | URG | 0 | 0 | 0 |
| 20 | ACK | 0 | 0 | 0 |
| 21 | PUSH | 0 | 0 | 0 |
| 22 | RST | 0 | 0 | 0 |
| 23 | SYN | 0 | 0 | 0 |
| 24 | FIN | 0 | 0 | 0 |
| 25 | TCP Hrd Length | 0 | 0 | 0 |
| 26 | UDP Src Port | 0 | 0 | 0 |
| 27 | UDP Dst Port | 0 | 0 | 0 |
| 28 | UDP Length | 0 | 0 | 0 |
| 29 | ICMP Code | 0 | 0 | 0 |
| 30 | ICMP Type | 0 | [14, 15, 13, 9, 10] | [15, 10] |
| 31 | ICMP Length | 0 | 0 | [11, 15, 7, 6, 10] |
| | | | | 0 |
| | | | | 0 |

Appendix C. Matrix of Code Version 2.44

| | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | |
|----|----------------|----------------|------------------------------|------------------------------|----------------------------------|------------------------------|------------------------------|------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| | IPSrc1 | IPSrc2 | IPSrc1 | IPDst1 | IPSrc1 | IPDst1 | IPDst1 | IPDst1 | MACSrc1 | MACSrc2 | MACSrc1 | MACSrc2 |
| 0 | IPSrc1 | IPSrc2 | | | | | | | | | | |
| 1 | IPSrc1 | IPDst1 | [7, 13, 11, 14, 9, 6, 10, 5] | [7, 14, 11, 13, 10, 5, 6, 9] | [6, 15, 9, 14, 11, 13, 7, 5, 10] | [6, 15, 9, 10, 7, 5, 11] | [6, 15, 9, 10, 7, 5, 11] | [6, 15, 9, 10, 7, 5, 11] | [7, 14, 11, 13, 6, 10, 5, 9] | [7, 14, 11, 13, 6, 10, 5, 9] | [7, 14, 11, 13, 6, 10, 5, 9] | [7, 14, 11, 13, 6, 10, 5, 9] |
| 2 | IPDst1 | IPSrc2 | | | [15, 6, 9, 14, 11, 7, 13, 5, 10] | [14, 7, 13, 11, 5, 10, 6, 9] | [14, 7, 13, 11, 5, 10, 6, 9] | [14, 7, 13, 11, 5, 10, 6, 9] | [14, 7, 13, 11, 5, 10, 6, 9] | [14, 7, 13, 11, 5, 10, 6, 9] | [14, 7, 13, 11, 5, 10, 6, 9] | [14, 7, 13, 11, 5, 10, 6, 9] |
| 3 | IPDst1 | IPDst2 | | | | | [14, 11, 13, 7, 9, 6, 10, 5] | [14, 11, 13, 7, 9, 6, 10, 5] | [14, 11, 13, 7, 9, 6, 10, 5] | [14, 11, 13, 7, 9, 6, 10, 5] | [14, 11, 13, 7, 9, 6, 10, 5] | [14, 11, 13, 7, 9, 6, 10, 5] |
| 4 | MACSrc1 | MACDst1 | | | | | | | [10, 15, 5, 6, 11, 9, 7, 13, 14] | [10, 15, 5, 6, 11, 9, 7, 13, 14] | [10, 15, 5, 6, 11, 9, 7, 13, 14] | [10, 15, 5, 6, 11, 9, 7, 13, 14] |
| 5 | MACSrc1 | MACDst2 | | | | | | | | | | |
| 6 | MACDst1 | MACSrc2 | | | | | | | | | | |
| 7 | MACDst1 | MACDst2 | | | | | | | | | | |
| 8 | TCPSrcPort1 | TCPSrcPort2 | | | | | | | | | | |
| 9 | TCPSrcPort1 | TCPDstPort1 | | | | | | | | | | |
| 10 | TCPDstPort1 | TCPSrcPort2 | | | | | | | | | | |
| 11 | TCPDstPort1 | TCPDstPort2 | | | | | | | | | | |
| 12 | TCPSeq#1 | TCPSeq#2 | | | | | | | | | | |
| 13 | TCPSeq#1 | TCPNxtSeq#2 | | | | | | | | | | |
| 14 | TCPNxtSeq#1 | TCPSeq#2 | | | | | | | | | | |
| 15 | TCPNxtSeq#1 | TCPNxtSeq#2 | | | | | | | | | | |
| 16 | Ack1 | Ack2 | | | | | | | | | | |
| 17 | Ack1 | Push2 | | | | | | | | | | |
| 18 | Ack1 | Rst2 | | | | | | | | | | |
| 19 | Ack1 | Syn2 | | | | | | | | | | |
| 20 | Ack1 | Fin2 | | | | | | | | | | |
| 21 | Push1 | Ack2 | | | | | | | | | | |
| 22 | Push1 | Push2 | | | | | | | | | | |
| 23 | Push1 | Rst2 | | | | | | | | | | |
| 24 | Push1 | Syn2 | | | | | | | | | | |
| 25 | Push1 | Fin2 | | | | | | | | | | |
| 26 | Rst1 | Ack2 | | | | | | | | | | |
| 27 | Rst1 | Push2 | | | | | | | | | | |
| 28 | Rst1 | Rst2 | | | | | | | | | | |
| 29 | Rst1 | Syn2 | | | | | | | | | | |
| 30 | Rst1 | Fin2 | | | | | | | | | | |
| 31 | Syn1 | Ack2 | | | | | | | | | | |
| 32 | Syn1 | Push2 | | | | | | | | | | |
| 33 | Syn1 | Rst2 | | | | | | | | | | |
| 34 | Syn1 | Syn2 | | | | | | | | | | |
| 35 | Syn1 | Fin2 | | | | | | | | | | |
| 36 | Fin1 | Ack2 | | | | | | | | | | |
| 37 | Fin1 | Push2 | | | | | | | | | | |
| 38 | Fin1 | Rst2 | | | | | | | | | | |
| 39 | Fin1 | Syn2 | | | | | | | | | | |
| 40 | Fin1 | Fin2 | | | | | | | | | | |
| 41 | ICMPCode | ICMPCode | | | | | | | | | | |
| 42 | ICMPType | ICMPType | | | | | | | | | | |
| 43 | ICMPDataLength | ICMPDataLength | | | | | | | | | | |

Figure C.1 Example Beta Crosschecking Matrix Trained on the First 100K Packets from Tuesday Week One Inside Lincoln Labs Data

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|-----|-----|-------|----------|----------|-----------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|
| | IPD | IPCTL | Donrfreq | Donrfreq | Donrfreq | Donrfreq | Monrfreq | Monrfreq | Monrfreq | Monrfreq | Unusdfreq | Unusdfreq | Unusdfreq | Unusdfreq | |
| | IPD | IPCTL | Donrfreq | Monrfreq | Unusdfreq | Donrfreq | Donrfreq | Monrfreq | Monrfreq | Monrfreq | Donrfreq | Monrfreq | Unusdfreq | Unusdfreq | FrqOffset |
| | | | | | | | | | | | | | | | |
| 80 | URG | ECHO | | | | | | | | | | | | | |
| 81 | URG | URG | | | | | | | | | | | | | |
| 82 | URG | ACK | | | | | | | | | | | | | |
| 83 | URG | PUSH | | | | | | | | | | | | | |
| 84 | URG | RESET | | | | | | | | | | | | | |
| 85 | URG | SYN | | | | | | | | | | | | | |
| 86 | URG | FIN | | | | | | | | | | | | | |
| 87 | ACK | CWR | | | | | | | | | | | | | |
| 88 | ACK | ECHO | | | | | | | | | | | | | |
| 89 | ACK | URG | | | | | | | | | | | | | |
| 90 | ACK | ACK | | | | | | | | | | | | | |
| 91 | ACK | ACK | | | | | | | | | | | | | |
| 92 | ACK | ACK | | | | | | | | | | | | | |
| 93 | ACK | ACK | | | | | | | | | | | | | |
| 94 | ACK | ACK | | | | | | | | | | | | | |
| 95 | ACK | ACK | | | | | | | | | | | | | |
| 96 | ACK | ACK | | | | | | | | | | | | | |
| 97 | ACK | ACK | | | | | | | | | | | | | |
| 98 | ACK | ACK | | | | | | | | | | | | | |
| 99 | ACK | ACK | | | | | | | | | | | | | |
| 100 | ACK | ACK | | | | | | | | | | | | | |
| 101 | ACK | ACK | | | | | | | | | | | | | |
| 102 | ACK | ACK | | | | | | | | | | | | | |
| 103 | ACK | ACK | | | | | | | | | | | | | |
| 104 | ACK | ACK | | | | | | | | | | | | | |
| 105 | ACK | ACK | | | | | | | | | | | | | |
| 106 | ACK | ACK | | | | | | | | | | | | | |
| 107 | ACK | ACK | | | | | | | | | | | | | |
| 108 | ACK | ACK | | | | | | | | | | | | | |
| 109 | ACK | ACK | | | | | | | | | | | | | |
| 110 | ACK | ACK | | | | | | | | | | | | | |
| 111 | ACK | ACK | | | | | | | | | | | | | |
| 112 | ACK | ACK | | | | | | | | | | | | | |
| 113 | ACK | ACK | | | | | | | | | | | | | |
| 114 | ACK | ACK | | | | | | | | | | | | | |
| 115 | ACK | ACK | | | | | | | | | | | | | |

