Intrusion Detection System:
Ideas from the Human Immune System

A Thesis Presented to

The Faculty of the Computer Science Program
California State University Channel Islands

In (Partial) Fulfillment
of the Requirements for the Degree of
Master of Science in Computer Science

By
Hassine Letaief
May 2007

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

_____

Advisor: Dr. Andrzej Bieszczad Date

_____

Advisor: Dr. William Wolfe Date

_____

Advisor: Dr. Peter Smith Date

APPROVED FOR THE UNIVERSITY

_____

Advisor: Dr. Gary A. Berg Date

Intrusion Detection System:
Ideas from the Human Immune System

by

Hassine Letaief

Computer Science Program
California State University Channel Islands

Abstract

Security has always been a major issue in computers, even more so with the modern systems and the exponentially growing use of the Internet. A well-planned security policy and the use of antivirus software make a first line of defense. A second line of defense is needed since, as has been proven over the past few years, it is not a question of "if" the first line of defense is going to be crossed by malicious code; it is a question of "when". This second line of defense consists of an Intrusion Detection System (IDS). The more efficient an IDS is at detecting real threats, the more robust the overall security of a computer system is. The Human Immune System provides a rich set of promising theories that could be used to help improve the efficiency of Intrusion Detection Systems. Self/Non-Self theory about how HIS detects foreign and malicious bodies is one area that I plan to explore and implement in an IDS application.

Acknowledgements

I would like to thank Dr Andrzej Bieszczad for his continuous support and great advice without which this work would not have been completed. I am also grateful to my fiancée Sarah for her encouragement, her support, and most of all, the fact that she was the one that introduced me to this great computer science program at CSUCI. I am also very lucky to have a great family that has been always there for me when I needed them most. Thank you all!

And finally I would like to express my deepest gratitude to my friend Kory Wollons for encouraging me to do this work and for being very understanding with my unpredictable schedule at work during the past few weeks.

Table of Contents

Chapter 1

1- Introduction

As computer networks are becoming more and more part of every day's activity of many industries, they play a very helpful role when used correctly and for the purposes they were originally made for. Unfortunately, like many other human inventions, computer networks effect can be devastating if misused or used intentionally to cause harm. This unintended consequence is even more dangerous when network security is vital for the survival and success of an organization. Making a network completely safe from hackers and malicious users is a dream of every network and system administrator.

For several decades researchers have tried to find a way to make a network completely safe. Unfortunately, for many reasons ranging from the lack of security awareness when designing new protocols to the human nature of hoping for the best but never expecting the worst, the goal of a completely safe and secure network was never reached.

Several software packages which are currently available on the market have the sole purpose of protecting computer networks from foreign intrusion as well as from insider sources. An Intrusion Detection System (IDS) is a major element of this rich set of network security software and hardware packages. The efficiency of IDS depends on how reliable it is in detecting possible malicious intrusion. False Positives and False Negatives constitute a hard challenge that faces a network security administrator. Ideas that improve the efficiency of IDS in detecting real threat, thus reducing false positives and false negatives are very valuable in reaching the goal of building a robust, reliable and secure network.

The Human Immune System (HIS) is an area that seems to be very promising to help reach this goal. The efficiency of HIS in detecting real threat has been proven over millions of years that made us resist millions of threats ranging from microbes to viruses to psychological stress. Modeling some techniques used in HIS to implement them in IDS sounds very promising.

In this document I will explore some of those HIS techniques and theories. I will also try to use them in designing and building an application to test and analyze the results and build conclusions on how efficient (or inefficient) the approach of using these techniques, is.

1.1 Human Immune System Theories

There are several theories on how the Human Immune System could be modeled. The following three spearhead the group:

1. Self/Non-Self Theory:

The Human Immune System defends the body by detecting harmful and previously unseen invaders. This is done by learning to discriminate between "self" (the normally occurring patterns in the body) and "non-self" (foreign pathogens, such as bacteria or viruses, or components of self that are functioning abnormally).

2. Immune Network Theory:

The network model suggests that, instead of the immune system being composed of a set of discrete agents that react only when triggered by an antigen, it actually acts as a regulated network of molecules and cells that recognize each other in the absence of antigen. This theory suggests that antibodies recognize each other, thus providing self-regulation of the immune response that could provide both inhibitory (to block and prevent malicious actions) and excitatory (to stimulate defense mechanisms) actions.

3. Danger Theory:

This theory suggests that the immune system reacts to threats based on the existence of various "danger" signals. It generates methods to formulate an immune response to target the intruder. This approach, which suggests that the immune system does not attack foreign molecules whenever it detects them, but only when they start to cause trouble, was first introduced in the early nineties. It is still a debated subject within the immunology community.

1.2 Suggested Research Areas and Goals

One major problem that current Intrusion Detection Systems suffer from is the large amount of false alarms. Such large number of alarms hinder the efforts of System Administrators to keep track of what is going on within their computer systems hence making it very difficult to act early enough to stop or limit damage whenever an intrusion occurs. Exploring ideas that could increase the efficiency of Intrusion Detection Systems to detect real threats and minimize false alarms is a major goal of this research.

In this document I will present some ideas on how to use the immune system models discussed earlier to provide backbone material for an Artificial Immune System that could be used to build an Intrusion Detection System. Furthermore, I will describe my explorations of these models that allowed me to better understand the complicated way our immune system functions. The potential benefits from such research are enormous: if we find a way to model the efficiency of the Human Immune System, a resulting Intrusion Detection System will be as efficient in detecting harmful patterns within a computer system as our immune system is in detecting harmful foreign elements in our bodies.

Chapter 2

2- Background

2.1 Human Immune System

The human immune system is a complex and amazing network of mechanisms, cells, and organs, put together to respond to attacks from foreign bodies that threaten the human body. Defending the body against these attacks can sometimes be an extremely difficult task to accomplish. Pathogens range from viruses to parasitic worms can cause irreversible damage to the human body if not stopped in time. So these diverse threats must be detected with absolute specificity amongst normal cells and tissues. Evolving pathogens constantly develop new ways to avoid detection by the immune system, and therefore improve their chances to successfully infect their hosts. Researchers have identified several techniques used by the immune system to efficiently detect and stop threats. Self/Non-Self recognition is one of these techniques. It is achieved by having every cell display a marker based on Major Histocompatibility Complex (MHC). Any cell not displaying this marker is treated as non-self and attacked **[6].**

There are two main fluid systems in the body: blood and lymph. The blood and lymph systems are intertwined throughout the body and they are responsible for transporting the agents of the immune system.

All blood cells are manufactured by stem cells, which live mainly in the bone marrow, via a process called *hematopoiesis*. The stem cells produce hemocytoblasts that differentiate into the precursors for all the different types of blood cells. Hemocytoblasts mature into three types of blood cells: *erythrocytes* (red blood cells or RBCs), *leukocytes* (white blood cells or WBCs), and *thrombocytes* (platelets).

The leukocytes are further subdivided into *granulocytes* (containing large granules in the cytoplasm) and *agranulocytes* (without granules). The granulocytes consist of neutrophils (55–70%), eosinophils (1–3%), and basophils (0.5–1.0%). The agranulocytes are *lymphocytes* (consisting of B cells and T cells) and *monocytes*. Lymphocytes circulate in the blood and lymph systems, and make their home in the lymphoid organs.

All of the major cells in the blood system are illustrated below (**Figure [2.1A]**).

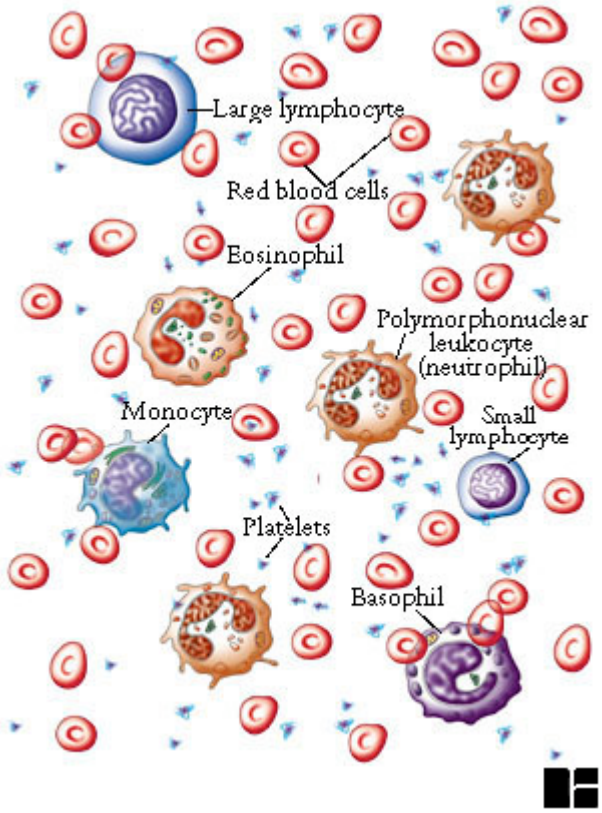


Figure [2.1A]: *Major cells in blood system [6]*

The lymph flows from the interstitial fluid through lymphatic vessels up to either the thoracic duct or right lymph duct, which terminate in the subclavian veins, where lymph is mixed into the blood. (The right lymph duct drains the right sides of the thorax, neck, and head, whereas the thoracic duct drains the rest of the body.) Lymph carries lipids and lipid-soluble vitamins absorbed from the gastrointestinal (GI) tract. Since there is no active pump in the lymph system, there is no back-pressure produced. The lymphatic vessels, like veins, have one-way valves that prevent backflow. Additionally, along these vessels there are small bean-shaped *lymph nodes* that serve as filters of the lymphatic fluid. It is in the lymph nodes where antigen is usually detected by the immune system **[6].**

The human *lymphoid system* consists of the following:

- **Primary organs**: bone marrow (in the hollow center of bones) and the thymus gland (located behind the breastbone above the heart).

- **Secondary organs** (at or near possible portals of entry for pathogens): adenoids, tonsils, spleen (located at the upper left of the abdomen), lymph nodes (along the lymphatic vessels with concentrations in the neck, armpits, abdomen, and groin), Peyer's patches (within the intestines), and the appendix.

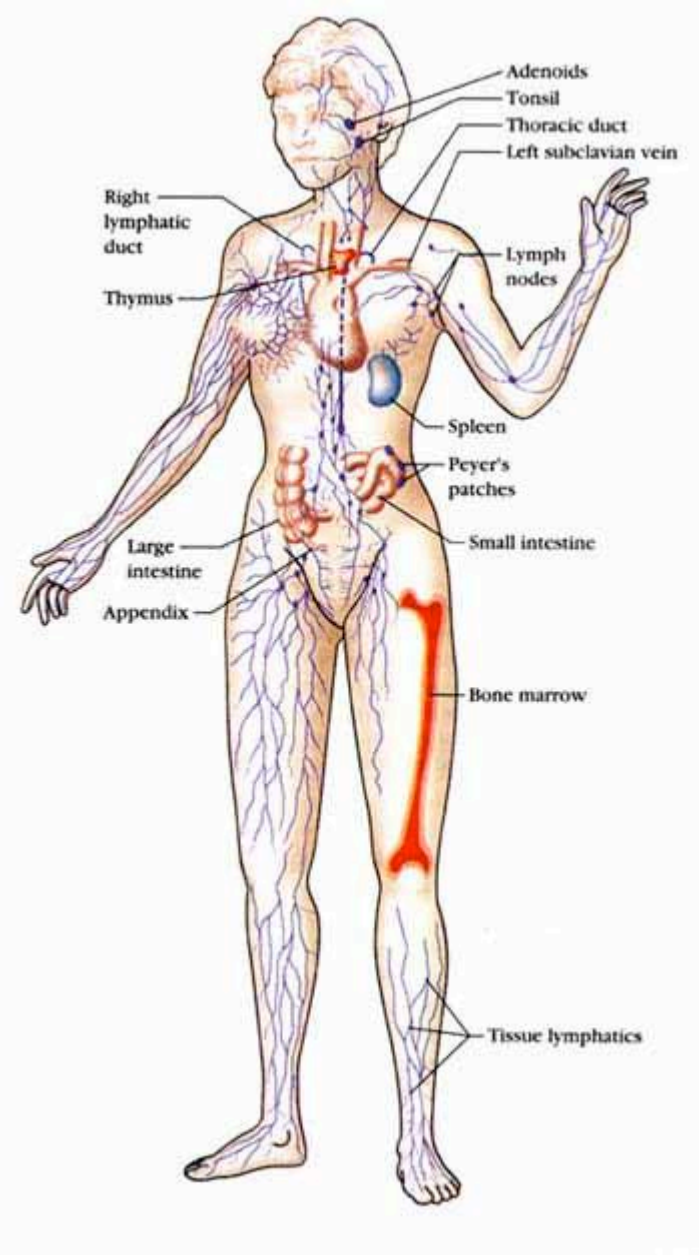


Figure [2.1B]: *Human lymphoid system [6]*

2.1.1 Self Non-Self Theory

The key to a healthy immune system is its remarkable ability to distinguish between the body's own cells (self) and foreign cells (nonself) **[1]**. The body's immune defenses normally coexist peacefully with cells that carry distinctive "self" marker *molecules*. But when immune defenders encounter cells or organisms carrying markers that say "foreign" they quickly launch an attack.

