SESSION-BASED INTRUSION DETECTION SYSTEM TO MAP ANOMALOUS NETWORK TRAFFIC

by

BRUCE D. CAULKINS
B.S. Furman University, 1986
M.S. University of Central Florida, 1995

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Modeling and Simulation
in the College of Arts and Sciences
at the University of Central Florida
Orlando, Florida

Fall Term
2005

Major Professor: Morgan C. Wang

© 2005 Bruce D. Caulkins

ABSTRACT

Computer crime is a large problem (CSI, 2004; Kabay, 2001a; Kabay, 2001b). Security managers have a variety of tools at their disposal – firewalls, Intrusion Detection Systems (IDSs), encryption, authentication, and other hardware and software solutions to combat computer crime. Many IDS variants exist which allow security managers and engineers to identify attack network packets primarily through the use of signature detection; i.e., the IDS recognizes attack packets due to their well-known "fingerprints" or signatures as those packets cross the network's gateway threshold. On the other hand, anomaly-based ID systems determine what is normal traffic within a network and reports abnormal traffic behavior. This paper will describe a methodology towards developing a more-robust Intrusion Detection System through the use of data-mining techniques and anomaly detection. These data-mining techniques will dynamically model what a normal network should look like and reduce the false positive and false negative alarm rates in the process. We will use classification-tree techniques to accurately predict probable attack sessions. Overall, our goal is to model network traffic into network sessions and identify those network sessions that have a high-probability of being an attack and can be labeled as a "suspect session." Subsequently, we will use these techniques inclusive of signature detection methods, as they will be used in concert with known signatures and patterns in order to present a better model for detection and protection of networks and systems.

To my wife Kim,

whose love, patience, and support

meant everything for my research

ACKNOWLEDGMENTS

Many thanks to the members of my committee, Drs. Morgan C. Wang, Joohan Lee, William H. Allen, Mark E. Johnson, J. Peter Kincaid, and Michael D. Proctor.

Much credit goes to Morgan Wang for his guidance and mentorship. In a multi-disciplinary area like modeling and simulations, it takes a special professor who has wide-ranging knowledge and patience to make this research work well and go smoothly. Special thanks also goes to Joohan Lee – a specialist in computer security and reliable computing, he provided an incredible insight into the problems that we encountered during our research. Bill Allen at FIT also gave key assistance with his extensive expertise as well as the FIT data sets.

The US Army Human Resources Command (HRC) provided support for this research through the Army's fully funded Advanced Civil Schooling (ACS) program. I thank them as well as my boss at my current assignment at the US Army Signal Center at Fort Gordon, Georgia – Colonel Mackey. He as well as the good folks at the Army's School of Information Technology (SIT) gave me the time and encouragement to complete my research properly and on time.

Finally, I'd like to thank my family and friends, particularly my wife Kim. Much has happened to all of us since Kim and I started this journey in August 2002 and my dad's onslaught of dementia was the absolute low point over the last three years. However, he always told me to study hard, never quit, take as many English classes as you can in college, and always put the family first. His current condition won't allow him to remember ever saying those words but that's fine – I've never forgotten that advice.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

ACK – Acknowledge (TCP flag)

ACM – Association of Computing Machinery

ACMCCS – ACM conference on Computer and Communications Security

ARPANET – Advanced Research Projects Agency Network

ASCII – American Standard Code for Information Interchange

BBC – British Broadcasting System

BBN – Bolt, Beranek, and Newman

CA – CERT Advisory

CART – Classification and Regression Trees

CCITT - International Telegraph and Telephone Consultative Committee

CD – Compact Disc

CERT – Computer Emergency Response Team

CERT/CC – Computer Emergency Response Team/Coordination Center

CLAD – Clustering Anomaly Detection engine

CS – Computer Science

CSO – Chief Security Officer

CSI – Computer Security Institute

DARPA – Defense Advanced Research Projects Agency

DDOS – Distributed Denial of Service attack

DHCP – Dynamic Host Configuration Protocol

DIMVA – Detection of Intrusions and Malware & Vulnerability Assessment

DISCEX II – 2nd DARPA Information Survivability Conference and Exposition

DNS – Domain Name System

DOS – Denial of Service attack

DTC – Decision Tree Classifier

DW – Deutsche Welle (German; German Wave)

EFSA – Extended Finite State Automata

FAQ – Frequently Asked Question

FBI – Federal Bureau of Investigation

FIN – Finished (TCP flag)

FIT – Florida Institute of Technology

FTP – File Transfer Protocol

GAO – Government Accountability Office

GB – Gigabyte (1,024 megabytes)

GHz – Gigahertz

GTE – General Telephone and Electronics

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IA – Information Assurance

ICMP – Internet Control Message Protocol

ID – Intrusion Detection

IDEVAL – Intrusion Detection Evaluation

IDS – Intrusion Detection System

IEEE – Institute of Electrical & Electronic Engineers

IEN – Internet Engineering Note

IIS – Internet Information Server

IMTEC – Information Management and Technology Division (GAO)

IP – Internet Protocol

IPS – Intrusion Protection System

IPSec – Internet Protocol Security

ISI – Information Sciences Institute

ISO – International Standards Organization

IT – Information Technology

KGB - Komitet Gosudarstvennoy Bezopasnosti (Russian; Committee for State Security)

LAN – Local Area Network

LERAD – Learning Rule Anomaly Detector

LL – Lincoln Labs

LSASS – Local Security Authority Subsystem

MB – Megabyte (1,024 Kilobytes)

MIT – Massachusetts Institute of Technology

NFS – Network File System

NIC – Network Interface Card

NID – Network Intrusion Detection

NIST - National Institute of Standards and Technology

OSI – Open Systems Interconnection

PBS – Public Broadcasting System

PHAD – Packet Header Anomaly Detection

PSH – Push (TCP flag)

PSS – Product Support Services

RAID – Recent Advances in Intrusion Detection

RFC – Request For Comments

RST – Reset (TCP flag)

SAND – Session-based Anomaly Notification Detector

SANS – Systems Administration, Networking, and Security

SAS – Statistical Analysis System Corporation

SEMMA – Sample, Explore, Modify, Model, and Assess

SIAM – Society for Industrial and Applied Mathematics

SMTP – Simple Mail Transfer Protocol

SNMP – Simple Network Management Protocol

SQL – Structured Query Language

SSH – Secure Shell

SSL – Secure Sockets Layer

SYN – Synchronized (TCP flag)

TCP – Transmission Control Protocol

TFN – Tribal Flood Network

TGT – Target Variable

UCF – University of Central Florida

UDP – User Datagram Protocol

URG - Urgent (TCP flag)

USENIX – Unix Users Group

UTF – Universal Transformation Format

VHS – Video Home System

WAN – Wide Area Network

WWW – World Wide Web

ZIP – Zone Improvement Plan code

CHAPTER 1: INTRODUCTION

One of the most pervasive issues in the computing world today is the problem of security – how to protect computer networks and systems from intrusions, disruptions, and other malicious activities from unwanted attackers (M. Bednarczyk, Guili, & J. Bednarczyk, 2005).

Many of the problems seen in today's connected society are preventable; that is, the vast majority of the system and network vulnerabilities that exist are known and preventable. RFC 2828, Internet Security Glossary, defines the terms threat and vulnerability (Shirey, 2000):

Threat – a potential danger that might exploit a vulnerability that can be either intentional or accidental

Vulnerability – a flaw in a system's design or implementation that can be exploited for malicious gain

To combat these threats and mitigate the latent vulnerabilities, security standards and protocols are developed to improve the security and confidentiality between computers and peripherals while ensuring the intended traffic successfully gets to its destination. Secure Sockets Layer (SSL), Hypertext Transport Protocol Secure (HTTPS), Secure Shell (SSH), and Internet Protocol Security (IPSec) are just a few of the protocols and standards available for use in the computer security field.

When the standards and protocols for today's Internet were established in the 1960s and 1970s, not much attention was given towards developing security protocols and equipment. Most of the work at that time was devoted to linking computers and local area networks together

over vast distances (Postel, 1977). Any introduction of security equipment at that juncture would have inevitably reduced the effectiveness of any interconnectivity testing or implementation.

Over time, however, international standards committees like the International Consultative Committee for Telegraphy and Telephony (CCITT) realized that more emphasis needed to be placed upon security services and standards. So, in 1991 the CCITT delineated the security services that can be provided within the framework of the OSI Reference Model (CCITT, 1991; Stallings, 2003):

1. Authentication – ensuring that the computer entity is who they claim to be.
2. Access Control – ensuring that unauthorized use of a particular resource (server, router) is prevented.
3. Data Confidentiality – ensuring that the data is not disclosed to unauthorized personnel.
4. Data Integrity – ensure that the data received is the same data sent by an authorized person.
5. Nonrepudiation – ensuring both the sending and receiving entities actually sent or received that message.

1.1 A Historical Look at Computer Security Issues

Computer crime has been around for a long time. Whether as an intellectual exercise or as a deliberate attack, hacking can cause much damage and havoc. Two major events – while

positive in nature – had some negative aspects to them as well: the creation of the predecessor of the Internet – the Advanced Research Projects Agency Network (ARPANET) (Postel, 1981a; Smithsonian Institution Online, n.d.) and the creation of the World Wide Web (Berners-Lee, 1992), which runs on top of the Internet backbone network.

While both events were welcome additions to the realm of information dissemination, they also were technologies that provided little in the way of information security and assurance. In fact, most of the efforts surrounding these events concentrated in getting the information from one computer to another; therefore, layers of security would have impeded those goals at that time. As time went on and more users and companies were dependent upon the Internet and the WWW, security issues became more important and less of a nuisance to network managers.

1.1.1 The 1980s and 1990s

1.1.1.1 Cuckoo's Egg

More than a decade ago, Cliff Stoll wrote a book detailing his encounter with computer espionage during the waning days of the Cold War (Stoll, 1989). In "The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage," Stoll describes how a 75-cent accounting error eventually led him to unravel a KGB-financed spy ring in West Germany that cracked dozens of highly-sensitive computer networks in the United States.

Although the events in the book occurred in the late 1980s, the lessons learned are still relevant today. Stoll points out that he used a rudimentary intrusion detection system (hard copy printouts) and tracked the hacker across many computer systems. The text is a must-read as well as the companion video from PBS NOVA (Stoll, 1990) that showcases the difficulty Stoll had while tracking the hacker.

1.1.1.2 Internet Worm and Mitnick's Attacks

Other high-profile attacks included the Morris Internet Worm in 1988 (GAO, 1989) and Kevin Mitnick's attacks and subsequent capture by authorities as described in Shimomura's book *Takedown* (Shimomura, 1995; Shimomura & Markoff, 1996). In a subsequent report on the Morris worm, the Government Accountability Office (GAO) determined that the estimated cost of the inadvertent attack was around $10 million and affected 6,000 computers (GAO, 1989b). One positive outcome of this event was the creation of the Computer Emergency Response Team/Coordination Center (CERT/CC) in 1988 (CERT/CC, 2005a).

The worm was preventable if systems administrators took proper steps to update and patch their operating systems. After all, the Internet Worm exploited software bugs in unpatched Unix operating systems and replicated rapidly to other accessible, non-patched (or poorly-patched) computer systems. The affected systems were eventually rendered useless as they were overloaded with processing multiple copies of the worm program (CERT/CC, 2005a).

The tools have improved over the past fifteen years but the patience and dedication required to track hackers have not changed. In 1998 I was part of an Army team that started a ten-day network and systems security course for US Army military and civilian personnel at Fort Gordon, Georgia. In that course the instructors describe and teach the tools needed to detect and hopefully stop hackers. As an introductory piece, we use excerpts from the Stoll book to underscore the need for accurate and complete network and systems security in any organization.

1.1.2 More Recent Malicious Events

More recent attacks include the so-called Love Letter worm (CERT/CC, 2000) (or, ILOVEYOU virus) and the CodeRed worm (Symantec, 2001). Both attacks proved to be damaging but ultimately manageable.

Other attacks like the SQL Slammer (Sapphire) (Moore, Paxson, Savage, Shannon, Staniford, & Weaver, 2003), SoBig.F and MyDoom (MessageLabs, 2004), and W32.Blaster/W32.Nachi (MyCERT, 2005; Microsoft, 2003) all underscore the need for constant vigilance and dedication on the part of systems administrators and security managers. In Figure 1.1 below (MyCERT, 2005), the damage to one network from the W32.Blaster and W32.Nachi worms are shown:

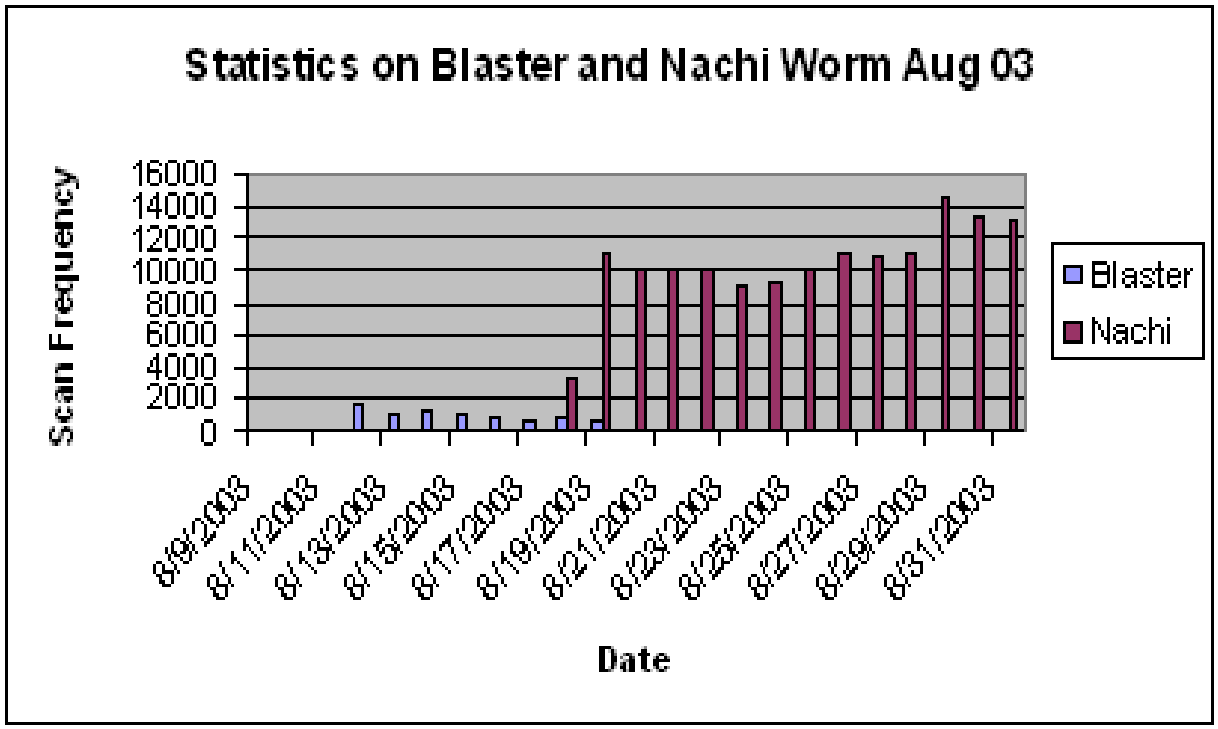


Figure 1. 1. W32.Blaster/W32.Nachi Worms Statistics

The Sasser Worm (Microsoft, 2004a) is another, more-recent example of a destructive application sent over the Internet. Sasser exploits a buffer overflow vulnerability in the Microsoft Local Security Authority Subsystem (LSASS) process and normally uses TCP port 445 on the victim's computer (ISS, 2004). Unfortunately, millions of computer administrators and users failed to patch their systems as Microsoft came out with a critical patch weeks before the worm actually hit the Internet (Microsoft, 2004b).

In an ironic twist to this story, the German teen who was charged and convicted of releasing the Worm in May 2005 was given a job at Securepoint in Germany (BBC, 2004). Recently the teen was convicted of computer sabotage and other crimes but since he confessed to everything, was seventeen at the time of the worm's release on May 1, 2004 and was remorseful, he was given a suspended sentence despite affecting almost twenty million computer systems worldwide and costing millions of dollars in lost data and productivity (DW-World, 2005).

Applying the latest operating system patches would have helped but newer attacks such as simple variants to all of these attacks would be hard to stop due to their uniqueness. Firewalls and Intrusion Detection Systems (IDS) can find known attacks by accessing their internal signature databases when scanning for attacks; but, a never-seen-before attack may slip through the protective devices unnoticed.

1.2 Security Issues of Today

In today's hyper-connected world of computer networks, security is a constant concern. As networks grow in size and complexity and computer services expand, vulnerabilities within the local area and wide area networks become more numerous and problematic.

Since the explosion of the Internet's usage, particularly with the introduction of the World Wide Web in the mid-1990s, more research and development dollars have been spent towards securing Internet data and transactions. However, much more needs to be done.

1.2.1 CSO Magazine Survey Results

CSO Magazine, in conjunction with the United States Secret Service and Carnegie-Mellon University's Computer Emergency Response Team (CERT), conducts an annual E-Crime Watch Survey (2004) which surveys various corporations and other assorted organizations; through the voluntary surveys, they discover trends and techniques of the criminals and how best to combat those electronic criminal acts.

Some of the survey's major findings are shown in Table 1.1:

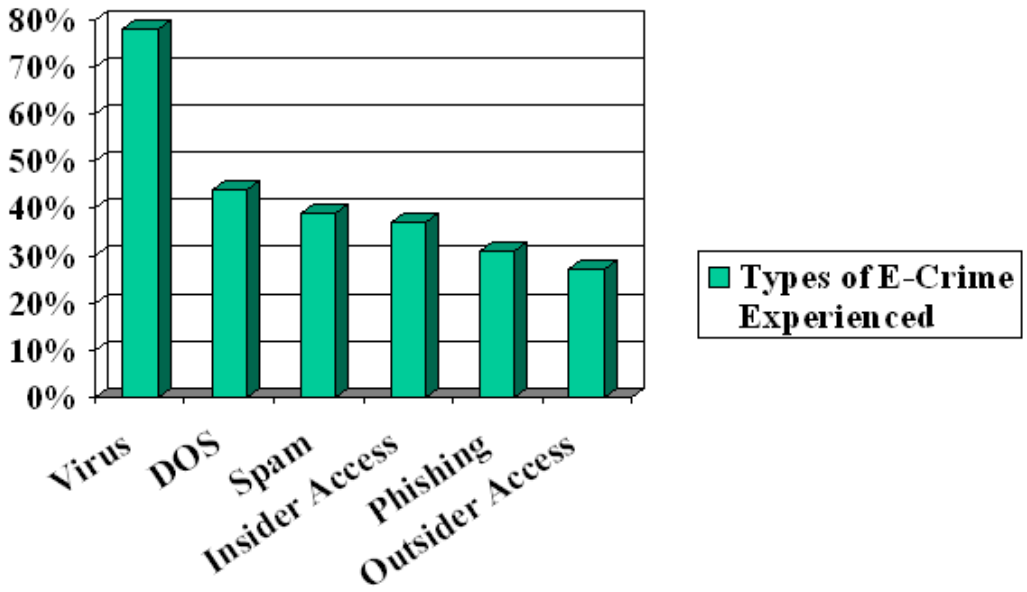
Table 1. 1. E-Crime Survey Findings 2004

Increase in e-crimes and intrusions from last year	43%
At least one e-crime or intrusion committed against them	70%
E-Crime costs in 2003	$666 million

Even more troubling than the information contained in the table above is the finding that nearly one-third of those organizations surveyed who experienced an electronic crime (e-crime) or intrusion did not know if it was an insider or an outsider who committed the particular e-crime. Well-run security organizations make it their business to know when and why breaches and attempted breaches in their security happen.

Out of the seventy percent of the surveyed organizations that reported an e-crime or intrusion, six types of crimes stand out amongst the rest – viruses; denial of service attacks (DOS); spam; unauthorized access by insiders; phishing – the use of spoofed emails or websites to fool users into divulging personal information (ISS, 2005); and, unauthorized access by outsiders. The frequency table for these six types of reported crimes is shown in Table 1.2 (CSO, 2004):

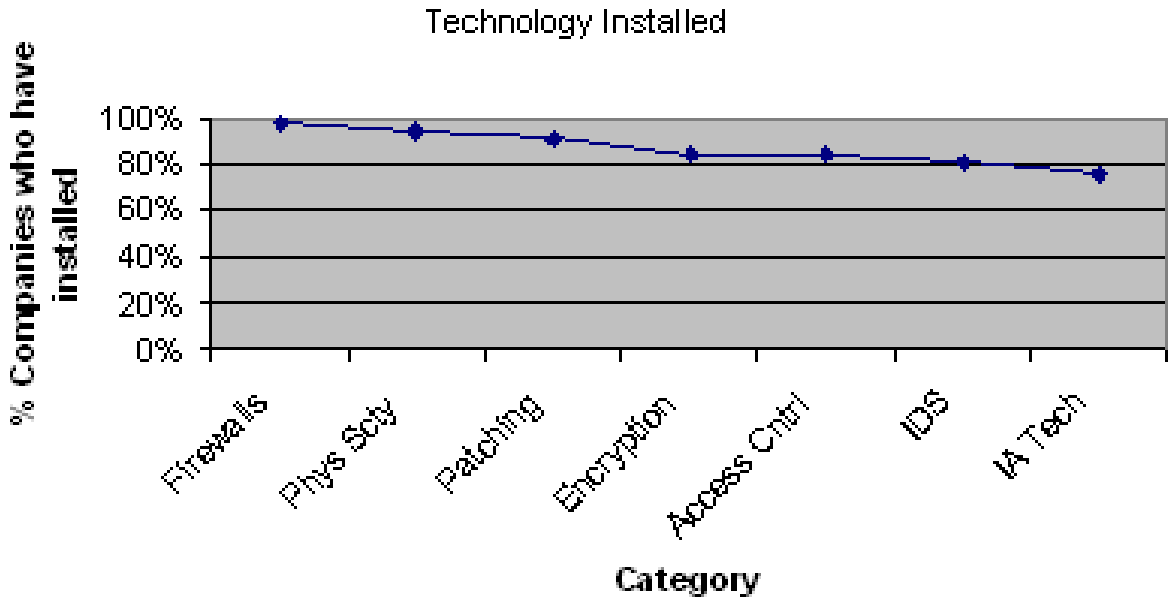
Table 1. 2. Reported Frequency of Six E-Crimes



To combat these troubling statistics, companies, government organizations, and other entities must employ a variety of security solutions. Many employ a "defense in depth" strategy that applies strict security measures across the spectrum of devices inside a company's network – routers, switches, servers, printers, desktop computers, etc. Therefore, if someone obtained root privileges on a gateway router, the effects will be mitigated since they could not automatically get access to anything else (Johnson, 2004).

The surveyed organizations reported a promising number of security solutions (hardware and software) that have been installed at their locations. Table 1.3 describes those items – firewalls; physical security devices; patches; encryption; access control; intrusion detection systems; and other information assurance (IA) technologies:

Table 1. 3. Security Solutions Installed in Surveyed Companies (CSO, 2004)



1.2.2 CERT/CC Statistics

The Computer Emergency Response Team/Coordination Center (CERT/CC) tracks these issues and publishes various statistics regarding the state of security and security reporting within the computer network community. Until 2003 the CERT/CC kept track of incidents reported to them and those numbers are shown in Figure 1.2 below (CERT/CC, 2005c).