12 packets wait 25 seconds to finish

Appendix E. Matrix of Code Version 1.0 Plus MAC Address

Trained on LLData TueWeek1 Inside 100K packets, tested on IPSpoofed Telnet Sessions

| | IPSRC(0) | SrcMac(1) | IPDest(2) | DstMac(3) | TCPSrcPort(4) | TCPDestPort(5) |
|----------------|----------|--------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| IPSRC(0) | 0 | [9, 15, 6, 10, 7, 5, 11] | [9, 15, 6, 14, 11, 13, 7, 5, 10] | [10, 15, 5, 6, 11, 9, 7, 13, 14] | [9, 15, 6, 5, 13, 14, 11, 7, 10] | [10, 15, 5, 9, 13, 14, 6, 7, 11] |
| SrcMac(1) | 0 | 0 | [5, 15, 10, 9, 14, 6, 13, 7, 11] | [6, 15, 9, 10, 14, 11, 5, 7, 13] | [5, 15, 10, 9, 13, 6, 14, 7, 11] | [6, 15, 9, 10, 13, 5, 14, 7, 11] |
| IPDest(2) | 0 | 0 | 0 | [6, 15, 9, 10, 7, 5, 11] | [5, 15, 10, 9, 13, 14, 6, 7, 11] | [6, 15, 9, 5, 13, 14, 11, 7, 10] |
| DstMac(3) | 0 | 0 | 0 | 0 | [9, 15, 6, 13, 10, 5, 14, 7, 11] | [10, 15, 5, 9, 13, 6, 14, 7, 11] |
| TCPSrcPort(4) | 0 | 0 | 0 | 0 | 0 | [6, 15, 9, 5, 7, 13, 11, 14, 10] |
| TCPDestPort(5) | 0 | 0 | 0 | 0 | 0 | 0 |
| TCPSeq#(6) | 0 | 0 | 0 | 0 | 0 | 0 |
| TCPNxtSeq#(7) | 0 | 0 | 0 | 0 | 0 | 0 |
| Ack(8) | 0 | 0 | 0 | 0 | 0 | 0 |
| Push(9) | 0 | 0 | 0 | 0 | 0 | 0 |
| Reset(10) | 0 | 0 | 0 | 0 | 0 | 0 |
| Syn(11) | 0 | 0 | 0 | 0 | 0 | 0 |
| Fin(12) | 0 | 0 | 0 | 0 | 0 | 0 |
| IPSRC(0) | 0 | SrcMac(1) | IPDest(2) | DstMac(3) | TCPSrcPort(4) | TCPDestPort(5) |
| IPSRC(0) | 0 | [5, 15, 10, 14, 6, 9] | [6, 15, 9] | [6, 15, 9, 14, 10, 5] | [5, 15, 10, 13] | [6, 15, 9, 13] |
| SrcMac(1) | 0 | 0 | [6, 15, 9, 11, 10, 5] | [6, 15, 9, 11, 10, 14] | [5, 15, 10, 9, 6] | [6, 15, 9, 11, 10, 13, 5] |
| IPDest(2) | 0 | 0 | 0 | [10, 15, 5, 14, 6, 9] | [9, 15, 6, 13] | [10, 15, 5, 13] |
| DstMac(3) | 0 | 0 | 0 | 0 | [9, 15, 6, 13, 11, 10, 5] | [10, 15, 5, 9, 6] |
| TCPSrcPort(4) | 0 | 0 | 0 | 0 | 0 | [6, 15, 9, 7, 13] |
| TCPDestPort(5) | 0 | 0 | 0 | 0 | 0 | 0 |
| TCPSeq#(6) | 0 | 0 | 0 | 0 | 0 | 0 |
| TCPNxtSeq#(7) | 0 | 0 | 0 | 0 | 0 | 0 |
| Ack(8) | 0 | 0 | 0 | 0 | 0 | 0 |
| Push(9) | 0 | 0 | 0 | 0 | 0 | 0 |
| Reset(10) | 0 | 0 | 0 | 0 | 0 | 0 |
| Syn(11) | 0 | 0 | 0 | 0 | 0 | 0 |
| Fin(12) | 0 | 0 | 0 | 0 | 0 | 0 |

They are not equal at [0][1] [2][3] Done!!!

Figure E.1 Less-than, Greater-than, Equal Matrix With MAC Addresses Added Shows Detection of IP Spoof Attack

Trained on LLDData 1

| | TCPSeq#(6) | TCPNxtSeq#(7) | Ack(8) | Push(9) | Reset(10) |
|----------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| IPSRc(0) | [10, 13, 5, 15, 6, 14, 9] | [11, 13, 9, 5, 15, 10, 14, 6, 7] | [11, 15, 7, 10, 5, 13, 14, 6, 9] | [11, 13, 9, 5, 15, 10, 7, 14, 6] | [11, 15, 7, 13, 9, 14, 6, 5, 10] |
| SrcMac(1) | [6, 13, 9, 15, 10, 14, 5] | [7, 13, 5, 9, 15, 6, 14, 10, 11] | [7, 15, 11, 10, 5, 13, 14, 6, 9] | [7, 13, 5, 9, 15, 6, 11, 14, 10] | [7, 15, 11, 13, 9, 14, 6, 5, 10] |
| IPDest(2) | [6, 13, 9, 15, 10, 14, 5] | [7, 13, 5, 9, 15, 6, 14, 10, 11] | [7, 15, 11, 6, 9, 13, 14, 10, 5] | [7, 13, 5, 9, 15, 6, 11, 14, 10] | [7, 15, 11, 13, 5, 10, 9, 6, 14] |
| DstMac(3) | [10, 13, 5, 15, 14, 9, 6] | [11, 13, 9, 5, 15, 10, 14, 6, 7] | [11, 15, 7, 10, 14, 9, 13, 6, 5] | [11, 13, 9, 5, 15, 10, 7, 14, 6] | [11, 15, 7, 13, 5, 14, 10, 9, 6] |
| TCPSrcPort(4) | [6, 13, 9, 15, 5, 14, 10, 7, 11] | [7, 13, 5, 9, 15, 6, 14, 10, 11] | [7, 15, 11, 6, 9, 13, 5, 10, 14] | [7, 13, 5, 9, 15, 6, 11, 14, 10] | [7, 15, 11, 13, 5, 6, 10, 9, 14] |
| TCPDestPort(5) | [10, 13, 5, 15, 6, 14, 9] | [11, 13, 9, 5, 15, 10, 14, 6, 7] | [11, 15, 7, 10, 6, 5, 13, 14, 9] | [11, 13, 9, 5, 15, 10, 7, 14, 6] | [11, 15, 7, 13, 9, 14, 6, 5, 10] |
| TCPNxtSeq#(7) | 0 | [11, 5, 9, 15, 13, 10, 6, 7, 14] | [11, 7, 15, 10, 5, 6, 9, 14] | [15, 5, 10, 11, 7, 6, 9, 14] | [11, 7, 15, 5, 9, 13, 10, 6] |
| Ack(8) | 0 | 0 | [15, 7, 11, 10, 14, 13, 5] | [15, 5, 10, 11, 7, 6, 9, 14] | [15, 7, 11, 9, 13, 14, 6] |
| Push(9) | 0 | 0 | 0 | [15, 13, 14, 10, 11, 7, 5] | [15, 11, 7, 9, 14, 6, 13] |
| Reset(10) | 0 | 0 | 0 | 0 | [15, 7, 11, 9, 13, 14, 6] |
| Syn(11) | 0 | 0 | 0 | 0 | 0 |
| Fin(12) | 0 | 0 | 0 | 0 | 0 |
| | TCPSeq#(6) | TCPNxtSeq#(7) | Ack(8) | Push(9) | Reset(10) |
| IPSRc(0) | [5, 13, 10, 15] | [7, 13, 5, 9, 11, 14, 10, 6, 15] | [7, 15, 11, 14, 6, 9, 13] | [7, 13, 5, 9, 11, 15, 14, 10, 6] | [7, 15, 11] |
| SrcMac(1) | [5, 13, 10, 15, 9, 6] | [7, 13, 5, 9, 11, 14, 10, 6, 15] | [7, 15, 11, 10, 5, 13] | [7, 13, 5, 9, 11, 15, 14, 10, 6] | [7, 15, 11] |
| IPDest(2) | [9, 13, 6, 15] | [11, 13, 9, 5, 7, 14, 6, 10, 15] | [11, 15, 7, 14, 10, 5, 13] | [11, 13, 9, 5, 7, 15, 14, 6, 10] | [11, 15, 7] |
| DstMac(3) | [9, 13, 6, 15, 10, 5] | [11, 13, 9, 5, 7, 14, 6, 10, 15] | [11, 15, 7, 14, 10, 9, 13] | [11, 13, 9, 5, 7, 15, 14, 6, 10] | [11, 15, 7] |
| TCPSrcPort(4) | [5, 13, 10, 15] | [7, 13, 5, 9, 11, 14, 10, 6, 15] | [7, 15, 11, 6, 9, 13] | [7, 13, 5, 9, 11, 15, 14, 10, 6] | [7, 15, 11] |
| TCPDestPort(5) | [9, 13, 6, 15, 5] | [11, 13, 9, 5, 7, 14, 6, 10, 15] | [11, 15, 7, 14, 10, 5, 13] | [11, 13, 9, 5, 7, 15, 14, 6, 10] | [11, 15, 7] |
| TCPNxtSeq#(7) | 0 | [7, 5, 9, 13, 11, 6, 10, 15] | [7, 11, 15, 6, 9, 5] | [7, 5, 9, 13, 11, 6, 10, 15] | [7, 11, 15] |
| Ack(8) | 0 | 0 | [15, 7, 11, 10, 14, 13, 5] | [15, 5, 7, 10, 11] | [15, 7, 11] |
| Push(9) | 0 | 0 | 0 | [15, 13, 14, 10, 11, 7, 5] | [15, 11, 7] |
| Reset(10) | 0 | 0 | 0 | 0 | [15, 7, 11] |
| Syn(11) | 0 | 0 | 0 | 0 | 0 |
| Fin(12) | 0 | 0 | 0 | 0 | 0 |