Anything that can trigger this *immune response* is called an *antigen*. An antigen can be a microbe such as a virus, or even a part of a microbe. Tissues or cells from another person (except an identical twin) also carry nonself markers and act as antigens. This explains why tissue transplants may be rejected.

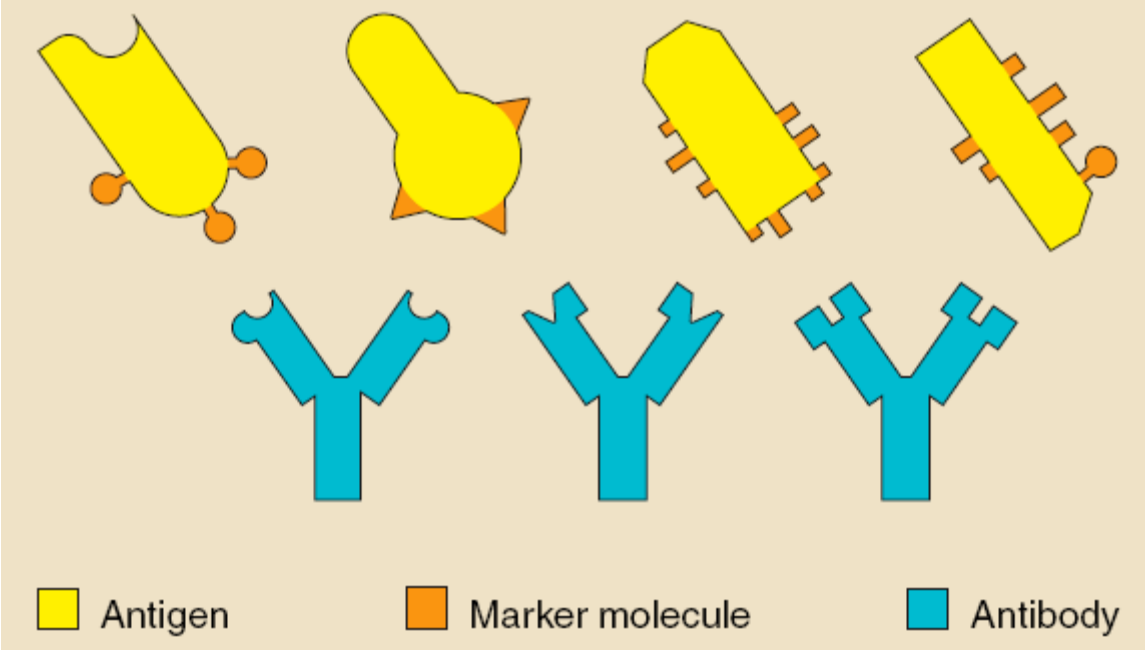


Figure [2.1.1A]: *Antigens carry marker molecules that identify them as foreign [1].*

In abnormal situations, the immune system can mistake self for nonself and launch an attack against the body's own cells or tissues. The result is called an *autoimmune disease*. Some forms of arthritis and diabetes are autoimmune diseases. In other cases, the immune system responds to a seemingly harmless foreign substance such as ragweed pollen. The result is allergy, and this kind of antigen is called an *allergen*.

2.1.2 The Danger Theory (DT)

In his DT based IDS research U. Aickelin et al **[2]** has examined the biological basis for the Self/Non-Self metaphor, and the alternative Danger Theory hypothesis. The Human Immune System (HIS) is commonly thought to work at two levels: innate immunity including external barriers (skin, mucus), and the acquired or adaptive immune system. As part of the latter level, B-Lymphocytes secrete specific antibodies that recognize and react to stimuli. It is this matching between antibodies and antigens that lies at the heart of the HIS and most Artificial

Immune System (AIS) implementations. The central tenet of the immune system is the ability to respond to foreign invaders or 'antigens' whilst not reacting to 'self' molecules. In order to undertake this role the immune system needs to be able to discern differences between foreign, and possibly pathogenic, invaders and non-foreign molecules. It is currently believed that this occurs through the utilization of the Major Histocompatability Complex (MHC). This complex is unique to each individual and therefore provides a marker of 'self'. In addition, the cells within the immune system are matured by becoming tolerised to self-molecules. Together, through the MHC and tolerance, the HIS is able to recognize foreign invaders and send the requisite signals to the key effector cells involved with the immune response.

The DT debates this and argues that there must be discrimination happening that goes beyond the self-nonself distinction because the HIS only discriminates 'some self' from 'some nonself'. It could therefore be proposed that it is not the 'foreignness' of the invaders that is important for immune recognition, but the relative 'danger' of these invaders. This theory was first proposed in 1994 by Polly Matzinger **[3]** to explain current anomalies in our understanding of how the immune system recognizes foreign invaders. For instance, there is no immune reaction to foreign bacteria in the gut or to food. Conversely, some auto reactive processes exist, e.g. against self-molecules expressed by stressed cells. Furthermore, the human body (self) changes over its lifetime. Therefore, why do defenses against nonself learned early in life not become auto-reactive later?

The DT suggests that foreign invaders, which are dangerous, will induce the generation of cellular molecules (danger signals) by initiating cellular stress or cell death. These molecules are recognized by the Antigen Presenting Cells (APCs), critical cells in the initiation of an immune response, which become activated leading to protective immune interactions. Overall there are two classes of danger signal: those which are generated endogenously, i.e. by the body itself, and exogenous signals which are derived from invading organisms, e.g. bacteria. Evidence is accruing as to the existence of myriad endogenous danger signals including cell receptors, intracellular molecules and cytokines. A commonality is their ability to activate APCs and thus drive an immune response.

2.2 Other Immune System Theories

2.2.1 Specificity is predetermined

In his theory, Niels K. Jerne **[8]** explains the development of a specific antibody response in the following way: Each individual has a large number of natural antibodies with specificities for all antigens towards which the individual can respond. These antibodies develop already during fetal life in the absence of external antigens. The foreign antigen then selects the antibody molecule which has the best fit. The antigen-antibody binding stimulates the production of this particular antibody specificity.

Jerne's natural-selection theory contrasted to the dogmatic views of the antibody response as formulated in the instruction theories which were prevailing at that time. According to these theories the antigen serves a template for the production of antibodies. In Jerne's natural selection theory it is implied that the generation of the enormous number of antibody

specificities is independent of exogenous antigens. This view on the nature of the immune system constitutes the basis for modern immunology.

2.2.2 Reactivity against self-antigens creates diversity

The natural-selection theory is mainly concerned with the maturation of the immune system after it has acquired the ability to react with antigen. In his second theory **[9]** Jerne explains how the immune system develops from stem cells to mature lymphocytes which can react to antigen. He suggests that every individual possesses all genes needed for the production of antibodies, and antibody-like molecules, which can bind all strong transplantation antigens of the species. Jerne also suggests that lymphocytes mature in the thymus gland and in other lymphoid organs where they are exposed to the transplantation antigens of the individual. Cells which recognize the antigens are stimulated and enter cell division.

As mutations accumulate in rapidly dividing cells new immunological specificities may develop. At the same time the specificities of the lymphocytes for self transplantation antigens are weakened. The mature lymphocytes will recognize foreign antigen associated with transplantation antigens. The theory explains how the immune system normally matures through the influence of self antigens. It also offers an explanation for the regulation of immunological specificity by genes belonging to the transplantation system.

2.2.3 The Network theory

In his third main theory **[10]**, Jerne explains how the specific immune response is regulated. The theory has greatly stimulated research and led to new insights into the immune system. Recently its principles have been applied to diagnosis and treatment of disease. A basis for the network theory was the observation that antibodies can elicit anti-antibodies directed against antigen binding structures on the first antibody. Moreover, anti-antibodies can stimulate the production of still another generation of antibodies, anti-anti-antibodies. Essentially, this antibody cascade is endless successively adding new specific properties to the immune system. The various antibody generations either stimulate or suppress the production of one another. Under normal conditions the network is balanced. When an antigen is introduced the equilibrium is disturbed. The immune system tries to restore balance which leads to an immune response against the antigen.

Chapter 3

3- Intrusion Detection System

3.1 Definitions of Intrusion Detection System

1- Software/hardware that detects and logs inappropriate, incorrect, or anomalous activity. Intrusion Detection Systems are typically characterized based on the source of the data they monitor: host or network. A host-based IDS uses system log files and other electronic audit data to identify suspicious activity. A network-based IDS uses a sensor(s) to monitor packets on the network to which it is attached **[13].**

2- An Intrusion Detection System is a software/hardware tool used to detect unauthorized access to a computer system or network. This may take the form of attacks by skilled malicious hackers, or Script kiddies using automated tools **[14]**.

3- The National Institute of Standards and Technology classifies Intrusion Detection as the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network. This definition captures the essence of intrusion detection but fails to address the methods by which Intrusion Detection Systems automate this process. The concepts of false positive and false negative are essential to this classification process. False positives are those sequences of innocuous events that an IDS erroneously classifies as intrusive, while false negatives refer to intrusion attempts that an IDS fails to report: the reduction of both false positives and false negatives is a critical objective in intrusion detection **[12]**.

3.2 Intrusion Detection System: A Brief History [4]

- Beginning in 1980, with James Anderson's paper, Computer Security Threat Monitoring and Surveillance, the notion of intrusion detection was born.
- In 1983, SRI International, and specifically Dr. Dorothy Denning, began working on a government project that launched a new effort into intrusion detection development.
- One year later, Dr. Denning helped to develop the first model for intrusion detection, the Intrusion Detection Expert System (IDES), which provided the foundation for the IDS technology development that was soon to follow.

- In 1988, the Haystack project at University of California Davis' Lawrence Livermore Labs released another version of intrusion detection for the US Air Force. This project produced an IDS that analyzed audit data by comparing it with defined patterns.
- In 1990, Heberlein was the primary author and developer of Network Security Monitor (NSM), the first network intrusion detection system (see Heberlein, L. et al. "A Network Security Monitor."
- Commercial development of intrusion detection technologies began in the early 1990s. Haystack Labs was the first commercial vendor of IDS tools, with its Stalker line of host-based products.
- The intrusion detection market began to gain in popularity and truly generate revenues around 1997. In that year, the security market leader, ISS, developed a network intrusion detection system called RealSecure.
- A year later, Cisco recognized the importance of network intrusion detection and purchased the Wheel Group, attaining a security solution they could provide to their customers. Similarly, the first visible host-based intrusion detection company, Centrax Corporation, emerged as a result of a merger of the development staff from Haystack Labs and the departure of the CMDS team from SAIC. From there, the commercial IDS world expanded its market-base and a roller coaster ride of start-up companies, mergers, and acquisitions ensued.
- Currently, market statistics show that IDS is amidst the top selling security vendor technologies and should continue to rise. Furthermore, government initiatives, such as the Federal Intrusion Detection Network, (FIDNet) created under Presidential Decision Directive 63 by former president Bill Clinton, are also adding impetus to the evolution of IDS. Advancements in IDS will ultimately push security technology into a whole new arena of automated security intelligence.

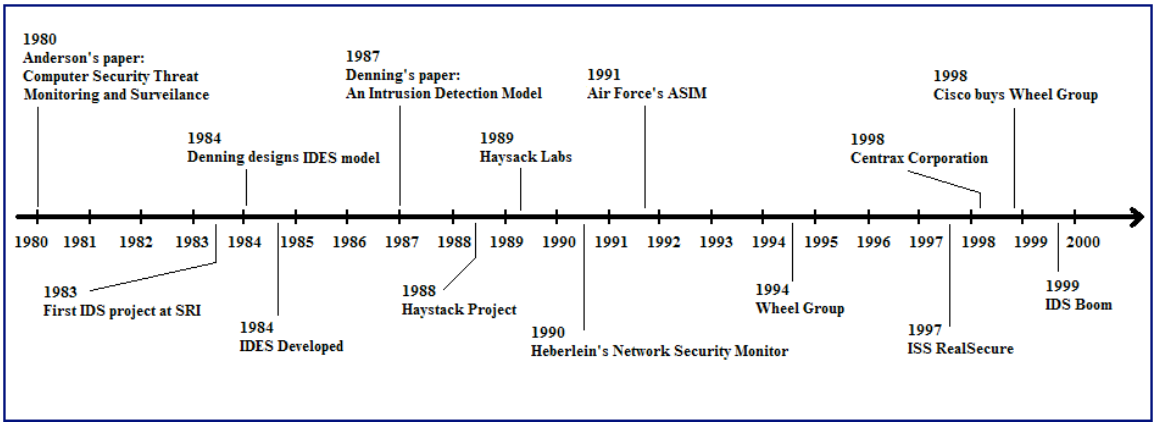


Figure [3.2A] *History of Intrusion Detection System.*

3.3 Different types of Intrusion Detection Systems

According to U. Aickelin **[5]** there are several ways to categorize an IDS. One way to do this is to focus on the analysis approach and another is to focus on the placement of the IDS. The former leads to two main classes: misuse detection and anomaly detection. The misuse detection approach examines network and system activity for known misuses, usually through some form of pattern-matching algorithm. In contrast, anomaly detection approach bases its decisions on a profile of normal network or system behavior, often constructed using statistical or machine learning techniques.

