Using Genetic Engineering Algorithm to Prevent Subversion of an Intrusion Detection System.

A Thesis Submitted to The Faculty of Science,
Department of Computer Science
University of Zimbabwe

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

By

Hector Kapelewela

February 2006

Using Genetic Engineering Algorithm to Prevent Subversion of an Intrusion Detection System.

A Thesis Submitted to
The Faculty of Science, Department of Computer Science
University of Zimbabwe

In Partial Fulfilment

of the Requirements for the Degree of

Master of Science (Computer Science)

By

Hector Kapelewela (R950635M)

February 2006

Dedications

To my Father who passed away before this work was delivered may his soul rest in peace, to my family Wadzanai my wife and Charles my son**.**

Acknowledgements

I acknowledge the help of so many people during the time of this project undertaking. My greatest thanks go to my supervisor Mr N. Ngoma, my former supervisor Mrs F. Mandizvidza, and the initial supervisor who initiated this topic and indeed made me understand what genetics are and what they can do to computer security.

Mr Ngoma who despite having a timetable completely full, worked so hard to understand this topic and the student at the same time. Thank you for helping me all round the clock through valuable discussions and telephone calls, without which this work could not have been possible.

Thanks to the Head of Department of Computer Science, and the course coordinators, for providing hotel presentation environments with masterpieces of equipment, which made me, appreciate the value of my project undertaking to the academic society and computer science community at large.

Abstract

Intruders mercilessly attack commercial, academic, defence; healthy centres distributed (networked) systems frequently, and often successfully. The challenges of intruders have become very critical. The most perceived effective defence today is the use of intrusion detection systems. (IDS), though it is widely considered to be impossible to build an effective distributed systems that completely eliminate unauthorized intrusions. It may be effective to thicken the wall of defence by building effective procedures in form of efficient algorithms inside IDSs. The target system should account for misuse detection and anomaly detection by reporting on the presence of an intruder, protecting the system from harm by the intruder, make intrusions into the system difficult, help locate the intruder for a possible prosecution with the law. Our solution analyse each string that is introduced into the computers to those residing in the system already by carrying out a pattern matching with detectors that match only strange patterns and recognise all friendly patterns that are legitimate to the system.

Blank Page

Table of Contents

CHAPTER I INTRODUCTION

1.0.0 Preamble 1

1.0.1 Background 1

1.1 Categories of Intrusion Detection system 2

1.2 Intrusion Type 3

1.3 Violation of security mechanisms 6

1.4 Some Historical Background 6

1.5 Why not Use a Firewall 7

1.6 The Immune System 8

1.7 Why Imitate Immune System 8

1.8 How does the Immune System Recognizes Antigens 9

1.9 The Memory Concept 12

1.10 Justification 13

1.11 Scope of Study 14

1.12 Mechanism and functionality of GEA 15

1.13 Imitations of Existing Intrusion Detection Systems -IDS 15

1.14 Definitions of Terms and concepts 16

1.14.1 Single System IDS 16

1.14.2 Disparate IDS 16

1.14.3 Distributed IDS 17

1.14.4 Network based IDS 17

1.14.5 Host Based IDS 17

1.14.6 Firewall Based IDS 18

1.14.7 Classifier Expert System 18

1.14.8 Network Security Manager 18

CHAPTER II LITERATURE REVIEW

2.0 Introduction to Literature Review 19

2.1 Details of Existing IDS 19

2.2 The Likeness of Artificial and HIS (Human Immune System) 24

2.3. How different is our presentation 25

CHAPTER III METHODOLOGY

3.1 Components of a Genetic Algorithm 33

3.2 Philosophy of Approach 35

3.3 Matching 36

3.4 Consider Illustration 36

3.5 Partial Matching 38

3.5.1 We illustrate partial matching r=3 38

3.6 Estimating Probability of Detection 39

3.7 Algorithm Performance 41

3.8 Detection Size 41

3.9 Sub-Equations 42

3.9.1 Boltzinann Selection 43

3.10 Holes 43

3.11 Linear Time Algorithm 45

CHAPTER IV FINDINGS AND CONCLUSIONS

4.0 Information Loss 46

4.1 Relationship between Detector Sets and Failure Probability 47

4.2 Experimenting with Y_m and Pf as constant 53

4.3 Experimenting with Theoretical N_{R0} and Experimental N_{R0} 54

4.3.1 Observed Results 55

4.4 Discussions 56

4.5 Conclusions and Recommendations 58

Annexes 59

Bibliography 68

List of Tables

Table 1 Experimental $N_{s,}$ Constant Pf and varying l values 47

Table 2 Experimental values 49

Table 3 Experimenting with Y_m and Pf as constants 53

List of Figures

Fig 2 Hypothetical Matching 26

Fig 3 Censoring 34

Fig 4 Monitoring by Matching 34

Fig 5 Existence of holes 48

Fig 6 Relationship between initial

detectors sets and failure probability 49

Fig7 Experimenting with a fixed NR. 54

Fig 8 Results depicting proportions

of generated detectors and actual size of detectors used. 56

Glossary of Symbols

IDS - Intrusion Detection Systems

GA or GEA – genetic Algorithm

OSIDS – operating System Intrusion Detection Systems

HIDS – host Intrusion Detection Systems

LAN – local Area Network

NFS – network File System

DoS – denial of Service attack

Signature – a set of condition when met indicate some kind of intrusion event.

Y_m - matching probability of between a randomly chosen string and a detector.

N_s - the number of self-strings

m - is the alphabet composition, binary =2

l - Strings length

Γ or r - non-contiguous matching bits, standing for the threshold

$\cap$ -Intersection sign

$\cup$ -Union sign

Pf - possible failure probability-

$\approx$ Symbol standing for identical to

Ω Big O notation symbol

OS Operating Systems

{} Sets

(p/t) matching possibilities of string power strings p over string

Chapter I Introduction

I.0.0 Preamble

The security personnel or a System Administrator has to deal with many security problems brought about by the computer system all the times. Computer systems bring together a series of vulnerabilities. There are human vulnerabilities throughout; this means humans cause them and individual acts can accidentally or deliberately jeopardize the system's information protection capabilities. Hardware vulnerabilities are shared among the computers, the communication facilities, and the remote units and consoles. There are software vulnerabilities at all levels of the machine operating system and supporting software; and there are vulnerabilities in the organization of the protection system (e.g., in access control, in user identification and authentication, etc.). How serious any one of these might be depends on the sensitivity (classification) of the information being handled, the class of users, the computational capabilities available to the user, the operating environment, the skill with which the system has been designed, and the capabilities of potential attackers of the system.

1.0.1 Background

Neither firewalls nor access control lists once thought as good solutions in preventing network intrusion, can provide the capability to respond to or provide real-time detection of an intrusion attempt, most of which has been

described above. IDS provide continual real-time or near-real-time monitoring of a host or a network. In this research work we concentrate much on creating mechanisms using a GA to police the existing system. Borrowing the words of Steven Hofmeyr:

‘The crossover between Biology and computer science can be fruitful for both disciplines: computers can be used to model biological systems to improve our understanding of those systems, and we can use understanding of mechanisms underlying biological systems to improve the way we design computer systems here we focus on the latter case: using biological metaphors to build better computer systems. “Through Genetic Algorithm design “

1.1 Categories of Intrusion Detection system

Intrusion Detection System (IDS) is a piece of software or hardware that is designed to recognize all pieces of code of software that are introduced into the computer system either though the network or direct injection via a stiff or any movable hardware.

Two primary categories of IDS are network-based and host-based. Network-based IDS monitor network traffic on the local LAN, analyzing traffic that "fits" a known signature for a given exploit, and then notifies the proper contacts of its findings. Host-based IDS tools provide detection of an intrusion on a system within the network. Since it is widely considered to be impossible to build IDS that completely eliminate unauthorized users, host-based IDS

should be able to determine whether the attacker who attempted to enter the system had succeeded in compromising the system.

The building blocks of Network-based IDS comprises of real-time and near real time. Real-time network-based IDS report suspicious traffic as soon as it is detected on the wire. Near-real-time IDS work by gathering network traffic and then at a predetermined time interval (such as once an hour) provide an analysis of the previous interval's data. One of the benefits of real-time IDS is the capability to respond to an attack as it is happening. Near-real-time IDS also provide sufficient notification of an attack in progress.

Host-based IDS monitor system files (log files) as well as check the integrity of system binaries to determine whether an intrusion has occurred. These types of IDS utilize an agent that resides on the host being monitored. An example of this attack would involve NFS and rlogin attacks. Initially the attacker determines NFS file-handle for a remote host (rhosts) file /etc/host.equiv. Using NFS file handle the attacker then rewrites the file to give himself login privileges to the attacked host, using rlogin from the formally intruding host. The attacker would be able to login to an account on the attacked host, since attacked host mistakenly now trust the attacker. At this point the system can be adversely be compromised. [GUGH].

1.2 Intrusion Types

Policies are a cornerstone of all security of the computer system. Any activities that are done outside the scope of the computer policy definition may become unacceptable hence intrusive. However, policies are difficulty to

formulate. Tight policies will almost bring to halt meaningful computer activities. Consider a computer server belonging to the head office of a bank being disconnected from the network in an environment where shared transactions need to take place on a continuous basis, obviously services will be down and clients will be disappointed.