Trained on LLData 1

| | Syn(11) | Fin(12) |
|----------------|----------------------------------|----------------------------------|
| IPSRC(0) | [10, 14, 7, 15, 11, 9, 5, 6, 13] | [11, 15, 7, 13, 5, 9, 10, 14, 6] |
| SrcMac(1) | [6, 14, 11, 15, 7, 9, 13, 10, 5] | [7, 15, 11, 13, 9, 5, 6, 14, 10] |
| IPDest(2) | [6, 14, 11, 15, 7, 5, 9, 10, 13] | [7, 15, 11, 13, 9, 5, 6, 14, 10] |
| DstMac(3) | [10, 14, 7, 15, 11, 9, 13, 6, 5] | [11, 15, 7, 13, 5, 9, 10, 14, 6] |
| TCPSrcPort(4) | [6, 14, 11, 15, 7, 5, 9, 10, 13] | [7, 15, 11, 13, 9, 5, 6, 14, 10] |
| TCPDestPort(5) | [10, 14, 7, 15, 11, 9, 5, 6, 13] | [11, 15, 7, 13, 5, 9, 10, 14, 6] |
| TCPSeq#(6) | [10, 6, 7, 15, 11, 9, 5] | [11, 7, 15, 5, 9, 10, 6, 13] |
| TCPNextSeq#(7) | [14, 6, 7, 15, 11, 9, 13] | [15, 7, 11, 9, 14, 13, 6, 5, 10] |
| Ack(8) | [14, 15, 9, 13, 7, 6, 11, 5, 10] | [15, 13, 11, 14, 10, 7, 5] |
| Push(9) | [14, 6, 7, 15, 11, 9, 13] | [15, 7, 11, 9, 14, 13, 6, 5, 10] |
| Reset(10) | [14, 15, 13, 7, 9, 11, 6] | [15, 13, 14, 7, 6, 11, 9] |
| Syn(11) | ∅ | [11, 15, 13, 7, 14, 6, 9] |
| Fin(12) | ∅ | ∅ |
| | Syn(11) | Fin(12) |
| IPSRC(0) | [6, 14, 11, 15, 7, 13, 5, 9, 10] | [7, 15, 11, 13, 9, 5, 6, 14, 10] |
| SrcMac(1) | [6, 14, 11, 15, 7, 9, 13, 10] | [7, 15, 11, 13, 9, 5, 6, 14, 10] |
| IPDest(2) | [10, 14, 7, 15, 11, 13, 9, 5, 6] | [11, 15, 7, 13, 5, 9, 10, 14, 6] |
| DstMac(3) | [10, 14, 7, 15, 11, 13, 9, 6] | [11, 15, 7, 13, 5, 9, 10, 14, 6] |
| TCPSrcPort(4) | [6, 14, 11, 15, 7, 5, 13, 9, 10] | [7, 15, 11, 13, 9, 5, 6, 14, 10] |
| TCPDestPort(5) | [10, 14, 7, 15, 11, 13, 9, 5, 6] | [11, 15, 7, 13, 5, 9, 10, 14, 6] |
| TCPSeq#(6) | [6, 11, 15, 7, 5, 9, 10] | [7, 11, 15, 5, 9, 6, 13, 10] |
| TCPNextSeq#(7) | [14, 6, 7, 15, 11, 9, 13] | [15, 7, 11, 9, 13, 14, 6] |
| Ack(8) | [14, 15, 9, 13, 7, 6] | [15, 13, 11, 14, 10, 7] |
| Push(9) | [14, 6, 7, 15, 11, 9, 13] | [15, 7, 11, 9, 13, 14, 6] |
| Reset(10) | [14, 15, 13] | [15, 13, 14] |
| Syn(11) | ∅ | [11, 15, 13, 7, 14, 6] |
| Fin(12) | ∅ | ∅ |

Appendix F. Run Time Data

| # of Packets | V 0.0 | V 1.0 | V 2.44 | V 2.116 | V 2.0 | V 3.0 | V 4.0 |
|--------------|-------|-------|--------|---------|-------|-------|--------|
| 1,000 | 1 | 1 | 19 | 118 | 6 | 124 | 424 |
| 5,000 | 4 | 6 | 73 | 457 | 33 | 455 | 2063 |
| 10,000 | 7 | 11 | 154 | 658 | 64 | 822 | 4033 |
| 25,000 | 20 | 28 | 344 | 1401 | 156 | 2053 | 11330 |
| 50,000 | 36 | 66 | 783 | 3218 | 314 | 4126 | 26624 |
| 100,000 | 96 | 136 | 1170 | 8134 | 697 | 8947 | 54203 |
| 200,000 | 179 | 331 | 2740 | 20710 | 2145 | 19363 | 118407 |

Figure F.1 Run Times of Code Versions
 *Times are indicated in seconds

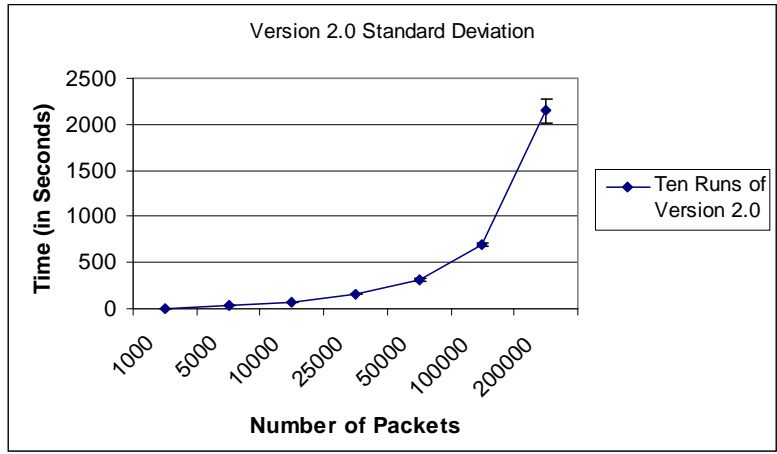


Figure F.2 Standard Deviation
 Ten Runs of Code Version 2.0 on Mon and Tue Week 1 LL Data