Each of these approaches offers its own strengths and weaknesses. Misuse-based systems generally have very low false positive rates but are unable to identify novel or obfuscated attacks, leading to high false negative rates. Anomaly-based systems, on the other hand, are able to detect novel attacks but currently produce a large number of false positives. This stems from the inability of current anomaly-based techniques to cope adequately with the fact that in the real world normal, legitimate computer network and system usage changes over time, meaning that any profile of normal behavior also needs to be dynamic.

A second distinction can be made in terms of the placement of the IDS. In this respect IDSs are usually divided into host-based and network-based systems. Host-based systems are present on each host that requires monitoring, and collect data concerning the operation of this host, usually log files, network traffic to and from the host, or information on processes running on the host. In contrast, network-based IDSs monitor the network traffic on the network containing the hosts to be protected, and are usually run on a separate machine termed a sensor. Once again, both systems offer advantages and disadvantages.

Host-based systems are able to determine if an attempted attack was indeed successful, and can detect local attacks, privilege escalation attacks and attacks which are encrypted. However, such systems can be difficult to deploy and manage, especially when the number of hosts needing protection is large.

Furthermore, these systems are unable to detect attacks against multiple targets of the network. Network-based systems are able to monitor a large number of hosts with relatively little deployment costs, and are able to identify attacks to and from multiple hosts. However, they are unable to detect whether an attempted attack was indeed successful, and are unable to deal with local or encrypted attacks. Hybrid systems, which incorporate host- and network-based elements, can offer the best protective capabilities, and systems to protect against attacks from multiple sources are also under development.

3.4 Intrusion Detection System Functional Blocks

In general any IDS falls within the following three different categories **[11]**:

- **Monolithic**: This has a single application acting as complete IDS. Here the architecture is very simple but at the same time this approach can not identify attacks made by distributed events which appear to be normal when analyzed with respect to one single system.
- **Hierarchic**: This architecture has evolved in order to identify distributed attack made in network environment. Here IDS's centralized module co-relates data available from different nodes and analyzes the results to detect any intrusion.
- **Agent Based**: It is purely distributed in nature. There is no centralized control over analysis and data source. This architecture is highly scalable and adapted by many modern IDSs.

Intrusion Detection Systems have evolved from monolithic batch-oriented systems to distributed real-time network of components. In recent systems, a number of common functional building blocks can be distinguished:

- **Sensors**: These modules form the most primitive data gathering components of an IDS. They track network traffic, log files or system behavior - translating raw data into events usable by IDS monitors.
- **Monitors**: They are the main processing components of an IDS, they receive events from Sensors. These events are then correlated against the IDS behavior models, potentially producing model updates and alerts. Alerts, events in themselves, indicate occurrence significant to the security of a system, and may be forwarded to higher level monitors or Resolvers.
- **Resolvers**: Resolver components receive suspicious reports from Monitors and determine the appropriate response - logging, changing the behavior of lower level components, reconfiguring security mechanisms and notifying operators.
- **Controllers**: They are facilitating component configuration and coordination and are most significant in distributed IDS, where manually starting, updating and configuring network wide series of components is almost impossible. In addition controllers provide single point of administration for an IDS.

3.5 Problems with Intrusion Detection Systems

3.5.1 Deriving an Expert Rule Set

One drawback of misuse detection systems is their reliance on an expert rule set that traditionally must be constructed by a human domain expert. This rule set is therefore expensive to produce and susceptible to human error **[12]**.

Snort, a real-time NIDS, addresses this conundrum by minimizing the effort required to develop new attack rules. In Snort, each attack rule is a single line of text that specifies exactly which characteristics of a packet are to be examined and what values these characteristics must equal in order to trigger the rule. This approach to the problem is helpful, but is limited in its ability to detect variations of codified attacks and does not resolve the issue of requiring a human expert to devise a knowledge base for the IDS.

A technique that allows for the automated construction of attack rules would therefore be tremendously valuable. The architecture for an artificial immune system (ARTIS) that Hofmeyr and Forrest [15] have proposed takes a bold step in this direction. Based on a model of the human immune system, ARTIS is concerned with developing a set of lymphocytes (analogous to attack rules) that can detect pathogens (intrusive activity) in a system. In an attempt to closely model the way this is performed in the human immune system, ARTIS continuously generates lymphocytes with random characteristic values and deploys these to detectors that are replicated throughout the system. Over time, lymphocytes randomly die off and are replaced; when a particular lymphocyte correctly binds to a particular event sequence in the system, it is reinforced and its probability of being replaced is reduced. In this fashion, those lymphocytes that are most effective in detecting misuse become more prevalent in the system.

This design is advantageous in that it eliminates much of the dependency on expert human knowledge in developing a misuse detection system. In reality, however, ARTIS does require human intervention in distinguishing the lymphocytes that bind correctly from those that do not in order to reduce the number of false positives that the system produces. Because lymphocytes are constructed at random, there is also a high probability that well-known attacks may not be detected at all if the characteristics needed to identify these attacks are never generated. ARTIS achieves one of the benefits of anomaly systems in the ability to detect new attacks without relying on a human-encoded expert knowledge base. It does this; however, at the expense of some of the advantage of traditional misuse systems, namely it is less effective at detecting known attacks and is more likely to produce false positives.

3.5.2 Detecting attack variations

Detecting subtle variations of known attacks presents a sizeable challenge to many systems that rely upon misuse detection **[12]**. Because of the way intrusion signatures must be codified into an expert knowledge base, it can be very difficult for misuse systems to identify attacks that may originate from more than one source, vary in the means by which they are conducted, or are protracted over long periods of time. State transition analysis (STAT) is one technique that addresses this issue. In such a system, attacks are represented by a state transition diagram: the start state represents a pristine system, intermediate states represent changes to the system that occur during an attack, and the final state represents a system compromise. For all of the codified attacks that exist in an IDS's rule base, the system retains internal data indicating which states of the attack have been reached. Because this data is global, it is independent of who causes a state change, the way in which a new state is reached, and the time scale on

which state transitions occur. Through this means, countless variations of a single attack can still be detected because the system monitors system state changes that are symptomatic of an intrusion attempt rather than monitoring the actions that cause those state changes.

This advantage is not gained, however, without considerable expense. Because the occurrence of state transitions in an attack may not be a simple linear progression, the system must maintain internal data for every intermediate state of an attack that has ever been reached. This can lead to an explosion in the amount of data that the system must maintain as the number of attacks and attack states monitored becomes large, a property that an attacker could exploit. Codifying attacks as state transition diagrams also complicates the process of developing the system's expert knowledge base: if a critical attack state is omitted from the diagram, false positives result when state progressions resembling an attack occur; if a non-critical state is inserted into an attack's diagram, false negatives occur when variations of the attack transpire that do not instantiate this non-critical state.

Colored Petri Nets: The technique belongs to a branch of mathematics called graph theory. A Petri net may be represented graphically as well as mathematically. The ability to visualize structure and behavior of a Petri net promotes understanding of the modeled system. Petri Nets is a formal and graphical appealing language which is appropriate for modeling systems with concurrency and resource sharing. Petri Nets has been under development since the beginning of the 60'ies, where Carl Adam Petri defined the language. It was the first time a general theory for discrete parallel systems was formulated. The language is a generalization of automata theory such that the concept of concurrently occurring events can be expressed

A similar tactic to the problem of detecting attack variations makes use of Colored Petri Nets to monitor codified event sequences. Colored Petri Nets allow for attacks to be encoded as graphs in such a way that the progression of attack steps is more flexible than the rigid order imposed by state transition analysis. Guards are used to define the situations in which attack signatures are matched rather than maintaining data indicating all current and previous states. This helps to achieve many of the benefits of using the STAT approach without making the system vulnerable to the state-consuming attacks.

Unfortunately, Colored Petri Nets do not present a viable option for practical intrusion detection because the process of state unification and matching in this model is prohibitively compute-expensive. The complexity of this matching is exponential on the number of states represented by the system, while partial order matching requires super-exponential time. Although an IDS based on Colored Petri Nets would not be vulnerable to a state-consuming attack, it would be very vulnerable to a CPU consuming attack. The efficiency of such a system would therefore be rapidly eroded in a production environment in which an attacker could render the IDS useless by overwhelming the analysis engine.

3.5.3 Training Behavioral Models

Anomaly systems universally suffer from the problem of how to correctly construct a baseline model of behavior that is sufficient for complete and correct operation of the system. Any successful means of training the system must expose the system to the full range of normal

behavior **[12]** in order to minimize false positives as well as avoid exposing the system to properties of anomalous activity that may desensitize the IDS to attacks.

Depending on the design of the anomaly system, systems can be trained with just normal data or with two sets of data that are correctly identified as normal and intrusive. In presenting intrusive data that is going to be used to train the system, it is important that this data represent a range of anomalies so that the trained system is not incapable of identifying certain classes of attack.

Ghosh et al. **[19]** have experimented with using randomly generated events to represent anomalous behavior in order to train the neural networks that provide the analysis for their IDS. In empirical tests, those networks that were trained with randomly generated anomaly data consistently out-performed those that did not receive this training by reducing the number of false negatives in the system (none of the networks produced any false positives). While these results are very promising, they suggest that a likely reason that the neural networks trained with random data performed well is because the normal data set with which they were trained defined a narrow range of behavior that very closely resembled the normal data that the system was tested against. Were this not the case, the system could reasonably have been expected to generate at least a small number of false positives as some of the randomly generated events trained as anomalies would fall into the range of normal activity. It is further unclear how well a system trained over random anomalous data would perform in correctly identifying actual attacks, as the attack data against which this system was tested also consisted of randomly generated events.

A more ideal solution to the training problem would be one that allows for an anomaly system to be correctly trained over noisy data, i.e. data that contains an assortment of both normal and anomalous behavior. This would allow for the system to be effectively trained in a production environment without relying on hypothetical data sets representing normal and anomalous behavior.

Eleazar Eskin **[18]** has developed a process that uses learned probability distributions to train an anomaly system over noisy data. This technique uses machine learning to create a probability distribution of the training data and then applies a statistical test to identify anomalies. Interestingly, Eskin's technique requires no domain-specific knowledge. It does, however, operate on three assumptions about the training data: normal data can be effectively modeled using a probability distribution; anomalous events differ significantly enough from normal events that they can be identified; and the number of anomalous events is small compared to the number of normal events. Furthermore, before the system is trained, one must define a value λ indicating the percentage of the training data that is expected to be anomalous. Because this data is noisy and not artificially constructed, choosing the value of λ to best ensure correct operation of the system is very difficult. This model is additionally limited by its assumption that normal data is distributed normally across noisy data: in reality, it is likely that intrusion attempts are sometimes clustered.

Regardless of the means by which one is trained, there is also the issue of evolving normal behavior with which anomaly systems must contend. One option is that a system be retrained periodically with new training data that represents current normal behavior. This, however, increases the dependency of the system on training and underscores the inherent difficulties in developing sufficient data or an appropriate technique for this task.

Adaptive anomaly systems have been suggested as a solution that would allow for systems to evolve their normal behavior models gradually as normal behavior evolves. Lane and Brodley's **[20]** machine learning system is an example of a system that makes use of this technique. This system maintains a finite-sized dictionary of normal event sequences and uses a least- recently-used (LRU) policy to replace seldom-occurring sequences with new ones that are determined to be normal. It is important to note that systems that use adaptive training techniques face the problem of preventing an attacker from gradually training the system over time to accept a range of anomalous behavior as normal. Resolving this difficulty remains an open challenge

3.5.4 Attack against the IDS

While the purpose of an intrusion detection system is to detect attacks against a host or set of hosts, an ironic consequence of its existence is that the IDS itself may draw attack from an attacker seeking to disable the IDS **[12]**. It is critical that the design of a system be performed within the framework that the IDS itself be resistant to and tolerant of attack attempts designed to obstruct its ability to correctly detect intrusions. One class of such attacks referred to as "crash attacks" attempts to disable an IDS by causing it to fault or to run out of some critical resources. Assuming that it is infeasible to totally prevent these attacks, the goal of an IDS in the face of such an attack is therefore to minimize the extent to which the attacker is successful in disabling the IDS.

Another class of attacks seeks simply to inject a large quantity of fake data into a monitor's event stream in order to distract an IDS while an attack on a monitored host takes place. Such attacks can be particularly lethal when launched against NIDS that is already under the stress of monitoring and performing analysis on data for a large number of hosts. In the face of attacks like these, Vern Paxson **[16]** suggests that such systems perform "triage" against incoming flows: if the system detects that it is nearing exhaustion, it can shed load by discarding state for monitored flows that do not appear to be making progress. This suggestion operates under the assumption that an attacker is less likely to have complicity from hosts on both sides of the monitor, therefore making it difficult for the attacker to fake a large number of active connections.

The idea of triage is a risky one. On the one hand, the load that is shed during triage can allow for IDS to operate continuously, helping to maximize the coverage of the system and prevent an attacker from denying service to the system by overwhelming it with illegitimate data. On the other hand, when a system enters a mode of triage, it is in essence performing denial of

service on itself. The efficiency of a triage mechanism therefore relies on the system's ability to properly determine which data it can safely ignore and which data it cannot: if the system were able to make this distinction perfectly, however, there would never be any need to examine the irrelevant data.