Security policies need to be tightened to a certain extent; this means the problem of computer attacks cannot be eliminated with policies alone. In some extents what contributes to system compromise is the lack of a sound policy on security.

The challenges in defining security policies is that they have to be written in natural languages, which becomes difficult determining if some usage violates the policies. A more formal way of describing these would remove the difficulties brought in by the natural language.

A suggestive sequence of implementation policy could model as follows**:**

Correct policy

Correct Implementation

Correct Figuration

This lead to **SECURE SYSTEM**, hence less attacks

The opposite being

Flawed policies

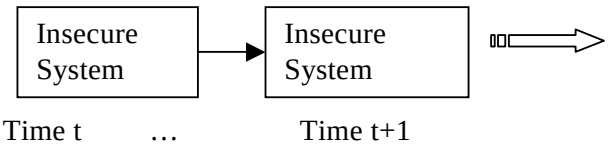
Flawed implementation

Flawed configurations

INSECURE SYSTEM

The second problem is caused by the vendor system, after sometimes the system goes through many changes. The changes are often not documented, from which security holes emerge.

A model of compromise can be drawn as



 Continued changing system

To understand whom the enemy is we need to understand a little bit about when and how a genuine user or individual becomes a misuse. (Intruders), through understanding some of the underlying meanings of the following:

Authorized: Is a person who should be able to use the system with full permission from the administrator. The individual may be aware or unaware of the violation of the security policies.

Unauthorized: Is a person who according to the security policy must not use the system.

1.3 Violation of Security Mechanisms

Not violating: the security policy is violated merely by misuse of privileges.

Bypassing: Security flaws are used to bypass the security mechanisms

Tempering: The security mechanisms are affected in a way that obstructs detection of intrusion i.e. erasure of log files.

Affect of Computer System

Confidentiality: information is leaked to unauthorized person

Integrity: Data, programs or resources are altered.

Availability: Access to programs or resources is limited or denied for authorized users

Systems evaluated following these guidelines, make it easier to understand what the intruders did and how to identify them.

1.4 Some Historical Background

From Mainframe to PCs, and PCs to Laptops the computer development has grown too fast. The need for shared knowledge and expertise, through wireless communication, the Internet and other network connections has become a reality and much sought technology in organizations, churches and homes and academic institutions.

Networked computers have transformed the world into a global village where dynamism in computer systems is fast becoming a common trend to our

academic and industrial operation. The Internet is the most perceived and cheaper tool that help us achieve text, visual, “voice” quality reception or and transmission of voluminous information within time bands unimaginable ten years ago.

The technology however brought with it threats of attacks from hackers. People are preempting the sensitive information in transit. They make illegal copies out of it; malicious worms and other contaminations are introduced to the information.

1.5 Why not Use a Firewall

In networked computer firewalling is the protection to the system, it controls packets that come in and out of the gateway. However it is inevitable that some of the packets firewalls allow in are malicious in nature, hence the need for a second layer of defense to complement fire walling [JUN], which we call IDS.

The ultimate goal of intrusion detection is to detect and classify instances of misuse of a system, while ignoring all instances of legitimate use. Intrusion Detection System work by analyzing one or more inputs event streams and by looking for manifestation of attack. An example of the event stream is packets sent on a network link, the audited record generated by the kernel-level auditing facility or the log produced by the user-level applications.

1.6 The Immune System

The immune system is a mysterious natural system. Though little is known about its main functionalities, there is a consensus among scientists that it protects the body from harmful organisms, which from time to time enter the human bodies through the nose, skin, mouth, and several other points. At the center of contention is the question on 'how does the immune system defense mechanism recognize novel attacks, the new harmful organism that enter the human body'. Assuming that the immune system recognizes new attacks by a way of learning, observing and comparing the behavior of the new intrusion through some inbuilt benchmarks, the challenge is that can computer scientist use the same analogy to build algorithmic tools that can recognize novel intrusions, worms, and virus, which attack computer systems, before extensive damage is done?

Biologically inspired computational intelligence approaches [AyaraTimis] have provided robust, error tolerant, scalable and flexible solutions to otherwise what would be intractable problems in the areas of medicine, neural networks, and swarm systems. [Bentley 2001].

1.7 Why Imitate Immune System

Imitating natural systems is a promising source of practically and feasible set of solutions to dynamic intrusion challenges, facing the computer experts today. When, the internal dynamics of a system is understood, in terms of functionalities, scope, and observable dynamics of the system through

experimentation. The knowledge can then be correctly used to build a similar natural system, and fine-tuned if significant deviations are observed. It is in this spirit that we give the detailed functionality of the immune system.

The immune system responds to foreign invaders known as pathogens, innate immune system provides its first defense, just like the pony piece on the chessboard. When the line of defense is broken, the adaptive (acquired immune system) take over, just like our chessboard analogy, we can liken the adaptive system to the bishops, queen, or rock pieces coming in the battle line, to rescue the kingdom.

1.8 How does the Immune System Recognizes Antigens?

The adaptive system is composed of B and T cells. These are capable of responding to certain antigenic patterns presented on the surface of the pathogens [AyaraTimis]. The receptor molecules on the surface of the immune cells are capable of recognizing limitless numbers of antigenic patterns. B cells recognize patterns in the blood and T-cells recognize antigens on other cells that come closer to it. [DeCastro].

Antigenic recognition is the prerequisite for the immune system to be stimulated, to mount an immune system response. Computer security or intrusion problems are concerned with distinguishing self (legitimate user, authorized actions, original source code, ‘uncorrupted data from non self (intruders’, computer viruses, and spoofing, Trojan horses, [PATRICD, 1997].

The HIS (Human Immune system) has been solving similar problems for hundreds of million of years, using the algorithms that follows. The complexity mechanisms the HIS use to defend itself remain the area of research. However, the defence mechanism can be divided into specific or non-specific. The specific provides specialist protection against a known type of intrusion, just like the body reacting to measles. Non-specific provides a more general protection or fight against such conditions like skin and inflammation of the skin cells.

The comparative analogy is that a computer protection can be divided into specific (virus checking with signature, and security analysis tools), [PATRICK D] and non-specific (good code, hygiene, firewalls, encryption). These do not detect intrusions in progress; they stop no self from entering on the system, this means they sensor all unrecognized pieces of code from entering the system.

Lymphocytes or T-cells are part of a large population of specialized cells in the immune system. They are generated from the thymus and are covered with receptors, which bind antigens (foreign proteins). Each T-cell has specific kind of receptor it binds to a small group of structurally related antigens. The receptors are randomly generated and T-cells mature in the thymus where they under go a censoring process called negative selection. All those cells that do not bind self-protein are released into the body to become part of its defence.

T-cells that leave the thymus to circulate through the body are tolerant to self

(they do not attack the body). The concept presents an alternative paradigm to perform pattern recognition by storing the information about the complement set (non self) of the pattern to be recognized.

The algorithm hereby provided is adapted from L. Castro and J Timmis that focus on anomaly detection, time series prediction, image inspection and hardware fault tolerant.

The parameters self-set (**P**) Based upon the negative solution algorithm generate a set of detectors (**M**) that will be responsible to identify all elements that do not belong to the self-set. Which runs as follows generate random candidates (**C)** using the same representation adopted?

Compare (match) the elements in **C** with the elements in **P**. If a match occurs that is if an element of **P** is recognized by an element t of C, then discard this element of **C** in detector set **M**. After generating the set of detectors (**M**), the next stage of the algorithm consists in monitoring the system for the presence of no self-pattern. In this case, assume a set **P*** of pattern protected. This might be composed of the set **P** plus other new pattern, or it can be completely novel set. The algorithm will be revisited in chapter 3.

Pathogens can replicate into thousands in a short time, hence the need for an efficient system that can bind and eliminate these pathogens. Learning, adaptation and remembering structure of attacking proteins for future reference are some of the techniques used by the human Immune System (HIS) defense system.

B-cells mature in the borne marrow, When activated its thresh hold affinity is exceeded, and it produces copies of itself (clones as result of cell division). The copying is subject to mutation rates that are nine orders of magnitude higher than ordinary cell mutation rates; known as somatic hyper mutation, which can produce an offspring, B-cells with receptors different from both parents. The new B-cell will be capable of binding different types of pathogens. If the affinity to bind pathogens exceeds their threshold they will in turn clone. [Patrick D]

The competition now become apparent on the cells that reproduce the most to create a cell with a perfect match to the pathogen about to be destroyed, a concept known as Darwinian process of variation and selection also called affinity maturation.

Complementary to the role of negative selection, cloned selection is the theory used to explain how an immune response is mounted when a non self antigenic pattern is recognized by a B-cell. When a B-cell receptor identifies a non-self antigen with a certain affinity, it is marked for proliferation and produces antibodies in high volumes. Antibodies bind to antigens leading to eventual elimination by other immune cells.

1.9 The Memory Concept

The Immune system has an adaptive response that enables it to learn protein structures that characterize pathogens it encounters, and remembers those structures so that future reactive response is swift. Primary response is a

response mounted by the cells when they face a completely new attack or intrusion. This may take several weeks to be eliminated completely. The secondary response is a response mounted against a known attack. The system remembers a similar attack and uses similar techniques that were successful on a former attack to defend the body. The secondary response may be used against a slightly modified attack to the exact antigens that formally attacked the body.