Appendix G. Lincoln Lab Attack Descriptions

| | |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Apache 2 | <p>The Apache2 attack is a denial of service attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash. This exploit was adapted from C code originally posted to the bugtraq mailing list. A C- shell wrapper was also created which executes the apache2 C program in a loop until the server being attacked is no longer responsive. As soon as the attack was launched the load average (as reported by the 'top' program) of the victim server jumped to 5 or more. As more and more requests were submitted to the web server the memory usage and load average of the victim continued to climb until eventually the httpd daemon ran out of memory and crashed. At this point the server no longer responded to http requests and the httpd daemon needed to be restarted by the superuser for service to be restored.</p> |
| DoSNuke | <p>DoSNuke is a Denial of Service attack that sends Out Of Band data (MSG OOB) to port 139 (NetBIOS), crashing the NT victim (bluescreens the machine). A Perl script, dosnuke.pl, runs on an NT attacker. Open dosnuke.pl for editing and set the time of day to run the attack. Then run dosnuke.pl or place it in the startup group. The script takes no arguments (always attacks Computer 172.16.112.100).</p> |
| IMap | <p>The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges. The Imap attack used in the 1998 DARPA intrusion detection evaluation was part of the Impack 1.03 attack toolkit. This toolkit contained precompiled binary programs for the Linux platform that would scan for vulnerable machines, as well as send the necessary message to exploit the buffer overflow and gain access to a root shell. The Impack contained detailed instructions on how to use these precompiled programs and took very little skill to use. The release of the Impack made this vulnerability especially dangerous, as any user with a Linux machine and the ability to follow instructions could use this attack to remotely gain root access to any vulnerable hosts.</p> |
| IPSweep | <p>An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines. There are many methods an attacker can use to perform an Ipsweep attack. The most common method and the method used within the simulation is to send ICMP Ping packets to every possible address within a subnet and wait to see which machines respond. The Ipsweep probes in the simulation were not stealthy the sweeps</p> |

Figure G.1 Attack Descriptions (adapted from Lincoln Lab Database [LLab99])

| | |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | were performed linearly, quickly and from a single source. |
| Land | The Land attack is a denial of service attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS 4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address. The land exploit program used in the DARPA evaluation was adapted from a C implementation found at http://www.rootshell.com . The exploit is quite simple and the code could easily be rewritten in any language with access to the TCP sockets interface. The code sends a single SYN packet with the source address spoofed to be the same as the destination address. Within the simulation, this exploit was run against a Sun SPARC WorkStation running SunOS version 4.1. when a TCP SYN packet with an identical source and destination address was received by this host, the system completely locked up. In order to restore service, the machine had to be physically turned off and on again. |
| Mscan | Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities. The Mscan program used in the simulation was compiled from source code found at. Mscan was easy to run and has several command line options for specifying the number of machines to scan and which vulnerabilities to look for. Within the simulation, mscan was used to scan the entire eyrie.af.mil domain for the following vulnerabilities: statd, imap, pop, IRIX machines that have accounts with no passwords, bind, various cgi-bin vulnerabilities, NFS, and open X servers. |
| NT InfoScan | NTInfoScan is a NetBIOS based security scanner. It scans the NT victim to obtain share information, the names of all the users, services running, and other information. The results are saved in an html file named .html where victim is the victim's hostname. View the page from mnemonic for more information. A Perl script runs on an NT attacker. Edit the first line of the ntis.pl with the time of day the attack should run and then run ntis.pl or put it in the Startup group and restart the machine. Ntis.pl automatically scans hume.eyrie.af.mil. The attack may take up to 20 min. to complete. |
| Ping Of Death | The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting. Several implementations of the Ping of Death exploit can be found at http://www.rootshell.com as well as many other sources on the web. This exploit is popular because early versions of the ping program distributed with Microsoft Windows95 would allow the user to create oversize ping packets simply by specifying a parameter at the command line (i.e. ping 65510). Thus, many users could potentially exploit this bug without even making the effort to download and compile a program. The Ping of Death attack affected none of the victim systems used |

| | |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | in the evaluation. The attack was included as an example of an attempted known attack that fails to have an effect. |
| PortswEEP | Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscans options include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order. At the time of the evaluation, Nmap was the most complete publicly available scanning tool. During the simulation, Nmap was used to perform portscans on between one and ten computers using SYN scanning, FIN scanning, and UDP scanning of victim machines. Both sequential and random scans were performed in the simulation, and the timeout between packets was varied to be anywhere from one second to six minutes. The number of ports scanned on each machine was varied between three and one thousand. |
| Queso | QueSO is a utility used to determine a what type of machine/operating system exists at a certain IP adress. QueSO sends a series of 7 tcp packets to any one port of a machine and uses the return packets it receives to lookup the machine in a database of responses. To make the attack more stealthy, we increased the delay between sending the packets. In the 1999 DARPA evaluation machines are sent the 7 QueSO packets with delays of 3, 5, and 10 minutes. |
| Satan | SATAN is an early predecessor of the SAINT scanning program described in the last section. While SAINT and SATAN are quite similar in purpose and design, the particular vulnerabilities that each tools checks for are slightly different. Like SAINT, SATAN is distributed as a collection of perl and C programs that can be run either from within a web browser or from the UNIX command prompt. SATAN supports three levels of scanning: light, normal, and heavy. The vulnerabilities that SATAN checks for in heavy mode are: ? NFS export to unprivileged programs ? NFS export via portmapper ? NIS password file access ? REXD access ? tftp file access ? remote shell access ? unrestricted NFS export ? unrestricted X Server access ? write-able ftp home directory ? several Sendmail vulnerabilities ? several ftp vulnerabilities Scans in light and normal mode simply check for smaller subsets of these vulnerabilities. |
| Self Ping | The selfping attack is a denial of service attack in which a normal user can remotely reboot a machine with a single ping command. This attack can be performed on Solaris 2.5 and 2.5.1. The ping command broadcasts echo request packets using the localhost as the multicast interface. Within a couple seconds the system panics and reboots. There are two version of this attack in the 1999 DARPA evaluation. One version creates an atjob on the victim machine and then logouts. The other, more malicious version, creates a cronjob which reboots the machine every 5 minutes. The administrator must remove the cronjob in order to keep the machine from rebooting. |
| SYNFlood | A SYN Flood is a denial of service attack to which |

| | |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <p>every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the ``tcpd" server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative. The neptune exploit code used in the simulation was compiled from C code originally posted to the bugtraq archive. The neptune program allows the user to specify a victim host, the source address to use in the spoofed packets, the number of packets to send, and the ports to hit on the victim machine (including an ``infinity" option that would attack all ports). The neptune exploit was effective against all three of the victim machines used in the simulation. Every TCP/IP implementation is vulnerable to this attack to a varying degree depending on the size of the data structure used to store incoming connections and the timeout value associated with half-open connections. As a point of reference, sending twenty SYN packets to a port on a Solaris 2.6 system will cause that port to drop incoming requests for approximately ten minutes. During the simulation, a neptune attack which sent 20 SYN packets to every port from 1 to 1024 of the Solaris server once every ten minutes was able block incoming connections to any of these ports for more than an hour.</p> |
| TearDrop | <p>The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments. The teardrop name is derived from a widely available C program that exploits this vulnerability. This exploit code can be found at http://www.rootshell.com and in the Bugtraq archives. Although many systems are rumored to be vulnerable to the teardrop attack, of the systems used in the DARPA evaluation, only the Redhat Linux 4.2 systems were vulnerable. The teardrop attack would cause these machines to reboot.</p> |
| Udpstorm | <p>A Udpstorm attack is a denial of service attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack;</p> |

no account access is needed. For example, by connecting a host's chargen service to the echo service on the same or another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. An illustration of such an attack is presented in Figure 6-2. The figure demonstrates how an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. First, the attacker forges a single packet that has been spoofed to look like it is coming from the echo port on the first victim machine and sends it to the second victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request, so when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in like kind, and the loop of traffic continues until it is stopped by intervention from an external source [10]. Code that exploits this vulnerability was posted to the bugtraq mailing list. This program sends a single spoofed UDP packet to a host. This single spoofed packet is able to create a never-ending stream of data being sent from the echo port of one machine to the echo port of another. This loop created network congestion and slowdown that would continue until the inetd daemon was restarted on one of two victim machines