The average annual increase of reported incidents to the CERT/CC was 104% from 1997 to 2003. This annual doubling of reported incidents is troubling and probably will not diminish in the near future.

With regards to Figure 1.2, emphasis should be placed on the word "reported." Many companies and governmental agencies are reluctant to report any security incidents for a variety of reasons. Primarily, these entities are concerned with loss of company secrets, revenues and profit (companies); loss of classified or sensitive information (government); and, loss of standing in the public arena (both companies and governmental agencies). Further, some security managers and administrators may be reluctant to report incidents for fear of losing their jobs or standing in their company (Symantec, 2005).

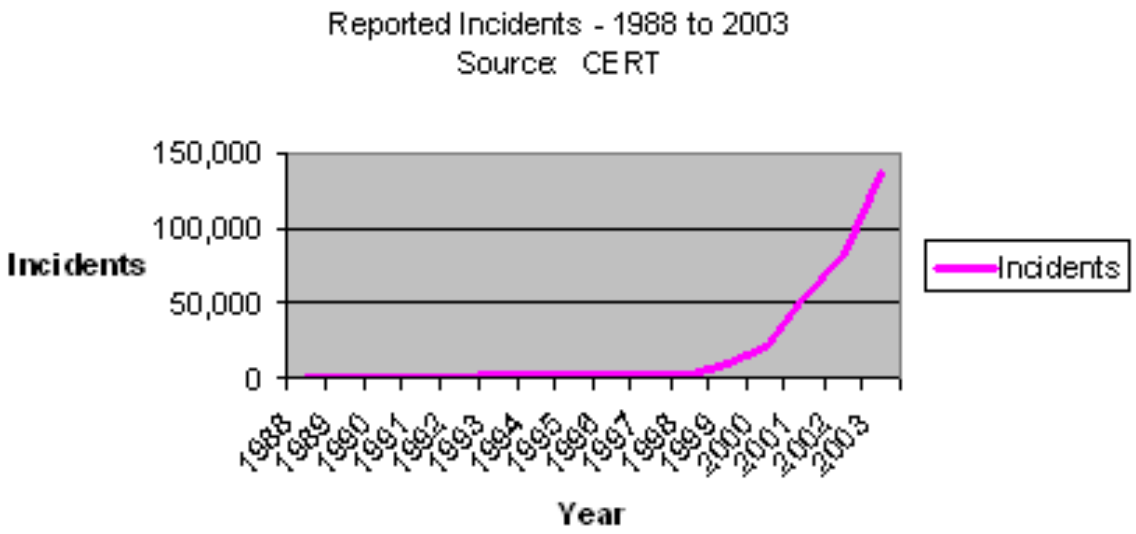


Figure 1. 2. Reported Incidents (CERT/CC)

In 2004, CERT/CC decided to stop keeping track of reported incidents partly due to the reasons stated in the previous paragraph. They also stated that proliferation of automated attack tools have caused the number of attacks to skyrocket against networks connected to the Internet; therefore, the CERT/CC felt that tracking the number of incidents would provide little helpful information regarding the assessment of the scope and impact of attacks. We disagree with this conclusion since automated attacks are still attacks against networks and tracking the number and severity of these attacks would continue to provide useful information. However, we can assume that the doubling of the number of reported attacks will continue and we must be prepared to use any means to combat this ever-increasing trend (CERT/CC, 2005c).

1.2.3 Trends and Attack Profiles

CERT/CC discussed the trends that they have seen in the changing profiles of most intrusions. Back in 1988, most of the attacks centered on the exploitation of passwords as well as the exploitation of known vulnerabilities. Today, however, the security managers have much more than simple exploitation of passwords to worry about since the attacks are usually contained in one of the following profiles (CERT/CC, 2005b):

- Exploiting passwords
- Exploiting known vulnerabilities
- Exploiting protocol flaws
- Examining source and executable files for new security flaws
- Defacing web servers
- Installing sniffer programs
- IP source address spoofing
- Denial of service attacks
- Widespread, automated scanning of the Internet
- Distributed attacks
- Building large networks of compromised computers
- Developing command and control networks to use compromised computers to launch attacks

While each of the twelve profiles raises concerns for security managers, many novel attacks combine two or more of the profiles listed above in their overall attack schemes. For example, an attacker could exploit a known vulnerability (profile #2), install a sniffer program (profile #6) that scans the internal victim network for unprotected executable files, then exploits those executable files that have new security flaws (profile #4) to create further problems for the victim network.

1.3 Thesis Statement

The main goal of this research is to detect network intrusions by analyzing the network traffic flow. The differences between our approach and traditional intrusion detection algorithms are as follows:

First, our system is a two-tier system. The first tier is a traditional signature based system that can be used to detect the known attacks. The second tier is a model-based system that can recognize currently unknown attacks through a learning algorithm developed using known attacks. A flag will be raised to catch the system administrator's attention if any traffic that is either significant deviated from normal network traffic pattern or has some similarities with existing attacks.

Our research will lay a step ahead of the traditional methods since the damages already have been done before any attack patterns can be known. The signature based system acts as a first line of defense that will filter known intrusions based upon carefully crafted heuristic rules.

The model driven rule based system will offer an added functionality where new rules can be incorporated into the database.

Second, we will analyze session based network traffic instead of analyzing packet-based traffic to increase the detection accuracy. This will consequently reduce the false alarm rate.

Our data mining system serves as the core engine to sift intrusion traffic from benign ones. We will explore support decision trees for their proven capabilities in mining useful patterns from a large corpus of data and learning through the extracted features. A dynamic analysis of the control and data flow in the traffic will provide us the insight of the program behavior that is avoided by a static examination.

CHAPTER 2: DATA MINING

Data Mining can best be described as an analytical process that sifts through a large set of data and provides the researcher with patterns and relationships contained within the data itself. From these patterns and relationships, the researcher can subsequently models to better understand and predict any future data that comes into the pre-existing data set.

For our research, these data mining patterns could provide a better understanding of any local area network (LAN). The problem with creating a model derived from these patterns is one of utility: how can we use such a model on a live network?

Further, the issues of traffic ambiguity and vantage points present problems for the accuracy of our network model (Dreger, Kreibach, Paxson, & Sommer, 2005; Ptacek & Newsham, 1998). The modeling of a LAN's traffic at selected choke points – like an Internet gateway – will give us a large amount of network packets.

2.1 Data Mining Basics

SAS Institute describes the process of data mining in an acronym – SEMMA – Sample, Explore, Modify, Model, and Assess (SAS, n.d.). This acronym accurately describes the steps we took to first sample the large amount of computer network traffic data in small pieces; explore the data in more depth by finding relationships between variables; modify and transform the data to meet our needs; model the entire data set; and, assess our model. Due to the extremely large file sizes of the network data (some were in excess of 8GB), following the SEMMA process helped us to properly dissect and analyze the data at the most-critical points.

Many other sources are available online and in book form that provide in-depth information on basic and advanced data mining principles. In (2001) Han & Kamber cite seven iterative steps needed to properly conduct knowledge discovery procedures, shown in the table below.

Table 2. 1. Knowledge Discovery Steps

Step	**Description**	**Notes**
Data Cleaning	Remove noise	Inconsistent packets removed – out of session boundaries, e.g.

Data Integration	Combine several data sources	Original data merged with attack database
Data Selection	Retrieve relevant data	Iterative process where most-needed variables are used in data set
Data Transformation	Modify data for mining	Create variables for sum, average, and other stats
Data Mining	Extract data patterns	Decision tree created to show rules for finding predictive attack patterns
Pattern Evaluation	Identify interesting patterns	Analyze attack patterns
Knowledge Presentation	Visualize mined patterns	N/A

Out of these seven steps, only knowledge presentation (visualization techniques to present mined knowledge) will not be discussed here in further detail.

2.1.1 Data Cleaning, Selection, and Transformation

These three initial steps comprise of manipulating the data to remove noise, inconsistencies, irrelevant data, and other non-essential elements. While somewhat tedious, these steps are crucial in the early stages of knowledge gathering and any missteps can lead to erroneous conclusions and models.

For the data cleaning and data selection tasks, data imputation and the use of global constants can help fill in missing values. Further, techniques for data reduction – such as discretization of values – assist the data miner by reducing the number of possible values in a continuous variable.

Data transformation techniques (smoothing, aggregation and normalization, to name a few) are critical operations that prepare the data for subsequent data mining analysis.

2.1.2 Data Mining and Pattern Evaluation

Generally, there are two types of data mining models – descriptive and predictive. While descriptive models create meaningful subgroups or demographics within data sets, predictive models create patterns that can be used to predict future values (Two Crows Corporation, 2005).

These models will be useful for our research by allowing us to create models of how the network should "look" and employing that knowledge at the network's edge to augment current signature-based IDS systems.

The most-common techniques used in data mining are shown below (Joshi, 2005):

1. *Artificial neural network*s: Models that resemble biological neural networks.

2. *Decision trees*: Tree-shaped structures that represent sets of decisions.

3. *Genetic algorithms*: Techniques that use processes such as genetic combination, mutation, and natural selection.

4. *Nearest neighbor*: Techniques that classify each record in a dataset based on similarity with nearest neighbor.

5. *Rule induction*: Techniques that extract pertinent if-then rules from data based on statistical significance.

6. *Data visualization*: Models that visually interpret complex relationships in multidimensional data.

In our research we will concentrate primarily on the decision tree technique used in data mining. These tree-shaped structures will assist our research by representing the sets of decisions required to accurately portray normal network activity and thereby showing us a more-effective way of finding potential attacks from an outside source.

2.2 Data Mining Patterns

Within the Data Mining framework, there are five sets of patterns that exist that we can strive to achieve and look at further (Han & Kamber, 2001): Characterization and Discrimination; Association; Clustering; Outliers; Evolution; and, Classification and Prediction.

2.2.1 Characterization and Discrimination

Data characterization allows us to summarize the general characteristics of a particular subset of our data. The subset is usually retrieved by a database query, such as an SQL statement. An example of data characterization would be describing – in a summary fashion – all grocery customers who spent more than $1,000 dollars per year shopping in the grocery store. The corresponding SQL statement follows:

```
SELECT      C.cid, C.amt
FROM        Customer C
GROUP BY    C.cid
WHERE       C.amt > 1000
```

Data discrimination, on the other hand, describes the process where two separate subsets of the data are compared and contrasted in order to achieve the desired results. These results can be expressed in the form of rules that are called discriminant rules. Additionally, when data characterization and data discrimination processes are combined, we achieve even more functionality.

2.2.2 Association

Association analysis allows us to discover attribute-value conditions within data subsets. These conditions that we find by using association analysis occur together quite frequently and give us association rules like (Han & Kamber, 2001):

age(X, "20..29") AND *income*(X, "20K..29K") ➔ *buys*(X, "CD player")

[support = 2%, confidence = 60%]

The example rule above indicates that 2% (support) of the customers who are between the ages of 20-29 and have an income of 20K to 29K have purchased a CD player at a particular

store. Therefore, there is a 60% probability (confidence) that a customer in this demographic will purchase a CD player.

Association Rule Mining allows us to discover interesting relationships among large data sets. These relationships are expressed in rules that can assist businesses in decision-making processes (i.e., market basket analysis). Additionally, association rules provide a good starting point for network traffic analysis and we may explore using these rules in the future.

2.2.3 Clustering

Clustering analyzes data objects without using a known class label (Han & Kamber, 2001). These labels are not present in the training data set since they are not known in the beginning; however, clustering can generate these labels if needed.

Clustering of data objects is grouped by maximizing the intraclass similarity while minimizing the interclass similarity. In other words, clusters are formed where objects within a particular cluster have a high degree of similarity with other objects in that cluster while also having a low degree of similarity with objects in other clusters.

Clustering of network traffic information has great potential for our research. Chan, Mahoney, & Arshad (2003) wrote extensively on this subject and developed a clustering anomaly detection engine (CLAD).

2.2.4 Outliers

Outlier analysis identifies objects within a data set that are abnormal and do not act normally. These objects generally do not comport to the general overall behavior of the larger data set.

While outlier analysis has limited uses, it has been used some; for example, credit card companies use outlier analysis to detect fraudulent credit card activity where a criminal may be using a stolen credit card to buy items that the legitimate user would not normally buy.

2.2.5 Evolution

Evolution analysis describes data regularities or trends whose objects' behaviors change over a certain length of time. An example of this would be predicting future stock market performance by looking at market regularities over time as well as during certain times of the year.

2.2.6 Classification and Prediction

Classification is the process (Han & Kamber, 2001) where the data miner can build a model from a training data set that will eventually describe data classes whose class label is known. This model would be used to predict a class of objects whose class label is unknown from a totally different data set.

Classification models can be shown as IF-THEN rules; decision trees; math formulae; or, neural networks. In our research, classification models that produce decision trees and IF-THEN rules will be extremely useful and beneficial for our end product.

2.3 A Decision Tree Analysis

Decision trees are readily applicable to large data sets such as the network traffic data set used for our experiments. In order to explore our data, we used a decision tree data mining technique that identified classification rules for an attack as "1" and a non-attack (normal traffic) as "0".

Safavian & Landgrebe (1991) discussed decision tree methodology and described the possible positive and negative effects of using a decision tree classifier (DTC) over other methods. They showed that DTCs are simple, flexible standards and can use a smaller number of features at each node without degrading performance in the model. An example of a general decision tree is shown in Figure 2.1 below:

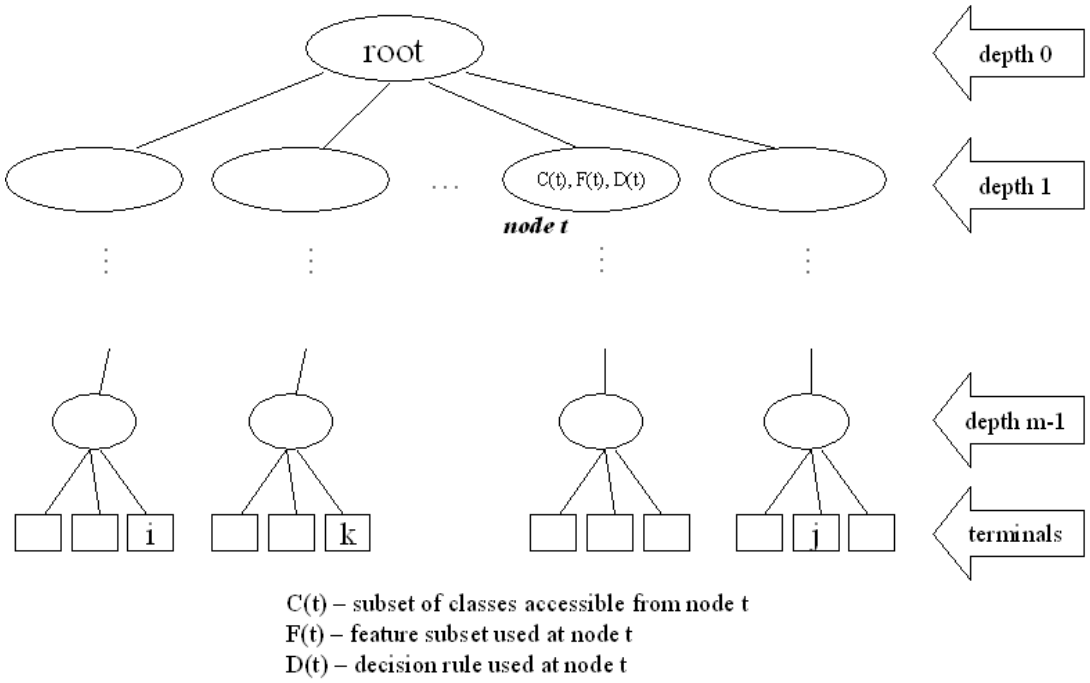


Figure 2. 1. Generic Decision Tree (Safavian & Landgrebe, 1991)

2.3.1 Decision Trees and Large Data Sets

One of the challenges seen when exploring a large data set is the issue of missing values. To deal with missing values, we do not have to impute values because these models have built-in mechanisms such as floating category approaches implemented by Enterprise Miner and the surrogate method as seen in Classification and Regression Trees (CART). For our data set, there are many missing values and imputation is a very difficult and time-consuming task.

Also, decision trees are among the most efficient methods for studying problems of this nature. For example, other methods such as logistic regression cannot efficiently handle all

variables under consideration. There are almost two dozen independent variables involved in our research; all of them are multi-leveled. This means that the logistic regression model must incorporate dozens of dummy variables and two-way interactions. Even with today's computers this process is very time consuming. On the other hand, the decision tree approach can perform this analysis very efficiently, even if the investigator considers higher order interactions.

Further, decision-trees constitute an appropriate method for studying this problem because many of the variables are ordinal in their scaling. Although we can assign numerical values to each of these categories, assignment of values to each of these categories is not unique. However, decision trees use the ordinal component of the variables to derive a solution analysis.

The rules found in decision trees have an IF-THEN structure that is readily comprehendible. Also, the quality of these rules can be assessed with percentages of accurate classification or odds-ratios that can be easily understood. The final procedure analysis produces tree-like rule structures that predict outcomes. Customarily, researchers test the quality of the rules on a data set independently of the one on which they were developed.

2.3.2 The Model Building Procedure

For this study, we used the classification and regression tree methods (Breiman, Friedman, Olshen, & Stone, 1984) executed with SAS Enterprise Miner (SAS, n.d.).

Tree-based methods are recursive; bisecting data into disjoint subgroups called terminate nodes or leaves. CART analysis incorporates three stages: data splitting, pruning methods, and homogeneous assessment.

Data splitting into two (binary) subsets at each stage is the first feature of the model. After splitting, the data in the subsets become more and more homogeneous. The tree continues to split the data until the numbers in each subset are either very small (i.e. say the number of observations is less than 100), or all observations in a subset belong to one category (e.g. all observations in a subset have the same rating). Typically, this "Growing the Tree" stage results in far too many terminate nodes for the model to be useful. The extreme case occurs when the number of terminate nodes equals the number of observations. Such models are uninformative because they produce very few rules with explanatory power.

The CART procedure solves this problem by using pruning methods that reduce the dimensionality of the system. In practice, CART splits the data into two pieces—the first data set grows the tree, and the second prunes the tree, thereby validating the model. In practice, CART methods, through the pruning process, reduce the original tree into a nested set of sub trees. Although homogeneousness based on the training data set can always be improved, it is not necessarily true in the validation set. Typically, because the validation data are not used in the growing process, they give an honest estimate of the best tree size.

The final stage of the analysis involves assessing homogeneousness in growing and pruning the tree. One way to accomplish this is to compute the misclassification rates. For example, with a rule that produces a .95 probability that an attack is present has an associated error of 5.0%.

An important feature of this approach involves a performance assessment of the finally developed model—accomplished by the application of rules that have been developed and validated initially to an independently collected data set.

2.3.3 Consequences of Using Decision Trees

Although decision tree techniques are effective for analyzing data sets such as those considered in our research, there are consequences of this procedure. First, decision trees only use ranks to handle both ordinal variables and interval variables. At times, this might lead to lost distribution information about some variables, although the use of ranks does not create any information lose in the analysis.

Second, decision tree algorithms will combine categories if a given category variable has excessive partitions. For example, most decision trees algorithms will combine ZIP Codes into several groups before applying split search. This feature, however, was not problematic in this research because we used no categorical variable that had more than 10 categories.

Third, the most serious weakness of the decision trees is that the results can be unstable because the technique is data driven and small variations can lead to substantially different final solutions. Techniques such as boosting (Schapire, 1996) and bagging (Breiman, 1996) provide some remedies to the instability of tree methodology. However, these treatments will make the

interpretation of the rules much less intuitive countermanding the fact that ease of interpretation is one of the most important advantages of decision tree modeling.

2.4 Data Mining In Our Research

For our research, the question remains: how do we use data mining tools in our research and which technique from the list in section 2.1 should we use?

Initially, the decision-tree method will be used to gather as much data as possible. Several unique models will be compared and contrasted against our score data set to see which model provides the best anomaly detector and why.

A comparison of classification methods would become necessary and we will use the following criteria to compare the methods. These methods are ranked in order of usefulness to our research, from most useful to least useful:

1. *Predictive Accuracy* – ability to correctly model the class label of new data. In our research, how accurate is the session-based IDS model when it predicts whether or not incoming network traffic is anomalous?

2. *Interpretability* – level of understanding and insight provided by the model. In our research, this is critical since the model's insight will help us to better protect our networks through different protective schema.

3. *Scalability* – ability to make a model efficiently, given a large data set. In our research, this is key since our data sets are extremely large (2 GB per day at least).

4. *Speed* – computational speed costs involved while (a) generating the model and (b) using the model. In our case, a fairly strong computer system is needed to generate the model monthly as well as using the model online to detect attacks.

5. *Robustness* – ability to make correct predictions given noisy data or data with missing values. In our case, no missing values exist and not much data is considered noisy.

So, a combination of the decision tree and rule induction techniques will be used in our research. The decision tree-based program segments the extremely large data sets (6 to 8 gigabytes each) well and creates a model that can be manually extracted and inserted into our anomaly-based program. Although this process is labor-intensive, it needs to be accomplished approximately once per month for a particular network. This will account for any changes in the structure of the defending network that needs to be accounted for in the anomaly-based model.

CHAPTER 3: INTRUSION DETECTION SYSTEMS

To mitigate the ever-present computer security issues, network and system security managers have an array of protective measures – both hardware- and software-based measures – which provide the basic tools for protecting the internal networks from outside influence. Secure and safe transactions can be provided through the use of firewalls, Intrusion Detection Systems (IDSs), Intrusion Protections Systems (IPSs), encryption, authentication, and other hardware and software solutions.

One of the main tools used by today's security managers is the Intrusion Detection System (IDS). The IDS provides a layer of security by "watching over" the network at key points and alerting the network manager of any malevolent traffic that enters or leaves the network. IDSs come in two flavors – network-based IDS and host-based IDS. While the latter concentrates on securing individual servers and clients, the former works at a much larger scale to actually secure the gateway from the internal network to the outside Internet.

3.1 Network Intrusion Detection Systems

Network Intrusion Detection Systems usually have multiple sensors (the eyes and ears of the program) that remotely watch over the network and monitor all connections. These sensors

report back to the main program and the suspicious events are logged and acted upon if necessary.

While much time and effort have been given towards improving firewalls, encryption, passwords, and securing operating systems, research into effective Intrusion Detection Systems has been uneven and concentrating mostly on misuse detection instead of anomaly detection.

3.2 Misuse Detection and Anomaly Detection

Many IDS variants exist which allow security managers and engineers to identify attack network packets primarily through the use of signature detection; i.e., the IDS recognizes attack packets due to their well-known "fingerprints" or signatures as those packets cross the network's gateway threshold. On the other hand, anomaly-based ID systems determine what is normal traffic within a network and reports abnormal traffic behavior (Axelsson, 2000).

Misuse detectors use traditional Intrusion Detection System technology where known attacks are catalogued in their internal databases and immediately recognized by the IDS when its known signature pattern is seen. The problem with misuse detection is that attack signatures can be altered easily and thereby "sneak in under the radar" of the misuse detection systems. On the other hand, *anomaly detectors* look at the network from a more macro point of view and concentrate on how a "normal" network should look like during the workday. The downside

with most anomaly detection techniques is the high number of false positive alerts that are usually seen in these systems.