Chapter 4

4- Network Tools

4.1 Using the Wireshark Network Analyzer

Gerald Combs, the creator of Ethereal **[21]**, has initiated the Wireshark network protocol analyzer project, a successor to Ethereal. The Ethereal core developer team has moved with Gerald to the Wireshark project. It is the world's most popular network protocol analyzer. It has a rich and powerful feature set, and runs on most computing platforms including Windows, OS X, and Linux. It is freely available as open source, and is released under the GNU General Public License.

4.1.1 Libpcap File Format (.pcap)

The libpcap file format **[22]** is the main capture file format used in TcpDump/WinDump, Wireshark/TShark, snort, and many other networking tools.

Overview

This file format is a very basic format to save captured network data. As the libpcap library became the "de facto" standard of network capturing on UNIX, it became the "common denominator" for network capture files in the open source world (there seems to be no such thing as a "common denominator" in the commercial network capture world at all).

Libpcap and it's Windows equivalent, WinPcap, use the same file format.

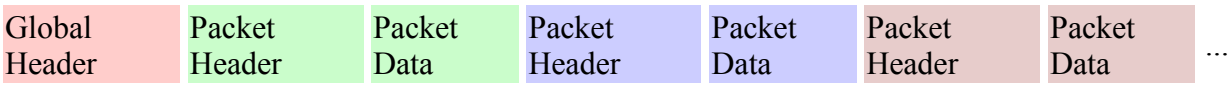
Although it's sometimes assumed that this file format is suitable for Ethernet networks only, it can serve many different network types, examples can be found at the Wireshark's Supported Capture Media page **[21]**; all listed types are handled by the libpcap file format.

The proposed file extension for libpcap based files is: *.pcap*

File Format

There are some variants of the format "in the wild", the following will only describe the commonly used format in its current version 2.4. This format version hasn't changed for quite a while (at least since libpcap 0.4 in 1998).

The file has a global header containing some global information followed by zero or more records for each captured packet, looking like this:



Global Header

- This header starts the libpcap file and will be followed by the first packet header:

```
typedef struct pcap_hdr_s {
        guint32 magic_number;   /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;       /* GMT to local correction */
        guint32 sigfigs;        /* accuracy of timestamps */
        guint32 snaplen;        /* max length of captured packets, in octets
        guint32 network;        /* data link type */
} pcap_hdr_t;
```

- magic_number: used to detect the file format itself and the byte ordering. The writing application writes 0xa1b2c3d4 with its native byte ordering format into this field. The reading application will read either 0xa1b2c3d4 (identical) or 0xd4c3b2a1 (swapped). If the reading application reads the swapped 0xd4c3b2a1 value, it knows that all the following fields will have to be swapped too.
- version_major, version_minor: the version number of this file format (current version is 2.4)
- thiszone: the correction time in seconds between GMT (UTC) and the local time zone of the following packet header timestamps. Examples: If the timestamps are in GMT (UTC), thiszone is simply 0. If the timestamps are in central European time (Amsterdam, Berlin ...) which is GMT + 1:00, thiszone must be -3600. In practice, time stamps are always in GMT, so thiszone is always 0.
- sigfigs: in theory, the accuracy of time stamps in the capture; in practice, all tools set it to 0

• snaplen: the maximum size of each packet (typically 65535 or even more, but might be limited by the user), see: *incl_len* vs. *orig_len* below.

• network: data link layer type, this can be various types like Token Ring, FDDI, etc.

Record (Packet) Header

• Each captured packet starts with (any byte alignment possible):