Appendix H. Screen Shots of Attacks Analyzed with GOPHER

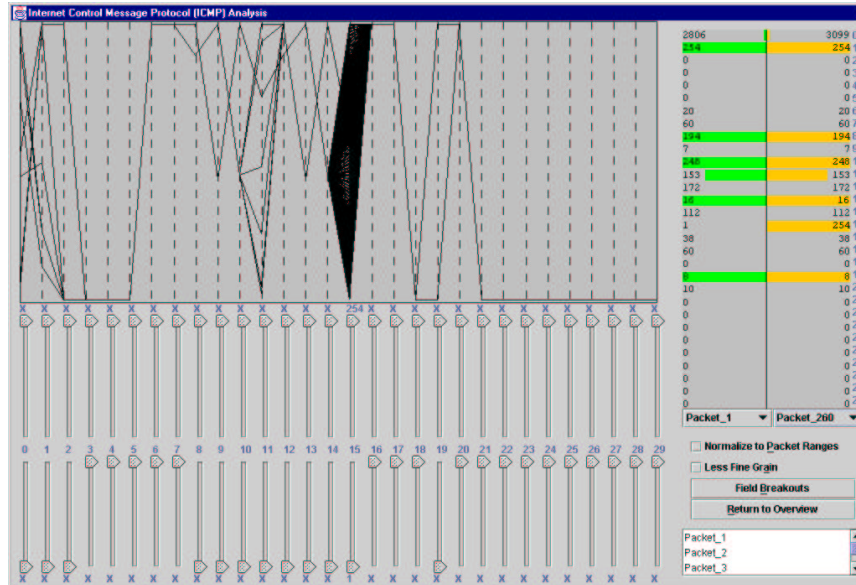


Figure H.1 Attack 45.192523 IP Sweep

This attack sweeps through IP destination addresses 172.116.112.1 to 172.116.112.254 consecutively, as can be seen in field 15

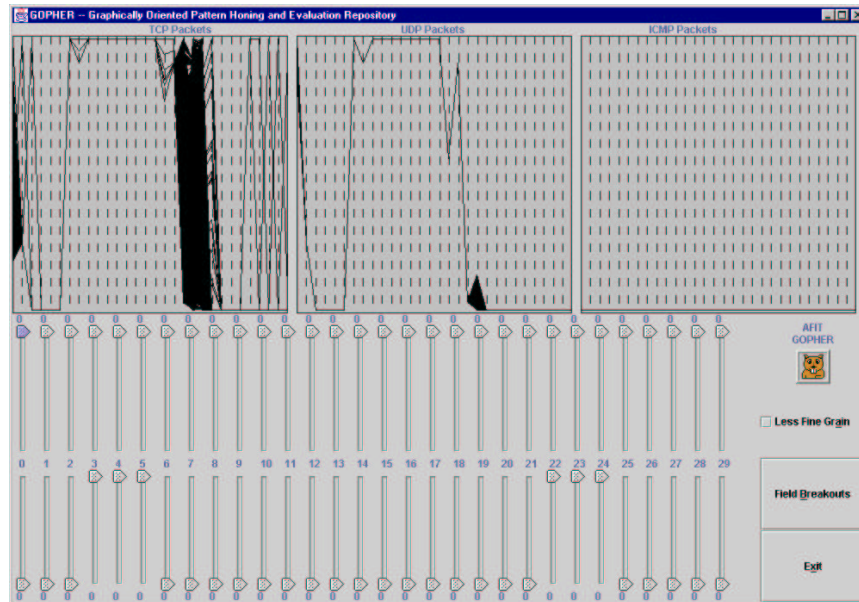


Figure H.2 Attack 54.145832 Satan Scan Main Screen

This screen shot shows that this scan was conducted in both TCP and UDP

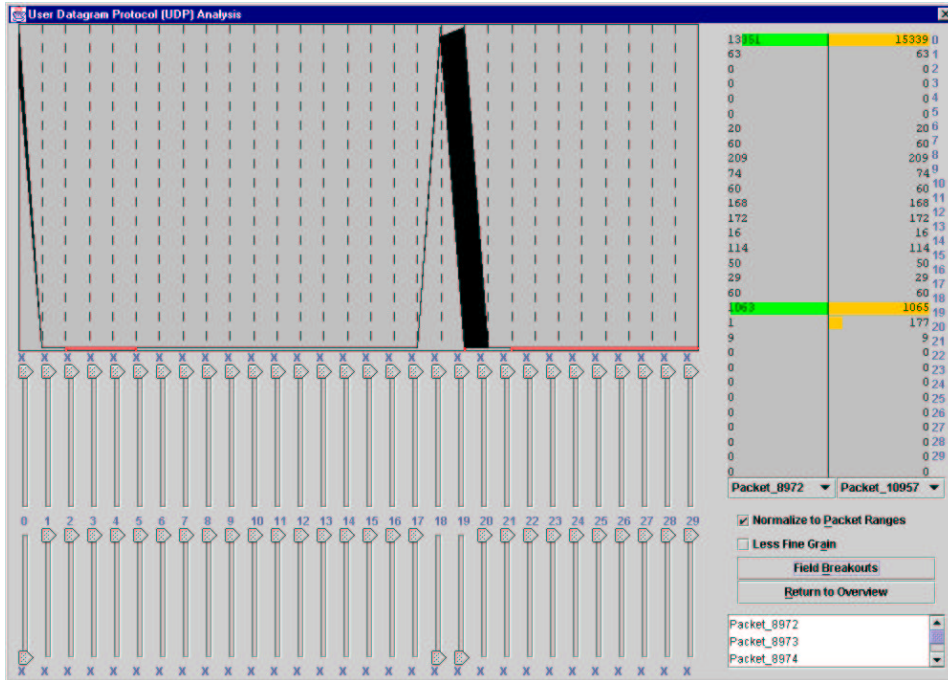


Figure H.3 Attack 54.145832 Satan Scan UDP Drill Down
 This attack scanned UDP destination ports 1 - 177 (field 19)

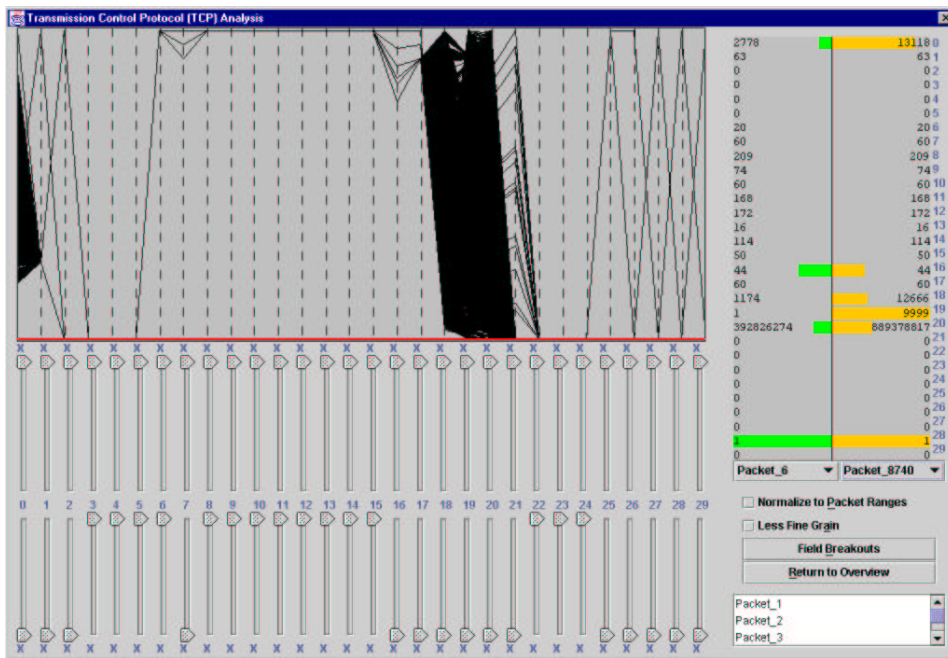


Figure H.4 Attack 54.145832 Satan Scan TCP Drill Down
 This attack scanned TCP destination ports 1 - 9999 (field 19) from source ports
 878-32830 (field 18)