1.10 Justification

According to the 2002, CSI/FBI computer Crime security Survey, the total revenue loss in industry due to intrusions was calculated at US$455 848 000.

The idea of using a GEA emanates from the belief that, different organisms, previously known by the body's immune system and unknown by the immune system attack the human body, but the body put an effective self-defense using the genetic defense system, allowing most people to survive up to 70 years or more under the immune system's protection.

We are motivated therefore to build computer systems algorithms with similar logic as those of the immune system. Since the immune system is complex and quite robust in nature, we choose and use selected features that may be compatible to be used in intrusions detect system.

GEA is biological inspired, self-regulatory, domain independent and has ability to automatically create a computer program from a high level statement

of a problem requirement. This characteristic is suitable for a system's defense in that it allows a system to recognize an attack, deploy counter measures and avoid a total collapse from the compromise.

1.11 Scope of Study

The research is being done under the guise of the discipline of theoretical computer science, with the mathematical basis of computing, as techniques for solving the research question or problem. We hope to analyse the existing genetic algorithm and demonstrate it to be correct or optimal in detecting intrusions.

The research shall rigorously cover Misuse detection system; hence, aspects of other type of detection will be reviewed in passing. We attempt to give statistical analysis, and concentrate on practical concerns such as execution time, storage space, communication, and the constraints imposed by hardware architectures, in direct relationship with the GEA.

The thesis investigates using a genetic engineering algorithm in a host-based platform. Though we use a particular operating System (OS) based platform, our ultimate goal is that the algorithmically solution should be run in all platforms without much modifications.

The second goal of the research is to use the well founded GEA in an attempt, to detect a substantial percentage of intrusions into the supervised GEA system, while keeping false positive and false negative at lower rate.

We borrow fuzzy logic where the concepts of GEA are complex, and difficult to Implement. Otherwise, we stick to the concepts of genetic engineering algorithm.

1.12 mechanism and functionality of GEA

GEA or interchangeably called GA were pioneered by (Holland 1975), they continuously breeds a population of computer programs over a series of generations. [KOZA99]. The technique is different from other approaches like AI, machine learning, neural networks, adaptive systems, reinforcement learning or automated logic in seven ways. It is based on the concept of nature of survival of the fittest.

The human body protects itself from antigens, through a mechanism of matching non-self (intruders, virus), and self co-habitants. The detectors are either nonspecific or specialized. The non-specific look for total strange invaders “novel” attacks the like of a computer attack, unrecorded before. The specific looks for well-known attacks, recurrence of the past.

1.13 Limitations of Existing Intrusion Detection Systems (IDS)

Most current Intrusion Detection System solutions generally implement an algorithm aimed at either Host Based or Network Based targets. When any anomalous behavior coming through is regarded as intrusion, despite the action being a legitimate one. This detection mechanism, accounts for a lot of

false positives, false negatives. The administrators tend to ignore systems alarm from such software and in the process more missed attacks, occur [JUN].

Little is known about the functionalities of the immune system; hence our algorithm may lack the most important component to run as efficiently as the immune system. The fact that only aspects that are interesting and show similarities are included in GEA detection algorithm, this introduces some incompleteness into the IDS. Very little is well understood about the functionalities and make up of immune systems hence less important aspects of detection mechanisms are likely to be incorporated into the algorithm, thus limiting the capabilities of detection.

1.14 Definitions of Terms and concepts

Throughout this Thesis, several terms will be used to describe features and components of IDS implementations. We explain the meaning of each as follows**:**

1.14.1 Single-System IDS – is an IDS architecture in which only one IDS system is implemented to monitor network activity. The system may be composed of multiple sensors and / or monitoring stations, but it is comprised of only a single type, brand, and model. This is currently a very common IDS model.

1.14.2 Disparate IDS – is a term to describe a security architecture in which multiple, different IDS systems are monitoring traffic. These systems each

have proprietary reporting and logging methods for handling suspicious activity, and the individual logs and reports must be managed and reviewed individually.

1.14.3 Distributed IDS – is a term to describe a security architecture of different IDS system types that all report to a single, centralized system. The reports are correlated, aggregated, and presented in a consolidated alert log format.

1.14.4 Network-based IDS – is a device that resides on a network segment and monitors traffic that traverses that network segment. The network-based IDS inspect each packet for anomalous (not matching standard patterns) or malicious (as defined by a signature set) traffic and report any traffic that it deems suspicious.

1.14.5 Host-based IDS – is similar in functionality to a network-based IDS, except that rather than watching traffic on the network, it monitors activity on a single host computer on which it is installed. Some host-based IDS systems actually monitor network traffic for the host and report suspicious traffic, while others monitor logs on the host on which they are installed and report anomalous log entries.

1.14.6 Firewall – is a packet filter. A firewall's main purpose is to deny network access to unauthorized traffic, and allow network access to authorized traffic. A firewall will usually contain a rule set against which it compares all

incoming traffic. From this rule set, each packet is determined as authorized or unauthorized, and the packet is either forwarded into the network, dropped, denied, reset, rate limited, or redirected. Properly configured, a firewall can enforce network policies, dramatically improving network security.

1.14.7 Classifier Expert System – is a device that takes input from several different devices (both network and host based, and potentially others), performs some processing on these inputs (i.e. correlation, aggregation, categorization, prioritisation, etc.), and then takes some action based on those inputs (i.e. logging to a database, notification, pre-programmed automated responses, etc.).

1.14.8 Network Security Manager – is a generic title for the individual within an organization who is responsible for that organization's network security. Generally, this is the person that configures the network security devices described in this list, and monitors their output.

Chapter II Literature Review

2.0 Introduction to Literature Review

Intrusion Detection systems protect important systems. Different versions of IDS monitor traffic and system activities. Many IDS were made for a particular OS and environment. The data collected provide the network and system security manager with invaluable insight into what traffic (both malicious and benign) is happening in the system and traversing the network. Valuable network and system activity intelligence that can lead to real-time (or near real-time) detection of significant network events, insight into network vulnerabilities and attacker techniques and procedures, can lead to evidence related to intrusion incidents and many other valuable network, enough to convince finance to upgrade a system or support the security manager to curb intrusions

2.1 Details of Existing IDS

Many commercial systems such Cisco, OS IDS Intrusion operating system, Opp-DIS application intrusion systems, are available in the market. However, the commercial algorithms are a patent product, which the host company does not intend to publish the init-grit of the algorithms used.

Some popular network IDS include commercial products, such as the Cisco Secure Intrusion Detection System (CSIDS, formerly NetRanger), ISS's RealSecure, and NFR's Network Flight Recorder. Popular host-based IDS tools include Tripwire, Symantec's Intruder Alert, and Intercepted by Entercept Security Technologies.

Several freeware network and host-based intrusion detection systems are available on the Internet, and they provide a comparable level of protection as their commercial brethren. To appreciate the functions of IDS, the freeware systems they are to be exploited and the results be presented as a perfect simulate of real world scenarios.

Snort systems are defined as "light weights Intrusion Detection System". By definition, lightweight IDS should have a small system footprint, provide for cross-platform support, and easy installation. Snort fits all three requirements. It utilizes the **libpcap** library (originally developed at Lawrence Berkeley Laboratory) for sniffing traffic and then analysing the packet payloads.

Snort is configured by command-line options as well as Berkeley Packet Filter commands. The heart of the Snort detection engine is a set of rules written in a simple language that allows for per packet tests and actions. Snort's detection rules can also be modified and extended by the end user.

There are three primary subsystems to Snort:

(i). Packet decoder,

(ii). Detection engine,

(iii). Logging and alerting system.

For the system to perform to optimum it needs adequate rule base, and as the rules accumulate resizing forward become apparent. A well-populated Rule-Base will easily pick up variety of violation. On the other hand, again a larger rule-base system

consumes lot more disk space and impact negatively on the speed of the system during rule search.

Search is a resource intensive process, and can slow down response time of the system. The use of subsystems make snort, a robust, large and monolithic system since It performs all of the monitoring, data gathering, data manipulation, and decision making for the whole system, [MARCK]. It can monitor system logs, user activities, and system state, seating on the system kernel. Snort then deduce metric systems, overall security and alert of intrusions.

Snort adds an overhead to the entire system, the large amount of data, it collects consume both disk space and CPU time.

The Linux Intrusion Detection System (LIDS) is a kernel patch for the Linux kernel as well as an admin tool for enhancing security. LIDS implements a reference monitor and mandatory access control in the Linux kernel. When LIDS is in effect, file access, system and network administration operations, raw device file access, and memory and I/O access can be made impossible, even for root. LID not only provides protection but detection as well.

Like PortSentry, LIDS can detect port scans against the host and notify the systems administrator. LIDS can also detect other access rule violations as well, and respond to any access rule violation. This response can either be through logging to `Syslog` or even terminating a user session. LIDS' flexibility makes it ideal as a host intrusion

detection and response tool. Unfortunately, an administrator usually views LIDS results when a user machine has been compromised already.