3.3 Why IDS Programs are Needed

Sundaram (1996) described the need for Intrusion Detection Systems (IDS) along with the two major types of IDS devices – *Anomaly Detectors* and *Misuse Detectors.* Anomaly detectors determine what is "normal" in a particular system or network and if an abnormal or anomalous event occurs, then it is flagged for further inspection. Misuse detectors are similar to modern anti-virus systems where an attack's signature is stored in the IDS for later use if needed. However, both types have their advantages and disadvantages, but the future seems to lie within the anomaly detectors' area of concern.

Sundaram continued his introduction by describing the problems with IDS – false negatives and false positives. False negatives are the situation when intrusive activities that are not anomalous result in a flagged event. False positives occur when anomalous activities that are not intrusive are flagged as intrusive.

3.4 Anomaly Detectors of Today

Liston (2005) and Phung (2000) describe the limitations of current signature-based ID systems. Both authors recognize that using data mining tools to support anomaly detection projects are critical for the overall advancement of ID systems.

Two sub-types of anomaly detection are statistical approaches and predictive pattern generation. The former method uses a detector that constantly generates the variance of the present profile from the original one while the latter method or sub-type tries to predict future events based on the events that have already occurred.

3.4.1 CLAD and LERAD

Chan, Mahoney, and Arshad (2003) discussed how signature detection ("misuse detection" in the previous reference) could not detect novel attacks. Therefore, they contended that it is better to focus on anomaly detection techniques in the IDS field to model normal behavior in order to detect these novel attacks. The authors described two methods for learning anomaly detectors: rule learning (LERAD) and clustering (CLAD).

Both methods – CLAD and LERAD – present interesting advantages and possibilities for future research. One example of a learned rule's semantics under LERAD is the probability

("P") where *P(DestPort = 80 | SrcIP = 128.1.1.3, DestIP = 128.4.5.6).* Here, we see that their models are predictive in nature so that false alarms are less likely to occur.

Finally, the authors described a proposed data-mining algorithm that has five main steps that aid in their research:

1. Generate candidate rules from the random data sample
2. Evaluate those candidate rules
3. Select a minimal set of candidate rules that covers the random data sample
4. Train the selected candidate rules on the training data set
5. Prune the rules that cause false alarms on the validation data set

3.4.2 EFSA

Sekar, Gupta, Frullo, Shanbhag, Tiwari, Yang, et. al, (2002) discussed anomaly detection based intrusion detection techniques and described a new approach to fix the main problem anomaly detection – high rates of false alarms.

They developed extended finite state automata (EFSA) to simulate the network's normal operation and determined that their state machine fairly accurately portrayed the training data sets. They used automated learning tools to map out the network behavioral patterns to become transitions in the state diagram.

3.4.3 Distributed Data Mining and Database Mining

Bala, Baik, Hadjarian, Gogia, and Manthome (2002) looked at distributed data mining approaches to build global profiles by applying tree induction procedures. Eventually, they generated classification rules that led them to build partial and final trees to show a pattern-based profile of the host network.

Hu and Panda (2004) proposed a data mining process and associated algorithm to detect malicious transactions in a database system. After two initialization steps, their algorithm generated sequential patterns from the transactional data based on the patterns recognized in the data. They conducted several experiments with their algorithm and found that they could effectively discover malicious activity within the database provided that certain data dependencies existed.

3.4.4 IDEVAL

Mahoney and Chan (2003) discussed the DARPA/MIT Lincoln Laboratory data set off-line evaluation (IDEVAL) (Lippmann, Haines, Fried, Korba, & Das, 2000), which covered many of the then-popular attacks against Windows NT and Unix hosts in 1999. Mahoney and Chan analyzed the popular network traffic benchmark and suggested that the presence of simulation artifacts can lead to "overoptimistic evaluation of network anomaly detection systems."

Simulation artifacts contain three conditions or attributes: well behaved in a simulation environment without attacks; poorly behaved in a real environment without attacks; and, poorly behaved in traffic containing simulated attacks. While the authors contend that these artifacts can cause misevaluations of anomaly detection systems, we contend that we can mitigate this problem by dynamically modeling networks continuously and tailoring our model to fit the network that requires an anomaly detection system.

Lee, Stolfo, and Mok (1999) proposed using association rules and frequent episodes found in network auditing data as the basis for guiding the data gathering and feature selection process. They also proposed using meta-learning in order to make the IDS models more adaptive and more effective.

The authors discussed the different types of detection systems available, as well as their advantages and disadvantages. Then, they described the use of association rules and frequent episodes programs. They further described data mining concepts of support and confidence, which allows the data miner to determine the viability of a given rule that is used at a particular point in time with a given level of confidence. They used these concepts against a variety of datasets, with excellent results.

3.4.5 Statistical Analysis on Network Packets

Bykova, Ostermann, & Tjaden (2001) described a statistical-based methodology in the analysis of network packet characteristics. Like the other papers that talk about the use of

statistical or data mining procedures to find anomalous traffic, the authors show that their research can analyze attack packets and create rules based around the characteristics of the packets that come before and after the intentionally-malformed packets.

While useful, this type of research does not directly affect our network anomaly research. While we consider the attack packets and their surrounding packets, we do not create models that simulate specific attack signature patterns. However, we do create models for our research that simulate normal network traffic and any anomalous traffic will get separated and inspected later in the process.

CHAPTER 4: SESSION-BASED INTRUSION DETECTION

The complexities in the field of modeling and simulations can be enormous. Floyd and Paxson (1997; 2001) described the inherent difficulties of simulating the Internet and networks in general. The heterogeneity of the Internet and networks makes it a difficult task to model those entities; however, if we properly choose the vantage point where our information is extracted, we can show that the model for that particular point is advantageous for our research.

While the choice of a network vantage point for data collection is crucial, the issue of modeling specifications also comes into play. How a model should be created and choosing the correct data from the network packets are key points in our research. We concentrated on using the data obtained from network packet headers but only use a certain number of fields. We then converted the packets into sessions for improved results.

The variable selection was conducted initially through trial and error but eventually evolved into intuitive matching of certain features with other like features. For example, we chose to concatenate the source IP address with the source port number. We concatenated the same variables on the destination side of things. Further, we combined temporal variables like date, hour, minutes, seconds, and milliseconds and converted those variables to have a nominal value since their numeric values mean little from one day to the next. Finally, we added housekeeping variables like numPKTS that showed the number of packets in the selected session we were examining.

4.1 Packet-Based IDS System

Mahoney and Chan (2001) described their Packet Header Anomaly Detection (PHAD) system that discovers the value ranges of packet header fields for data link (layer 2), network (layer 3), and transport (layer 4) layers of the traditional seven-layer Open System Interconnection (OSI) networking model as described in ISO 7498 (ACM, 1994). Figure 4.1 shows the OSI reference model. Further, PHAD does not look at application (layer 7) protocols like DNS, HTTP, or SMTP, so its scope is limited as are most anomaly-based engines.

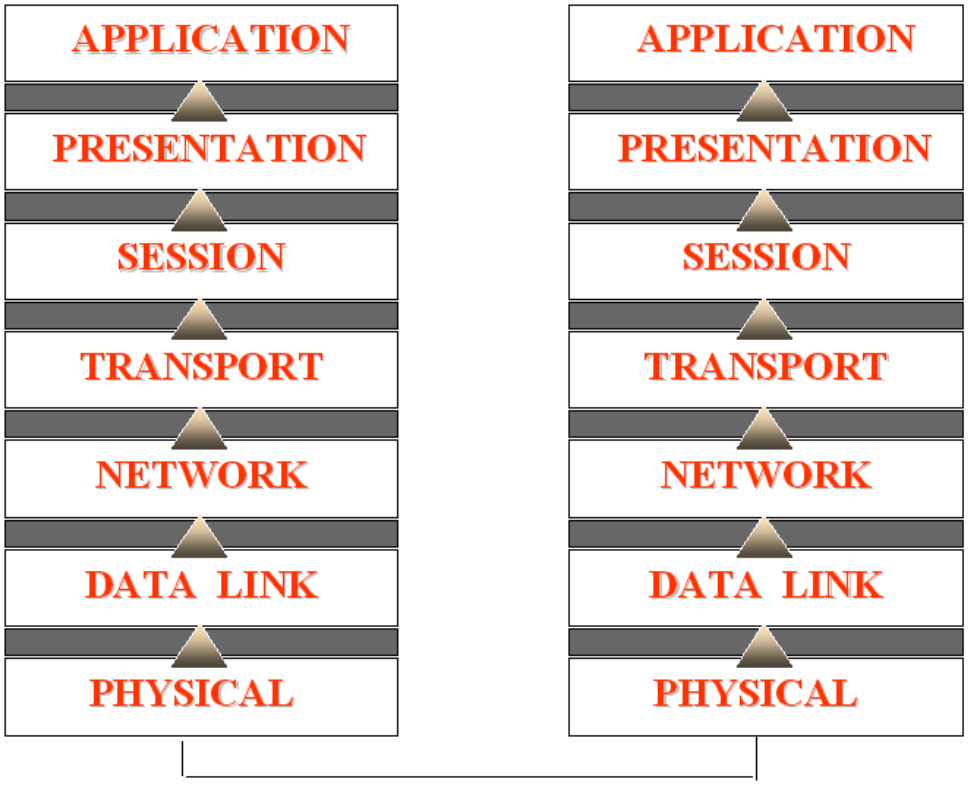


Figure 4. 1. OSI Reference Model

While we start off by using network packets, our research expands on programs like the PHAD system and collapses the packets into sessions. These sessions – while not pure and true TCP data sessions – represent a leap forward in anomaly-based detection since they provide a new way at looking at network traffic and modeling of the network traffic at key vantage points.

4.2 Session-Based IDS Tools

For this research, we used a 2.4 GHz Pentium IV computer (single processor) running Windows XP with 528MB of RAM with an 80GB internal hard drive and a 160GB external hard drive. Also, several software packages were used to expedite processing of the data and for the creation of the model itself: Editcap, Snort, Cap2Txt2, and SAS® Enterprise Miner™. In the near future, we will work to export most of these model-creation processes to a Java- or C-based program; however, we will conduct that exportation process as follow-on work to this dissertation.

4.2.1 Tcpdump and Editcap

Tcpdump is a program that is used to capture packets in the background while the network interface card (NIC) is in promiscuous mode instead of its normal broadcast mode

(Hegde, 2005). When saved, the captured packets are stored in a binary format termed, "tcpdump format."

For MS Windows systems, Snort and Ethereal are more widely used packet capture programs than Tcpdump; however, the underlying libpcap packet capture library still supports those programs as well as Tcpdump. Both Snort and Ethereal (Combs, n.d.) are open source programs as well.

Editcap is a command line program that works with Ethereal (Networkdictionary.com, 2005); editcap takes a saved capture file as its input and then writes some or all of the packets into another capture file (Orebaugh, 2004). We used this program to take the original Tcpdump-formatted file and slice the file up into smaller files, each containing 100 thousand packets. This allowed us to process the data faster as our Java program could not handle resulting text files with much more than 100 thousand packets.

4.2.2 Snort IDS Freeware

Snort is a lightweight, Network Intrusion Detection freeware software package (Roesch, 1999). Although it is open source and readily available for anyone to use, it is also a highly regarded IDS system in use today.

From its inception in 1998, Snort has evolved from a niche program to a solid NID system that is capable enough to handle real-time traffic and the analysis of protocols (Carlson, 2005).

Snort is customizable and the open-source nature of Snort allows users to mold the program to fit their individual needs. Further, registered Snort users can modify and update the online signature rules database to highlight a newly found attack signature or simply to match their needs (Vossen, 2005).

For this research we used Snort in two distinct ways. First, we used Snort to simply take the output from the editcap routine and pipe it to a text file. Second, we used Snort to analyze the entire October 2004 training data set and tag the packets that were attacks or probable attacks.

A key point in our research is that we assume that Snort's current rule set – which is currently six months younger than the training data set – will find the vast majority of attacks in the training and validation data sets. After all, any new attacks (i.e., new attacks as of October 2004) will be identified and tagged by the Snort user community. So, modeling the October 2004 data in June 2005 should present very few errors and misfires.

4.2.3 Cap2Txt2 Java Program

To finish off the initial processing of the data, I created a Java program – cap2txt2 – that processes the text-based network packet data and formats the data further for exportation into the SAS® Enterprise Miner™ program. This program goes through each packet, line by line, and subsequently formats the data properly. Malformed packets will be ignored and discarded.

In the future, this Java program will form the basis for the overall SAND program (see section 4.5). While the core of the modeling activity will still occur within SAS® Enterprise Miner™, most of the formatting, editing, and other preprocessing tasks will be done online within the SAND program itself. We envision that editcap, tcpdump, and most of the SAS Code modules will become obsolete, as SAND will take up most of those tasks. This project should occur within the next year or so, outside the scope of this dissertation.

4.2.4 SAS® 9.1 Enterprise Miner ™ 4.3

In version 4.3 of SAS® Enterprise Miner™, we used the SEMMA process as described in chapter 2 to properly analyze our flood of network data. Several steps in this process – most

notably, explore, modify, and assess – were repeated several times to get better results as well as to verify the results themselves.

SAS® Enterprise Miner™ is a solid data mining software suite that allowed us to create a model in the form of a decision tree and subsequently extract the vital rules inside that decision tree. These rules were then exported into a SAS code program, which we ran in order to create a new data set that told us which sessions were probable attacks and which were not probable attacks. The overall SAS model is shown in the figure below.

We chose to use SAS for our research for several reasons – ease of use; accuracy of decision-tree models; and, flexibility with large data sets on Windows-based platforms. Other similar tools like Classification and Regression Trees (CART) could have been used but SAS met our needs best. Future research should include other data mining software products in order to analyze and compare the results generated. Like any other software program, SAS is simply a tool to help validate our theories.

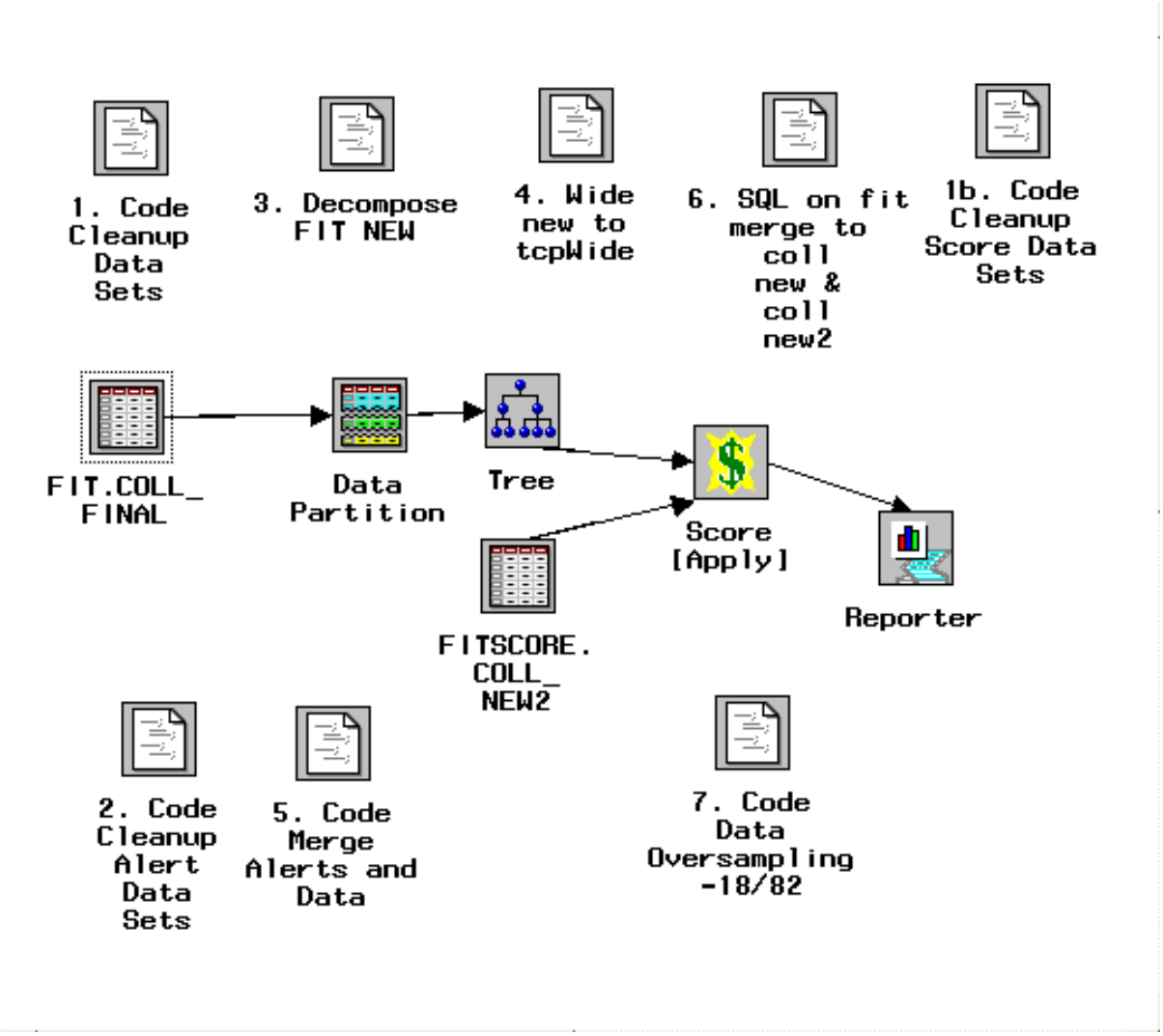


Figure 4. 2. SAS Enterprise Miner Model of our Data

Most of the modules shown above simply act as placeholders for the code we created to handle the data better and more efficiently. The modules without any arrows are SAS code modules used for this purpose. Although SAS is touted as a Windows-supported program, most of the power of the data-mining environment in SAS is derived from the user creating code in the SAS modules.

4.3 Research Methodology

We started our research towards developing a robust network modeler through the use of data-mining techniques using the well-known MIT Lincoln Lab data sets from their work in 1999. The MIT researchers created a synthetic network environment test bed to create their data sets (Haines, Rossey, & Lippmann, 2001).

These contrived data sets – which contained both attack-filled and attack-free data – allowed us to look at network traffic stored in tcpdump format where the attack types, duration, and times were known and published.

When looking at the data sets, the individual rows represent network packets while the columns represent the associated variables found in that particular network packet. The figure below (Allen, 2003) shows a sample of the types of TCP traffic that occurred in the synthetic test bed.

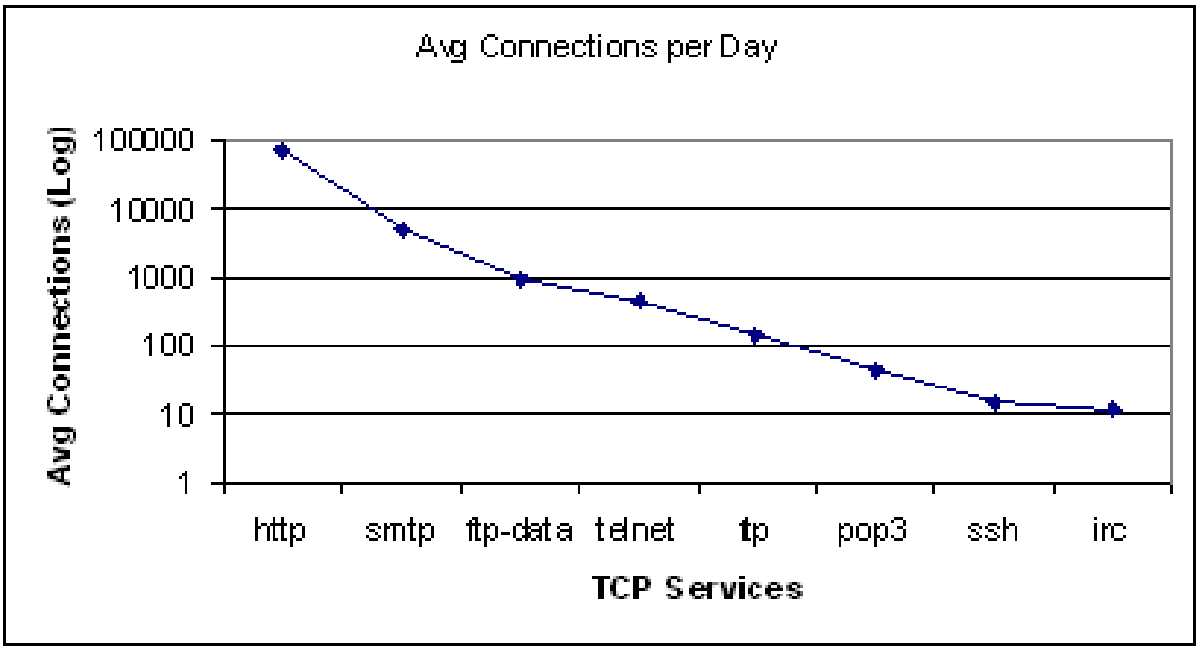


Figure 4. 3. Average TCP Connections Per Day (non-attack weeks)

We started off looking primarily at the Transmission Control Protocol/Internet Protocol (TCP/IP) header information. Based on earlier work in the 1970s and 1980 (Postel, 1977; Postel 1980), these two protocols form the basis for most Internet traffic today. While the original Internet Protocol specified a security option (Postel, 1981a), this option determined the labeling and routing of a message and provided little in the way of current security standards and practices.

4.3.1 Data Preparation

Figure 4.4 shows the TCP segment information that we used in our research with the first 20 bytes being the header information of the TCP segment, as specified in RFC 793 (Postel, 1981b). Using data mining techniques on the individual packets' data set, we were able to estimate an importance value for each of the fields prior to our final modeling attempts with the Lincoln Lab data set. Further, we decided that additional information like IP addresses, which are important for security managers to know and use for later inspection, are not important for our research since we do not need to worry about where the packets start or finish, just what is contained in them.

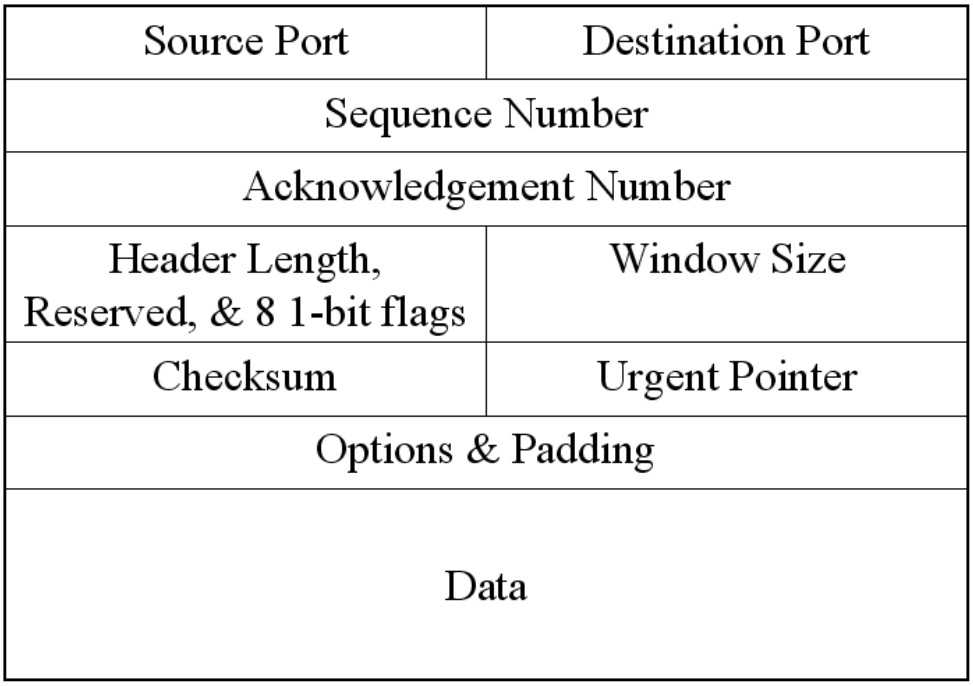


Figure 4. 4. TCP Segment (Postel, 1981b; Leon-Garcia & Widjaja, 2004)

4.3.1.1 Target Variable

From there, we studied the data sets' patterns and modeled the traffic patterns around a generated target variable, "TGT". This variable is a Boolean value (true/false) where a true value is returned for TGT if the packet is a known attack packet as designated by the studies done in the Lincoln Labs' tests. We used this variable as a predictor target variable for setting the stage when we applied a new, separate data set to the model and subsequently scored that new data set against our model.