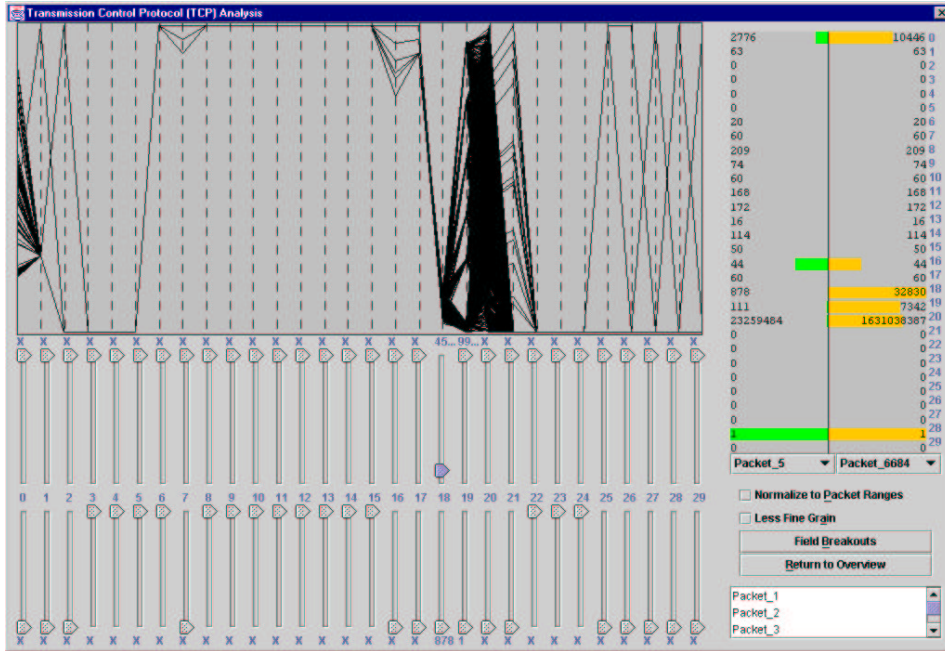


Figure H.5 Attack 54.145832 Satan Scan TCP Slice
 Field 18 sliders were used to show a window of the attack. This shows groups of TCP destination ports being scanned at once

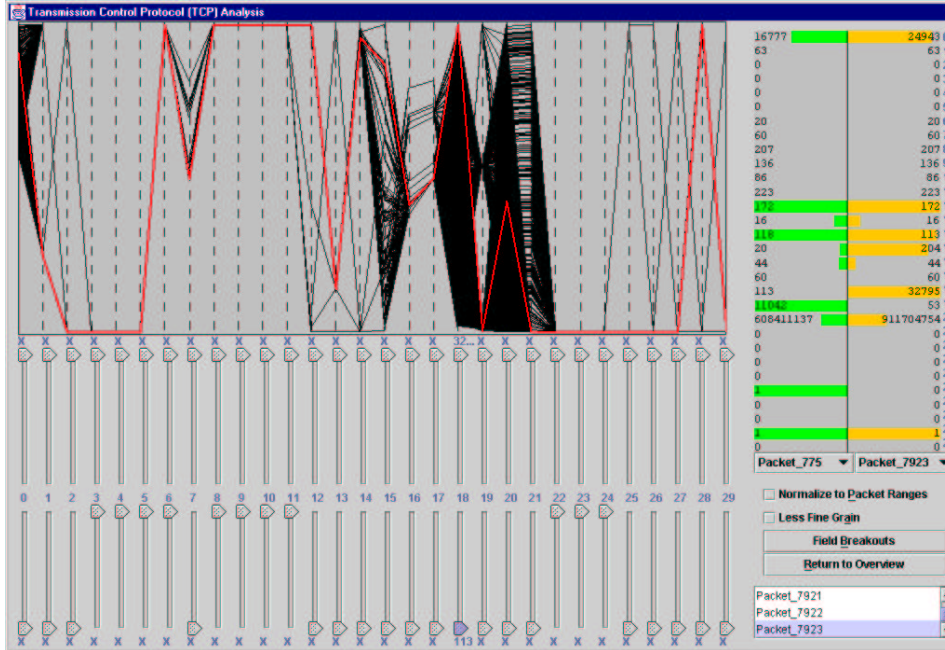


Figure H.6 Attack 54.195951 Mscan
 Field 18 shows this attack scanning from TCP source ports 113-3279

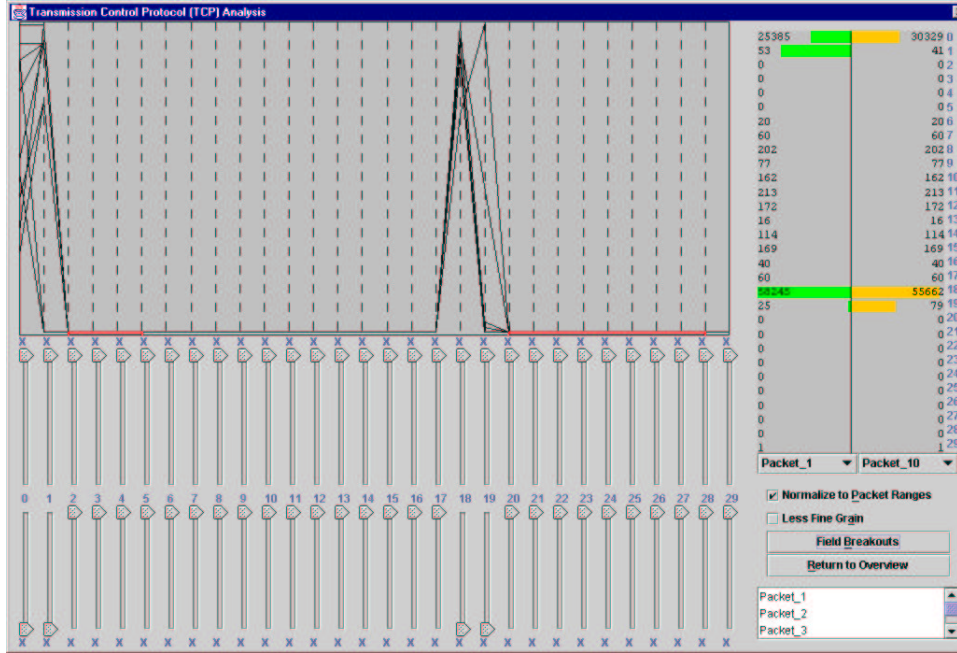


Figure H.7 Attack 41.162715 Portsweep

This shows all 10 packets of this attack have a TCP sequence number of 0 (field 20)

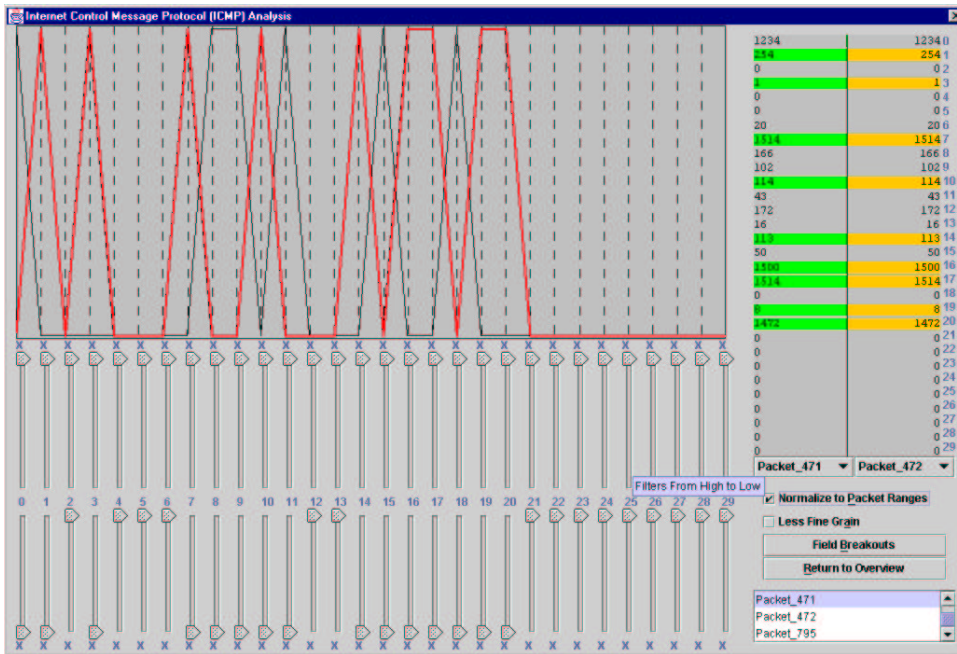


Figure H.8 Attack 52.130655 Ping of Death

The 17 packets of this attack look like two since they are repeated packets

Appendix I. Source Code Availability

The Java code for the different multiple-packet intrusion detection sensors and GOPHER are not included as part of this document. Those interested in obtaining a copy of the source code should direct their requests to:

Dr. Gregg Gunsch

AFIT/ENG

2950 P Street

WPAFB, OH 45433-7765

gregg.gunsch@afit.edu

Bibliography

- [Sans99] “Global Incident Analysis Center Special Notice–ECN and its Impact on Intrusion Detection.” World Wide Web, 1999. World Wide Web Page. URL <http://www.sans.org/y2k/ecn.htm>.
- [Alle00] Allen, Julia, et al. *State of the Practice of Intrusion Detection Technologies*. Technical Report CMU/SEI-99-TR-028, Carnegie Mellon Software Engineering Institute, January 2000.
- [Almg01] Almgren, Magnus and Ulf Lindqvist. “Application-Integrated Data Collection for Security Monitoring.” *Recent Advances in Intrusion Detection (RAID 2001)*. LNCS. 22–36. Davis, California: Springer, October 2001.
- [Amor99] Amoroso, Edward. *Intrusion Detection; An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*. Intrusion.Net Books, Sparta, NJ, USA, 1999.
- [Axen01] Axent Technologies, Inc., “NetProwler Dynamic Network Intrusion Detection.” World Wide Web, 2001. World Wide Web Page. URL http://enterprisesecurity.symantec.com/PDF/AxentPDFs/NetProwler_Dsht_Sy%mantec.pdf.
- [Bace01] Bace, Rebecca and Peter Mell, “Intrusion Detection Systems.” White Paper, 2001. Available electronically at URL <http://www.snort.org/docs/nist-ids.pdf>.
- [Brid00] Bridges, Susan M. and Rayford B. Vaughn. “Fuzzy Data Mining and Genetic Algorithms Applied to Intrusion Detection.” *Proceedings of the 23rd National Information Systems Security Conference*. 13–31. National Institute of Standards and Technology, National Computer Security Center, 2000.
- [Chap01] Chappell, Laura, “You’re Being Watched: Cyber-Crime Scans.” White Paper, 2001. World Wide Web Page. URL http://www.nwconnection.com/2001_03/cybercrime/.
- [Cisc01] Cisco, Systems, “Cisco Intrusion Detection.” World Wide Web, 2001. World Wide Web Page. URL <http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/index.shtml>.
- [CEnc00] Columbia University, in the City of New York. *The Columbia Electronic Encyclopedia, Sixth Edition*, 2000.
- [Cert98] Computer Emergency Response Team Coordination Center, Carnegie Mellon Software Engineering Institute, “CERT Advisory CA-1998-12 Remotely Exploitable Buffer Overflow Vulnerability in mountd.” World Wide Web, 1998. World Wide Web Page. URL <http://www.cert.org/advisories/CA-1998-12.html>.
- [Dasg98] Dasgupta, Dipankar, editor. *Artificial Immune Systems and Their Applications*. Berlin: Springer-Verlag, 1998.

- [Deba92] Debar, Hervé, et al. "A Neural Network Component for an Intrusion Detection System." *Proceedings of the 1992 IEEE Computer Society Symposium on Reserach in Security and Privacy*. 240–250. IEEE, IEEE Service Center, Piscataway, NJ, 1992.
- [Durs99] Durst, Robert, et al. "Testing and Evaluating Computer Intrusion Detection Systems," *Communications of the ACM*, 42(7) (1999).
- [Forr98] Forrest, Stephanie, et al. "Artificial Neural Networks for Misuse Detection." *Proceedings of the 21st National Information Systems Security Conference*. 441–454. 1998.
- [Forr97] Forrest, Stephanie, et al. "Computer Immunology," *Communications of the ACM*, 40:88–96 (1997).
- [Ghos99a] Ghosh, A. and A. Schwartzbard, "A study in using neural networks for anomaly and misuse detection," 1999.
- [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/.
- [Glob01] Global Spy Shop, "Intrusion Detection Alarms and Cameras." World Wide Web, 2001. World Wide Web Page. URL <http://www.globalspyshop.com/IntrusionDetection.htm>.
- [Grah00] Graham, Robert, "FAQ: Network Intrusion Detection Systems." World Wide Web, 2000. World Wide Web Page. URL <http://www.robertgraham.com/pubs/network-intrusion-detection.html>.
- [Harm00] Harmer, Paul. *A Distributed Agent Architecture for a Computer Virus Immune System*. MS thesis, AFIT/GCE/ENG/00M-02, School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 2000.
- [Harm01] Harmer, Paul, et al. "A Distributed Agent Based Architecture for Computer Security Applications," *To Appear in IEEE Transactions On Evolutionary Computation, Special Issue on Artificial Immune Systems* (2001).
- [Hofm99] Hofmeyr, Steven and Stephanie Forrest. "Architecture for an Artificial Immune System," *Evolutionary Computation*, 7(1):1289–1296 (1999).
- [Hofm99a] Hofmeyr, Steven A., "An Interpretive Introduction to the Immune System." White Paper, 1999. University of New Mexico, Albuquerque, NM 87131.
- [Inse97] Inselberg, Alfred. "Multidimensional Detective." *Proceedings of IEEE Symposium on Information Visualization, InfoVis '97*. 1997. Published in Readings in Information Visualization Using Vision to Think, Morgan Kaufmann Publishers, Inc.
- [ISS00] Internet Security Systems, Inc., "Internet Security Systems Ships RealSecure for Windows 2000." World Wide Web, 2000. World Wide Web Page. URL <http://bvllive01.iss.net/issEn/delivery/prdetail.jsp?oid=14714>.

- [ISSI98] Internet Security Systems, Inc. *Network vs. Host Based Intrusion Detection*, 1998. World Wide Web Page. URL http://secinf.net/info/ids/nvh_ids/.
- [Jane97] Janeway, Charles and Paul Travers. *Immuno Biology The Immune System in Health and Disease*. Garland Publishing Inc, New York and London, 1997.
- [Jonc95] Joncheray, Laurent. *A Simple Active Attack Against TCP*, 1995. World Wide Web Page. URL <http://citeseer.nj.nec.com/joncheray95simple.html>.
- [Kemmm01] Kemmerer, Richard A. *STAT Projects*, 2001. World Wide Web Page. URL <http://www.cs.ucsb.edu/~kemm/netstat.html/projects.html>.
- [Keph97] Kephart, Jeffrey and William Arnold, "Automatic Extraction of Computer Virus Signatures," 1994.
- [Keph97a] Kephart, Jeffrey, et al., "Blueprint for a Computer Immune System," 1997.
- [Kim01a] Kim, Jungwon and Peter Bentley. "An Evaluation of Negative Selection in an Artificial Immune System for Network Intrusion Detection." *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*. 2001.
- [Kim01] Kim, Jungwon and Peter Bentley. "Towards an Artificial Immune System for Network Intrusion Detection: An Investigation of Clonal Selection with a Negative Selection Operator." *Proceedings of the Congress on Evolutionary Computation (CEC-2001)*. 2001.
- [LLab99] Laboratory, Lincoln. *Lincoln Laboratory Database of Attacks*. MIT Lincoln Laboratory, 1999. Available electronically at <http://ideval.II.mit.edu/Links/attackDB.html>.
- [Lamo99] Lamont, Gary B., et al. *New Ideas in Optimization, A Distributed Architecture for a Self-Adaptive Computer Virus Immune System*, chapter 11, 167–183. McGraw-Hill, 1999.
- [LeeW00a] Lee, Wenke, et al. "Adaptive Intrusion Detection: a Data Mining Approach," *Artificial Intelligence Review*, 14(6):533–567 (2000). Available electronically at URL <http://www.csc.ncsu.edu/faculty/lee/publications.html>.
- [LeeW00] Lee, Wenke, et al. "A Data Mining and CIDF Based Approach for Detecting Novel and Distributed Intrusions." *Proceedings of Recent Advances in Intrusion Detection, Third International Workshop*. 49–65. Berlin: Springer, 2000. Available electronically at URL <http://www.csc.ncsu.edu/faculty/lee/publications.html>.
- [LeeW99a] Lee, Wenke, et al. "Automated Intrusion Detection Methods Using NFR." *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*. 1999.
- [LeeW98] Lee, Wenke and Salvatore Stolfo. "Data Mining Approaches for Intrusion Detection." *Proceedings of the 7th USENIX Security Symposium*. 1998.
- [LeeW99] Lee, Wenke, et al. "A Data Mining Framework for Building Intrusion Detection Models." *Proceedings of 1999 IEEE Symposium on Security and Privacy*.