EMERALD (Porras and Neuman, 1997) is a complex system, and was developed by Stanford Research Institute, the design put emphasis on distributed tasks, and the acute need for a scalable solution and it is highly modular. Emerald employs the concept of hierarchical organization to achieve scalability. There are three tiers of components, operating on progressively lager portions of network and on higher level of abstraction.

Each service monitor, the lowest –tier component, which has a job to overseeing the operation of one service (e.g. FTP server), each of the monitor employ both rules rule-based, misuse detection and anomaly detection algorithms. The other monitors include Enterprise-level and Domain security.

Emerald is a disparate IDS system, which provides in-depth views into what types of traffic are traversing the network. Chances of detection are increased with disparate IDS implementations due to the fact that they apply different detection logic to their traffic analysis procedures.

However, disparate IDS implementations contain several different types of IDS systems, none of which will interoperate with others to provide a consolidated view of network activity. Each individual IDS system within the Disparate IDS architecture will create its own summary of network traffic and each summary must be reviewed separately and correlated with all other reports to provide the “total picture” of

network activity. This provides a potentially overwhelming amount of data to the network security administrator.

GEAs are mostly based on off-line learning algorithms. A data set is collected and manually labelled by an expert. Subsequently, a general purpose is evoked to induce the rules. The popular of them all is RIPPER (Cohen, 1995). RIPPER finds rules of the implicative form by generating a large number of candidate rules and evaluating directly on the data, the improvement they yield, which may be viewed as instances of genetic algorithm.

All classifications of IDS have varying degree of flaws in them, for example: Some solutions like Single-System IDS (both signature and anomaly-based) implementations only look for what they are programmed to look for. Many have a fixed “signature” set, or hard-coded logic as to what to look for. In the case of anomaly-based systems, logic flaws or oversights can potentially allow an attacker to slip by unnoticed.

NIDS algorithms only analyse the traffic traversing the network segment to which they are directly connected. If there is an alternate route into the network, open socket, a way to avoid detection by the NIDS, then an attacker can potentially enter and manipulate a network without being detected by the IDS.

HIDS algorithms only analyse events that are occurring on the individual host that they are monitoring, quite valuable on high-value assets such as servers because they are likely to be targeted by attackers. However, HIDS are potentially negative on system performance.

2.2 The Likeness of Artificial and HIS (Human Immune System)

In this chapter, we elaborate more in detailed overview the major mechanisms and properties of HIS. A complete detail of the functionalities of HIS is beyond the scope of this research and can be found in specialized texts. The ability to protect our bodies lies with the immune system, as it plays a role of eliminating dysfunctional endogenous cells, commonly called infectious self, and exogenous micro organisms (infectious non self) such as bacteria and virus which enters the body through many routes that include respiratory, digestive system and more commonly through damaged dermal tissues. In comparison to the Artificial Immune System (AIS) that we build in chapter 3 HIS plays an inspirational role for providing us with the fundamentals from which we imitate the immune system and build our own protective algorithm. Biological inspired computing is complex and we need a detailed understanding of the mechanisms of functionalities to build a realistic algorithm.

HIS is constituted of different podiums of layers, each layer stands in defence against infectious material or pathogens. The physical layer that comprises of skin, nasal hairs, and reflex actions such as coughing and sneezing blocks the ingestion of pathogens. The physiological layer comprises fluids secreted by the body, which is saliva, sweat and tears these are used as transport of pathogens out of the body, and as additional function, they dissolve the pathogens. The cellular layer is a set of cells, e.g. T-cells, B-cells and many more. The cell classification further subdivides into innate, specific, acquired or adaptive immune system. Innate is the defence system invoked by the body minutes or hours after an attack or infection, using non-specific responses. Adaptive immunity drags for longer times, usually days before becoming

effective. These require a specific response that is adapted to remove a specific pathogenic infection, and body allocates its defence in a controlled and dynamic way.

The following classification is identified as describing innate and acquired immune system. Leukocyte family commonly called the white blood cells is the root node of innate comprising of granulocytes, monocytes, and lymphocytes all of these originates from stem cells in the bone marrow. Granulocytes make up 50% -60% of all leukocyte family. Carry granules containing various chemicals and are fragmented into 3 groups namely neutrophils, eosinophils and basophils. Leukocytes,monocytes mature into macrophages, which play key roles in both innate and adaptive immune system responses. Macrophages locate and destroy pathogens. The two cells are collectively referred to as phagocytes.

2.3 How different is our Presentation

Most IDS are limited in scope on their capabilities to operations. For example an IDS specializing to protecting a network attack, may have a different focus with an IDS meant for a host based IDS.

The IDS operational environments differ greatly hence, the need to employ a variety of differing techniques for producing alerts. The signature alerts used in networked-based system should be different from host-based signature. Where these are similar, the underlying difference is just a smaller intersection. Most systems that do both types of detection suffer from anomalies of slowness, system overloads, and high failure rate in detecting intruders in real time. Another reason for these larger systems

failure is because they need to search larger reservoirs of signature on already over burdened system with other tasks as those of analysing the data and producing ideal findings.

In this dissertation, we present an effective tool that attempts to remove the signature reservoir requirements. The tool will produce detector strings probabilistically. Through negative selection all useful system data, programs, and software, binary strings will be called protected data (S) and be kept separately .A random set of strings will be produced(R), of which a one to many matching will be done a between the two sets. If a string in R is found to be matching a string in S. Then the string in R is discarded. Any string in R not matching a string in S will become a potential detector and will be stored in set M. All strings in M will be left to roam the system if they happen to match another string then that will be noted and reported as intruder and the string shall be destroyed.

To illustrate the hypothetical system we give the following illustration adapted from Forest et al.

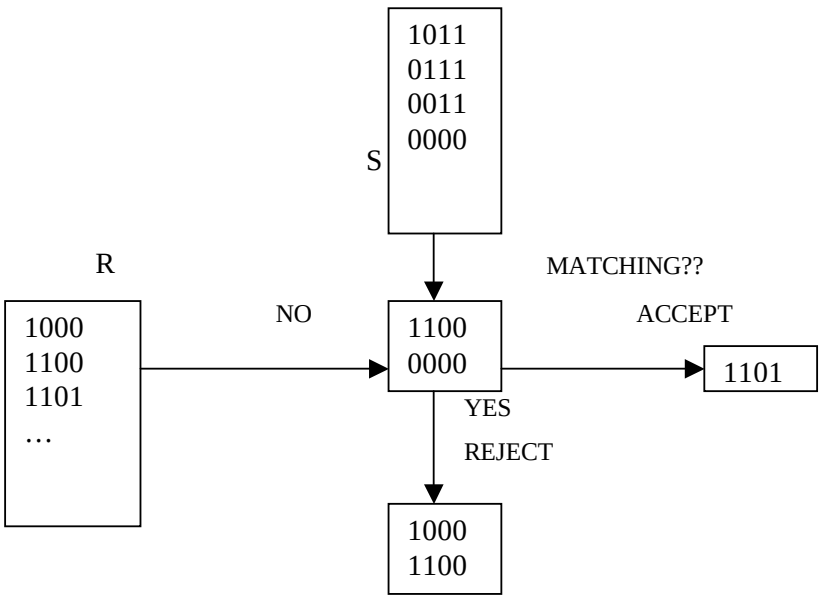


Fig 2 illustrates a Hypothetical Match, example, generating random strings

Hypothetically we can assume the entire system to be composed of one string of a certain length, $D=\{d_{1,}d_{2...}\ d_{n}\}$ to achieve a goal of running a scan again and again we deliberately subdivide that string into smaller strings. Our method however uses a contiguous matching technique. Instead of trying to match the entire length of the string, only a small portion of the binary string need be matched against the smallest strings. The merit of the system is to consider a string as matching if the contiguous string as defined by r matches, and then the entire string matches. In the above example contiguous match of $\Gamma=2$ was used.

The researchers have debated about the methods, on how to prevent an exponential growth of set S. Many limiting factors have been suggested mathematical formulae, and novelty mechanisms such as those ideas of likening the calculations of volume of a funnel and a windowing strategy. We do not cover the detailing aspects of these, as for the purpose of this dissertation; we will adopt original controlling formula by Stephanie Forrest [PATRICD 1997].

Pioneering ideas are based on rewarding a good performer. Strings or set of strings that performed well in detecting malicious code worms and virus, would be promoted through lengthening their life spun and leave the life spun of less performers constant, in some cases mutating all those with an average performance record. We suggest crossing over the record performers with the average and retaining both for further detection. Our deviation from the original idea, because we recognise the concept of polymorphism and metamorphism where the malicious code are built incrementally

with each successive coming iteration and successive or earlier code, introducing, a small change to the new virus.

A polymorphism intrusion tool will encrypt a malicious code and decrypt it during execution. To disguise its victims from identifying a virus, several transformations such as null insertions, comments, code transpositions, and use of GOTO statements are heavily used in decryption routines.

Metamorphic viruses attempt to invade heuristic detection techniques by using complex disguising methods. When they replicate the malicious system changes their code in a variety of ways. Such as code transpositions, substitution of equivalent instruction sequences, change of conditional jumps, and register reassignment. They can insert a malicious code right inside a host program. Since the beginning of the code is now hidden, the identification of such a worm, virus or malicious system becomes difficult by an ordinary IDS tool, which looks for signature of the intruder.