Other TCP header information like Sequence Number offsets – which are important factors when considering what hackers look for when attacking networks (Northcutt, Cooper, Fearnow, & Frederick, 2001) – were not found to be important factors in our research.

4.3.1.2 Shaping and Collapsing the Data

Eventually, we shaped the data set to become "wider" and "shorter"; i.e., we added more variables (columns) and reduced the number of observations (rows). To achieve this we collapsed the data set through the use of Structure Query Language (SQL) constructs to mesh the associated packets into sessions, thereby reducing the size and complexity of the data while maintaining the essential components of the data set.

Since the responding (i.e., victim) computer was involved in an attack session, its sending "TGT" value was reset to "1" to describe that it's receiving an attack from an outside source. The original test data set went from 12.5 million observations (rows) to 61,000 observations. A visual diagram of this action is shown in Figure 4.5 below:

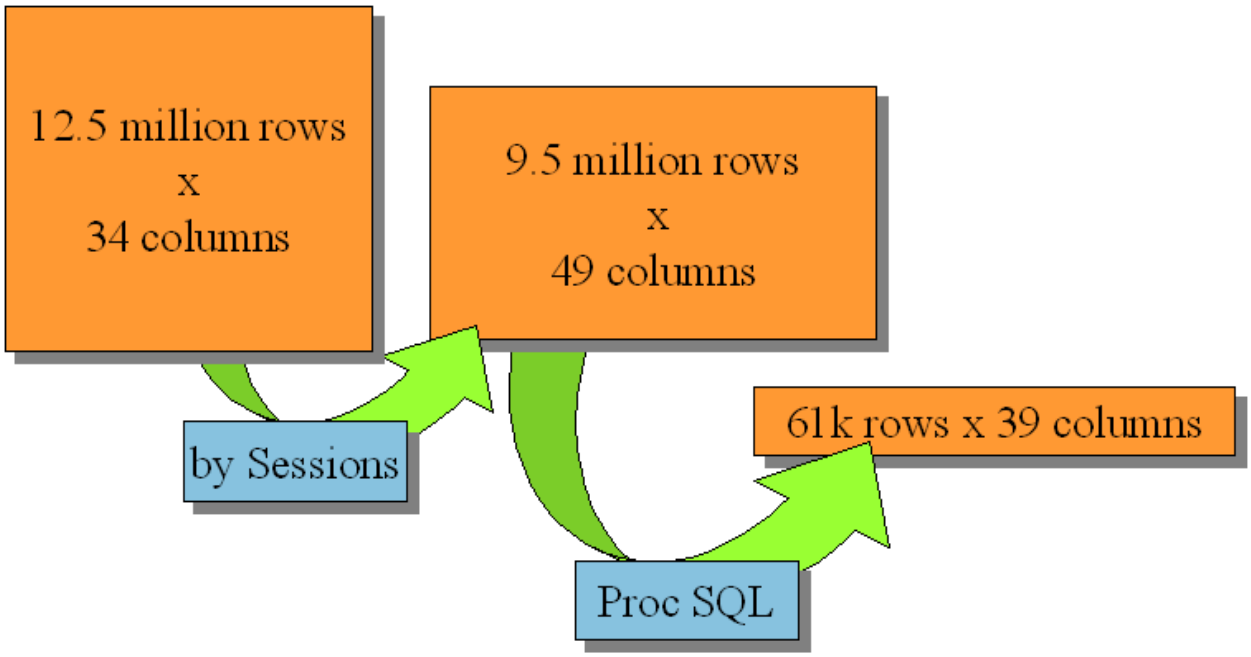


Figure 4. 5. Data Collapsing

4.3.2 Classification Tree Modeling

We used a predictive, classification tree method for our model. This technique allowed us to determine which categorical variables where pertinent to our ability to accurately predict the attack sessions in our network.

The advantage with using this method over traditional pattern-recognition methods is that the classification tree is an intuitive tool and relatively easy to interpret. When classifying network sessions from its predictive categorical variables, the resulting hierarchical tree produced two major types of sessions – probable attacks and non-probable attacks (see section 4.5).

4.3.3 Data Modeling and Assessment

We strived to accomplish our data-mining goals through the use of supervised learning techniques on the binary target variable, "TGT". We also used the concept of over-sampling, which allowed us to highlight the infrequent and unusual attacks that occurred during the 1999 IDEVAL. From the initial data set we over-sampled the labeled attack sessions and created a new data set that contained a mix of 43.6% attack sessions and 56.4% non-attack sessions.

4.3.3.1 Variable Deletion

From this mixture we further deleted a few extraneous variables – date/time, source IP address, and destination IP address variables.

All other header variables were kept, including the source and destination port numbers, in order to allow the system to model the important facets of the network better.

4.3.3.2 Model Creation

We then partitioned the data using stratified random sampling and allocated 67% of the observations (network sessions) to the training data set and 33% to the validation data set. The TGT variable was then specified to form subsets of the original data to improve the classification precision of our model.

We created a decision tree using the Chi-Square splitting criteria and modeled the data from the IDEVAL. Our decision tree model determined that the number of leaves required to maximize the "profit" of the model was around four or five leaves.

Our model also shows that after five leaves of depth, the tree maximizes its profit and further calculations past that point would prove extraneous for our studies. Further, the statistics from our model show a low misclassification rate and average squared error, which indicate stability. Our model's fit statistics are shown below:

Table 4. 1. Model's Fit Statistics

Fit Statistic	Training	Validation
Average Squared Error	0.05	0.05
Sum of Squared Errors	4194.35	2121.60
Root Average Squared Error	0.23	0.23
Maximum Absolute Error	0.91	0.91
Divisor for ASE	81816.00	40298.00
Total Degrees of Freedom	40908.00	.
Misclassification Rate	0.06	0.06
Number of Estimated Weights	2.00	.
Sum of Frequencies	40908.00	20149.00
Sum Case Weights * Frequencies	81816.00	40298.00

4.4 Assessment of Risk

Once employed in a network security framework, this system of anomaly detection would prove invaluable to the network security managers and administrators. However, one potential problem area is the introduction of new technologies into a network, since traffic from these technologies would be tagged as anomalous from our anomaly-based filter.

Further, as Stoneburner, Goguen, & Feringa (2002) show in their Risk Management Guide for IT Systems, the IT security practitioners must use the risk management process continuously to identify and resolve news risks when any of the following events occur:

1. Expansion in network connectivity
2. Changes to the existing infrastructure
3. Changes to the existing organizational policies
4. Introduction of new technologies

The authors later described the nine risk assessment steps required, shown below. After the first step is completed, steps 2, 3, 4, and 6 can be accomplished in parallel.

1. System Characterization
2. Threat Identification
3. Vulnerability Identification
4. Control Analysis
5. Likelihood Determination
6. Impact Analysis
7. Risk Determination
8. Control Recommendations
9. Results Documentation

The risk assessment methodology flowchart for these nine steps is shown in Figures 4.6 and 4.8.

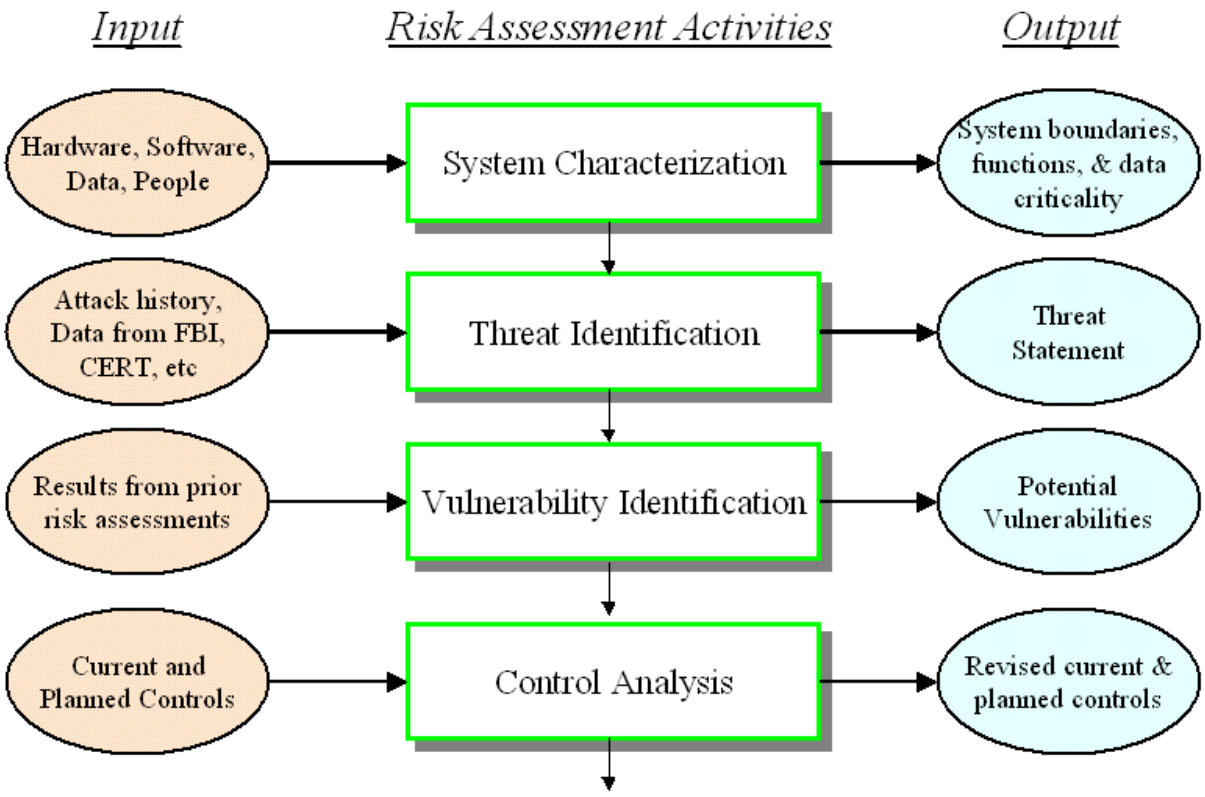


Figure 4. 6. Steps 1-4 of Risk Assessment (Stoneburner, Goguen, & Feringa, 2002)

The first four steps shown above entail getting the problem identified properly. That is, the security personnel must determine if and when there was an authorized access of data, for example, and what exactly is the threat that the victim network encountered.

From there, they describe the remaining five steps of the risk assessment model (shown below) and show that the security personnel must continue to assess and re-assess the impact of the break-in and steps that need to be taken immediately.

All nine steps are qualitative in nature. However, steps 5, 6, and 7 have a quantitative aspect to them. In these steps, the security personnel are asked to evaluate the level motivation of the threat against the security controls that are in place (step 5 – "likelihood") as well as to evaluate the impact of a particular vulnerability being exploited (step 6 – "impact"). From there an impact vs. likelihood matrix is created (step 7) in order to show the level of risk that is being assessed. Figure 4.7 shows an example of this matrix.

Threat Likelihood	Impact		
	Low (10)	Medium (50)	High (100)
High (1.0)	Low (10x1.0 = 10)	Medium (50x1.0 = 50)	High (100x1.0 = 100)
Medium (0.5)	Low (10x0.5 = 5)	Medium (50x0.5 = 25)	Medium (100x0.5 = 50)
Low (0.1)	Low (10x0.1 = 1)	Low (50x0.1 = 5)	Low (100x0.1 = 10)
Risk Scale:	High: >50 to 100	Medium: >10 to 50	Low: 1 to 10

Figure 4. 7. Level of Risk Assessed Matrix (Stoneburner, Goguen, & Feringa, 2002)

They finish with long-term outputs – Recommended Controls and Risk Assessment Reports – that detail the steps that need to be taken to ensure the network is stronger, healthier, and more resistant to these types of attacks in the future.

The inputs received from steps 1 to 7 provide valuable data and insight for the security managers to take action upon. The resulting report and recommended controls are passed along to senior managers who undertake a risk mitigation strategy that is designed to limit risk until it gets to manageable and acceptable levels. Getting their risk down to or close to zero is probably impossible. At best, it's probably nearly impossible and too cost prohibitive. So, the senior managers use the risk mitigation steps to reduce the risk into "residual risk" that is acceptable.

4.4.1 New Technologies' Impact

We saw this problem of new technologies causing false IDS alarms early in our research when we analyzed the traffic (packet by packet) and noticed that DHCP, SSH, and secure FTP packets in the UCF data set caused alarms in our model since our training data set (IDEVAL) contained no instances of any of those protocols.

To counter this threat, constant updates are required for each network's model as the network managers add new protocols and equipment to the network. Additionally, our research shows that each network should have its own unique model to filter out and identify anomalous behavior properly.

4.4.2 Low Level Attacks' Effect

An additional risk comes from the possibility of low-level attacks that evade detection from some IDS systems. The packets from these attacks appear so infrequently that they are regarded as noise in the network; however, our anomaly filter may be able to detect these attacks and future research will look into this risk to see if this threat is manageable or not.

4.4.3 Two Levels

A further extension of our research would be the inclusion of signature-based detection working in conjunction with our anomaly-based detection system. Future work would include a proposed system where network sessions from the outside wide area network (WAN) environment pass through two sets of filters before reaching the internal local area network (LAN).

This setup would potentially solve many of the problems anticipated with the anomaly-based system, but one new problem would arise with this two-tiered framework: denial of service (DOS) attacks. A malicious outside user could determine that flooding these network devices with millions of attack packets would cause a successful DOS attack by forcing the filters to examine each packet and session individually, thereby allowing the system to eventually stall and go off-line to correct the problems.

The Tribal Flood Network (TFN) Distributed Denial of Service (DDOS) attacks in 1999 were an example of these types of attacks. TFN is made up of several programs that can flood and overwhelm networks with Internet Control Message Protocol (ICMP), TCP (SYN floods), and User Datagram Protocol (UDP) traffic. It can also accomplish Smurf style attacks and provide a root shell bound to a TCP port of an unwitting victim computer system (Dittrich, 1999). Smurf attacks are ICMP floods

To mitigate this risk, packet filters must be used to quickly block traffic from the offending IP subnet once the DOS attack commences.

4.5 IDEVAL Modeling Results

Once we created and refined our network model using the IDEVAL data, we collected hours of traffic from the UCF network. Then, we looked at the TCP/IP header information and ran that information against our newly-created model.

Web servers, mail servers, routers, desktops, and other devices contributed data to our study. These devices used a variety of operating systems, to include Windows XP, Windows 2000, Linux, and Solaris. Each day of our UCF data set contained 20-30 million packets, averaging 300 packets captured per second.

4.5.1 IDEVAL and UCF Data Differences

The UCF data differed from the IDEVAL in the following ways:

1. DHCP is used at UCF and the IP addresses are dynamically allocated.

2. Telnet and FTP connections are almost non-existent on the UCF networks. Most of these services are handled through Secure Shell (SSH) and secure FTP protocols.

3. In addition to using port 80 for HTTP requests, the UCF web servers use port 443 for the incoming HTTPS requests to handle course scheduling and other personal information for the students that require secure transactions.

4. All of the UCF workstation and operating systems utilized operating systems that were later versions than the IDEVAL. Although this is an obvious point, it needs to be reiterated due to the large number of viruses, worms, and Trojan horses that still traverse our networks almost unabated due to improperly updated operating systems and networks.

4.5.2 UCF Data Set Score Results

We scored the UCF data with our network model and found that approximately 32.9% of the sessions were identified as having a probability of 1.0000 of being an attack session, given the data from our original model. A "score" of 1.0000 indicates that the classification model has used its predictive data-mining techniques to determine that this session is has a high probability of containing attack packet(s).

Although a probability of 1.0000 normally represents absolute certainty, here it represents a starting point where we can further examine the data and see how well our model performed. Conversely, a session that scored a probability lower than 1.0000 does not necessarily mean that session is a "good" session; however, the likelihood is high that session is innocuous and poses no threat to our campus networks.

The vast majority of the sessions captured had a low or non-existent probability of being an attack session. Our studies show that more than sixty-six percent of the sessions captured have an attack probability of 0.0129.

Most of the anomalous sessions that were tagged in the high-probability zone were web-based port 80 traffic; further, many suspicious sessions also contained high-numbered ports from the source address that were not commonly used. Finally, for the majority of this research, we limit our discussion to TCP packets only. Later research will model more than just TCP packet traffic. Figure 4.6 shows the number of packets per suspicious session.

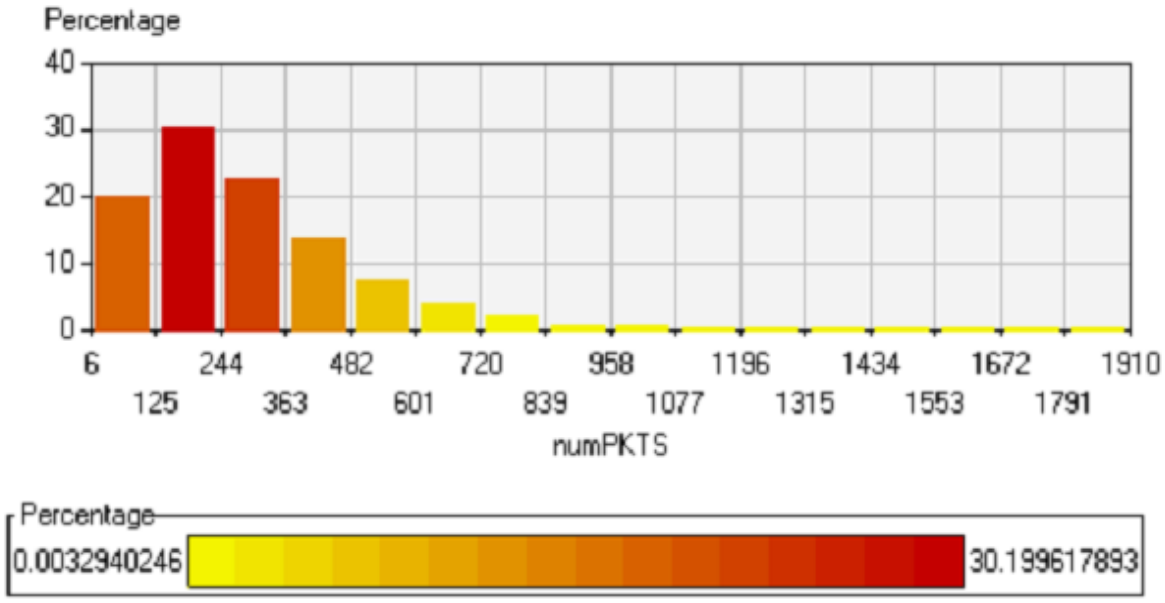


Figure 4. 9. Number of Packets / Suspect Session

Similarly, most of the TCP packets could be removed from the attack list; for example, a substantial number probable attack TCP packets were NFS packets – a protocol not used in the 1999 IDEVAL. Further, little-used services like di-traceware were found in the UCF data that ran against our model and caused those packets to be tagged as highly probable attacks.

Overall, out of the approximately 30,000 sessions that were labeled as highly probable attack sessions by our model, a small percentage of those sessions were deemed as probable attacks from the use of a freeware signature-based IDS system called Snort. Further, our model did label approximately eighty-five percent of the probable attacks properly.

4.6 SAND Modeling

While our initial network model using the IDEVAL data set proved promising, a more-recent and more-relevant set of data was needed to continue to improve on the modeling methodology. A newer training and validation model that more-closely resembled the scored data set would be needed to produce better predictive results; so, we developed this model using the Session-based Anomaly Notification Detector (SAND). The figure below shows the SAND Modeling Algorithm:

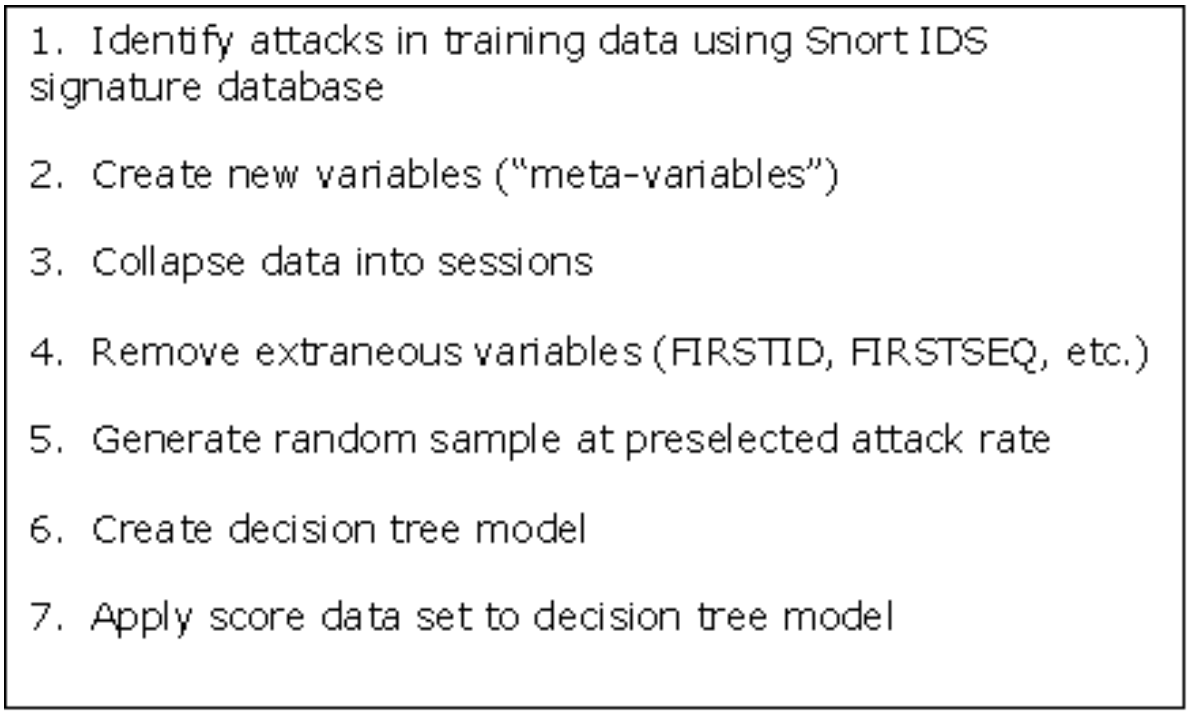


Figure 4. 10. SAND Modeling Algorithm

4.6.1 SAND Process

SAND begins the process by loop modeling the network traffic at critical points in the network. For our research, we chose one critical point to retrieve the network data. All data retrieved are stored in tcpdump format.