1999. Available electronically at URL
<http://www.csc.ncsu.edu/faculty/lee/publications.html>.
- [LeeW00b] Lee, Wenke, et al., “Real Time Data Mining-based Intrusion Detection.” White Paper, 2000. Available electronically at URL
<http://citeseer.nj.nec.com/452795.html>.
- [Mand01] Manderscheid, Scott, “An Intrusion Detection System Process: Defense in Depth.” World Wide Web, 2001. World Wide Web Page. URL
<http://www.sans.org/infosecFAQ/intrusion/process.htm>.
- [Mans00] Mansfield, Glenn, et al. “Towards trapping wily intruders in the large,” *34*(4):659–670 (October 2000).
- [McCl99] McClure, Stuart and Joel Scambray. *Hacking Exposed*, 1. Osborne/McGraw-Hill, 1999.
- [McHu00] McHugh, John. “The 1998 Lincoln Laboratory IDS Evaluation: A Critique.” *Proceedings of Recent Advances in Intrusion Detection, Third International Workshop*. 145–161. Berlin: Springer, 2000.
- [Me98] Mé, Ludovic. “GASSATA, A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis.” *Proceedings of the 1st Workshop on Recent Advances in Intrusion Detection*. Berlin: Springer-Verlag, 1998.
- [Mich00] Michalewicz, Z. and D. Fogel. *How to Solve It: Modern Heuristics*. Berlin, Germany: Springer-Verlag, 2000.
- [Neri00] Neri, Filippo. “Comparing Local Search with Respect to Genetic Evolution to Detect Intrusions in Computer Networks.” *Proceedings of the 2000 Congress on Evolutionary Computation*. 238–243. Piscataway, NJ: IEEE Service Center, 2000.
- [NFRS01] Network Flight Recorder Security, Inc. *Overview of NFR Network Intrusion Detection*. Technical Report, Network Flight Recorder Security, Inc., July 2001.
- [Nort00] Northcutt, Stephen, “Intelligence Gathering Techniques.” White Paper, 2000. World Wide Web Page. URL <http://www.allright.com/S/Security/Intelligence%20Gathering%20Technique%20s.htm>.
- [Paxs98] Paxson, Vern. “Bro: A System for Detecting Network Intruders in Real-Time.” *Proceedings of the 7th USENIX Security Symposium*. San Antonio, TX: USENIX, 1998.
- [RFC826] Plummer, David C., “An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware.” White Paper, 1982. Available electronically at URL
<http://sunsite.dk/RFC/rfc/rfc826.html>.
- [Porr97] Porras, Phillip A. and Peter G. Neumann. “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances.” *1997 National Information Systems Security Conference*. oct 1997.

- [Proc01] Proctor, Paul E., "Computer Misuse Detection System (CMDS) Concepts." Technical Paper, 2001. World Wide Web Page. URL <http://publications.saic.com/>.
- [Ptac98] Ptacek, Thomas H. and Timothy N. Newsham. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*, 1998. Secure Networks paper, available electronically at URL http://www.nai.com/media/ps/nai_labs/ids.ps.
- [Rhod00] Rhodes, Brandon C., et al. "Multiple Self-Organizing Maps for Intrusion Detection." *Proceedings of 23rd National Information Systems Security Conference*. IEEE, IEEE Service Center, Piscataway, NJ, 2000.
- [Roes01a] Roesch, Marty, "SNORT Users Manual." World Wide Web, 2001. Available electronically at URL <http://www.snort.org/>.
- [Roes01] Roesch, Marty. *Writing Snort Rules: How to Write Snort Rules and Keep Your Sanity*, 2001. World Wide Web Page. URL http://www.snort.org/writing_snort_rules.htm.
- [Sans01] SANS Institute. *Intrusion Detection FAQ*, 2001. World Wide Web Page. URL http://www.sans.org/newlook/resources/IDFAQ/ID_FAQ.htm.
- [SANS01a] SANS Institute, "SANS Alert Consensus Number 119." World Wide Web, 2001. Available electronically at <http://www.mail-archive.com/archive@jab.org/msg47460.html>.
- [Soma97] Somayaji, A., et al. "Principles of a Computer Immune System." *1997 New Security Paradigms Workshop*. 75–82. Association for Computing Machinery, 1997.
- [Stan00] Staniford, Stuart, et al. "Practical Automated Detection of Stealthy Portscans." *Proceedings of the 7th ACM Conference on Computer and Communication Security*. 2000. Available electronically at <http://www.silicondefense.com/pptntext/spice-ccs2000.pdf>.
- [Stev99] Stevens, W. Richard. *TCP/IP Illustrated: The Protocols, 1*. Addison Wesley, 1999.
- [USGAO01] United States General Accounting Office. Washington, DC. *Information Security : Challenges to Improving DOD's Incidents Response Capabilities*. Technical Report GAO-01-341, United States General Accounting Office. - Washington, DC, March 2001. report to the Chairman, Cmte. on Armed Services, House / United States General Accounting Office.
- [Whit94] Whitley, Darrell. "A Genetic Algorithm Tutorial," *Statistics and Computing*, 4:65–85 (1994).
- [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 Recent Advances in Intrusion Detection, Fourth International Workshop*. Berlin: Springer, 2001.

Vita



Lieutenant Bebo enlisted in the Air Force in 1985 as an Airborne Cryptologic Linguist and has flown in several reconnaissance aircraft including the RC-135V/W (Rivet Joint), the RC-135U (Combat Sent), the RC-135S (Cobra Ball), the C-130 (Senior Scout), and the E-3B/C (AWACS). He rose to the rank of Technical Sergeant prior to being selected for the Air Force Reserve Officer Training Corps (AFROTC). After graduating Magna Cum Laude from the University of Nebraska at Omaha with a bachelor's degree in Computer Science, he was assigned to Offutt Air Force Base, where he worked as the Officer in Charge of MILSTAR Operations. While at Offutt, he applied and was accepted to pursue his Masters of Science degree at AFIT. Lieutenant Bebo's military awards include the Air Medal, five Aerial Achievement Medals, three Air Force Commendation Medals, two Air Force Achievement Medals, the Combat Readiness Medal, the Armed Forces Expeditionary Medal, the Southwest Asia Service Medal, and the Kuwait Liberation Medal. Additionally, Lieutenant Bebo has been selected as a distinguished graduate from the Korean and Slavic Cryptologic Linguist courses, as well as from AFROTC Field Training and AFROTC Detachment 470. Upon graduation Lieutenant Bebo will be assigned to the Air Force Research Laboratories, Rome, New York.

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) 28 02 2002 | 2. REPORT TYPE Master's Thesis | 3. DATES COVERED (From - To) Jan 2001 - Mar 2002 |
|--------------------------------------------------|------------------------------------------|------------------------------------------------------------|

| | |
|------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| 4. TITLE AND SUBTITLE USING RELATIONAL SCHEMATA IN A COMPUTER IMMUNE SYSTEM TO DETECT MULTIPLE-PACKET NETWORK INTRUSIONS | 5a. CONTRACT NUMBER |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |

| | |
|--------------------------------------------------------------|-------------------------------------|
| 6. AUTHOR(S) Bebo, John L., First Lieutenant, 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/GCS/ENG/02M-02 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|

| | |
|------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFGB Attn: Mr. John Feldman 525 Brooks Rd. Rome, NY 13441-4505 | 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 Given the increasingly prominent cyber-based threat, there are substantial research and development efforts underway in network and host-based intrusion detection using single-packet traffic analysis. However, there is a noticeable lack of research and development in the intrusion detection realm with regard to attacks that span multiple packets. This leaves a conspicuous gap in intrusion detection capability because not all attacks can be found by examining single packets alone. Some attacks may only be detected by examining multiple network packets collectively, considering how they relate to the "big picture," not how they are represented as individual packets. This research demonstrates a multiple-packet relational sensor in the context of a Computer Immune System (CIS) model to search for attacks that might otherwise go unnoticed via single-packet detection methods. Using relational schemata, multiple-packet CIS sensors define "self" based on equal, less than, and greater than relationships between fields of routine network packets. Attacks are then detected by examining how the relationships among attack packets may lay outside of the previously defined "self." |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

| |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15. SUBJECT TERMS intrusion detection, relational schemata, relational schema, anomaly detection, computer networks, packet visualization, multiple-packet detection |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

| | | | |
|----------------------------------------|-----------------------------------|----------------------------|-----------------------------------------------------------------------------------------------------|
| 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 | Professor Gregg Gunsch, ENG |
| | UU | 143 | 19b. TELEPHONE NUMBER (Include area code) (937) 255-6565, ext 4281; gregg.gunsch@afit.edu |