It is from such a behavioural analysis that we feel the negative selection ideology becomes hand. A hidden malicious code can be identified if the detector sample is larger enough. The mechanism of detecting using a GEA tool requires no prior knowledge of nature of intrusions in complete contrast with signature based virus scan. The detection is probabilistic that means we can generate as many times as needed detectors. The fact that we have formulae that control our generations we have a room for variability in how much detectors we can have depending on attack expectations. Detection is local which means focus can be directed to a small section of data to be checked and when an anomaly is identified, a proportionate solution can be mobilized. This aspects means the tool can find a virus in its earlier phases before propagation. The sets at each site may be unique, which means if one site were

compromised, other would still be protected. There is no communication between detectors or detector sets is needed until a change is detected.

Chapter III Methodology

In this chapter we describe in details the material and methods used. The immune system begins with a classifier system. Here the algorithm needs to classify the detectors that will be used in protecting the system from the attackers. The classification is very essential because we are dealing with very large generation of agents.

When the number of the elements is large, characteristics of the cascade depend more on its structure than on the quality of an individual separating element (classifier). The search of the optimal structure becomes a difficult task because of a large number of possible variants of interconnection between the elements. The genetic method of synthesis allows us to determine a nearly optimal structure of a classifier cascade. Particularly, a set of non-trivial structures need synthesized, and excluded without compromising the detection ability of detectors.

Most of algorithms that optimize a search space are NP-Complete as shown by Garey and Johnson (1979). However the GEA heuristics try to keep the search space within the polynomial time. GEA works similarly to an abstract automation running sequentially through a set of states (generations) until termination criteria ***t*** holds. The term generations in this thesis refer to time steps between successive states as well as to the population ***pt*** at time ***t***.

The important part in using a GEA effectively is in choosing an appropriate representation. The used representation must be appropriate and minimal and completely expressive. The defined representation must be able to represent a solution

in the set of solutions sought to a given problem at one time. (Robustness). The designed representation should avoid infeasible solutions being included in the solution set.

Genetic makeup contains characteristics that go beyond our scope of study. We hence include information that is minimal to represent our solution set to our problem of detecting of intruders. The inclusion of several genetic factors has the usual draw back of increasing the search space and reducing the algorithms performance.

The complexity of using numeric representation such as array of real numbers far outperforms most other known representations. The difficulties is how to choose a cross over operator that generate reordered list without duplicating an element in the list, Another challenge is mixing of contiguous and discrete elements in which case it may be desirable that a new structure to hold the mixed information be created. For example a solution with both integer and floating parts might require the use of a cross over that cross integer part and floating parts, but taking care that floating parts and integer parts never mixes.

We use the pure binary representation for our solution for simplicity, and as a standard computer data representation. The universe of binary strings is rich enough to allow us to study how a relatively small number of recognizers (detectors) can evolve to recognize a much larger number of different patterns (intruders).

For our experiments, we use test data taken from repositories and dumps collected from specialized sites that collect data when the actual attacks were in progress, in

different locations. The data colleted forms the input to our algorithm's test data, from which comparisons and statistical efficiencies will be given. Comparisons done using historical figures and similar findings above will form our observable conclusions and findings.

We calculate standard deviations, and mean to ascertain how much we deviate from others findings that used the data for their experiments using different methods of classifications. Additional statistical tools may be used to measure the effectiveness of system performance to match detectors (Regression analysis). Algorithms performance analysis in terms of running time and space complexity will be verified using standard techniques of the discipline of design and analysis of algorithms.

Regression analysis will be used to identify how correlated documented intrusions and perceived intrusions compare this means that intrusions in the past studies and intrusions as identified by our genetic algorithm.

Our emphasis is exploring the theoretical basis of our method and addressing the important question of practicality including, the feasibility of generating and matching detectors, and discussing the implications for the real world problems The proposed solution is a prototype intrusion detection Algorithm (IDA) that use a genetic algorithm, to investigate how a group of free running processes which are acting independent but cooperative of each other, just like the human cells, can flag and identify each behaviour they consider to be anomalous.[MARKC]

In building the system, the following assumptions have been taken into cognisance: Used Algorithms are observable, via system auditing mechanisms; Normal and intrusion activities have distinct indicators. Intrusion detection hence involves capturing audit data and derives the evidence from the data to predict whether the system is under attack. The use of mathematical parameters as basis of the algorithm is our main instrument of operation.

3.1 Components of the Genetic Algorithm

1. A representation for a real potential problem
2. A way to create an initial population of potential solution (this is done randomly).
3. An evaluation function that plays the role of environment, rating solutions in terms of their “fitness”.
4. A genetic operator that alter composition of the children (selection, mutation, crossover)
5. Values for various parameters that the genetic algorithm uses (population size, probability of applying genetic operators).

The first aspect to this process is the encoding (representation) of solutions as chromosomal strings that GA can evolve.

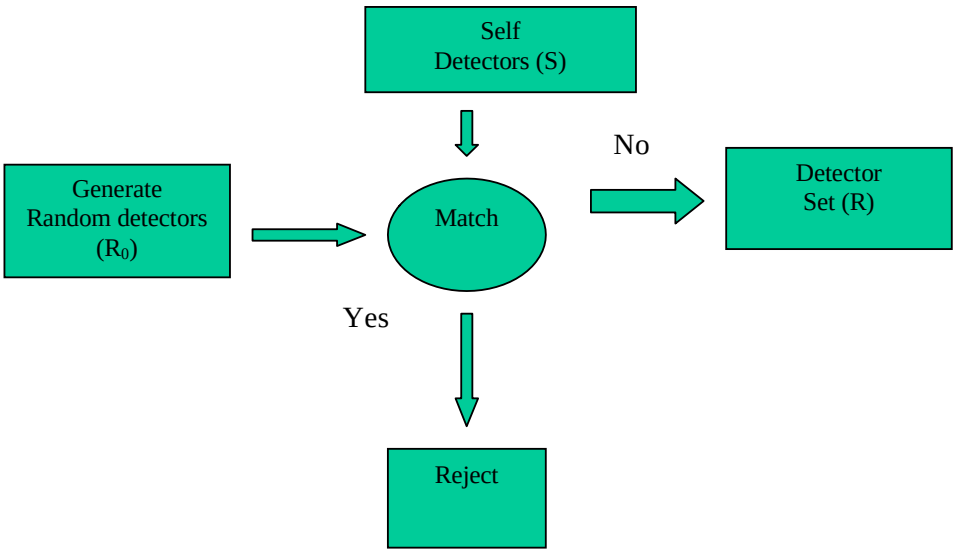


Fig 1 on the above diagram we present the preliminaries of a GEA algorithm, the diagram illustrate the censoring part adapted from forest et al, which also stands for a classifier.

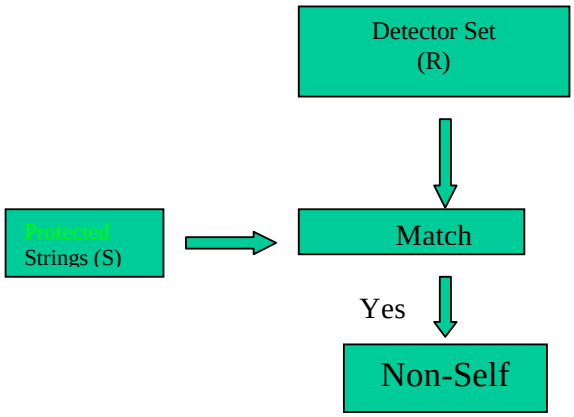


Fig 2 illustrates the second part of the algorithms that does the monitoring part it Follows from the fig 1.

3.2 The philosophy of Approach

The following fundamentals will steer our approach

- We deploy a unique detection algorithm copy for a given site. Where detection is required, a complete regeneration of detectors has to be produced.
- The detectors will be generated probabilistically and this will remove component of multiple storage for each phase of detection.
- Novel attacks are covered in our detectors, since the system produces detectors that will potentially match even a new pattern. The probabilistic method contrast significantly with signature based techniques, which use a known signature as a basis of matching attacks.

The algorithm in fig 2 generates a set of potential detectors. Each detector is a binary string that does not match any protected data. The protected data comprises software code, system files, and all the useful staff that we have stored in a machine. This phase is commonly known as the censoring phase.

The mechanism of protecting data is achieved by comparing them with the detectors. If a detector reacts with the protected data then we discard that detector as not useful. When a detector fails to match self (protected data), we accept that string as a potential detector, and pass it to fig2. Censoring is also known as Negative Selection Technique of intrusion detection science.

3.3 Matching

We give an overview of how our technique works, before moving into implementation details. We assume that a set of string, (care should be taken in interpreting the word set in the context that is used here, the word set is simply a group of strings without any mathematical rigor as is in mathematical set) in our set we have duplicates.

The self-string is an unordered collection of strings or concatenated sub-strings. To generate valid detectors we split self-strings (logically) into equal size segments. We firstly split the string to facilitate a mathematical analysis of the system, which will allow us to determine the probability of detection.