As the collected data was six months old, we were able to analyze the entire data set using the Snort IDS tool. From this initial analysis, Snort found 1,386 alerts in the data set.

These alerts were tagged in the data set as attacks and the corresponding TGT variable was given a value of 1.

After the attack packets are identified and labeled, we created a SAS module that merges the attack packet data set with less than one percent of the original data set that contains thirty-two percent of the identified attacks. This methodology would thereby create a solid data set to properly model the network correctly and identify anomalous behavior coming into or out of the network at the gateway.

Table 4. 2. Training and Score Sets

Data Set	**Size in Packets and Sessions**	**# of Attacks ID'd**	**Notes**
Entire Set	***109 million (100%)***	***1,365 attacks (100%)***	***Only 2.5 hours***
FIT Training	1.5 million (1.38%) = 1,740 sessions	463 attacks (33.92%)	Non-contiguous
FIT Score 1	633 thousand (0.58%) = 9,257 sessions	2 attacks (0.15%)	1 min, 20 sec

FIT Score 2	465 thousand (0.42%) = 5,658 sessions	6 attacks (0.44%)	1 min, 0 sec

From the above table, we can see that the entire data set contained more than 109 million packets. Out of those packets, we used 1.38% of the data (1.5 million packets) to create our training and validation data set. This data set formed the nucleus of our network's model at that particular juncture. We used a non-contiguous set of packets to form this model – overall there were five sections of the entire data set that were merged to form this non-contiguous model.

Within each of the five sections were several hundred thousand packets that are contiguous with respect to time. We chose this methodology in order to maximize the number of attacks (33.92%) relative to the number of packets in the data set itself. This is the first instance of oversampling the data in order to bring out the data set characteristics that exist on this network. The figure below shows this oversampling of non-contiguous data.

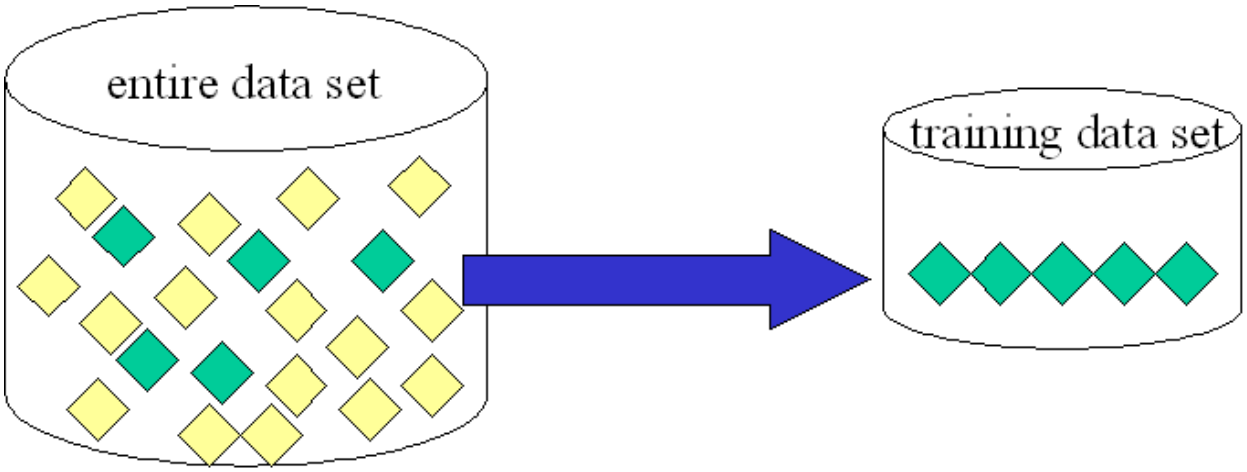


Figure 4. 11. Training Data Set

4.6.2 Score Data Set 1

As seen in table 4.2 above, score data set 1 has over 633 thousand packets but only two actual attacks to find. Within the first score data set, we have a total of two attacks and two types of attacks:

- SCAN FIN – Attack "3176" – This is a scan/probe where a TCP packet is detected where only the FIN flag is set to 1; all other flags are set to 0. Most Windows machines will respond with an ACK-RST regardless of whether or not the port is open (SourceFire, 2005e).
- Bare Byte Encoding – Attack "3202" – This is an attack against IIS servers that uses non-ASCII characters as valid values in decoding UTF-8 values. Bare byte encoding allows the user to emulate an IIS server (SourceFire, 2005c).

In the figures below, we show the effectiveness of the four models in finding these two attacks shown above. The four models are as follows:

1. 9% attack sessions ("9/91")
2. 18% attack sessions ("18/82")
3. 36% attack sessions ("36/64")
4. 40% attack sessions ("40/60")

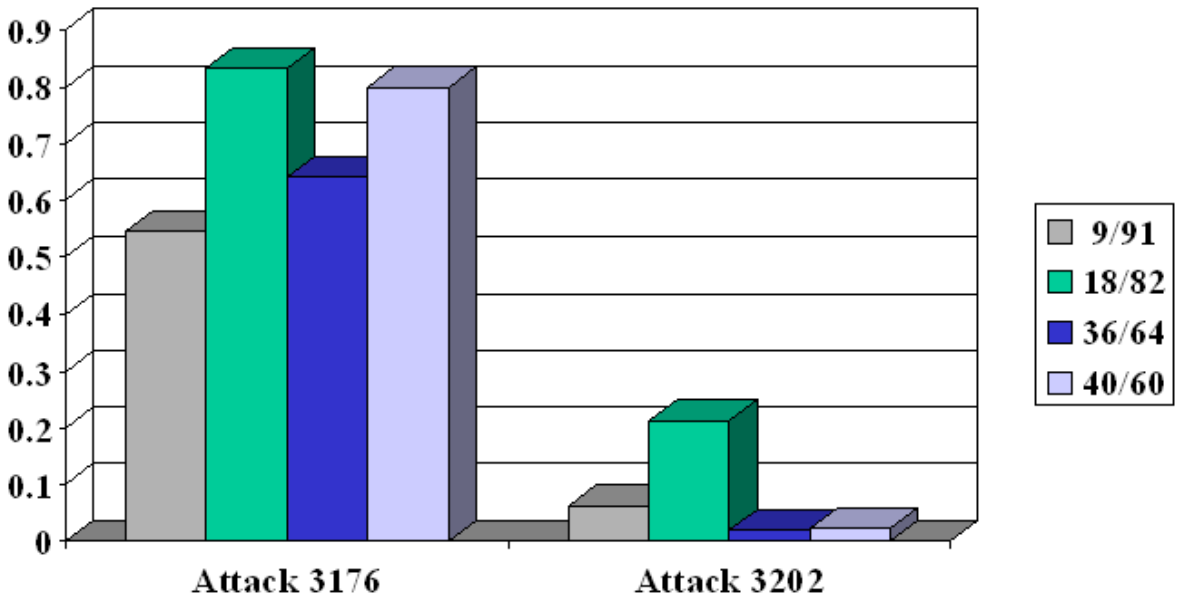


Figure 4. 12. Unnormalized Results - Score Set 1

From Figure 4.11, we see that models 18/82 and 40/60 are particularly effective finding the SCAN FIN attack; in fact, all four models met the minimum threshold of a 0.5 predictive value when finding the SCAN FIN attack in this scored data set. However, none of the models found the second attack (Bad Byte) but 18/82 performed the best, percentage-wise, in both attack cases. Figure 4.12 shows the normalized results with no major change for any of the models.

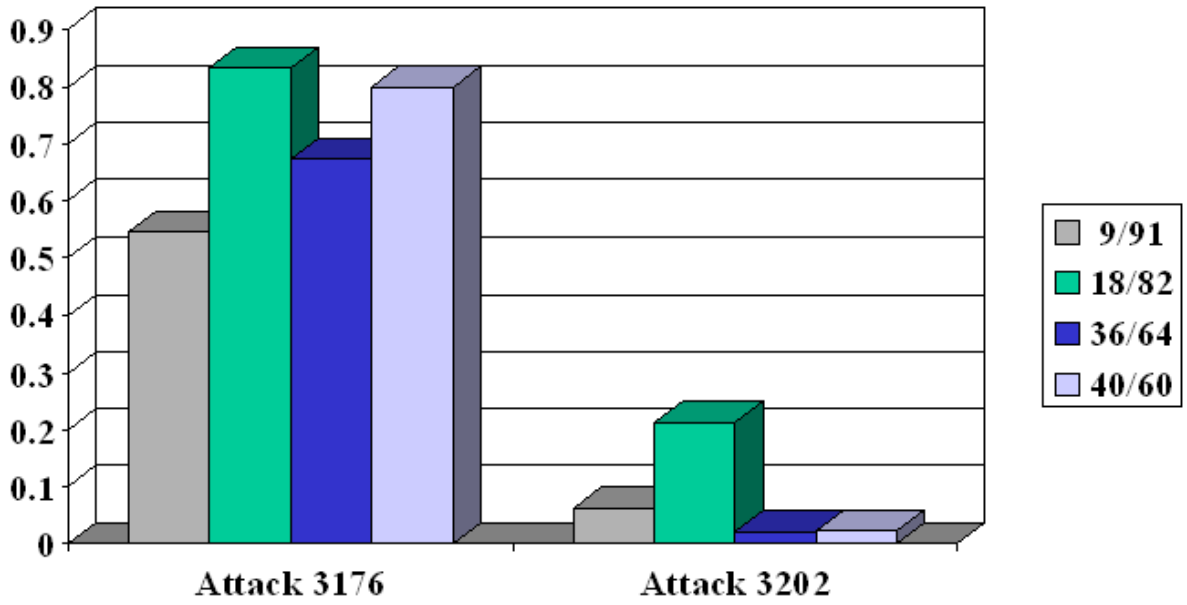


Figure 4. 13. Normalized Results - Score Set 1

A corresponding chart (Figure 4.13) shows the false alarm rate in these four models. Both 9/91 and 18/82 models performed the best and it showed in the results seen in the previous figures.

Newman, Snyder, & Thayer (2002) discussed false alarm rates in various commercial and noncommercial IDSs and their impact on the IDS's effectiveness. However, their discussion centered on signature-based IDS systems and not anomaly-based systems.

Although the false-alarm rates seem to be rather high, this is not unusual for anomaly-detection schemes (Allen, 2003; Axelsson, 1999; Tan, Steinbach, & Kumar, 2005). The key is to reduce the false alarm rate in any production system. For our research, the path to false alarm rate reduction is accomplished by adding more information to our training and validation data sets and making our decision tree leafier. These subjects will be discussed in more detail in chapter 5.

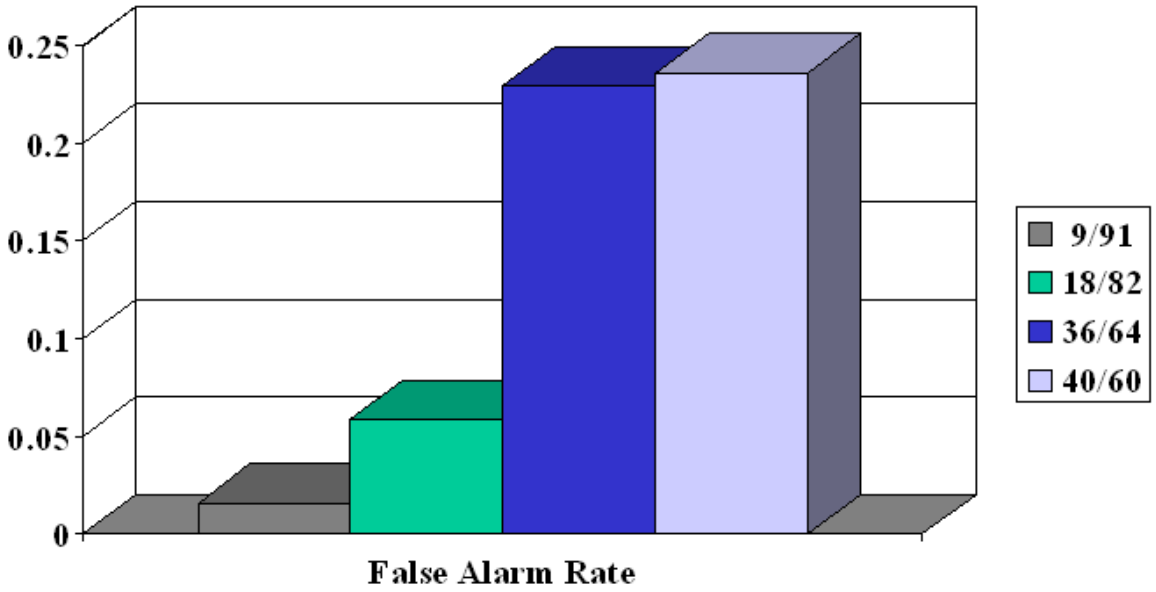


Figure 4. 14. False Alarm Rate - Score Set 1

4.6.3 Score Data Set 2

As seen in table 4.2 above, score data set 2 has almost 500 thousand packets but only a few attacks to find. Within the second score data set, we have a total of six attacks and two types of attacks:

- Bare Byte Encoding – Attacks 1 & 2 – This is an attack against IIS servers that uses non-ASCII characters as valid values in decoding UTF-8 values. Bare byte encoding allows the user to emulate an IIS server (SourceFire, 2005c).
- Bad traffic – loopback traffic – Attacks 3-6 – Under normal circumstances traffic to the localhost (127.0.0.0/8) should only be seen on the loopback interface (lo0). This attack is an indicator of unauthorized network use, reconnaissance activity or system compromise (SourceFire, 2005a).

In the two figures below, we show the effectiveness of our twelve models as we oversample the training and validation set at different levels. The x-axis shows the different models used, from 4% attacks to 81% attacks in the database. The higher the data goes on the y-axis, the better the model performed in predicting the attacks. Each different color in the bar codings represent a different attack (six in total).

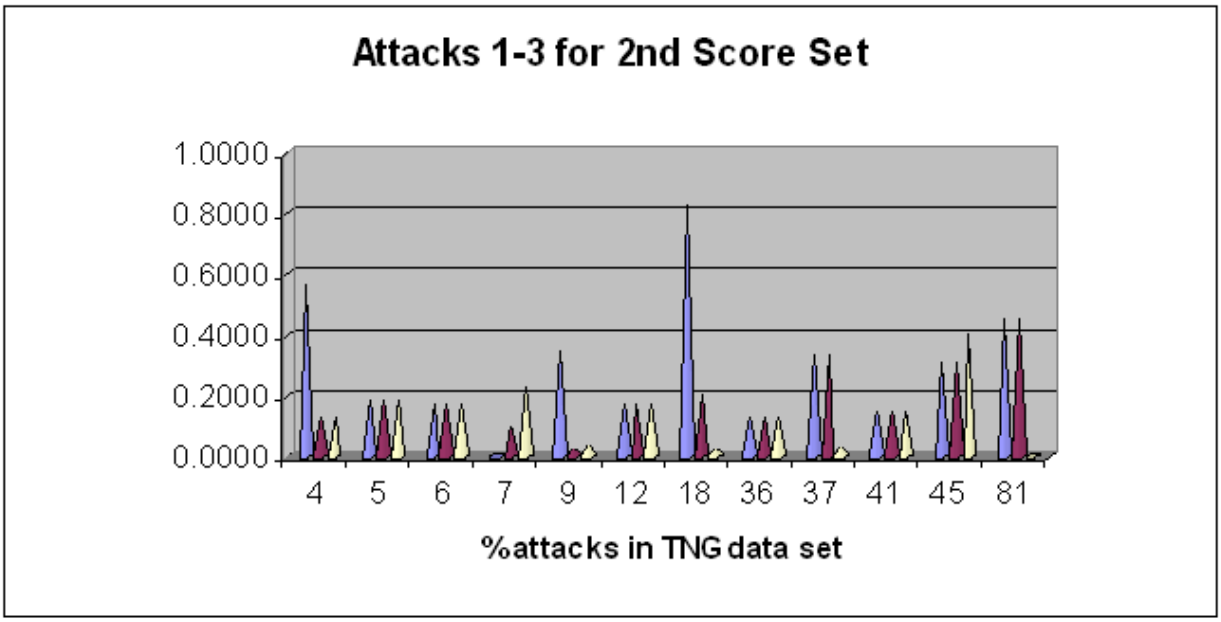


Figure 4. 15. Attacks 1, 2, and 3 Results

Although the first set of results (attacks 1-3) was promising, the second set proved more effective when predicting the attacks. Within either set, however, the oversampled model that contained 18% attacks proved to be the best predictor for finding the six attacks.

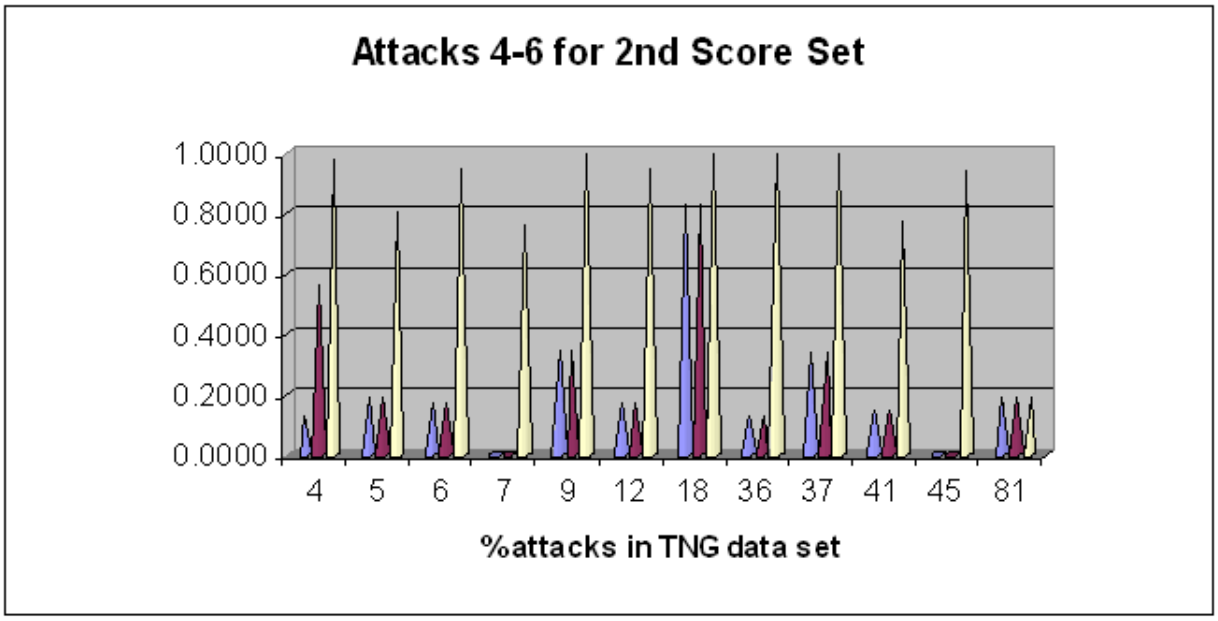


Figure 4. 16. Attacks 4, 5, and 6 Results

CHAPTER 5: ADVANCED TECHNIQUES ON SESSION-BASED IDS

The analysis shown in the previous chapter on the FIT data set provided a glimpse into how a session-based network IDS model would work. However, the modeling of the training and validation data sets provided a snapshot of how to obtain a good model only; indeed, the existence of a "good" model was sometimes due to pure chance. We now need to look more closely at the types of attacks that are present and re-model the network to account for those types of attacks as well as to include advanced data mining techniques like bootstrapping that can improve our results and lessen the effects of random chance.

5.1 The Seven Attack Types

As we analyzed the training and validation data sets, we used the Snort IDS application to sift through the data and identify all of the attacks present in the six-month-old data sets. From this action we determined that seven attacks occurred.

These attacks were tagged and catalogued during the process of session building as mentioned in the previous chapter. In contrast to the previous chapter, we removed the "TGT" variable from the target list and inserted an "alert-id" variable to be the new target. This "alert-id" variable – unlike the previous target variable that simply determined the presence of an attack

(yes/no Boolean variable) – shows us what type of attack was present and provides insight into the model's effectiveness in predicting future attacks.

Below we describe the seven attacks and how accurate our models were in predicting the instance of future attacks in the separate score data sets.

5.1.1 Bad Traffic – Loopback Traffic

SourceFire describes (2005a) this attack as a reconnaissance probe where an attacked uses a spoofed address (in this case, the localhost IP address for the owner's computer, which is always 127.0.0.1) and tries to gather more information on the potential victim's computer and associated network.

This usually indicates a probe since it is seen coming from the network and not from the computer's loopback interface, lo0. It is somewhat possible for an attacker to spoof this address and use it to attack a target computer on another interface. But, since this event can only indicate malicious intent, it is a very reliable signature event if it occurs. Advanced data mining techniques like bootstrapping (see section 5.2) will assist in detecting these types of attacks since there is a possibility that an error and not a malicious attempt had caused this event to occur.

5.1.2 Bad Traffic – TCP Port 0 Traffic

SourceFire describes (2005b) this event as another possible reconnaissance probe where an attacker tries to verify the existence of a host at a particular address or address range. TCP traffic over port 0 is not normal traffic. Although it is possible that an incorrectly configured network device could trigger this event filter, the more-likely scenario is an attacker trying to see if a computer exists by sending traffic over port 0 on the TCP side.

5.1.3 Bare Byte Unicode Encoding

This event is generated when the Snort preprocessor program, http_inspect, detects network traffic that may be malicious in nature. SourceFire describes (2005c) the bare byte Unicode encoding attack as a possible attempt to evade detection by an IDS system watching the network's gateway traffic. The attack is detected when a web browser sends non-ASCII values to a Microsoft IIS web server without encoding those values with a % sign. This non-standard format request is regarded as a probable attack against that web server.

5.1.4 FTP Exploit Stat

SourceFire describes (2005d) this attack as a malicious attempt to execute the STAT command – in conjunction with the file globbing (wildcard) "*" character – against Microsoft IIS FTP servers. When this denial of service (DOS) command is received by an unprotected IIS FTP server, the server crashes. Microsoft has released several security patches to address this issue but hackers still use this type of attack against FTP servers to see if the patch has been applied.

5.1.5 Scan FIN

SourceFire describes (2005e) the Scan FIN attack as a crafted packet that has only its FIN flag set. All other flags are set to zero. Most Windows machines will respond with a packet that has the ACK and RST flags set regardless of whether or not the corresponding port is open. Most Unix systems, however, will respond with an ACK and RST packet only if the port is closed.

The Scan FIN is a probing attack and may indicate that an attacker is trying to figure out which ports are open and which ports are closed on a victim machine. The principle of locality comes into play when looking at the data surrounding this probing attack since you have a definite action (FIN packet) followed by a definite response (ACK-RST, usually) that will be picked up during our modeling process. When crafting the training data set, keeping the "nearby" sessions will prove crucial to our detecting this attack in the score data sets.

5.1.6 SNMP Request TCP

The SNMP Request TCP attack is an event that is generated when an attacker is trying to gather information by directing a packet to TCP port 161 (SourceFire, 2005f). If the SNMP daemon responds to the trap request, the attacker can then proceed to launch subsequent attacks against the SNMP server program.

Using a packet-filtering firewall that only allows well-known servers to send this connection request to port 161 can block these types of probes. However, using a bootstrapping data mining technique on these sessions in order to improve our model's effectiveness can bolster our training and validation data sets. Bootstrapping is discussed in further detail in section 5.2.