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;     /* timestamp seconds */
        guint32 ts_usec;    /* timestamp microseconds */
        guint32 incl_len;   /* number of octets of packet saved in file */
        guint32 orig_len;   /* actual length of packet */
} pcaprec_hdr_t;
```

• ts_sec: the date and time when this packet was captured. This value is in seconds since January 1, 1970 00:00:00 GMT; this is also known as a UNIX time_t. If this timestamp isn't based on GMT (UTC), use *thiszone* from the global header for adjustments.

• ts_usec: the microseconds when this packet was captured, as an offset to *ts_sec*. Beware: this value shouldn't reach 1 second (1 000 000), in this case *ts_sec* must be increased instead!

• incl_len: the number of bytes actually saved in the file. This value should never become larger than *orig_len* or the *snaplen* value of the global header.

• orig_len: the length of the packet "on the wire" when it was captured. If *incl_len* and *orig_len* differ, the actually saved packet size was limited by *snaplen*.

Packet Data

• The actual packet data will immediately follow the packet header as a data blob of *incl_len* bytes without a specific byte alignment.

4.1.2 Wireshark Interface

Wireshark (formerly known as Ethereal) is a great open source network analyzer tool that is available in several platforms including Windows and Linux. It is very useful in capturing and filtering network traffic at the Data-link layer (Network Layers: **Section [4.2]**). I used Wireshark to capture SMTP traffic to be used as data sample for testing S.F.A.M (Chapter 6).

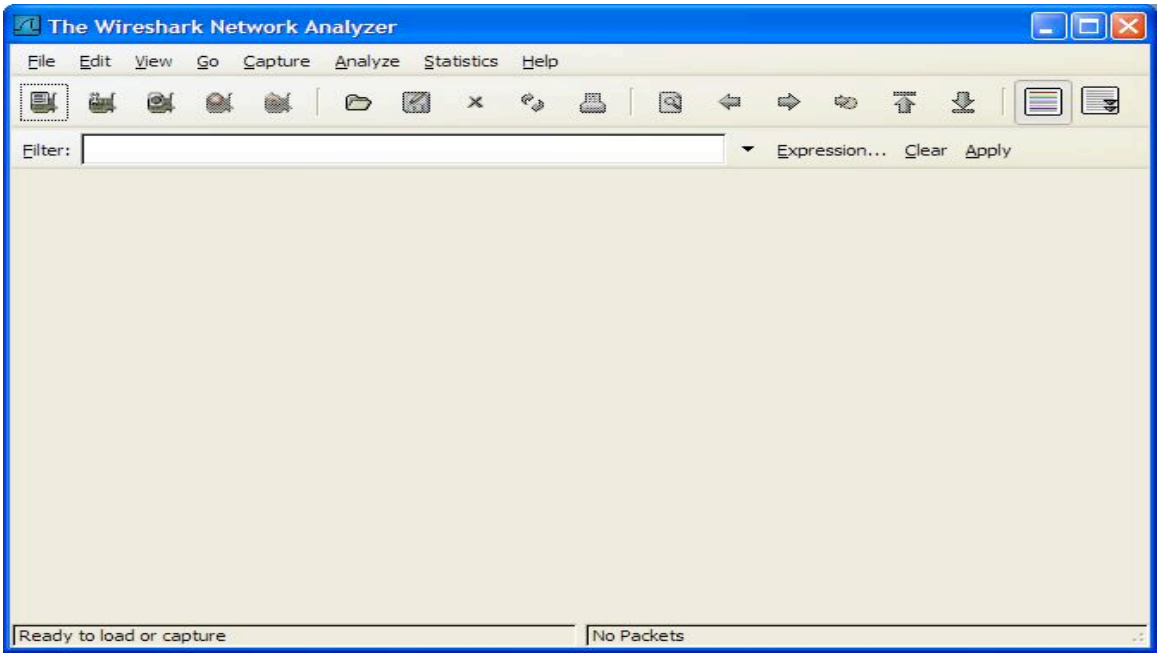


Figure [4.1.2A] *Wireshark main menu.*

Several filters are available when using Wireshark. This made it very easy for me to filter through a wide variety of network packets and capture only the ones that are relevant to my application. Specifically, I used a filter to capture email traffic (SMTP protocol). The following screen capture (Figures **[4.1.2B]**) shows the menu item [Options] which is used to choose the filter(s) settings. Figure **[4.1.2C]** shows the actual filter used to capture SMTP traffic (Notice the "Capture Filter" option set to "tcp port 25" which the default port number used by the email protocol SMTP).

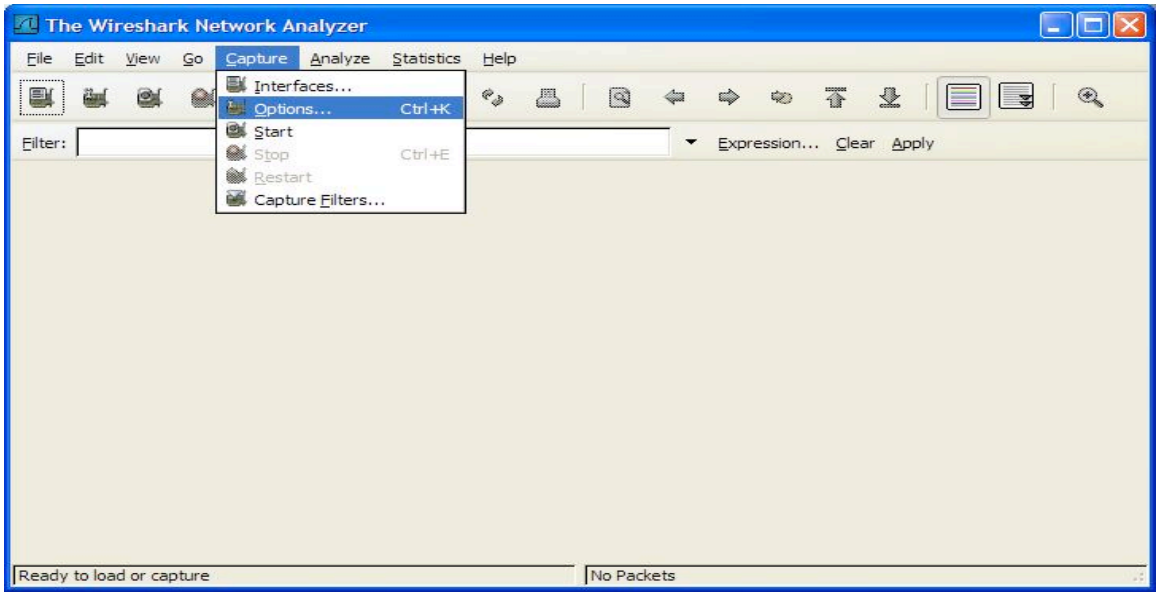


Figure [4.1.2B] *Menu item [Options] used to set several filter options including filter capture settings.*

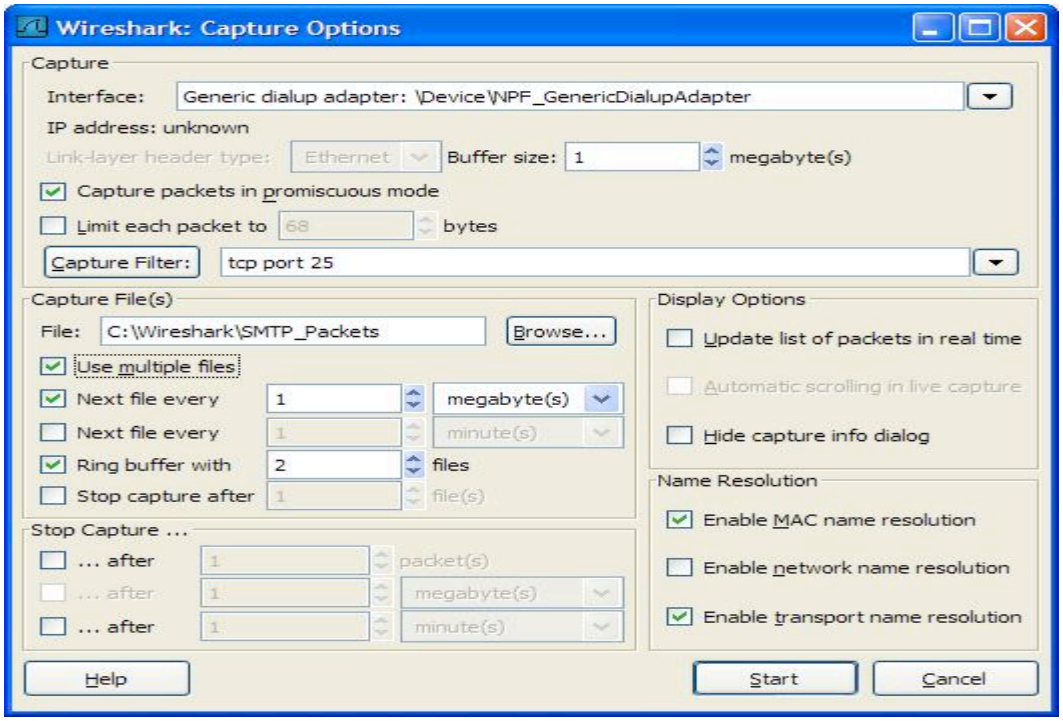


Figure [4.1.2C] *Options screen: used among other things to set the capture filter to "tcp port 25" to capture SMTP traffic only.*

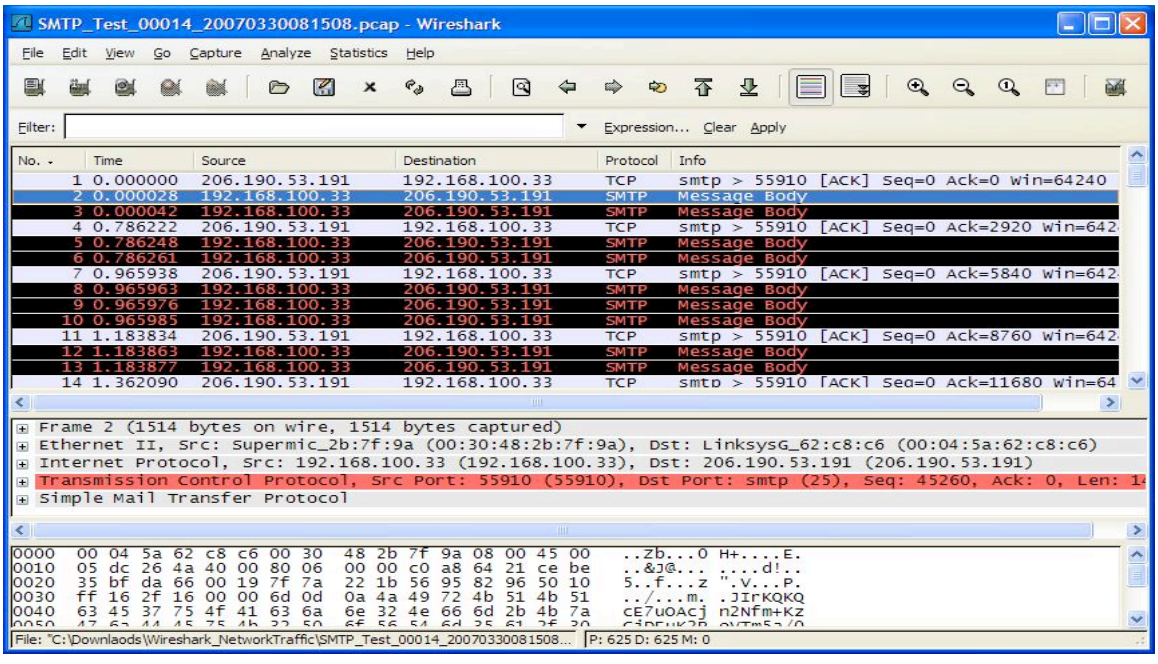


Figure [4.1.2D] *Screen showing a list of captured network packets with several properties identifying each packet (Source IP, Destination IP, Protocol, etc.).*

Wireshark supports several file formats to save captured network traffic. The file format ".pcap" is the default one. It is also possible to export captured packets into several other formats such as "Plain Text", "PostScript", "CVS" and XML (**Figure [4.1.2E]**).

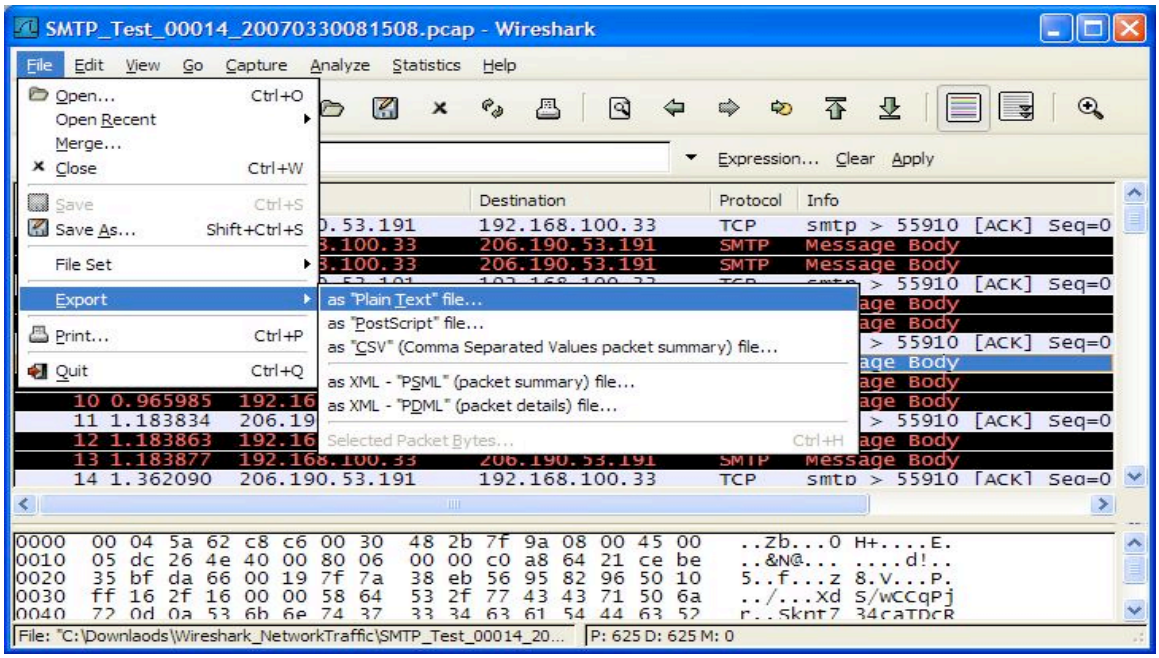


Figure [4.1.2E] *Export captured packets into several formats (Text, PostScript, etc.)*

4.2 Network Layers (ISO/OSI Model)

Layer 7 – Application: It provides network services to the end-users.

Layer 6 – Presentation: It converts local representation of data to its canonical form and vice versa. The canonical uses a standard byte ordering and structure packing convention, independent of the host.

Layer 5 – Session: It defines the format of the data sent over the connections.

Layer 4 – Transport: It divides user-buffer into network-buffer sized datagrams and chooses transmission control. It uses two protocols (TCP or UDP). TCP establishes connections between two hosts on the network through 'sockets' which are determined by the IP address and port number. It keeps track of the packet delivery order and the packets that must be re-sent. Maintaining this information for each connection makes TCP a stateful protocol. UDP on the other hand provides a low overhead transmission service, but with less error checking.

Layer 3 – Network: It is responsible for routing, directing datagrams from one network to another. The network layer may have to break large datagrams into smaller packets and the host receiving the packet will have to reassemble the fragmented datagram.

Layer 2 - Data Link: It defines the format of data on the network. A network data frame, also referred to as packet, includes checksum, source and destination address, and data. The largest packet that can be sent through a data link layer defines the Maximum Transmission Unit

(MTU). The data link layer handles the physical and logical connections to the packet's destination, using a network interface.

Layer 1 – Physical: It defines the cable or physical medium itself.

4.3 Known Protocols: (Five Network Layers (TCP/IP model)

The following is a list of common protocols used at each network layer:

Application layer (5)

DHCP • DNS • FTP • HTTP • IMAP4 • IRC • NNTP • XMPP • MIME • POP3 • SIP • SMTP • SNMP • SSH • TELNET • BGP • RPC • RTP • RTCP • TLS/SSL • SDP • SOAP • ... • IrDA

Transport layer (4)

TCP • UDP • DCCP • SCTP • GTP • ...

Network layer (3)

IP (IPv4 • IPv6) • IGMP • ICMP • RSVP • IPsec • ...

Data link layer (2)

802.11 • ATM • DTM • ***Ethernet*** *• FDDI • Frame Relay • GPRS • PPP • ARP • RARP • L2TP • PPTP • ...*

Physical layer (1)

Ethernet physical layer • ISDN • Modems • PLC • SONET/SDH • G.709 • ...

Chapter 5

5- Approach

One of the main problems that network and system administrators have to deal with on a daily basis is the large flow of unsolicited email (spam) directed towards their email servers. That huge amount of 'useless' email uses resources that could have been reserved for other useful tasks. Solving this issue would save a lot of resources and would make the job of administrators much easier.

Spam filters which are becoming very common nowadays, are in the frontline of defense against spam. In most cases a spam filter consists of software installed on a server which receives all email traffic. The job of the spam filter is to scan all email sent to a network and split it into 'clean' email which is then, forwarded to the email server(s) and spam which is quarantined until further analysis by a system administrator or a regular user.

Like many other network security related software, spam filters suffer from the lack of guaranteed efficiency. In other words, spam filters are usually marketed as a certain percentage accurate of detecting real spam. For example numbers ranging from 90% to 99% accuracy in detecting spam is a common description used to advertise spam filters **[24]**. This results in False Negatives (FN) and False Positives (FP). A false negative happens when a spam email passes the spam filter as 'clean' and thus forwarded to the server than to the user who discovers that the email is actually spam. A false positive on the other hand is much problematic because it happens when a 'clean' email is flagged by the spam filter as a spam email and thus quarantined. That email which could be a very important one might never be delivered without a human intervention.

Several techniques such as Bayesian filtering proposed by Sahami et al. (1998) are used in modern spam filters to make them as efficient as possible. The general idea is that some words occur more frequently in known spam email, and other words occur more frequently in legitimate email. Using Bayes's **[26]** mathematical theorem, it is possible to generate a "spam-indicative probability" for each word, thus determining the overall "spam probability" of an email based on the set of words it contains.

No matter how efficient those techniques are when used to build a spam filter, the fact that spam generators are so creative in finding new schemes, makes those filters vulnerable to becoming obsolete very quickly if not updated continuously. To maintain their products efficiency, spam filters manufacturers use updatable database of information to feed their programs with changes and updates to reflect new schemes used by spam generators. Unfortunately, like the case of many other security-related software products (such as anti-virus programs) manufacturers of spam filters are always one step behind those who generate spam.

Finding a way to stay ahead might seem impossible but doing everything possible to keep up with the bad guys is vital for spam filters.

One way to help achieve that, is to create a spam filter that is extendible not only by a mean of information store in the form of a database but also in a way which makes it possible for a spam filter administrator to add "modules of analysis" to the spam filter engine itself (more details in Section **[6.4]** Traffic Analysis Engine). To my knowledge, none of the spam filters available nowadays offers this feature.

Although, quickly outdated, Spam statistics give a good idea of how big of a problem spam is for many users and how important it is to find an efficient solution. **Table [5A]** shows a summary of some statistics (from 2006) collected from several sources including Google, Brightmail, Jupiter Research, eMarketer, Gartner, MailShell, Harris Interactive, and Ferris Research.

Email considered Spam	40% of all email
Daily Spam emails sent	12.4 billion
Daily Spam received per person	6
Annual Spam received per person	2,200
Spam cost to all non-corp Internet users	$255 million
Spam cost to all U.S. Corporations in 2002	$8.9 billion
States with Anti-Spam Laws	26
Email address changes due to Spam	16%
Estimated Spam increase by 2007	63%
Annual Spam in 1,000 employee company	2.1 million
Users who reply to Spam email	28%
Users who purchased from Spam email	8%
Corporate email that is considered Spam	15-20%
Wasted corporate time per Spam email	4-5 seconds

Table [5A] *(2006 statistics show among other things, the huge financial effect of spam email).*

Radicati Group **[25]**, a market research firm in Palo Alto, California, predicts that by 2009, there will be 228 billion spam messages each day, representing the vast majority of email traffic on the Internet.

In this thesis I tried to model some techniques used in the Human Immune System (discussed above) to design and implement a spam filter and try to increase its efficiency in reducing false positives and false negatives by using the Self/Non-Self technique.

While researching spam filters and IDS, I also noticed the lack of products that implement both, a spam filter and a network traffic analyzer in the same system. The use of a network analyzer within a spam filter could be very beneficial for a network administrator because of the transparency that it could offer by monitoring and analyzing network traffic in addition to detecting spam. Adding the capability to monitor and analyze network traffic to a spam filter system would be very useful from a network security point of view. For instance, in a scenario where a network security administrator needs to know if certain confidential files are being sent via outgoing email, monitoring and analyzing email traffic would be very helpful to accomplish that.

This idea of building a bridge between a spam filter and a network monitoring tool is not hard to implement since the spam filter already scans and analyzes the email traffic. Therefore in addition to using ideas from the HIS to improve the efficiency of a spam filter I built a system that has the potential to be used for monitoring and analyzing network traffic.

Chapter 6

6- Solution (Spam Filter And Monitor)

The system that I have built is called Spam Filter And Monitor (SFAM). It is a solution that could be used by network and system administrators to detect spam and help monitor and analyze network traffic. It uses ideas from HIS and IDS to offer a solution that combines spam filtering and network traffic analysis in one system.

The idea of building a spam filter which uses the Self/Non-Self technique inspired from HIS is realized by scanning email traffic and detecting addresses (From, To, CC and BCC), then based on some Analysis Modules the system labels some of these addresses as 'Self' or clean and the rest as 'Non-Self' or spam. This solution also uses several levels of analysis to increase the efficiency of detecting real spam. I built three levels of analysis for this solution and more levels could be added later. The first level is analyzing the email addresses (mentioned above), the second level is accomplished by scanning the subject line of an email and flag spam words. The third layer is to scan the whole email body text for spam words. Once the email traffic is scanned and analyzed, SFAM splits the traffic into clean or 'self' and spam or 'non-self'. It then forwards the 'clean' email to users and quarantines the spam traffic for later analysis by the system administrator.

6.1 System Architecture

SFAM consists of several components as shown in **Figure [6.1A]**. Email traffic is directed through those components as follows: It starts at the firewall level of a secure network. All email traffic (SMTP protocol) is routed to an Email Server Simulator (ESS). This is done by setting the firewall to forward all traffic coming to port 25 (SMTP) to the internal IP address of the server hosting SFAM and a specific port number where ESS is expecting email traffic. ESS is then responsible for communication with the outside server sending email to ensure the reception of email. This communication goes as was defined in RFC 821. An example is shown in **Figure [6.1B]** and explained in more details in [**Section 6.2**] about ESS.

Once the communication is established between ESS and the outside email server where email originated from, email traffic can be captured by SFAM. For demonstration purpose only, SFAM will be fed SMTP traffic from .pcap files captured by Wireshark as shown in **Figure [6.1A]**.

The Traffic Analysis Engine (TAE) is the next component in the path of the email traffic flow. This component is the core of SFAM. It is responsible for scanning, analyzing and deciding which email is 'clean' and which email is 'spam'. It communicates with two other components to achieve this task. A database component which holds all data needed for analysis and a user interface component needed to communicate with the system administrator.

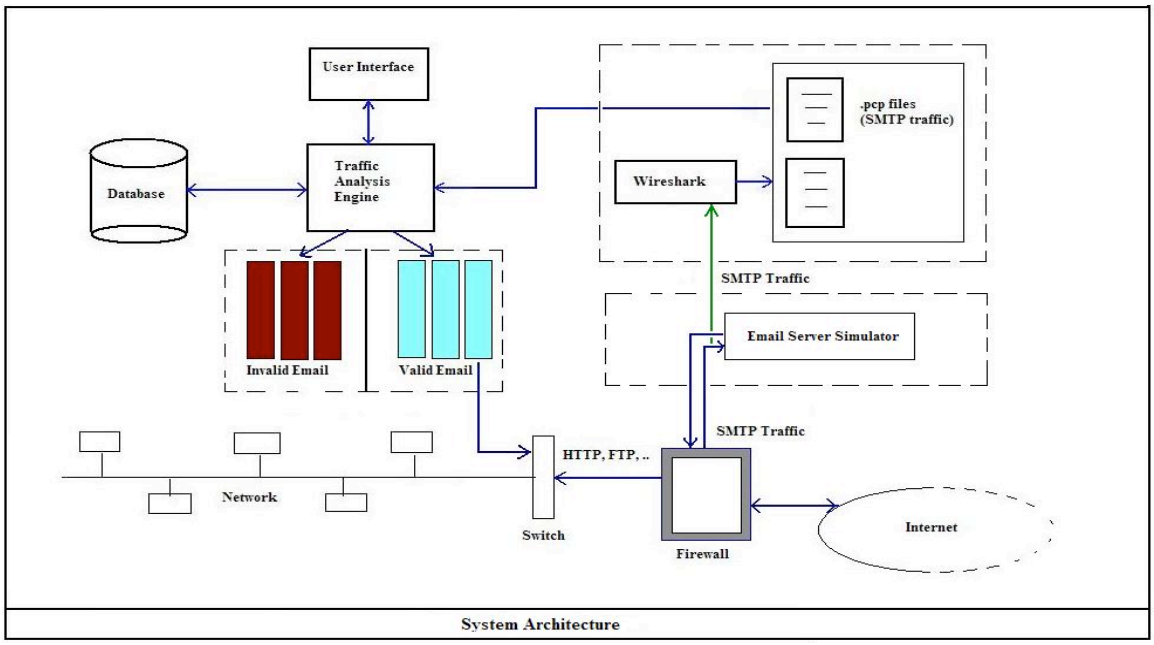


Figure [6.1A] *System architecture of S.F.A.M*