3.4 Consider As Illustration

Given a 32-bit string, break it into 8 sub strings, each of length four:

0010

1000

1001

0000

0100

0010

1001

0011

i. S ={collection of all self set (sub strings) to be protected}

ii. R_o ={Random strings}

iii. Match $\{R_i \mid S_i\}$ =Boolean Valid or Invalid Any string in R matching string in S is eliminated.

iv. R ={composed of strings that were invalidated iii mechanism also known as Repertoire}

Suppose $\mathbf{R}_o$ ={0111,1000,0101,1001} **∴ R ={0111,0101}**

The strings 1000, 1001 are automatically eliminated they are members of S.

With the R set populated with the collection of strings. A control mechanism has to be put in place that will test self occasionally by re-matching S against R to detect a hazardous mutation of intrusion in R and S respectively.

We decided to pair the matching alphabets deterministically, each string being chosen for matching in a fixed order, and the detectors being checked in the order they were produced. The reason we do this is to control the process of population generation to avoid an exponential growth. Randomization is another used technique though for the purpose of this thesis we stick to the fixed order given above.

A match of $X^1_{a\ldots n}$ and $X^2_{a\ldots n}$ strings, to be exact need be of equal length, and have equal alphabet at each location in the string. The difficulty with exact matching is that it is a rare scenario, especially when matching strings of relatively longer length.

3.5 Partial Matching

The commonly used matching is the contiguous r matching technique. The method scans for r contiguous matches between symbols in corresponding positions. For any two strings t and p, we say that match (t | p) is valid if p and t agree (match) at least r contiguous locations. The advantage of the rule is that this matching rule can be applied to strings defined over any alphabet of symbols. For our thesis, the string is defined over the alphabet {0,1}, representing any bit pattern that can be stored in a computer.

3.5.1 We illustrate a censoring or a partial matching of r=3

t ={247346289}

p={571**346**989}

The two strings, t, p defined over the 9-letter alphabet {2,3,4,5,6,7,8,9} match four contiguous locations underlined. Thus match (p|t) is invalid for r=3 or match (p|t) is true for r=4 or less.

In general two random string matches at least Γ contiguous location if the Probability

$Y_m \approx m^{-r}[(l-r)(m-1)/m+1]$.

Where

m = the number of alphabet symbol s

l or ⌊= the number of symbols in a string) and

Γ = The number of contiguous matches required for a match.

The test will come in form of varying the variables (Γ, ⌊, m, $Y_{m)}$ to test the ranges that will allow a room of conducting varying analysis across the performance of our algorithm.

According to Forrest the approximation is good when if the variable m^{-r} <1. Where the approximation fails, we use exact formulae.

All discrimination between self and non-self in the Immune System is based upon chemical bonds that form between protein chains. To preserve generality, we model protein chains as binary strings of fixed length *l*. The Immune System must distinguish self from non-self based on proteins. The set of strings of length *l* form the universe, *U*, which is partitioned into two disjoints subsets which we call Self, *S* and non-self, *N*. Formally *U*=SUN, $S \cap N$ =0. Given arbitrarily strings from *U*, classify it as either normal (corresponding to self) or anomalous (corresponding to non self).

If *r*=*l* the matching is completely specific that means the detector will only match a single string (Itself). If *r*=0 the matching is completely general the detector will match every single string of length *l.*

3.6 Estimating Probability of Detection

Good range of detectors can be obtained if our estimated probabilities numbers are accurate. The following description outlines how the predictions are arrived at.

Consider that the strings to be protected $S_{0,}$ are application programs, some data, or any other elements of the computer system that is stored in memory. Using the above detailed algorithm we need to:

i. Estimate the number and size of detector strings, which will be required to ensure that an arbitrary change to the protected string is detected with some fixed probability. The following definitions are given

ii. N_{R0} = The number of initial detector strings before negative selection

iii. N_R = Number of Detector string after censoring (size of potential detector)

iv. N_s= Number of self strings

v. Y_m = the probability of a match between two random strings.

vi. f= The probability of a random string not matching any of the N_s self strings

$=(1- Y_m) N_s$

vii Pf =probability that N_R detectors fail to detect an intrusion.

If Y_m is small and N_s is large then

$$f \approx e^{-Y_m N_s}$$

and

$$N_R = N_{R0} \times f \qquad (1)$$

$$Pf = (1-Y_{m})^{N_R} \qquad (2)$$

If Y_m is small and ^{N}R is larger, then $Pf \approx e^{-Y_m N_{R.}}$

$$N_R = N_{R0} \times f = -\ln Pf / Y_m \quad (3)$$

Solving 1 and 2 for N_{R0}, we get the following:

$$N_{R\,0} = \frac{-LnPf}{Y_m} \times \frac{1}{(1-Y_m)^{N_s}}$$

The given formula allows us to predict the number of initial strings (N_{R_0}) that will be required to detect a random change.

As f (x) of Y_m, (1-P_f), The number of self-strings (N_s) being protected and the matching rule (Y_m). Then $N_{R\,0}$ is minimized by choosing a matching rule such that

$$Y_m \approx \frac{1}{N_s}$$

3.7 Algorithms Performance

We delay for now giving details of the performance of our algorithms in terms of cost on computational resources, runtime, performance on comparisons; space and time complexities are all deferred for now. However a careful observation should discriminate liable areas of analysis. Namely, when generating random strings of fixed length and when doing comparisons at a censoring stage and real detection.

3.8 Detector Size

To determine the number of size to achieve a specific failure rate Pf is an important part of measuring what affects time delay, and storage space when generating

detectors. Theoretical Lower bounds measurements can be obtained using specific equations.

According to [Peres 79], a first lower bound for N_R from the average matching probability Y_m The best case would be to distribute potential detectors so that no two detectors can match the same no-self string. All detector set should cover an approximate space of N_R. This implies

$$N_R >= (1-Pf) / Y_m$$

3.9 Sub Algorithms

As illustrated above on fig1 and 2 respectively. Our algorithm is a combination of several algorithms. It would be impractical cumbersome and complex to give a one rundown of a huge algorithm to accomplish the task.

Genetic algorithms are a robust set of solutions that change form with the problem solution being sought. For example Selection is a method of fitness evaluation of genes. , And comes in many forms.

Prepositional selection also called the roulette wheel selection is derived from the fact that fitness can be ascertained from dimensions of sectors representing individuals in a population. The fitness is thus calculated over x representing an individual score over the sum of the entire population. Practically the formula becomes

$$P(x) = \frac{\delta(x)}{\sum y\delta(y)} ,$$ where P(x) is the fitness probability.

3.9.1 Boltzmann selection

Under this selection method, exponentiation rescaling of proportional selection is used. Proportional selection on $e^{ß f(x)} / \Sigma_y e^{ß(y)}$. The strength of method is controlled by ß parameter.

Some additional method under the same group are Ranking selection, Tournament selection, Truncation selection. However a combination of two or one more selection method can be applicable at any one time.

3.10 Holes

For the Γ contiguous bits matching rule bit technique, there exist other non-self strings known as holes. For these holes it is not possible to generate valid detectors.

Consider two binary strings t_1 and t_2 that matches over Γ-contiguous bits, they may create additional two bits y_1, and y_2 that are not detected because candidate detectors match either t_1 or t_2 as an example of strings with Γ=4

t_1 è 00101000

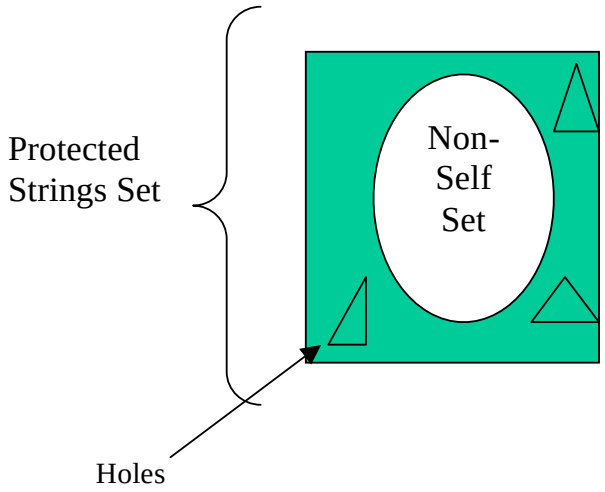
t_2 è 11101011

↓ ↓

y_1 è 00101011

y_2 è 11101000

Another example of a “Set” inducing hole is a set up given that the number of Non-self strings is smaller than the number of strings matched by the detectors.



Protected Strings set is larger than the potential detectors Non-self set, as illustrated above. Fig 3.1

Holes reduce the effectiveness of the intrusion systems objectives, by opening up possibility of intrusions. The non-self potential detectors offer inadequate cover. The problem of holes can be reduced through using matching rules where Γ-contiguous bit rules are different. For instance Γ=1 (detectors matches entire string space), Γ=⌊ detector matches a string (itself). Potential holes are then eliminated by using closer and potentially more specific match as opposed to scanning a smaller Γcontiguous positions, string, and string length ⌊ over the possibility of a match

3.11 Linear Time algorithm

It can be verified that most candidate detectors are rejected. That has no overhead when done by the human body during censoring. However it becomes rather ineffective when a computer is involved.

As previously stated, two ⌊-bit strings match each other if they are identical in at least r contiguous Γ positions that runs in linear time with respect to size of input.

The data structure used for this algorithm consists of (⌊-Γ)×$2^{r.}$ Array representation covering all possible ways two strings may combine over Γ contiguous bits.