5.1.7 Whisker Tab Splice Attack

SourceFire describes (2005g) the Whisker tab splice attack as a way for attackers to evade IDS detection. Since some web servers interpret tabs as they would spaces, then an attacker can send a malicious request to the web server using tabs in place of spaces. This could evade the IDS since most IDS variants will probably only be looking for the attacks with spaces in the command lines, not tabs.

5.2 Bootstrapping

When using data mining techniques to find pertinent results in our network traffic data, we tried several variations on the data to attain the most productive models. Even small modifications to the training data set – like deleting one row – caused great differences in the effectiveness and reliability of our model.

After our initial set of experiments, we concentrated on using our SAND methodology on models that exist in the 4/96 to 9/91 (attack/non-attack) range. We noted in our earlier experiments that models that fall outside of this range either have low rates of true positives (i.e., those models with less than 4% attacks in the data set) or have high misclassification rates and correspondingly high false positive rates (i.e., those models with more than 9% attacks in the data set).

5.2.1 Scalable Data Mining and Bootstrapping

In their white paper on this subject, Small and Edelstein (2005) described how scalable data mining concepts like bootstrapping produced positive results in their research. They further describe the benefits of improvements in computing power, storage capacity, and computing cost – all of which helped them to analyze large amounts of data.

The authors state how a computationally-intense process like bootstrapping can assist when validating and model: if you have 1,000 rows of data, you build a sample of 1,000 rows by choosing one row at a time, at random, and extracting that row by placing it into the sample data

set. You repeat this process over and over again and keep choosing one row until you have 1,000 rows in your sample data set. Some rows may be repeated; some may not appear at all. Then, you split the sample set into two groups – one for training and the other for validation. You repeat this model building cycle many times. The mean error rate is the mean over all of the repetitions (bootstraps). As you build more models, you create a very low error rate.

5.2.2 SAND Bootstrapping

When modifying our original algorithm as stated in chapter 4, we decided to create a looping algorithm that maintained the advantages of the original algorithm but added the characteristics of a bootstrapping mechanism. This allowed us to generate dozens of random samples and add to our model, layer by layer, until a more robust model was created.

Figure 5.1 describes the process of the SAND Bootstrapping Algorithm. Steps 1 through 4 and 6 through 8 were maintained from the original algorithm, but the latter three steps are now encased in a loop within the algorithm. Step 5 was added and it simply creates a null set (initializes the variable) for the variable *newModel.*

Once the model is created and we've scored the data set against the decision tree model (step 8), we added two new steps that allow us to determine if the scoring of the data set was successful; if so, we add the training data set (*ok_subModel*) to *newModel*. If not, we discard the training set. Then we loop back to step 6 until our desired bootstrapped model is achieved.

1. Identify attacks in training data using Snort IDS signature database

2. Create new variables ("meta-variables")

3. Collapse data into sessions

4. Remove extraneous variables (FIRSTID, FIRSTSEQ, etc.)

5. *newModel* = { }

SAND BOOTSTRAPPING LOOP:

6. Generate random sample at pre-selected attack rate

7. Create decision tree model

8. Apply score data set to decision tree model

9. *ok_subModel* = high scoring sub-model not yet in *newModel*

10. Add *ok_subModel* to *newModel*

11. Go to step 6

Figure 5. 1. SAND Bootstrapping Algorithm

5.2.3 SAND Bootstrapping Results – Part I

As we built each of the twenty layers for the bootstrapped model, we found that successive runs of the SAND Bootstrapping Algorithm produced better results. In Figure 5.2, we note the correlations between the number of attacks found per run versus the number of false alarms and normalized misclassification rates per same run. The maximum number of attacks per run is three, shown in blue in Figures 5.2 and 5.3. While high misclassification rates (to get "normalized" misclassification rates, we simply multiplied the rates by 100) usually produced low false alarms, we also noted the fact that when we were successful in finding all three attacks, we had a correspondingly high false alarm rate. This produced unacceptable levels of false alarms, so our research concentrates, in later stages, upon specific attacks to form better models.

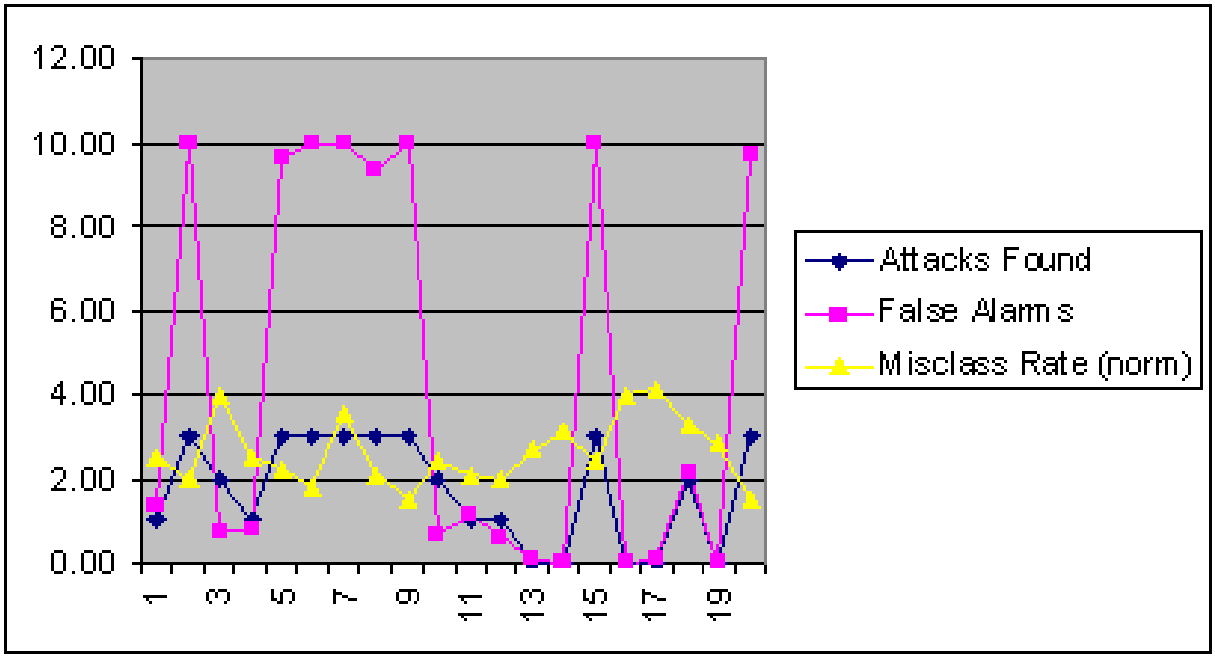


Figure 5. 2. Attacks Found vs False Alarms and Misclass Rate, part I

Figure 5.3 shows the same data without the false alarm information. The attacks found versus normalized misclassification rate graph shows that a low misclassification rate produces a good probability of finding the attack. This bit of information was key for our further research since it enabled us to build our twenty models per bootstrap with a relatively low number of attacks (4 to 9 percent range) versus non-attack sessions. Sessions with a higher number of attacks usually produced unnormalized misclassification rates of 0.15 or higher for both the training and validation data sets.

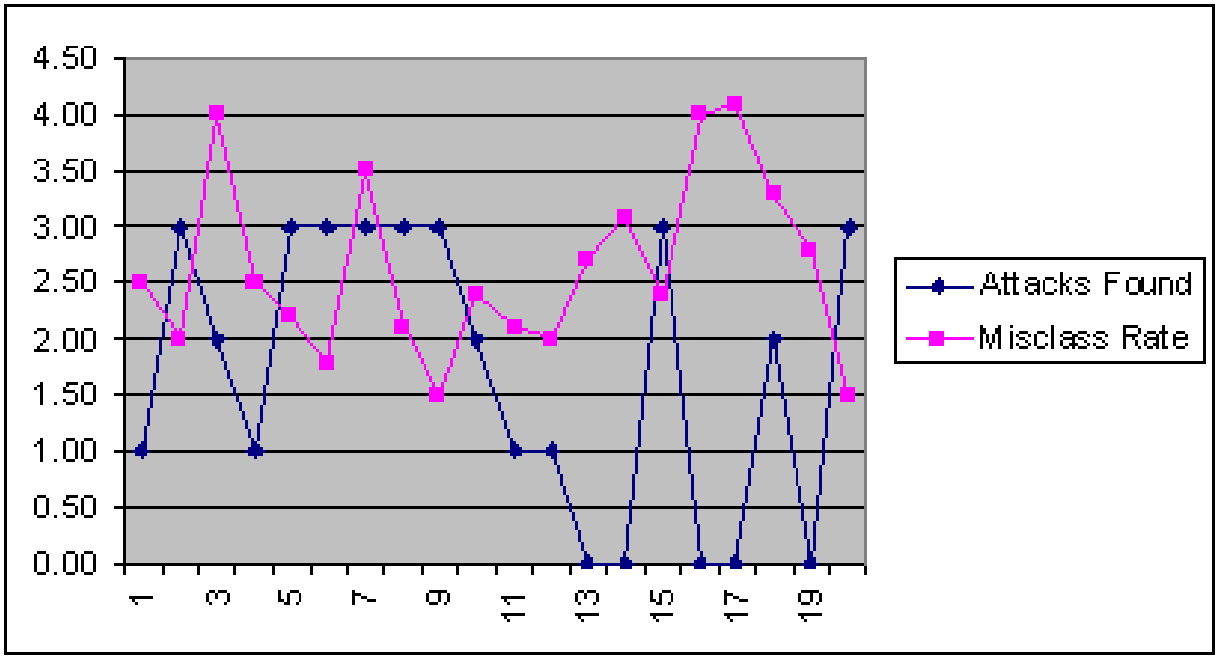


Figure 5. 3. Attacks Found vs Misclass Rate only, part I

Figure 5.4 shows a typical sub-model resulting structure, as seen in the SAS Enterprise Miner program. This decision tree model is shown as a yellow pie chart that graphically delineates the variables that are important in the development of the model. The misclassification rate is shown for both training (blue line) and validation (red line) data sets. As the decision tree is created layer by layer, the misclassification rate is adjusted.

The lower the misclassification rate, the better for the overall performance of the model. Once the model "tops out" and cannot improve any further, the upper limit on the number of leaves in the decision tree is met. This is shown below with the white vertical line that bisects the red and blue data set lines. In SAS, you can manually adjust the tree to have more or less leaves, but this adjustment usually provides no real improvement in the model's performance overall.

The sub-model shown below is one of the better performing models; indeed, it had one of the lowest misclassification rates of any of the sub-models and it discovered one of the attacks (attack 3176) well. It also had a very low false alarm rate, especially when compared to the other sub-models.

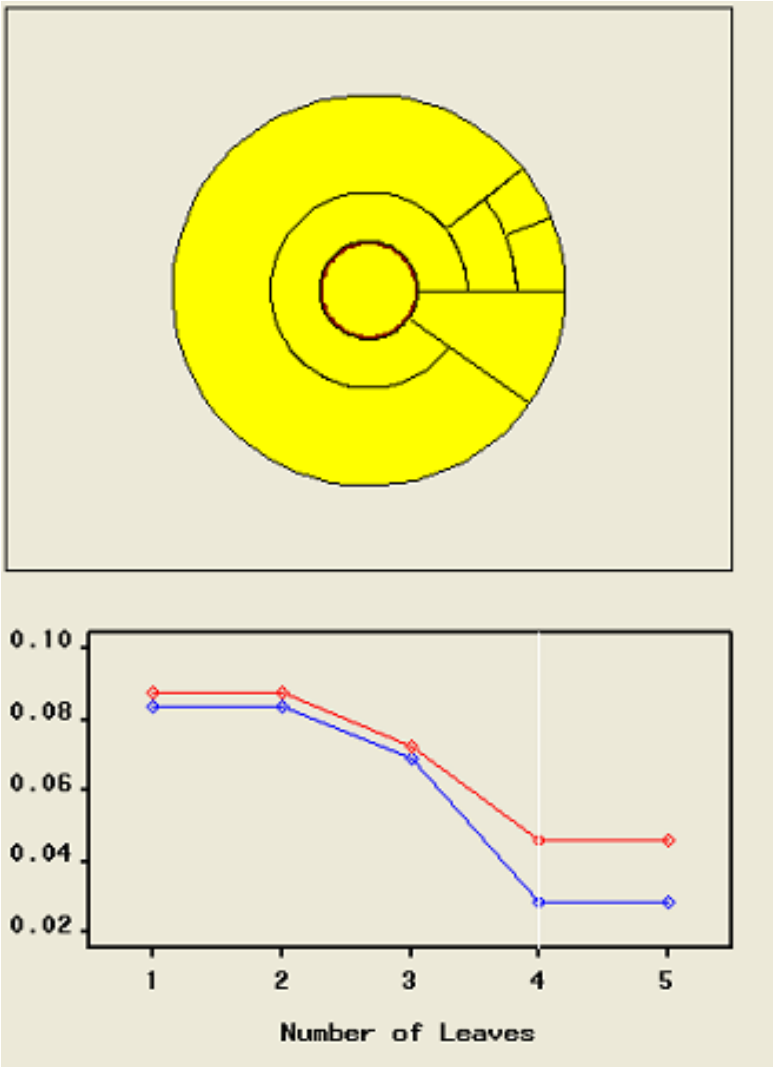


Figure 5. 4. Screenshot of SAS for Sub-Model X6

The sub-model seen in Figure 5.5 is similar to the previous sub-model in that it has a low misclassification rate coupled with a low false alarm rate. This sub-model discovered only attack 3176 as well and the yellow pie chart indicates that the variables that were important for the development of this sub-model were important for the development of the previous sub-model.

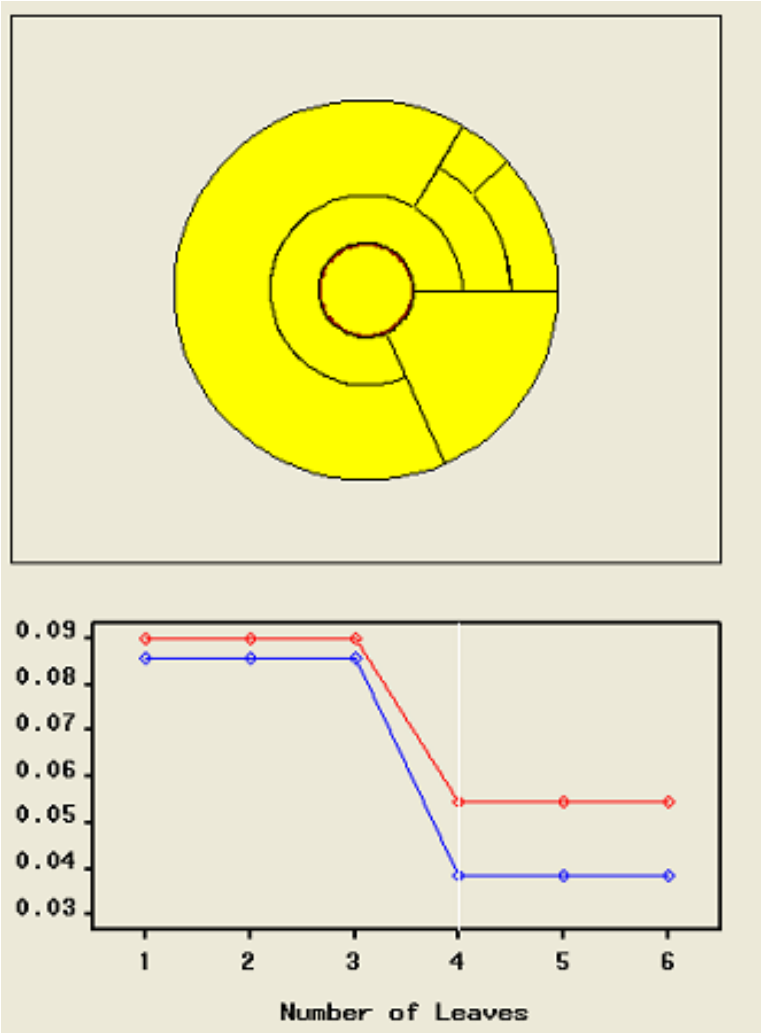


Figure 5. 5. Screenshot of SAS for Sub-Model X7

Another useful bit of information was derived from the variable importance tables. When SAS processes the decision tree, it determines the relative importance of the variables when creating the tree. Not surprisingly, the destination meta-variable (which is a nominal input variable that consists of the destination IP address concatenated with the destination port

number) proved to be the most-important variable for our first bootstrapped model thirteen out of twenty times. Table 5.1 below shows the entire important variable selections along with the sub-model's corresponding misclassification rate.

Table 5. 1. Misclass Rates and Variable Importance, part I

	Misclass Norm	Misclass	1st Var	2nd Var	3rd Var	4th Var
X1	2.5	0.025	Dest	Firstseq	Avgtos	
X2	2	0.02	Dest	Source	Sumttl	
X3	4	0.04	Dest	Firstseq	Avgtos	
X4	2.5	0.025	Dest	Firstseq	Sumtos	
X5	2.2	0.022	Dest	Firstseq		
X6	1.8	0.018	Dest	Firstack	Source	
X7	3.5	0.035	Source	Firstack	Firstseq	Avgtos
X8	2.1	0.021	Dest	Firstseq		
X9	1.5	0.015	Dest	Source	Avgwin	
X10	2.4	0.024	Dest	Sumdgml		
X11	2.1	0.021	Source	Firstseq	Dest	Minttl
X12	2	0.02	Dest	Firstseq	Minttl	
X13	2.7	0.027	Firstack	Session#	Firstseq	
X14	3.1	0.031	Firstseq	Sumttl		
X15	2.4	0.024	Dest			
X16	4	0.04	Firstseq	Maxtos		
X17	4.1	0.041	Session#	Firstseq		
X18	3.3	0.033	Dest	Maxtos	Sumwin	
X19	2.8	0.028	Firstseq			
X20	1.5	0.015	Dest	Firstseq		

These twenty sub-models collectively make up the first bootstrap model that we constructed. This bootstrap model attained true positives nearly 65% of the time while reducing the number of false positives as well. We reduced the false positive number even further (to 0.062) when we removed the instances when we found all three attacks but also achieved an abnormally high amount of false positives.

5.2.4 SAND Bootstrapping Results – Part II

We learned several model-building lessons from the first bootstrap model that was shown in section 5.2.3. First, since the vast majority of our most-important variables was the destination variable, we determined that ensuring that a few dozen nearby (i.e., closest to the three attack sessions with regards to time) non-attack sessions were kept in our training and validation sets. These nearby sessions provided some extra information for the data mining application to use in its discovery phase while it was automatically constructing the decision tree model.

Second, since some of our meta-variables were created for bookkeeping purposes, we chose to reject those variables to ensure that they did not give misleading information to the data mining application while it was constructing the model. Variables like "firstseq," which indicated the sequence number of the first packet in a session, provide little usable information for our data mining operation. Therefore, it was rejected or removed from consideration while model construction continued. Similarly, variables like "firstack" and "firstobs" also were removed from contention. Incidentally, these three variables were not removed from consideration until after the fourth sub-model was constructed, as seen in Table 5.2.

Figure 5.6 describes the SAND bootstrap part II improvements made from SAND bootstrap part I. For each sub-model in part II, we had more successes finding the attacks while the false alarm rate was reduced by almost 25 percent. Although the average normalized misclassification rate increased by 40 percent overall, the number of sub-models with extremely high false alarms was reduced substantially.

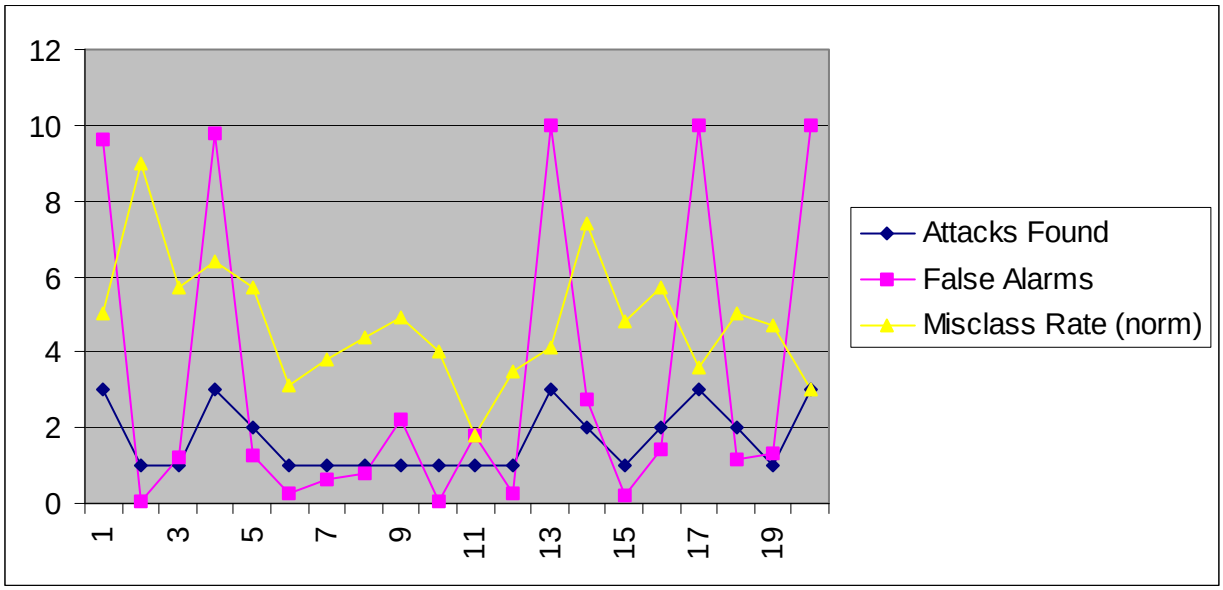


Figure 5. 6. Attacks Found vs False Alarms and Misclassification Rate, part II

Figure 5.7 shows the same data without the false alarm information. With three notable exceptions – sub-models X2, X17, and X20 – most of the data that showed success in finding at least one attack was directly related to the fluctuations of the misclassification rates.

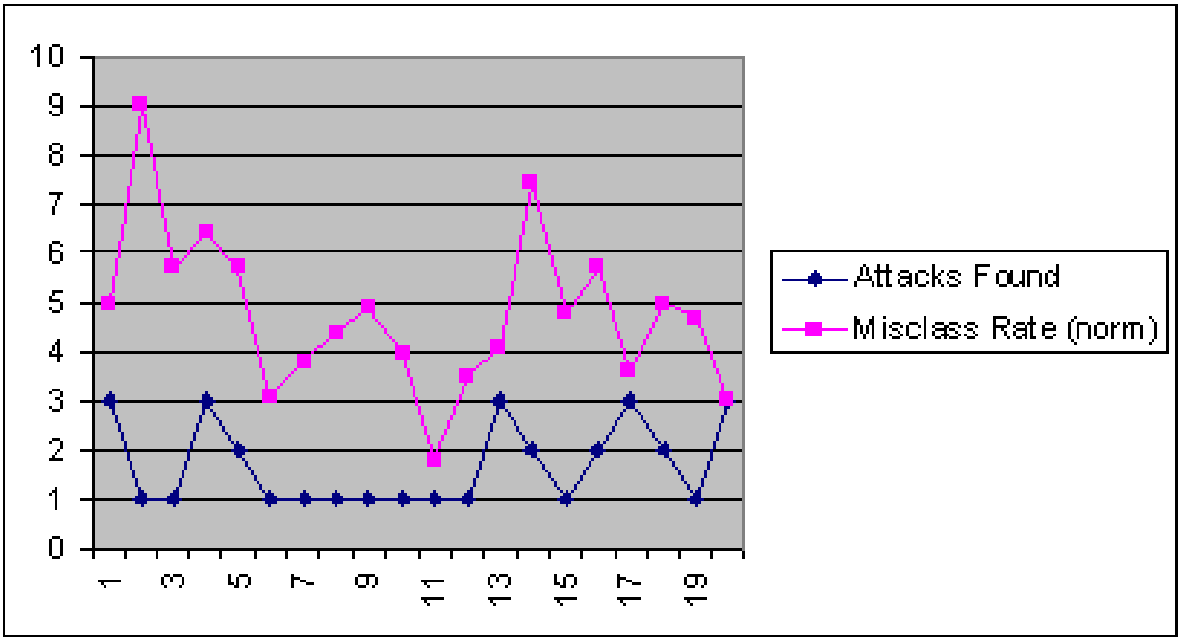


Figure 5. 7. Attacks Found vs Misclassification Rate only, part II

Collectively, these twenty sub-models in part II make up our entire second bootstrap model. We had a similar set of important variables as seen in part I. These variables are shown in Table 5.2. As before, the "destination" variable played the most-important variable role more than a dozen times. The best combination of variables – when considering misclassification rate, hit rate, and false alarm rate – appears to be "destination" along with "sumtos".