```
S: 220 www.example.com ESMTP Postfix
C: HELO mydomain.com
S: 250 Hello mydomain.com
C: MAIL FROM:<sender@mydomain.com>
S: 250 Ok
C: RCPT TO:<friend@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: test message
C: From: sender@mydomain.com
C: To: friend@example.com
C:
C: Hello,
C: This is a test.
C: Goodbye.
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```

Figure [6.1B] *SMTP Client/Sever Communication. "C" stands for client and "S" for server.*

6.2 Email Server Simulator (ESS):

The first component in our system which receives email traffic is the Email Server Simulator (ESS). This component is vital to the system as it assures the normal flow of email traffic by answering requests from external email servers. As described in **Figure [6.1B]** SMTP which is a client-server email communication protocol, is used here to route traffic to SFAM at the

firewall level of our secure network. An email server outside our network is considered a client to our ESS which answers the client's request by issuing command (220) to tell the client that it is ready to receive emails. The client then introduces itself (HELO mydomain.com), to which ESS says HELO back. The client identifies its sender's email address (FROM), to which ESS replies OK. Same thing happens with the destination address (TO). After that, the client starts sending DATA with the pre-determined agreement that at the end it should send a full stop (.) in a separate line. At the end of the communication the ESS component sends a command (250) after receiving a full stop (.) in a separate line, then the client sends [QUIT] command to inform ESS that it is ready to end the communication, to which ESS agrees and sends command (221) to close the communication channel.

6.3 Wireshark component:

This component is used for demonstration purpose only. It listens to SMTP communication between ESS and its client, and then saves the packets in "**.pcap**" files described in **[Section 4.1]** to be analyzed by the Traffic Analyzer Engine (TAE). TAE is perfectly capable of listening to the same traffic but for a demonstration it is more practical to feed the traffic caught by Wireshark directly to our Traffic Analyzer Engine. As explained in more details in **[Section 4.1]**, Wireshark is an efficient networking tool that can easily capture network traffic at the Data-link layer **[Section 4.2]**. Through the use of a set of filters it is easy to capture traffic based on a specific protocol. In our case it was used to capture SMTP traffic (port 25).

6.4 Traffic Analysis Engine (TAE):

This is the main component in SFAM. It scans SMTP packets and determines what is considered 'Self' or clean email and what is considered 'Non-Self' or spam email. TAE is designed and built in a way that supports multiple layers of traffic analysis. Three layers have been built, and the system supports additional layers that could be built in the future. TAE iterates through a collection of layers of analysis (or modules) that are available. Each layer generates a probability of how likely an email is to be considered spam. After going through all available layers, TAE calculates the overall average probability and decides whether the scanned email is spam or 'clean'. **Figure [6.4A]** shows the module structure of a layer of analysis. It has four main public methods: Initialize(), Terminate(), ReadData() and Analyze(). The first two methods are used to initialize and terminate the module respectively. ReadData() is used to feed email traffic to the module and Analyze() is used to calculate and return the probability of the email being spam. The three layers of analysis that I have built are explained in more details in the following paragraphs.

```
Initialize()
…
ReadData(email_packets)
Analyze() returns Probability P
…
Terminate()
```

Figure [6.4A] *Interface of a Layer of Analysis Module.*

First layer of analysis is defined by scanning the packets to detect all email addresses (From, To, CC and BCC) and compare them to a pre-built database of email addresses. Based on records in the database the system can identify 'valid' email addresses and flag those that are 'invalid'. **Figure [6.4B]** shows real email traffic example captured by Wireshark and fed to TAE in the form of network packets for analysis. After scanning all packets, TAE detects two email addresses *(joe@anotherdomain.com*) and (*christine@mydomain.com*). It communicates with the database component to retrieve information indicating how likely that they were used in spam email before. For more details please see the Database Component [**Section 6.5]**.

```
220 mailserver.mydomain.com Microsoft ESMTP MAIL Service, Version:
5.0.2195.6713 ready at Thu, 12 Apr 2007 14:41:04 -0700
EHLO mail.anotherdomain.com
250- mailserver.mydomain.com Hello [192.168.100.55]
250-TURN
250-ATRN
250-SIZE
250-ETRN
250-PIPELINING
250-DSN
250-ENHANCEDSTATUSCODES
250-8bitmime
250-BINARYMIME
250-CHUNKING
250-VRFY
250-X-EXPS GSSAPI NTLM LOGIN
250-X-EXPS=LOGIN
250-AUTH GSSAPI NTLM LOGIN
250-AUTH=LOGIN
250-X-LINK2STATE
250-XEXCH50
250 OK
MAIL FROM:<joe@anotherdomain.com> SIZE=8588 BODY=8BITMIME
RCPT TO:<christine@mydomain.com>
DATA
```

```
250 2.1.0 joe@anotherdomain.com....Sender OK
250 2.1.5 christine@mydomain.com
354 Start mail input; end with <CRLF>.<CRLF>

Subject: Hello!

From:  joe@anotherdomain.com

TO: christine@mydomain.com

…

Hello,

How are you doing?

Bye!

.

QUIT

250   2.6.0      <f32623c236692d5e36520a7ad7d1a412@mailing.cyberwebnet.com>
Queued mail for delivery

221 2.0.0 mailserver.mydomain.com Service closing transmission channel
```

Figure 6.4B *real email traffic example captured by Wireshark.*

The second layer of analysis built consists of scanning email packets to find the Subject Line and using a pre-defined database of 'spam terms' or expressions, it assigns a 'spam score' to the words used in the subject. For instance if the subject line contains the following expression ("While Supplies last") or ("You've been selected") its spam score increases. The exact score for each term is defined by the System Administrator and should be fine tuned periodically to be more efficient. More details about the choice of particular terms and expressions to be flagged as spam are available in the database component **[Section 6.5]**

The third layer is similar to the second one as it scans the whole email content and scans for spam words. Obviously this layer is more resource demanding than the second one as it scans every single word in the email. But it is meant to increase the efficiency of detecting real spam.

After analyzing the email traffic, TAE splits the traffic into two sections. The first one includes all the 'clean' traffic to be placed in a queue of "**.pcap**" files then forwarded to our secure network. The second one consists of all 'spam traffic' which sits in a queue until further analysis by the system administrator. This 'spam' email is saved just in case some of it turns out to be a false positive. In such case, the system admin will release it.

6.5 Database Component:

This component is the main information source for TAE. As described in [**Section 6.4],** it contains a list of email addresses. Each address has two numbers a [Valid Votes] count and an [Invalid Votes] count. Each time an email address is used as a sending address in a spam email it gains an additional vote in its [Invalid Votes] count. The same scenario applies with a valid

email address and [Valid Votes] count. Initially, the idea behind this voting system was based on a multi-user contribution to the database. In other words all users (those that use email system in our secure LAN) would contribute to the database. When they receive a spam email, they report it as 'invalid' which increases its [Invalid Votes] count. The system admin can overwrite this process by assigning any number of votes. Unfortunately, implementing this idea is not very practical as users have no easy way of providing data without wasting time that could be used to do their jobs instead of trying to increase the efficiency of the spam filter. One automated way to do this is to build a component that reads information from the internal email server and gets a list of email addresses from the users' 'junk' folders and feeds it directly to our database. The challenge of this solution lies within the very large variety of email servers available in the market and their different architectures make it very difficult if not impossible to build a single component that can read from all different servers. The positive side of this though, is that whenever a component is built for a specific server, it can have direct access to our database and can contribute to enhancing the efficiency of TAE without any changes to the overall system.

Another section of the database includes a list of common spam words. Words, expressions or sentences that are more likely to appear in a spam email than in a clean email are added to our database. This list is continuously updated and refined to reflect the latest changes that spam generators make to avoid being caught by spam filters. There are several commercial websites that offer a database of such spam words and sentences. For demonstration purposes, I filled this section of the database with sample data that is available over the internet. The following are some examples:

- "Special Promotion"
- "Easy Terms"
- "Meet Singles"
- "No cost, No fees"
- "Order Now"
- "Unsecured debt or credit"

6.6 Interface Component:

Using Visual Studio .NET I built a user interface to allow the system administrator to communicate with the Traffic Analysis Engine (TAE) and the rest of the system's components, such as the database component. This interface is also used to modify the settings of TAE and report the result of email traffic analysis.

Main screen:

As shown in the following screenshot (**Figure [6.6A]**), the interface offers two ways to access different features of the program. Either through a menu or a set of command buttons on a toolbar the user can access all the features. Moving the mouse over any button of the toolbar should show a 'tool tip' which describes briefly what the feature is supposed to accomplish.