Hence the running time can be thought of as

1. Ω((⌊-Γ)*N_s initializing for those entries in N_R that match a self string.
2. Plus Ω((⌊-Γ)*2^r time to recursively fill in the rest of the array.

The time complexity of this algorithm was derived based on two factors

i. Time to generate a number of candidates (N_R)

ii. Time to compare each with one of them with self ($N_{s)}$

The space complexity depends on the self-population, whose individuals are of length ⌊

Chapter IV Findings and Conclusions

4.0 Information loss

Since binary strings, representing data in the computer have to be split into Γ-contiguous bits according to the agreed threshold. We find that as bits are split across, no consideration is assumed of the existence of unique bit strings s_i, which fosters a loss of information through this division.

As an example, assume $N_{s,}$ and these sets of k unique strings in S (s_1, s_2, …s_k). they are $\left[\frac{(N_s - 1)!}{(k!, (k+1)!, \ldots (k+n)!} \right]$

Being possible combinations of ways of assigning the value $N_{i...s}$ which if recombined form S perfectly.

But since we are using the alphabet {0,1} then the above can be written to

$$\text{Log}_2 \left[\frac{(N_s - 1)!}{(k!, (k+1)!, \ldots (k+n)!} \right]$$

In general, we tend to ignore the change k in $N_{s,}$ since as it can be inferred on the second equation; the value of information loss is apparently small.

Larger strings of ⌊, tend to reduce the number of duplicate strings, an subsequently reducing the amount of information cost due to string splitting.

4.1 Relationship between Protected Sets and Failure Probability

We experimented with varied lengths and an arbitrary incremented value of failure probability as illustrated a lower bound for N_R as a function of increased length of the binary strings as follows:

Length l	N_S	"N_R (Pf=0,2)"	"N_R (Pf= 0,02)"
12	100000	160934	391202
14	95401	153542	33928
16	51607	83058	201888
22	50000	80442	195601
26	43103	69372	168620
28	41000	65987	160393
30	30902	49735	120889
32	27940	44968	109302
38	23811	38322	93149
42	18971	30533	74215

Table 1 above is an experimental relationship between string lengths and protected sets of strings against the failure probabilities $\mathbf{p}f$

Using a formula $\mathbf{Y_m} = \frac{1}{N_s}$

We calculated N_R, for value 0.02 and 0.02 an increase of ten percent to failure probability $\mathbf{p}f$; the detector numbers grew a 100% only. We make assertion that if the figures grew exponentially then we could conclude on the infeasibility of the solution set, being used on a modest environment like a PC, Laptop etc. The fact that the detector numbers only doubled it implies smaller detectors as well as larger detectors still can be used to detect anomalous behaviours much the same way. The detection ability of detectors is independent of the size of detectors numbers.

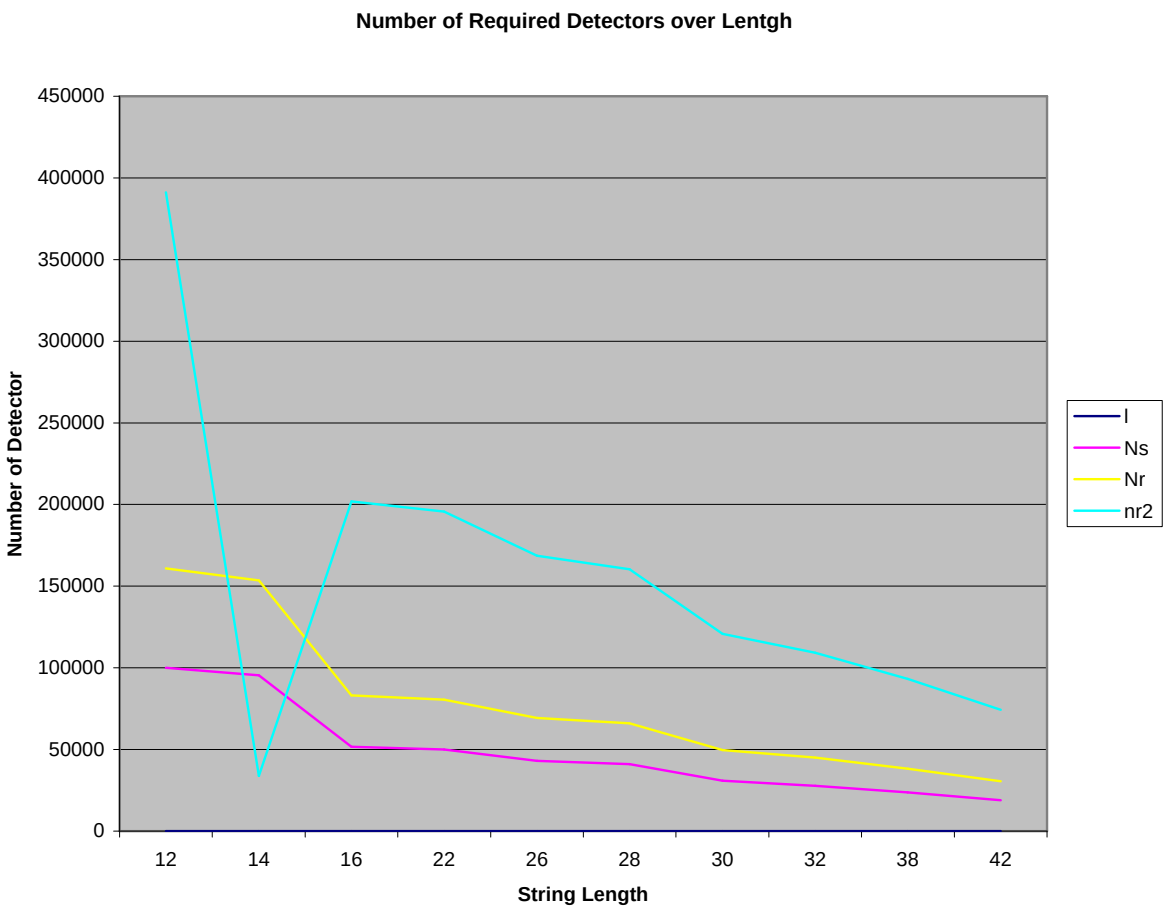


Fig 5 illustrates a graphical representation of the Relationships between various binary string lengths and failure probabilities.

Above the graphed scenario, illustrate the relationship between string lengths ⌊ and nr2 (nr2= N_{R0}). N_{R0} represents potential detectors before censoring phase, and nr. (nr = N_R) representing detectors after censoring. When strings lengths reduce in size by a larger dimension then we have numerous holes. Holes as defined earlier are pieces of strings that have both characteristics of self and non-self.

The observation is that as the string gets bigger in the direction of Γ, we generally require considerably fewer detectors to representing self to match non-self. In other words we can control our detection capacity and still enjoy a reliable protection.

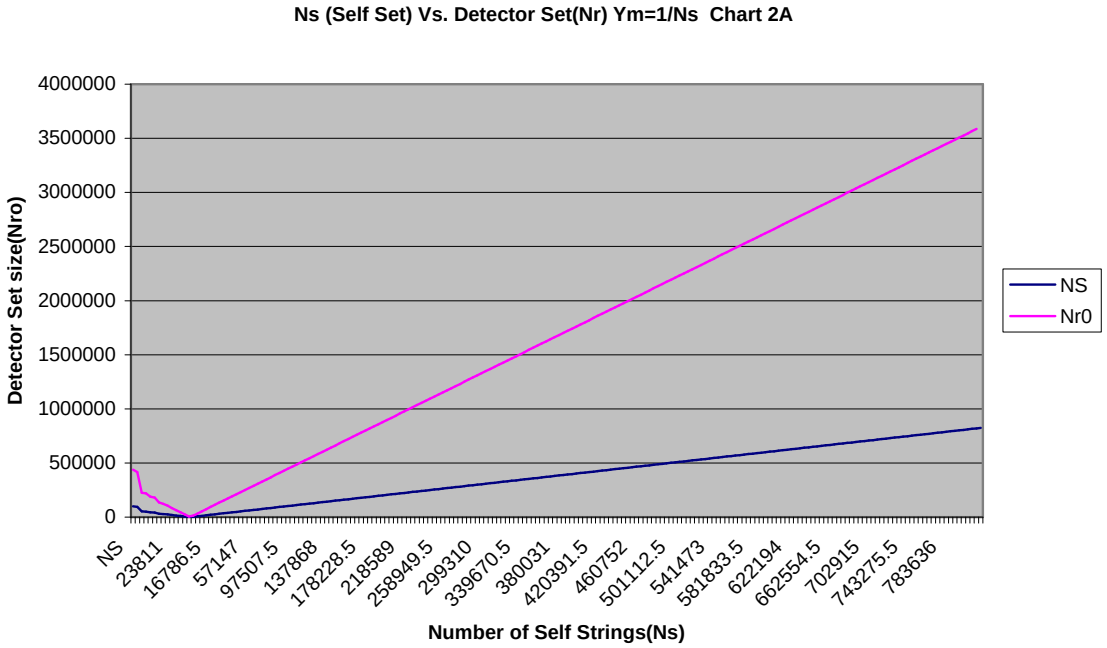


Fig 10 shows a population of detectors graphed against a set of protected string using a fixed probability of matching using the equation $1/N_s$. N_s on X-axis and a set of detector strings on Y-axis which were generate using equation N_{R0}

$= \frac{-LnPf}{Y_m} \times \frac{1}{(1-Y_m)^{N_s}}$ Despite N_{R0} values being shown fractional, the assumption is that we are working with rounded numbers, the solutions were included for completeness sake.