The worst situation occurred when "destination" was determined to be the only important variable, as seen in sub-models X13, X17 and X20. While these three sub-models found all three attacks, they also had a very high false alarm rate.

Table 5. 2. Misclass Rates and Variable Importance, part II

	Misclass	1st Var	2nd Var	3rd Var	4th Var
X1	0.05	Firstseq	Dest	Firstobs	
X2	0.09	Dest	Numpkts		
X3	0.057	Firstobs	Firstack	Firstseq	Avgwin
X4	0.064	Firstobs	Firstseq		
X5	0.057	Dest	Sumtos	Avgwin	
X6	0.031	Dest	Sumtos		
X7	0.038	Dest	Source	Sumtos	
X8	0.044	Dest	Sumtos		
X9	0.049	Dest	Firstobs	Minttl	
X10	0.04	Dest	Numpkts		
X11	0.018	Dest	Minttl		
X12	0.035	Dest	Sumtos		
X13	0.041	Dest			
X14	0.074	Firstobs	Avgtos	Minttl	
X15	0.048	Dest	Firstobs	Numpkts	
X16	0.057	Dest	Minttl	Maxtos	
X17	0.036	Dest			
X18	0.05	Dest	Source	Numpkts	Minttl
X19	0.047	Dest	Avgtos		
X20	0.03	Dest			

5.2.5 Future SAND Bootstrapping Research

Further research will be conducted regarding the modeling of specific attacks in conjunction with two layers of our bootstrapping model. These unique models should be able to more accurately portray a computer network by bootstrapping sub-models as shown above.

The difference with these sub-models is that only one type of attack will be used when creating the twenty decision trees (sub-models). These twenty sub-models will be bootstrapped together. Then, a second layer of bootstrapping will occur when we overlay other bootstrap models that addressed different attack types. The misclassification rates for these models will be extremely low and the resulting false alarm rate should be reduced another 50 percent as well. Further research will be conducted on increasing the number of sub-models to fifty to bring down the misclassification rates and false alarm rates even further.

CHAPTER 6: OVERVIEW AND CONCLUSIONS

This dissertation began by describing the challenges facing the designers and engineers of intrusion detection systems, particularly when using signature-based implementations. We proceeded with a review of current anomaly and data mining-based efforts, most of which primarily examined pattern matching and rule creation techniques. We then discussed data mining principles and their applications towards detecting novel intrusions in computer networks, exploring the modeling capabilities of our system against pure network packets contained in the UCF and IDEVAL data, as seen in section 4.5. Finally, we applied these concepts towards our two sets of solutions – SAND and SAND with bootstrapping, producing encouraging results.

During our discussion of current data mining principles and techniques, we turned our attention to the usefulness of decision tree modeling in our type of research. Decision tree modeling techniques are effective for analyzing very large data sets that we used in our research. The biggest drawback is the fact that the results can be unstable because decision trees are data driven constructs and small variations can lead to substantially different final solutions. We saw this over and over again in our research when simple alterations of a few variables produced dramatically different results. This disadvantage, however, was offset by the fact that decision trees are accurate modeling tools that are intuitive and relatively easy to implement.

In our first implementation of Session-based Anomaly Notification Detector (SAND), we chose data that were retrieved at one critical juncture in the network and modeled the network to reflect the situation at that juncture alone. We then used our SAND algorithm to create meta-

variables, collapse the packets back into session-like constructs, remove extraneous variables that proved to be of little value, and create the tree model itself.

Our second implementation of SAND (with bootstrapping) gave us improved results over the non-bootstrapped model. Bootstrapping is a somewhat new concept where you repeat the modeling process in an iterative fashion and keep choosing one row until you have multiple rows in your sample data set. During this process, we may see that some rows may be repeated; on the other hand, some rows may not be selected at all. Subsequently, we split the sample set into two groups – one for training and the other for validation. After we repeated this model-building process many times, we see that our mean error rate – which is the mean over all of the repetitions (bootstraps) – becomes very low as we build more models.

6.1 Summary of Results

Our experiments tested whether the SAND and SAND with bootstrapping techniques could detect a set of attacks in the FIT data set. We first experimented with modeling pure TCP packets contained in the UCF and IDEVAL data; however, the issues of attack predictability, false alarm rates, and efficiency of the program came into play and supported our assertions that a better way could be found to produce anomaly-based results.

In the SAND without bootstrapping methodology as seen in Section 4.6, we produced promising results with our session-based models (see Figures 4.12 through 4.16) but still maintained a relatively high number of false alarms when compared to the SAND with

bootstrapping modeling (Figures 5.2 through 5.7). Interestingly, the best-fit percentage of attacks in the SAND without bootstrapping modeling occurred with eighteen percent of the model's sessions consisting of attack information; the best-fit percentage of attacks in the SAND with bootstrapping, however, occurred with 4 to 9 percent of the model's sessions containing attacks, which was shown in Section 5.2.

For each sub-model in the second set of network modeling with the FIT data, we had a greater amount of success finding the attacks while the false alarm rate was reduced by almost 25 percent. Although the average normalized misclassification rate increased by 40 percent overall, the number of sub-models with extremely high false alarms was reduced substantially to a more manageable level.

6.2 Future Work

Both of the techniques shown in this dissertation – SAND with and without bootstrapping – provide a glimpse into the future of network modeling and anomaly identification techniques. Further research will be conducted on these projects to improve upon the results found over the last year.

The first modification to our research will be the introduction of a multi-variate variable (i.e., non-binary variable) as the target variable in our data mining research. This target variable will have the type of attack as its value. Currently, we simply use a binary (yes/no) target variable to indicate the presence of an attack in our training and validation data sets. We will

strive to introduce an alert identification target variable, which will allow us to build decision trees around the type of attack seen in the training and validation data. We started to create these models but processing power and storage capacity became an issue quickly in this scenario. Creating and modifying millions of packets into sessions with non-binary target variables create very complex models.

Further, other programs like CART may provide better solution sets for this issue. As stated in chapter 4, SAS was viewed upon simply as an efficient and convenient software tool to support our research methodology; however, the tool itself that we choose to use can be easily changed.

In chapter 5 we discussed the SAND bootstrapping process and mentioned that twenty iterations were sufficient for our bootstrapping needs; however, more research is required to be conducted in order to see if there is an upper threshold of bootstrapping. In other words, how many sub-models can we create and continue to improve upon the hit rates and false alarm rates until we stop making improvements by adding more sub-models to our data? In our current data set, it appears that five to ten more sub-models would improve our results but any more additions to the model would prove extraneous. In other situations, however, the creation of fifty to one hundred sub-models may prove worthwhile to explore.

Other work in this area is needed to concentrate our models on specific attack types. This research could be conducted in conjunction with the modified target variable environment discussed above. Our research showed that the models' performance measures were improved when we directed the model to use only one type of variable in the attack pool. Creating sub-models on this basis – that is, each sub-model is created and keyed against a specific, separate attack type – should produce solid results when these sub-models are bootstrapped together.

Finally, our future research should concentrate on automating the entire data mining process better and more efficiently. Using C++ instead of Java as well as converting a lot of our SAS code into pure SQL statements would provide substantial efficiencies in the storage and processing times of our research work. Also, using these new and better-automated tools would help immensely when we acquire more data from various sources to test our theories out.

6.3 Conclusions

We have demonstrated a unique concept in modeling anomalous traffic – using data mining tools and techniques in order to model network behavior and catch anomalous traffic on a given network.

Both techniques have great potential for future research. However, SAND with bootstrapping provides a more-promising vision for the future of modeling networks to detect anomalous behavior.

All of the limitations of the original SAND model are mitigated when bootstrapping is introduced. False alarm rates go down substantially while the probability of finding an attack is increased. While not perfect, we have shown that these techniques provide a promising approach to future research in this area of network security.

APPENDIX A: EXAMPLE RUNS OF SAND

Appendix A describes the process and the results generated by the implementation of our Session-based Anomaly Notification Detector (SAND). First, we describe the algorithm first seen in Chapter 4; then, we run through the process, step-by-step and describe the results.

A.1 The SAND Algorithm

The algorithm originally shown in Figure 4.10 is shown again in Figure A.1.

1. Identify attacks in training data using Snort IDS signature database
2. Create new variables ("meta-variables")
3. Collapse data into sessions
4. Remove extraneous variables (FIRSTID, FIRSTSEQ, etc.)
5. Generate random sample at preselected attack rate
6. Create decision tree model
7. Apply score data set to decision tree model

Figure A. 1. SAND Algorithm

A.1.1 Step 1 – Identify Attacks in Database

We used the editcap to take a saved capture file as its input and then write some or all of the packets into another capture file. We used this program to take the original file and slice the file up into smaller files, each containing 100 thousand packets. This allowed us to process the data faster as our Java program could not handle resulting text files with much more than 100 thousand packets.

An example set of editcap commands are shown below. Generally, this is the first step when slicing apart a raw network data dump; here, we used editcap to take apart a file called "tcp.2004-10-30.tcpdump."

We then created a corresponding file with our own naming convention. The crafted file on line 1 – "10-30tcp101300k.tcpdump" simply means that it is a file with 100,000 packets from the October 30th (10-30) data set; and, the last bit of information is from packet number 101,300,000 (101300k).

Batch files were created to handle these types of operations in order to reduce the drag on our system's resources since more than 100,000 packets usually presented technical problems during our data preparation phase.

```
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101300k.tcpdump 101200001-101300000
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101400k.tcpdump 101300001-101400000
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101500k.tcpdump 101400001-101500000
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101600k.tcpdump 101500001-101600000
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101700k.tcpdump 101600001-101700000
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101800k.tcpdump 101700001-101800000
```

```
c:\>  editcap -r tcp.2004-10-30.tcpdump 10-30tcp101900k.tcpdump 101800001-101900000
```

We then turned to our Java program to properly format our data sets. The original data was then split into training and validation data sets, where the IDS program called Snort played a key role. Snort analyzed one portion of the entire data set – this portion contained 2.4 million packets and hundreds of attack packets.

A.1.2 Step 2 – Create New Variables ("meta-variables")

After we completed the initial formatting of the data, we turned to using SAS code to create new variables and to clean up the current set of variables present. The SAS code is shown below.

The first set of the code simply copies the file called "wide_new" under the "fit" directory and creates a new file called "tcpWide" in the same directory in SAS.

```
data fit.tcpWide; set fit.wide_new;
```

Then, we strip out the non-TCP packets from the data set and start counting the number of observations (countobs).

```
IF PROTO = 'UDP' THEN DELETE;

IF PROTO = 'ARP' THEN DELETE;

IF ID = 'ID' THEN DELETE;

IF SRC_PT = 'ICMP' THEN DELETE;

IF DEST_PT = 'ICMP' THEN DELETE;

sp = ' : ';

ar = ' -> ';

DROP count;

RETAIN countobs;

IF countobs > 0 THEN countobs = countobs + 1;

ELSE countobs = 0;

IF countobs = 0 THEN countobs = countobs + 1;
```

Next, we take the flag values from our Java program and reset them to be binary values:

```
IF R1 = 'NoR1' THEN _R1 = 0; ELSE _R1 = 1;

IF R2 = 'NoR2' THEN _R2 = 0; ELSE _R2 = 1;

IF URG = 'NO-U' THEN _URG = 0; ELSE _URG = 1;

IF ACK = 'NO-A' THEN _ACK = 0; ELSE _ACK = 1;

IF PSH = 'NO-P' THEN _PSH = 0; ELSE _PSH = 1;

IF RST = 'NO-R' THEN _RST = 0; ELSE _RST = 1;

IF SYN = 'NO-S' THEN _SYN = 0; ELSE _SYN = 1;

IF FIN = 'NO-F' THEN _FIN = 0; ELSE _FIN = 1;
```

Then we create a new variable called "_newID" – which will eventually be called "newID" due to SAS coding idiosyncrasies – which is a metavariable containing source IP and

port numbers; destination IP and port numbers; SYN and FIN flags; and, number of observations counted. The DROP and RENAME functions are quirks of the programming environment which force us to use temporary variable names – i.e., those preceded by an underscore character – and then drop the real names (R1, R2, URG, etc) prior to renaming the temporary variable names back to the real names.

```
_newID = SRC_IP || sp || SRC_PT || ar || DEST_IP || sp || DEST_PT || sp || _SYN ||
sp || _FIN || sp || countobs;

DROP R1 R2 URG ACK PSH RST SYN FIN;

RENAME _R1 = R1;
RENAME _R2 = R2;
RENAME _URG = URG;
RENAME _ACK = ACK;
RENAME _PSH = PSH;
RENAME _RST = RST;
RENAME _SYN = SYN;
RENAME _FIN = FIN;
DROP newID sp ar;
RENAME _newID = newID;
```

The FORMAT and INFORMAT statements allow us to associate formats and informats to our set of variables. The FORMAT statement gives a special SAS instruction for writing a

variable. Conversely, the INFORMAT statement gives special SAS instructions when reading a variable. The statement below are for integers (interval variables) of length 1 or 2.

```
FORMAT TGT 1.;

FORMAT DF 1.;

FORMAT newIPL 1.;

FORMAT newTCPL 1.;

FORMAT newTOS 1.;

FORMAT newTTL 1.;

FORMAT newFRAG 1.;

FORMAT R1 1.;

FORMAT R2 1.;

FORMAT URG 1.;

FORMAT ACK 1.;

FORMAT PSH 1.;

FORMAT RST 1.;

FORMAT SYN 1.;

FORMAT FIN 1.;

FORMAT MO 2.;

FORMAT DY 2.;

FORMAT HR 2.;

FORMAT MN 2.;


INFORMAT TGT 1.;

INFORMAT DF 1.;

INFORMAT newIPL 1.;

INFORMAT newTCPL 1.;

INFORMAT newTOS 1.;
```

```
INFORMAT newTTL 1.;

INFORMAT newFRAG 1.;

INFORMAT R1 1.;

INFORMAT R2 1.;

INFORMAT URG 1.;

INFORMAT ACK 1.;

INFORMAT PSH 1.;

INFORMAT RST 1.;

INFORMAT SYN 1.;

INFORMAT FIN 1.;

INFORMAT MO 2.;

INFORMAT DY 2.;

INFORMAT HR 2.;

INFORMAT MN 2.;

run;
```

After we ran the SAS code shown above, we then created a "fitmerge" file that combined the results from the "tcpwide" file with the "fitnewalerts" file that contained all of the Snort-identified alerts in our training data set. The resulting file contained the one-to-many merged results that were keyed on the MN (minutes) and SEC_MSEC (seconds and milliseconds) variables.

```
data fit.fitmerge;

MERGE fit.tcpwide fit.fitnewalerts;

BY MN SEC_MSEC;

run;
```

We then executed a series of commands in SAS that continues to create metavariables and also collapses the data into session-like structures. Collapsing the data had two practical effects on our research – it created unique and helpful data patterns not present in the original network packet format and it also drastically reduced the amount of processing time and processing power required to create the decision tree model.

The code below creates a new file called "coll_new" in the fit directory from the "fitmerge" file mentioned earlier. The RETAIN statement in SAS allows the system to preserve a variable's value from the previous iteration of the DATA step.

```
data fit.coll_new; set fit.fitmerge;

RETAIN sessionNUM;
RETAIN ok;
RETAIN source;
RETAIN destination;
RETAIN firstobs;
RETAIN lastobs;
RETAIN firstsec;
```

```
RETAIN firstID;

RETAIN firstSEQ;

RETAIN firstACK;
```

A.1.3 Steps 3 & 4 – Collapse data into sessions and remove extraneous variables

Then we created a collapsed session for the first observed packet. New variables like firstobs, firstsec, firstID, firstSEQ, and firstACK are used for bookkeeping and to also see if the new variable can attain some importance in our model later.

```
IF countobs = 1 THEN DO;

   source = SRC_IP || SRC_PT;

   destination = DEST_IP || DEST_PT;

   firstobs = countobs;

   firstsec = HR || MN || SEC_MSEC;

   firstID = ID;

   firstSEQ = SEQNUM;

   firstACK = ACKNUM;

   sessionNUM = 1;

END;
```

Next, we created collapsed sessions for the remaining observed packets in our training data set.

```
ELSE IF SYN = 1 THEN DO;
   source = SRC_IP || SRC_PT;
   destination = DEST_IP || DEST_PT;
   sessionNUM = sessionNUM + 1;
   firstobs = countobs;
   firstsec = HR || MN || SEC_MSEC;
   firstID = ID;
   firstSEQ = SEQNUM;
   firstACK = ACKNUM;
END;

run;
```

PROC SQL statement was then used to create a table of data ("coll_new2") and to group the data together. Further, we established four new variables each for seven of the interval variables. These four new variables were derived from the minimum (min) function, maximum (max) function, average (avg) function, and summation (sum) function. It turns out that most of these new variables were not useful in our model creation stages, but a few were – especially SUMTTL and MAXTTL.

We then deleted extraneous variables (FIRSTID, FIRSTSEQ, etc) along the way as we determined which ones were not important to our research.

```
proc sql;

   create table fit.coll_new2 as

      select sessionNUM, firstobs,
             firstsec, count(*) as numPKTS,
             firstID, source, destination, max(TGT) as newTGT,
             firstSEQ, firstACK, min(TTL) as minTTL,
             max(TTL) as maxTTL, avg(TTL) as avgTTL,
             sum(TTL) as sumTTL, min(IPL) as minIPL,
             max(IPL) as maxIPL, avg(IPL) as avgIPL,
             sum(IPL) as sumIPL, min(DGML) as minDGML,
             max(DGML) as maxDGML, avg(DGML) as avgDGML,
             sum(DGML) as sumDGML, min(TCPL) as minTCPL,
             max(TCPL) as maxTCPL, avg(TCPL) as avgTCPL,
             sum(TCPL) as sumTCPL, min(LEN) as minLEN,
             max(LEN) as maxLEN, avg(LEN) as avgLEN,
             sum(LEN) as sumLEN, min(TOS) as minTOS,
             max(TOS) as maxTOS, avg(TOS) as avgTOS,
             sum(TOS) as sumTOS, min(WIN) as minWIN,
             max(WIN) as maxWIN, avg(WIN) as avgWIN,
             sum(WIN) as sumWIN

      from fit.coll_new
      group by sessionNUM, firstobs, firstsec,
           firstID, source, destination, firstSEQ, firstACK;
```

A.1.4 Step 5 – Generate random samples

To boost our attack rate in our data and pre-selected rates, we kept the sessions that contained attack packets and removed only non-attack sessions in order to create a more meaningful sample of data to train.

In the code shown below, we established a uniform random variable for each session called "rand." If the session was a non-attack session (newTGT = 0) and the rand variable was greater than a certain value (here it is 0.03) then we deleted that session and dropped the rand variable from the data set.

This last step was key since we found out that keeping the rand variable in our data set skewed the results and produced models that gave top importance to the rand variable and found a small number of attacks in our score data set.

```
data fit.coll_final; set fit.coll_new2;

rand = ranuni(0);

if newTGT = 0 and rand > 0.03 THEN DELETE;

DROP rand;

run;
```

A.1.5 Step 6 – Create decision tree model

The decision tree model was then created, as seen in Figure A.2. The pie chart shown in the tree visualization represents the top, root node of the tree (inner-most circle) and the subsequent branches of the decision tree (outer circles extending from the inner-most circle).

Generally, any training or validation set that had a misclassification rate of greater than 0.08 were unusable. These relatively high misclassification rates normally produced extremely high false alarm rates in our data. Further, trees that had three or less leaves were usually unusable as well. These trees usually produced results where no sessions were found to be attacks.

In a similar vein, decision trees that contained unusually low misclassification rates (i.e., those less than 0.015) normally produced results where no sessions were found to be attack sessions in the score data.

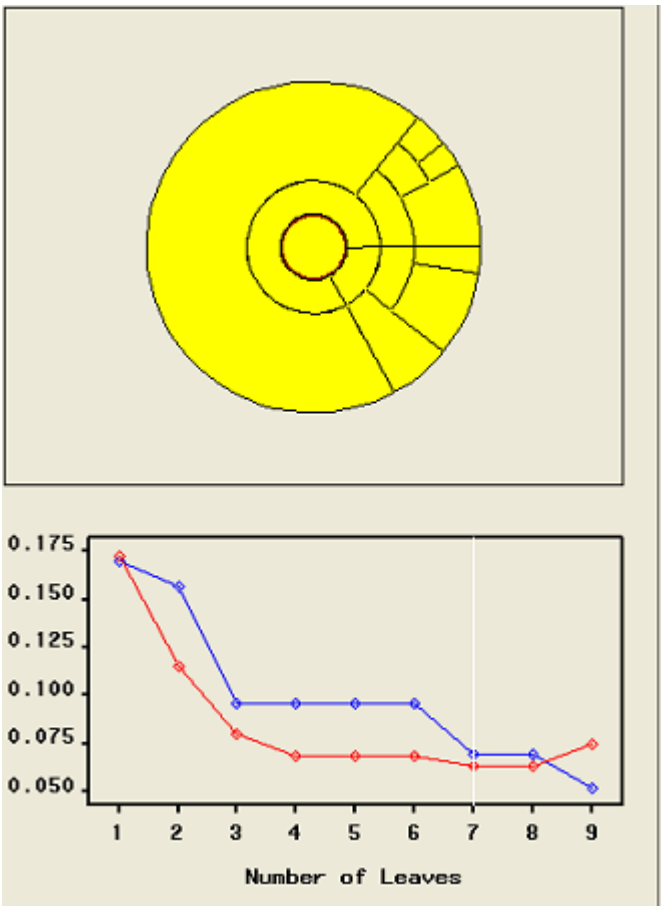


Figure A. 2. Example Decision Tree Visualization

A.1.6 Step 7 – Apply score data set to decision tree model

After creating the decision tree model from our training and validation data sets, we ran that model against a separate set of data from the same source. Earlier tests against different sets of data from different sources produced surprisingly good results. Our research with sets of data from the same source did not disappoint either.

The SAS code below shows several macro variables (delineated by the "%" character), which allow us to use them over and over again for the same function in different spots during the execution of the SAS program.