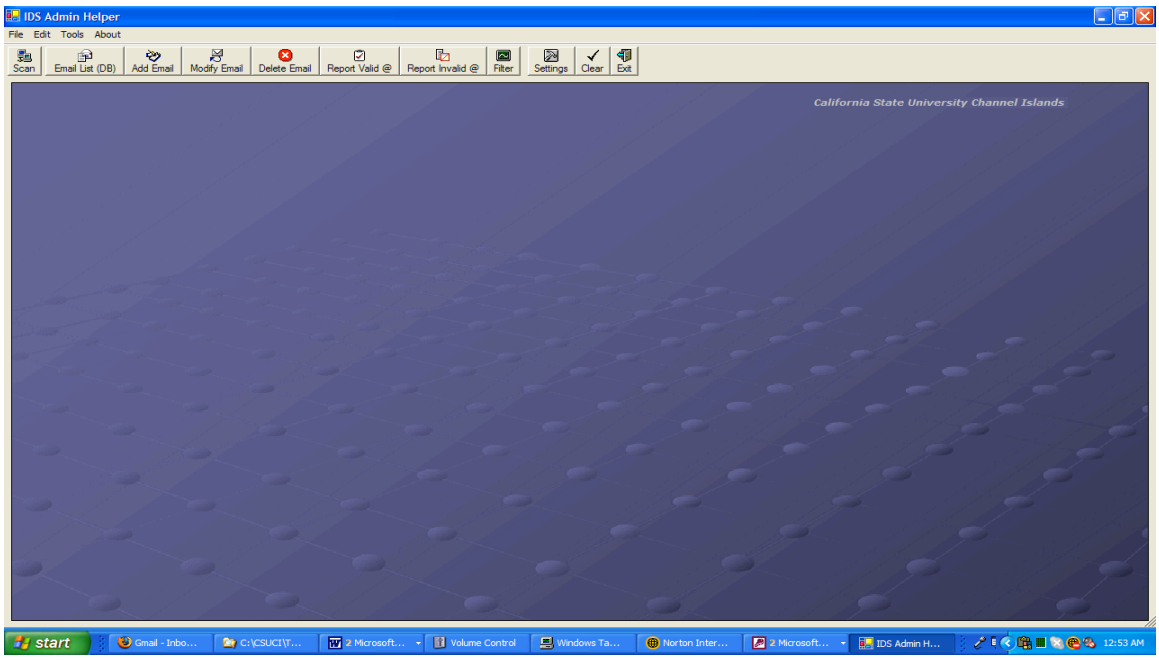


Figure [6.6A] *Main menu of the S.F.A.M program.*

Email List screen:

For instance, the tool 'Email List' is used to display the list of all email addresses stored in the database (**Figure [6.6B]**). Each email address is shown along side with its 'valid votes' and 'invalid votes' count as well as a brief description.

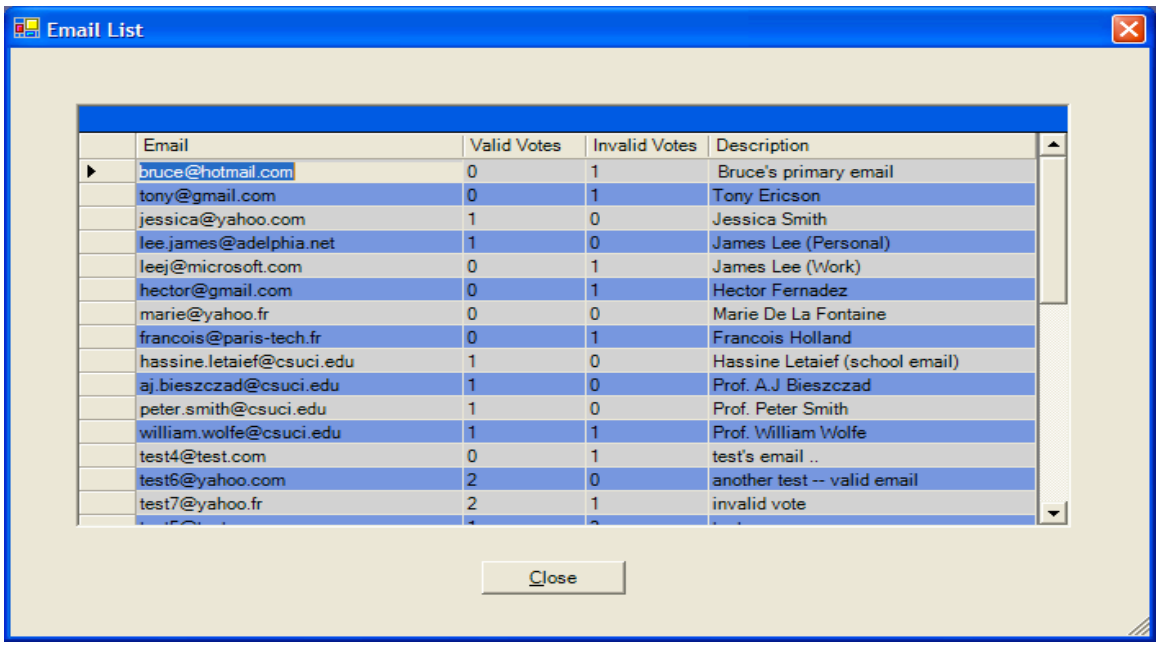


Figure [6.6B] *List of email addresses in the S.F.A.M database with 'valid' and 'invalid' votes counts.*

Add Email screen:

The 'Add Email' tool (**Figure [6.6C]**) permits the user to add an email address to the database and set its 'valid votes' or 'invalid votes' count to 1 depending on whether the 'Valid Email' option is checked or not. This tool does not allow duplicate entries; it prompts the user if he/she tries to add an address that already exists in the database.

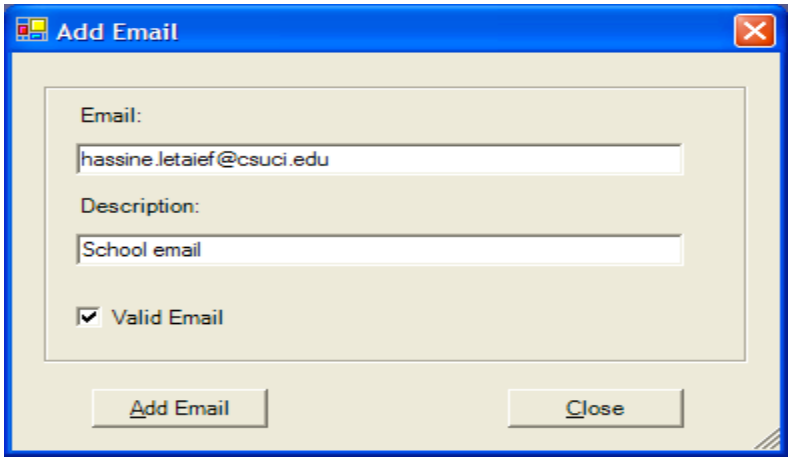


Figure [6.6C] *The 'Add Email' form is used to add new email addresses to S.F.A.M database and initialize their 'valid' or 'invalid' votes count.*

Modify Email screen:

The 'Modify Email' tool as **Figure [6.6D]** indicates allows the user to modify or delete an existing email address. The need for this tool might not rise unless an email address was entered by mistake and thus should be modified or removed from the database.

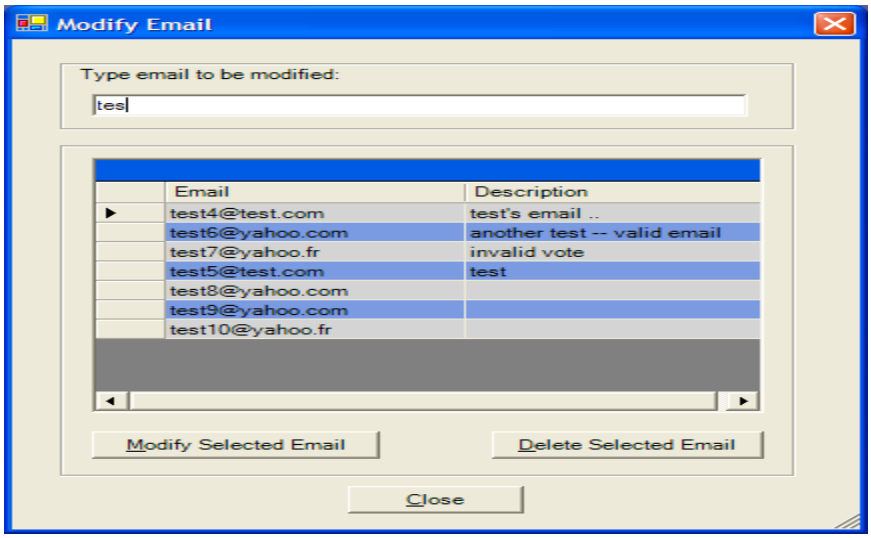


Figure [6.6D] *Modify or delete an existing email address.*

Vote Email (valid/invalid) screen:

The tool 'Vote Email (valid/invalid)' (**Figure [6.6E]**) is used to report an increase in the 'valid votes' or 'invalid votes' count of a given email. When the user starts typing an email address, the program starts showing all email addresses which start with the same sequence of letters entered. If for example, the user wants to increase the 'valid votes' count of an existing email address, he/she needs to select that address from the list and then click on 'Validate selected email'. On the other hand if the user enters an email address that does not exist in the database, chooses to 'validate' or 'invalidate' it, the program should add it and set its 'valid votes' or 'invalid votes' count to 1 (similar to the tool 'Add Email' discussed earlier).

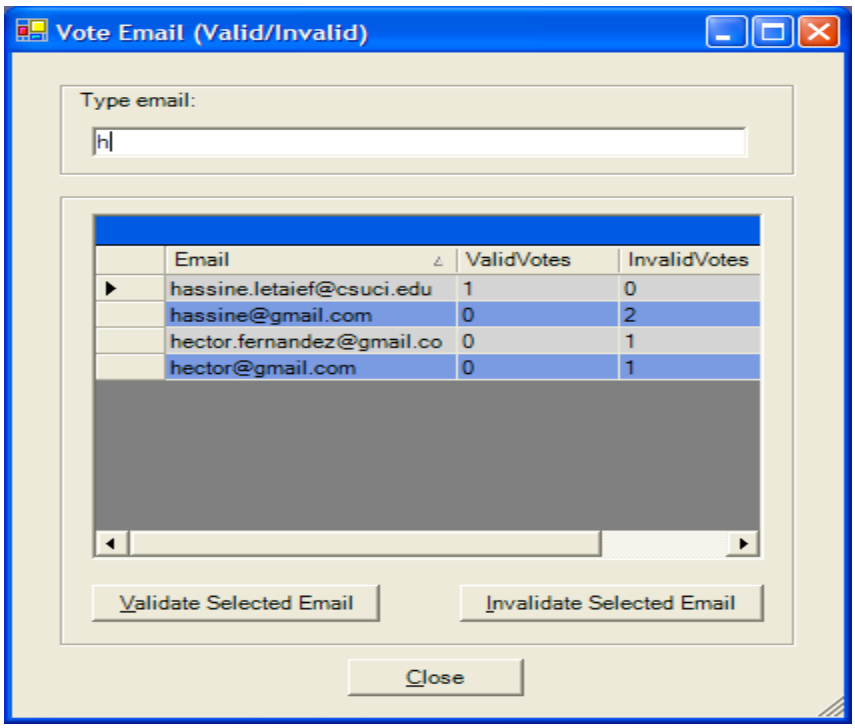


Figure [6.6E] *Report an email address as 'valid' or 'invalid' increments its corresponding votes count by one.*

Settings Screen:

Before the Traffic Analysis Engine starts scanning SMTP packets a group of settings must be initialized to determine which features the program is going to execute when scanning (**Figure [6.6F]**). The settings represent the different layers of analysis described earlier. The first one is scanning for valid/invalid email addresses. The second is scanning the subject line of an email for 'spam' terms and expressions. The third is scanning the whole email text body for 'spam' expressions.

Another setting is the Spam Threshold Score, it is a number chosen and fine tuned by the system admin to increase the efficiency of detecting real spam and decreasing the number of false positives.

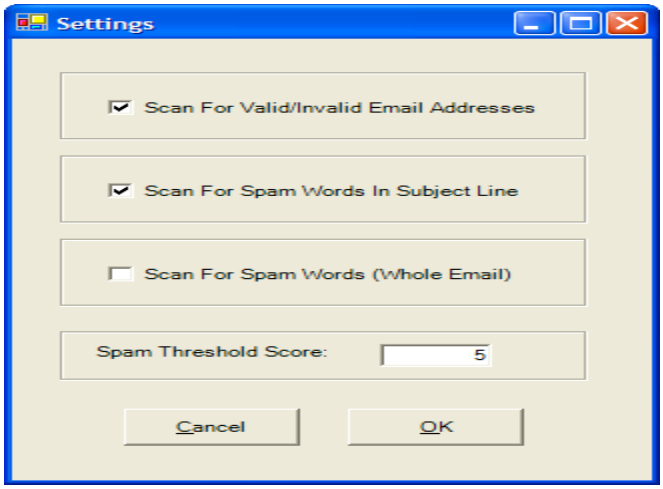


Figure [6.6F] *Initialize S.F.A.M settings by enabling/disabling different layers (modules) of analysis.*

Traffic Analysis Results Screen:

The following screenshot (**Figure [6.6G]**) shows several lists of data representing the result of scanning and analyzing SMTP traffic. One list shows all the 'valid' email addresses detected, another one shows the 'invalid' ones. A list of all subject lines detected is also displayed (for demonstration purpose) and next to it, is a list that shows all the 'spam' expressions detected in those subject lines and/or detected in the email text body (depending on the settings chosen before the scanning begins). A timer is also displayed to indicate whether or not the scanning is finished.