The %let statements at the beginning denote the score data set that we are working on ("coll_new2") and the resulting predicted data set ("X") that shows how well our model worked in trying to find the attacks in the score set using our anomaly-based methodology.

```
%let _Score=fitscore.coll_new2;
%let _predict=fitscore.X;

%macro DMNORLEN; 32 %mend DMNORLEN;

%macro DMNORMCP(in,out);
   &out=substr(left(&in),1,min(%dmnorlen,length(left(&in))));
&out=upcase(&out);

   %mend DMNORMCP;

   %macro DMNORMIP(in);
   &in=left(&in);
   &in=substr(&in,1,min(%dmnorlen,length(&in)));
   &in=upcase(&in);
   %mend DMNORMIP;
```

```
DATA &_PREDICT ; SET &_SCORE ;

%macro DMNORLEN; 32 %mend DMNORLEN;

%macro DMNORMCP(in,out);
&out=substr(left(&in),1,min(%dmnorlen,length(left(&in))));
&out=upcase(&out);
%mend DMNORMCP;

%macro DMNORMIP(in);
&in=left(&in);
&in=substr(&in,1,min(%dmnorlen,length(&in)));
&in=upcase(&in);
%mend DMNORMIP;


DATA &_PREDICT ; SET &_SCORE ;


*-------------------------------------------------------*;
*  START_CHUNK 1431122153.5:T2Z7NB6Y                    *;
*-------------------------------------------------------*;
*  END_CHUNK 1431122153.5:T2Z7NB6Y                      *;
*-------------------------------------------------------*;

*-------------------------------------------------------*;
*  START_CHUNK 1431179796.9:T3CIS7YA                    *;
*-------------------------------------------------------*;
```

```
*  END_CHUNK 1431179796.9:T3CIS7YA                    *;
*-----------------------------------------------------*;


*-----------------------------------------------------*;
*  START_CHUNK 1431179812.4:T00UYZOX                  *;
*-----------------------------------------------------*;
*                                                     ;
*  TOOL : Tree                                        ;
*  TYPE : MODEL                                       ;
*  NODE : Tree [T00UYZOX]                             ;
*                                                     ;
*-----------------------------------------------------;
*                                                     ;
*  MODEL NAME : Untitled                              ;
*  DESCRIPTION :                                      ;
*                                                     ;
*  TARGET : NEWTGT                                    ;
*-----------------------------------------------------;
```

Then, new variables are created in the predicted data set (“X”) called I_NEWTGT, F_NEWTGT, and so on.

```
*******************************************************;
      ****** DECISION TREE SCORING CODE ******;
*******************************************************;
```

```
****** LENGTHS OF NEW CHARACTER VARIABLES ******;

LENGTH I_NEWTGT $ 12;
LENGTH F_NEWTGT $ 12;
LENGTH _WARN_ $ 4;
****** LABELS FOR NEW VARIABLES ******;
LABEL _NODE_ = 'Node';
LABEL _LEAF_ = 'Leaf';
LABEL P_NEWTGT0 = 'Predicted: NEWTGT=0';
LABEL P_NEWTGT1 = 'Predicted: NEWTGT=1';
LABEL I_NEWTGT = 'Into: NEWTGT';
LABEL U_NEWTGT = 'Unnormalized Into: NEWTGT';
LABEL F_NEWTGT = 'From: NEWTGT';
LABEL R_NEWTGT0 = 'Residual: NEWTGT=0';
LABEL R_NEWTGT1 = 'Residual: NEWTGT=1';
LABEL _WARN_ = 'Warnings';

****** TEMPORARY VARIABLES FOR FORMATTED VALUES ******;
LENGTH _ARBFMT_2 $ 12;
DROP _ARBFMT_2;
_ARBFMT_2 = ' ';

/* Initialize to avoid warning. */
LENGTH _ARBFMT_4 $ 27;
DROP _ARBFMT_4;
_ARBFMT_4 = ' ';

/* Initialize to avoid warning. */
```

```
_ARBFMT_2 = PUT( NEWTGT , BEST12.);
%DMNORMCP( _ARBFMT_2, F_NEWTGT );
```

The next sequence of SAS code assigns the observations (sessions) to a certain node and gives it a corresponding predictive value for the probability of whether or not that session is an attack. This is represented by the P_NEWTGT1 variable. If it is greater than 0.5 then that session is probably an attack session.

```
****** ASSIGN OBSERVATION TO NODE ******;
IF NOT MISSING(FIRSTSEQ ) AND
1026865988.5 <= FIRSTSEQ THEN DO;
   IF NOT MISSING(FIRSTID ) AND
   37468.5 <= FIRSTID THEN DO;
      _NODE_ = 7;
      _LEAF_ = 6;
      P_NEWTGT0 = 1;
      P_NEWTGT1 = 0;
      I_NEWTGT = '0';
      U_NEWTGT = 0;
   END;

ELSE DO;
_ARBFMT_4 = PUT( DESTINATION , $27.);
%DMNORMIP( _ARBFMT_4);
IF _ARBFMT_4 IN ('163.118.222.94        27663' ,
```

```
   '163.118.98.10          2112' ,'163.118.178.0         57731' ,
   '163.118.178.0         44381' ,'163.118.218.237           0' )
THEN DO;
       _NODE_ = 11;
       _LEAF_ = 5;

       P_NEWTGT0 = 1;
       P_NEWTGT1 = 0;
       I_NEWTGT = '0';
       U_NEWTGT = 0;
   END;


   ELSE DO;
       _NODE_ = 10;
       _LEAF_ = 4;
       P_NEWTGT0 = 0;
       P_NEWTGT1 = 1;
       I_NEWTGT = '1';
       U_NEWTGT = 1;
   END;

END;
END;
```

The code below shows how a crafted variable like SUMTTL can have a positive impact on the predictive nature of our model. Here, the P_NEWTGT1 variable has a value of 0.83333.

```
ELSE DO;
IF NOT MISSING(SUMTTL ) AND
35142.5 <= SUMTTL THEN DO;
_NODE_ = 5;
_LEAF_ = 3;
P_NEWTGT0 = 0.16666666666666;
P_NEWTGT1 = 0.83333333333333;
I_NEWTGT = '1';
U_NEWTGT = 1;
END;

ELSE DO;
IF NOT MISSING(NUMPKTS ) AND
80.5 <= NUMPKTS THEN DO;
    _NODE_ = 9;
    _LEAF_ = 2;
    P_NEWTGT0 = 0.78723404255319;
    P_NEWTGT1 = 0.2127659574468;
    I_NEWTGT = '0';
    U_NEWTGT = 0;
END;

ELSE DO;
```

```
    _NODE_ = 8;
    _LEAF_ = 1;
    P_NEWTGT0 = 0.97692307692307;
    P_NEWTGT1 = 0.02307692307692;
    I_NEWTGT = '0';
    U_NEWTGT = 0;
END;
END;
END;

***** RESIDUALS R_ ************;
IF F_NEWTGT NE '0'
AND F_NEWTGT NE '1' THEN DO;
    R_NEWTGT0 = .;
    R_NEWTGT1 = .;
END;
ELSE DO;
    R_NEWTGT0 = -P_NEWTGT0;
    R_NEWTGT1 = -P_NEWTGT1;
SELECT( F_NEWTGT );
    WHEN( '0' ) R_NEWTGT0 = R_NEWTGT0 +1;
    WHEN( '1' ) R_NEWTGT1 = R_NEWTGT1 +1;
END;
END;
*******************************************************;
****** END OF DECISION TREE SCORING CODE ******;
*******************************************************;
```

```
 RUN  ;

 QUIT ;

/*-------------------------------------------------------*/
/*  ENTERPRISE MINER: END SCORE CODE                     */
/*-------------------------------------------------------*/


options nocenter pagesize=2000 linesize=80 nodate
  nonumber;
  proc sort data=&_predict;
      by descending P_TGT1;
  run;
```

A.2 The SAND Algorithm with Bootstrapping

The algorithm originally shown in Figure 5.1 is shown again in Figure A.3. Most of the steps are similar to the original SAND algorithm; however, several steps (5, 9, 10, and 11) are inserted to create the bootstrapped model from twenty or more sub-models.

1. Identify attacks in training data using Snort IDS signature database

2. Create new variables ("meta-variables")

3. Collapse data into sessions

4. Remove extraneous variables (FIRSTID, FIRSTSEQ, etc.)

5. *newModel* = { }

SAND BOOTSTRAPPING LOOP:

6. Generate random sample at pre-selected attack rate

7. Create decision tree model

8. Apply score data set to decision tree model

9. *ok_subModel* = high scoring sub-model not yet in *newModel*

10. Add *ok_subModel* to *newModel*

11. Go to step 6

Figure A. 3. Sand Bootstrapping Algorithm

A.2.1 Bootstrap Loop

After we go through the initial steps towards creating the decision tree model, we enter a continuous loop where we continually create new training data sets and eventually creating a large model made up of twenty or more sub-models.

These loops are beneficial since we were able to pick and choose the good models and discard the not-so-good models. Then, we added the chosen models to become a sub-model within our larger bootstrap-looped model.

A.2.2 Bootstrap results

The benefit of using bootstrapping is that it minimizes misclassification rates and it also minimizes the effects of any remaining high misclassification rates. Bootstrapping also significantly lowered the false alarm rates as well as increased the hit rates.

A more automated system outside of the SAS environment where thousands of sub-models would be created (instead of twenty) would likely produce a very close model of the actual network. Further, when we separated and segregated our sub-models by attack type, the number of false positives dropped dramatically, sometimes by 30-fold.

LIST OF REFERENCES

Allen, W. H. (2003). Analysis, Detection, and Modeling of Attacks in Computer Communication Networks. (Doctoral Dissertation, University of Central Florida, 2003).

Association for Computing Machinery (1994). ISO 7498 – Open Systems Interconnection Model. OSI Reference Model is currently maintained at http://www.acm.org/sigs/sigcomm/standards/iso_stds/OSI_MODEL/.

Axelsson, S. (2000). Intrusion Detection Systems: A Survey and Taxonomy. Technical report 99-15, Chalmers University of Technology.

Axelsson, S. (1999). The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection. In *Proceedings, ACM Computer and Communications Security Conference,* 1999.

Bala, J., Baik, S., Hadjarian, A., Gogia, B. K., & Manthome, C. (2002). Application of a Distributed Data Mining Approach to Network Intrusion Detection. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3*.

BBC News (2004). Firm justifies job for virus writer. Retrieved July 8, 2005 from http://news.bbc.co.uk/1/hi/technology/3677774.stm.

Bednarczyk, M., Guili, C., & Bednarczyk, J. (2005). Network Awareness: Adopting a Modern Mindset. BlackHat Consulting, Inc.

Berners-Lee, T. (1992). WorldWideWeb – Summary. Retrieved June 29, 2005 from http://www.w3.org/Summary.html.

Breiman, L. (1996). Bagging predictors. *Machine Learning* 24, 123-140, 1996.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and Regression Trees. Wadsworth, Pacific Grove, CA.

Bykova, M., Ostermann, S., & Tjaden, B. (2001). Detecting Network Intrusions via a Statistical Analysis of Network Packet Characteristics. In *Proceedings, 33rd Southeastern Symposium on System Theory*, 2001.

Carlson, K. (2005). IDS: Security at its Finest. In the May 30, 2005 issue of the online version of VAR Business, retrieved May 31, 2005 from http://www.varbusiness.com/showArticle.jhtml?articleID=163105275.

CCITT (1991). X.800 – Security Architecture for Open Systems Interconnection for CCITT Applications. 1991.

CERT/CC (2000). CERT® Advisory CA-2000-04 Love Letter Worm. Retrieved June 28, 2005 from http://www.cert.org/advisories/CA-2000-04.html.

CERT/CC (2005a). CERT/CC History and Policies. Retrieved June 28, 2005 from http://www.cert.org/research/JHThesis/Chapter3.html.

CERT/CC (2005b). CERT/CC Incident and Vulnerability Trends. Retrieved April 23, 2005 from http://www.cert.org/present/cert-overview-trends/.

CERT/CC (2005c). CERT/CC Statistics – 1988 to 2005. Retrieved April 22, 2005 from http://www.cert.org/stats/cert_stats.html.

Chan, P. K., Mahoney, M. V., and Arshad, M. H. (2003). Learning Rules and Clusters for Anomaly Detection in Network Traffic, *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava & A. Lazarevic (editors), Kluwer.

Chief Security Officer Magazine (2004). 2004 E-Crime Watch Survey.

Combs, G. Ethereal Introduction. Retrieved May 18, 2005 from http://www.ethereal.com/~gerald/presentations/Ethereal_Intro/.

Computer Security Institute (2004). 2004 CSI/FBI Computer Crime and Security Survey.

Dittrich, D. (1999). Tribal Flood Network Analysis. Retrieved July 4, 2005 from http://staff.washington.edu/dittrich/misc/tfn.analysis.

Dreger, H., Kreibich, C., Paxson, V., & Sommer, R. (2005). Enhancing the Accuracy of Network-based Intrusion Detection with Host-based Context, In *Proceedings of DIMVA 2005*, ***to appear***.

DW-World.DE. Sasser Worm Maker Gets Light Sentence. Retrieved July 8, 2005 from BBC News (2004). Firm justifies job for virus writer. Retrieved July 8, 2005 from http://www.dw-world.de/dw/article/0,1564,1643909,00.html.

Eskin, E., Miller, M., Zhong, Z.-D., Yi, G., Lee, W.-A. and Stolfo, S. (2000). Adaptive Model Generation for Intrusion Detection. In *Proceedings of the ACMCCS Workshop on Intrusion Detection and Prevention*, Athens, Greece, 2000.

Ethereal.com. Editcap Manual Page. Retrieved May 3, 2005 from http://www.ethereal.com/docs/man-pages/editcap.1.html.

Floyd, S. & Paxson, V. (2001). Difficulties in Simulating the Internet. In *IEEE/ACM Transactions on Networking*, Vol.9, No.4, pp. 392-403, August 2001.

Haines, J. W., Rossey, L. M., & Lippmann, R. P. (2001). Extending the DARPA Off-line Intrusion Detection Evaluations. In *Proceedings IEEE DARPA Information Survivability Conference and Exposition (DISCEX II)*.

Han, J. & Kamber, M. (2001). Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, first edition.

Hegde, V. (2005). Exploring TCP/IP with TCPdump and Tethereal. Retrieved May 1, 2005 from http://www.faqs.org/docs/gazette/tcp.html.

Hu, Y. & Panda, B. (2004). A Data Mining Approach for Database Intrusion Detection. In *Proceedings of the 2004 ACM Symposium on Applied Computing,* March 2004.

InfoSysSec (2005). Denial of Service Attacks. Retrieved July 4, 2005 from http://www.infosyssec.com/infosyssec/secdos1.htm.

Internet Security Systems (2004). Internet Security Systems Security Alert - Microsoft LSASS Sasser Worm Propagation, May 1, 2004.

Internet Security Systems (2005). Phishing and other significant threats of 2004. *X-Force® Threat Insight Quarterly,* February 2005.

Johnson, J. T. (2004). Security today means playing 'defense in depth'. *NetworkWorld*, August 16, 2004. Retrieved July 4, 2005 from http://www.networkworld.com/columnists/2004/081604johnson.html.

Joshi, K. P. (2005). Analysis of Data Mining Algorithms. Retrieved March 22, 2005 from http://userpages.umbc.edu/~kjoshi1/data-mine/proj_rpt.htm.

Kabay, M. E. (2001a). Studies and Surveys of Computer Crime. Retrieved June 29, 2005 from www.securitystats.com/reports/.

Kabay, M. E. (2001b). Understanding Studies and Surveys of Computer Crime. Retrieved June 29, 2005 from http://www2.norwich.edu/m**kabay**/methodology/**crime**_stats_methods.pdf.

Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A, & Srivastava, J. (2003). A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In *Proceedings of Third SIAM Conference on Data Mining*, San Francisco, May. 2003.

Lee, W., Stolfo, S. J., & Mok, K. W. (1999). A Data Mining Framework for Building Intrusion Detection Models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press.

Leon-Garcia, A. & Widjaja, I. (2004). In *Communication Networks: Fundamental Concepts and Key Architectures.* McGraw-Hill Publishing, Second Edition.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). The 1999 DARPA Off-Line Intrusion Detection Evaluation, *Draft of paper submitted to Computer Networks*, In Press.

Liston, K. (2005). Can you explain traffic analysis and anomaly detection? SANS Intrusion Detection FAQ. Retrieved June 1, 2005 from http://www.sans.org/resources/idfaq/anomaly_detection.php?printer=Y.

Mahoney, M. V. & Chan, P.K. (2001). PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic, Florida Tech. technical report CS-2001-04, http://cs.fit.edu/~tr/.

Mahoney, M. V. & Chan, P.K. (2003). An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection, in ***Proceedings of Recent Advances in Intrusion Detection (RAID)***, September 8-10 2003; ***Lecture Notes in Computer Science***, Vol. 2820, 220-237, 2003.

MessageLabs (2004). Mydoom Surpasses Sobig.F to Become Fastest Spreading Virus Ever. January 27, 2004. Retrieved June 29, 2005 from http://www.messagelabs.com/news/virusnews/detail/default.asp?contentItemId=734&reg.

Microsoft (2003). PSS Security Response Team Alert - New Worm: W32.Slammer. Retrieved July 2, 2005 from http://www.microsoft.com/technet/security/alerts/slammer.mspx.

Microsoft (2004a). What You Should Know about Sasser. May 1, 2004. Retrieved July 8, 2005 from http://www.microsoft.com/security/incident/sasser.mspx.

Microsoft (2004b). Windows Security Updates for April 2004. Retrieved July 8, 2005 from http://www.microsoft.com/security/bulletins/200404_windows.mspx.

Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., & Weaver, N. (2003). The Spread of the Sapphire/Slammer Worm. Retrieved June 29, 2005 from http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html.

MyCERT (2005). Statistics on W32.Blaster and W32.Nachi Worm on a Single Network. Retrieved June 29, 2005 from http://www.mycert.org.my/graphs/W32.Blaster_W32.Nachi/blaster_nachi.html.

Networkdictionary.com. Ethereal: A Network Packet Sniffing Tool. Retrieved June 4, 2005 from http://www.networkdictionary.com/software/ethereal.php.

Newman, D., Snyder, J., & Thayer, R. (2002). Crying Wolf: False Alarms Hide Attacks. Retrieved July 5, 2005 from http://www.networkworld.com/techinsider/2002/0624security1.html.

Northcutt, S., Cooper, M., Fearnow, M. & Frederick, K. (2001). *Intrusion Signatures and Analysis*. New Riders Publishing, First Edition.

Orebaugh, A. D. (2004). Top Ten Ethereal Tips and Tricks. Retrieved June 11, 2005 from http://www.onlamp.com/pub/a/security/2004/05/13/etherealtips.html.

Paxson, V. & Floyd, S. (1997). Why We Don't Know How To Simulate The Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

Phung, M. (2000). Data Mining in Intrusion Detection. SANS Intrusion Detection FAQ. October 24, 2000.

Postel, J. (1977). IEN 2 – Comments on IP and TCP. ISI, 1977.

Postel, J. (1980). RFC 760 – DoD Standard Internet Protocol.

Postel, J. (1981a). RFC 791 – Internet Protocol. DARPA Internet Program Protocol Specification.

Postel, J. (1981b). RFC 793 – Transmission Control Protocol. DARPA Internet Program Protocol Specification.

Ptacek, T. H. & Newsham, T. N. (1998). Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc.

Roesch, M. (1999). Snort – Lightweight Intrusion Detection for Networks. In *Proceedings, 13th USENIX Systems Administration Conference*, November, 1999.

Safavian, S. R. & Landgrebe, D. (2001). A Survey of Decision Tree Classifier Methodology. In *IEEE Transactions on Systems, Man, & Cybernetics,* Vol. 21, No. 3, May 1991.

SAS Institute, Inc. Getting Started with SAS® Enterprise Miner 4.3™.

SAS Institute, Inc. (2005a). Data Mining Overview. Retrieved May 30, 2005 from http://v9doc.sas.com/cgi-bin/sasdoc/cgigdoc?file=../emgs.hlp/a000167823.htm.

SAS Institute, Inc. (2005b). SAS Corporate Statistics. Retrieved May 30, 2005 from http://www.sas.com/presscenter/bgndr_statistics.html.

Schapire, R.E. (1996). The Strength of Weak Learnability. Machine Learning 5, 197-227, 1996.

Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., & Zhou, S. (2002). Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security,* November 2002.

Shimomura, T. (1995). Takedown Internet site. Retrieved June 28, 2005 from http://www.takedown.com.

Shimomura, T. & Markoff, J. (1996). Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw-By the Man Who Did It. Hyperion Books, 1st edition.

Shirey, R. (2000). RFC 2828 – Internet Security Glossary. GTE/BBN Technologies, 2000.

Small, R.D. & Edelstein, H.A. (2005). Scalable Data Mining. White paper retrieved September 18, 2005 from http://www.twocrows.com/whitep.htm.

Smithsonian Institution Online. Birth of the Internet – Internet History. Retrieved June 29, 2005 from http://smithsonian.yahoo.com/internethistory.html.

SourceFire (2005a). Bad Traffic – Loopback Traffic. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=1%3A528.

SourceFire (2005b). Bad Traffic – TCP Port 0 Traffic. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=1%3A524.

SourceFire (2005c). Bare Byte Unicode Encoding. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=119%3A4.

SourceFire (2005d). FTP Exploit Stat. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=1%3A1777.

SourceFire (2005e). Scan FIN. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=1%3A621.

SourceFire (2005f). SNMP Request TCP. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=1%3A1418.

SourceFire (2005g). Whisker Tab Splice Attack. Snort Signature Database. Retrieved June 22, 2005 from http://www.snort.org/pub-bin/sigs.cgi?sid=1%3A1087.

Stallings, W. (2003). *Cryptography and Network Security: Principles and Practice.* Prentice Hall Publishing, Third Edition.

Stoll, C. (1989). *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*. Pocket Books.

Stoll, C. (1990). The KGB, the computer, and me. PBS-Nova broadcast on VHS tape. Original broadcast – October 3, 1990.

Stoneburner, G., Goguen, A., & Ferina, A. (2002). Risk Management Guide for Information Technology Systems. National Institute of Standards and Technology Special Publication 800-30, July 2002.

Sundaram, A. (1996). *An Introduction to Intrusion Detection*. ACM Crossroads, Vol. 2, No. 4, 1996.

Symantec (2001). CodeRed Worm. Retrieved June 28, 2005 from http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html.

Symantec (2005). Reporting Incidents for Security's Sake. Retrieved June 22, 2005 from http://www.symantec.com/region/in/smallbiz/library/report.html.

Tan, P.-N., Steinbach, M., & Kumar, V. (2005). Support material for *Introduction to Data Mining*. Retrieved July 5, 2005 from http://www-users.cs.umn.edu/~kumar/dmbook/index.html.

Two Crows Corporation (2005). About Data Mining. Retrieved March 7, 2005 from http://www.twocrows.com/about-dm.htm.

U.S. Government Accountability Office (1989a). Executive Summary of the Morris Internet Worm Incident.

U.S. Government Accountability Office (1989b). GAO/IMTEC 89-57: Virus Highlights Need for Improved Internet Management. June 1989.

Vossen, J. P. (2005). How to modify and write custom Snort rules. Retrieved May 31, 2005 from http://searchsecurity.techtarget.com/tip/0,289483,sid14_gci998669,00.html.