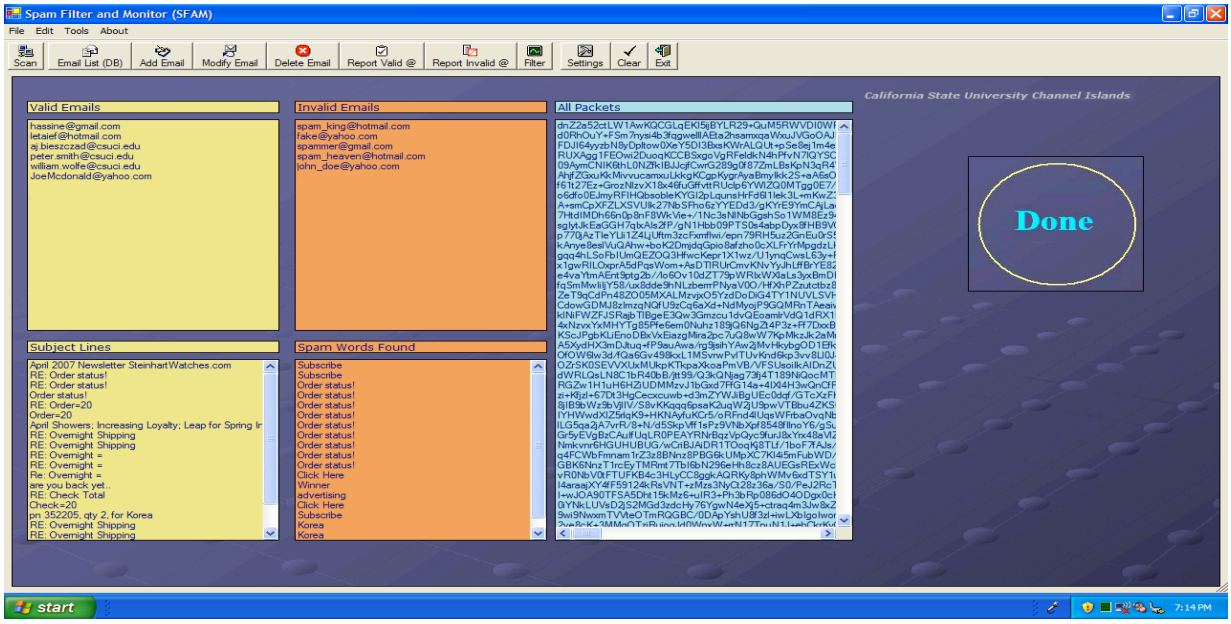


Figure [6.6G] *Traffic analysis result shows lists of detected spam words, valid/invalid email addresses as well as the whole 'text-equivalent' of the SMTP traffic analyzed.*

Chapter 7

7- Evaluation

In order to test SFAM's efficiency in detecting spam, several .pcap files captured by Wireshark were used (about 5MB of SMTP traffic). The following is a summary of the results obtained by using three levels of analysis (Addresses, Subject Line and Email Body) described in [**Section 6.4**]:

Information about the hardware used:

HP Pavilion
AMD Athlon ™ 64 Processor 3200+
994 MHz
768 MB of RAM

Scanning Results:

All email addresses and 'spam expressions" were detected successfully by TAE in seven seconds, when scanning a 500 Kbytes ".pcap" file of SMTP traffic with the first layer analysis only. The time was not significantly different when adding the second layer of analysis (Subject Line). On the other hand, adding the third layer of analysis (scanning the email text body) took about three seconds more (about 40% increase). These time measurements obviously depend on how large the email is and how much data TAE reads from the database for analysis. I used a list of 58 expressions in the 'spam' word section of the database. I also used a list of 35 recognizable email addresses in the database.

As expected, the results of scanning several files of SMTP traffic showed that the efficiency of detecting real spam, thus reducing false positives and false negatives depends on how efficient each layer of analysis is in doing the job separately and what effect combining several layers has on the overall result. For instance, the first layer which consists of validating email addresses was very efficient in detecting all addresses and flagging those that are marked as 'invalid addresses' in the database but its efficiency decreases dramatically when a spam email uses fake 'valid' addresses. The following example describes this in more details:

Consider the following spam email (**Figure [7A]**):

```
From: spamking@hotmail.com
To: hassine.letaief@csuci.edu
Subject: Special Promotion!
Body: Check this offer, easy terms, no down payment!
```

Figure [7A]. *An example email that could be interpreted as 'clean' or 'spam' depending on which module(s) of analysis are enabled.*

- The email addresses: (hassine.letaief@csuci.edu) and (aj.bieszczad@csuci.edu) are both marked as valid addresses in SFAM's database whereas (spamking@hotmail.com) is 'invalid'.
- When this email (**Figure [7A]**) was scanned with first layer of analysis only, the email was flagged as spam since the address (spamking@hotmail.com) is marked as 'invalid' (or very likely to be used in spam email) in the database.
- This might sound good at first, but knowing that it is very easy to send an email with a fake source email address makes the first layer of analysis insufficient in this case. For example the spam sender could easily send a spam email that shows (aj.bieszczad@csuci.edu) which is a valid address as the source of the email (which is not true) but the first layer of analysis won't be able to detect that and it will pass it as "clean" email.
- Adding the second layer of analysis however helps detect this spam email. Notice that the subject line contains the expression "Special Promotion" which is a member of the spam terms list in our database. This however could also be tricked by the spam sender if the subject line did not contain that expression.
- The same principle applies when adding the third layer of analysis (scanning the text body of the email) since the expression "easy terms" in the email body also is a member of the spam list.
- The challenge for these layers of analysis lies within the potential high level of false positives. For example, if the email shown above (**Figure [7A]**) was a legitimate email sent from (aj.bieszczad@csuci.edu) to (hassine.letaief@csuci.edu) and happens to include the expression "easy terms". If the first layer and the third layer are the only layers used, it would be flagged as 50% chance of being spam since the addresses are valid but there is a spam expression in the email text. Adding the second layer of analysis would be vital in this case. If the subject line also happens to contain a 'spam' expression then the email is flagged as spam. Thus if a legitimate email happens to have spam expressions in both subject line and text body, it will be flagged by TAE as spam. This is why adding other layers of analysis to SFAM is very important to improve its efficiency.
- As for the monitoring part of SFAM, and to test the efficiency of detecting a key word in the entire email such as file attachment type, I added a keyword which I know exists in one of the emails to the database and I set its spam score to a number higher than the spam threshold. Then I started scanning the ".pcap" file containing that email. The word was detected and the email was flagged as spam.

Chapter 8

8- Conclusion and future work

Summary:

When I started this research one of my target goals was to try to solve some problems that most Intrusion Detection Systems suffer from, namely the huge number of false positives occurring when an IDS flags a pattern of normal traffic as malicious. Another problem that I had as a target to try to solve is the massive log size produced by most current IDS which makes the job of a system administrator very difficult.

To accomplish those goals I started by researching what was already done in this field to get some ideas on what direction I need to point my research to achieve something practical as well as useful. A promising research field that attracted my attention more than others was the known efficiency of the Human Immune System in detecting malicious foreign bodies that try to invade it. If we could build an IDS that imitates the HIS in its techniques and hopefully in its efficiency, that would be a great step into improving the overall outcome of an IDS in securing and protecting a network.

Self/Non-Self, Immune Network, and Danger Theory are some of the most promising theories that if borrowed from HIS would make an IDS potentially far more efficient than any product currently available in the market.

One problem that a system/network administrator faces on a daily basis is the overwhelming number of spam email sent to network's email server and thus crowding the users input boxes with useless and very annoying amount of email. The idea of using the Self/Non-self technique in a spam filter is simply achieved by building a database of 'valid' and 'invalid' email addresses and analyzing SMTP traffic to detect and identify 'valid email' and 'invalid email'. This leads to filtering email and blocking spam before it arrives to users input boxes.

To increase the SFAM's efficiency I added more layers of analysis, such as scanning the subject line and/or the whole email text body for 'spam expressions'.

In addition to the spam filter role that SFAM attempts to fulfill, it also offers the ability to monitor outgoing email traffic to flag any violation of security procedures.

Future enhancements

SFAM as IDS tool:

In addition to its use as a spam filter, SFAM can also detect 'miss-use' of resources or, even worse, violation of security procedures within a secure network. Consider the following scenario where a company gets audited periodically to ensure that all necessary security measures are put in place to protect confidential data. One question that a network security auditor might ask is: "*Does your company have any way of monitoring your outgoing emails?*

In other words, if a user who is cleared to access a confidential document decides to send an email to an outside email address, does the company have any way to detect that?" The answer to that question is 'yes' if we use SFAM. All the system administrator has to do, is add a key term such as a file extension of the confidential document or a file name or any other keyword related to the document to our database of spam expressions and set its spam score to a high value (high enough to be considered as spam). TAE should be able to detect it and flag it as 'spam'. In this case a minor modification to SFAM could log all detected violations of that nature and build periodic reports to be analyzed by the company's management team.

Block spam before it gets to our server:

Instead of capturing all email traffic and analyzing it, our system could be enhanced with the addition of another database that keeps track of IP addresses known of sending spam. Our Email Server Simulator (ESS) with instructions from our Traffic Analysis Engine (both described above) would refuse email coming from those IP addresses. This would save the system a lot of overhead and free more of the network resources.

This is very similar to what the Spamhaus Project **[24]** offers with its SBL (Spamhaus Block List). "*The SBL is a real-time database of IP addresses of verified spam sources and spam operations (including spammers, spam gangs and spam support services), maintained by the Spamhaus Project team and supplied as a free service to help email administrators better manage incoming email streams*."

Improve first layer of analysis dynamically:

Another possible enhancement that could be added to our system is based on its ability to use other layers of analysis to increase its efficiency in detecting real spam, thus minimizing false positives. One way to do that is to build another component that can read the Information Store of an email server (such as Microsoft Exchange Server) and automatically retrieve a list of email addresses from the 'junk' folders of users, then add it to our database. This will make the system up to date and more resistant to spam email.

Analysis Module Using Bayes's Theorem:

To improve the efficiency of SFAM we could add a new module as another layer of analysis. This module would use Bayes's theorem which is very commonly used in modern spam filters. Similar to the second and the third layers of SFAM, it consists of using a predefined list of "spam words" but it differs in its way of concluding whether or not an email is spam. Bayes's theorem applied to spam email is as follows:

$$\Pr(spam \mid words) = \frac{\Pr(words \mid spam)\Pr(spam)}{\Pr(words)}$$

“The probability that an email is spam, given that it has certain words in it, is equal to the probability of finding those certain words in spam email, times the probability that any email is spam, divided by the probability of finding those words in any email:”

9- References

[1] U.S National Institute of Health, Understanding the Immune System How It Works (2003).

[2] U Aickelin, P Bentley, S Cayzer, J Kim, J McLeod, Danger Theory: The Link between AIS and IDS (2003)

[3] Polly Matzinger, Tolerance Danger and the Extended Family, Annual reviews of Immunology. (1994)

[4] Paul Innella, The Evolution of Intrusion Detection Systems (2001)

[5] U Aickelin, Julie Greensmith, Jamie Twycross, Immune System Approaches to Intrusion Detection (2003)

[6] Paul Bugl, University of Hartford (Connecticut), Immune System (2001)

[7] Niels K. Jerne, Georges J.F. Köhler and César Milstein, (Nobel Prize) Development and control of the immune system. The discovery of "the principle for production of monoclonal antibodies". (1984)

[8] Niels K. Jerne, Natural-Selection Theory of Antibody Formation (1955).

[9] Niels K. Jerne, Somatic Generation of Immune Recognition (1971).

[10] Niels K. Jerne, The Network Theory (1974).

[11] Marmagna Desai, Intrusion Detection System (2004).

[12] Douglas J. Brown, Bill Suckow, and Tianqiu Wang, A Survey of Intrusion Detection Systems. (1999)

[13] Source: The Federal Financial Institutions Examination Council (ffiec.gov)

[14] Source: wikipedia.org

[15] Steven A. Hofmeyr and S. Forrest. Architecture for an Artificial Immune System, Evolutionary Computation Journal, (2000)

[16] Mark Handley, Vern Paxson, and Christian Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics, 10th USENIX Security Symposium, Washington, D.C., (2001)

[17] Eleazar Eskin. Anomaly Detection over Noisy Data Using Learned Probability Distributions," Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000), Palo Alto, California, (2000).

[18] Anup K. Ghosh, James Wanken, and Frank Charron. Detecting Anomalous and Unknown Intrusions Against Programs, Annual Computer Security Applications Conference (ACSAC'98), Scottsdale, Arizona, (1998).

[19] Terran Lane and Carla E. Brodley. An Application of Machine Learning to Anomaly Detection, 20th Annual National Information Systems Security Conference, 1, pp. 366-380, (1997).

[20] The Wireshark Network Analyzer (http://www.wireshark.org)

[21] Libpcap File Format (http://wiki.wireshark.org/Development/LibpcapFileFormat)

[22] Ethernet (http://en.wikipedia.org/wiki/Ethernet)

[23] Spamhaus project (http://www.spamhaus.org/index.lasso)

[24] John Graham-Cumming. Understanding Spam Filter Accuracy, (2004).

[25] The Radicati Group. A Technology Market Research Firm (http://www.radicati.com/)

[26] Thomas Bayes (1702 – 1761) was a British mathematician (created the Bayesian Theorem used in many spam filters).