The Following values were used

N_S	Y_m	N_{r0}
1151.5	0.000868432	5039.893261
3333	0.00030003	14583.74919
5636	0.000177431	24659.15707
7817.5	0.000127918	34203.01405
10120.5	9.88093E-05	44278.42212
12302	8.12876E-05	53822.27915

14605	6.84697E-05	63897.68726
16786.5	5.95717E-05	73441.54431
18971	5.2712E-05	82998.52609
21271	4.70124E-05	93060.80949
23811	4.19974E-05	104173.0703
25755.5	3.88267E-05	112680.0747
27940	3.5791E-05	122237.0565
30240	3.30688E-05	132299.3399
30902	3.23604E-05	135195.5275
34724.5	2.87981E-05	151918.6051
39209	2.55043E-05	171537.8702
41000	2.43902E-05	179373.3266
43103	2.32002E-05	188573.7536
43693.5	2.28867E-05	191157.1354
48178	2.07564E-05	210776.4006
50000	0.00002	218747.4791
51607	1.93772E-05	225777.9527
52662.5	1.89888E-05	230395.6658
57147	1.74987E-05	250014.931
61631.5	1.62255E-05	269634.1962
66116	1.51249E-05	289253.4614
70600.5	1.41642E-05	308872.7266
75085	1.33182E-05	328491.9918
79569.5	1.25676E-05	348111.257
84054	1.18971E-05	367730.5222
88538.5	1.12945E-05	387349.7874
93023	1.075E-05	406969.0526
95401	1.04821E-05	417372.5787
97507.5	1.02556E-05	426588.3178
100000	0.00001	437492.7706
101992	9.80469E-06	446207.583
106476.5	9.39174E-06	465826.8482
110961	9.01218E-06	485446.1134
115445.5	8.6621E-06	505065.3786
119930	8.3382E-06	524684.6438
124414.5	8.03765E-06	544303.909
128899	7.75801E-06	563923.1742
133383.5	7.49718E-06	583542.4394
137868	7.25331E-06	603161.7046
142352.5	7.02482E-06	622780.9698
146837	6.81027E-06	642400.235
151321.5	6.60845E-06	662019.5002
155806	6.41824E-06	681638.7654
160290.5	6.23867E-06	701258.0306
164775	6.06888E-06	720877.2958
169259.5	5.90809E-06	740496.561
173744	5.75559E-06	760115.8262
178228.5	5.61077E-06	779735.0914
182713	5.47306E-06	799354.3566
187197.5	5.34195E-06	818973.6218
191682	5.21697E-06	838592.887
196166.5	5.09771E-06	858212.1522
200651	4.98378E-06	877831.4174
205135.5	4.87483E-06	897450.6826
209620	4.77054E-06	917069.9479
214104.5	4.67062E-06	936689.213
218589	4.5748E-06	956308.4782

223073.5	4.48283E-06	975927.7435
227558	4.39448E-06	995547.0087
232042.5	4.30956E-06	1015166.274
236527	4.22785E-06	1034785.539
241011.5	4.14918E-06	1054404.804
245496	4.07339E-06	1074024.069
249980.5	4.00031E-06	1093643.335
254465	3.92981E-06	1113262.6
258949.5	3.86176E-06	1132881.865
263434	3.79602E-06	1152501.13
267918.5	3.73248E-06	1172120.395
272403	3.67103E-06	1191739.661
276887.5	3.61158E-06	1211358.926
281372	3.55401E-06	1230978.191
285856.5	3.49826E-06	1250597.456
290341	3.44423E-06	1270216.721
294825.5	3.39184E-06	1289835.987
299310	3.34102E-06	1309455.252
303794.5	3.2917E-06	1329074.517
308279	3.24381E-06	1348693.782
312763.5	3.1973E-06	1368313.047
317248	3.15211E-06	1387932.313
321732.5	3.10817E-06	1407551.578
326217	3.06544E-06	1427170.843
330701.5	3.02388E-06	1446790.108
335186	2.98342E-06	1466409.373
339670.5	2.94403E-06	1486028.639
344155	2.90567E-06	1505647.904
348639.5	2.86829E-06	1525267.169
353124	2.83187E-06	1544886.434
357608.5	2.79635E-06	1564505.699
362093	2.76172E-06	1584124.965
366577.5	2.72794E-06	1603744.23
371062	2.69497E-06	1623363.495
375546.5	2.66279E-06	1642982.76
380031	2.63136E-06	1662602.025
384515.5	2.60068E-06	1682221.291
389000	2.57069E-06	1701840.556
393484.5	2.5414E-06	1721459.821
397969	2.51276E-06	1741079.086
402453.5	2.48476E-06	1760698.351
406938	2.45738E-06	1780317.617
411422.5	2.43059E-06	1799936.882
415907	2.40438E-06	1819556.147
420391.5	2.37874E-06	1839175.412
424876	2.35363E-06	1858794.677
429360.5	2.32905E-06	1878413.943
433845	2.30497E-06	1898033.208
438329.5	2.28139E-06	1917652.473
442814	2.25828E-06	1937271.738
447298.5	2.23564E-06	1956891.004
451783	2.21345E-06	1976510.269
456267.5	2.1917E-06	1996129.534
460752	2.17036E-06	2015748.799
465236.5	2.14944E-06	2035368.064
469721	2.12892E-06	2054987.33
474205.5	2.10879E-06	2074606.595

478690	2.08903E-06	2094225.86
483174.5	2.06965E-06	2113845.125
487659	2.05061E-06	2133464.39
492143.5	2.03193E-06	2153083.656
496628	2.01358E-06	2172702.921
501112.5	1.99556E-06	2192322.186
505597	1.97786E-06	2211941.451
510081.5	1.96047E-06	2231560.716
514566	1.94339E-06	2251179.982
519050.5	1.92659E-06	2270799.247
523535	1.91009E-06	2290418.512
528019.5	1.89387E-06	2310037.777
532504	1.87792E-06	2329657.042
536988.5	1.86224E-06	2349276.308
541473	1.84681E-06	2368895.573
545957.5	1.83164E-06	2388514.838
550442	1.81672E-06	2408134.103
554926.5	1.80204E-06	2427753.368
559411	1.78759E-06	2447372.634
563895.5	1.77338E-06	2466991.899
568380	1.75939E-06	2486611.164
572864.5	1.74561E-06	2506230.429
577349	1.73205E-06	2525849.694
581833.5	1.7187E-06	2545468.96
586318	1.70556E-06	2565088.225
590802.5	1.69261E-06	2584707.49
595287	1.67986E-06	2604326.755
599771.5	1.6673E-06	2623946.02
604256	1.65493E-06	2643565.285
608740.5	1.64274E-06	2663184.551
613225	1.63072E-06	2682803.816
617709.5	1.61888E-06	2702423.081
622194	1.60722E-06	2722042.346
626678.5	1.59571E-06	2741661.612
631163	1.58438E-06	2761280.877
635647.5	1.5732E-06	2780900.142
640132	1.56218E-06	2800519.407
644616.5	1.55131E-06	2820138.672
649101	1.54059E-06	2839757.937
653585.5	1.53002E-06	2859377.203
658070	1.5196E-06	2878996.468
662554.5	1.50931E-06	2898615.733
667039	1.49916E-06	2918234.998
671523.5	1.48915E-06	2937854.264
676008	1.47927E-06	2957473.529
680492.5	1.46952E-06	2977092.794
684977	1.4599E-06	2996712.059
689461.5	1.45041E-06	3016331.324
693946	1.44103E-06	3035950.589
698430.5	1.43178E-06	3055569.855
702915	1.42265E-06	3075189.12
707399.5	1.41363E-06	3094808.385
711884	1.40472E-06	3114427.65
716368.5	1.39593E-06	3134046.916
720853	1.38725E-06	3153666.181
725337.5	1.37867E-06	3173285.446
729822	1.3702E-06	3192904.711

734306.5	1.36183E-06	3212523.976
738791	1.35356E-06	3232143.242
743275.5	1.3454E-06	3251762.507
747760	1.33733E-06	3271381.772
752244.5	1.32936E-06	3291001.037
756729	1.32148E-06	3310620.302
761213.5	1.31369E-06	3330239.568
765698	1.306E-06	3349858.833
770182.5	1.29839E-06	3369478.098
774667	1.29088E-06	3389097.363
779151.5	1.28345E-06	3408716.628
783636	1.2761E-06	3428335.893
788120.5	1.26884E-06	3447955.159
792605	1.26166E-06	3467574.424
797089.5	1.25456E-06	3487193.689
801574	1.24755E-06	3506812.954
806058.5	1.2406E-06	3526432.219
810543	1.23374E-06	3546051.485
815027.5	1.22695E-06	3565670.75
819512	1.22024E-06	3585290.015
823996.5	1.2136E-06	3604909.28

The N_s set was generated probabilistically from different sets and sorted as per above illustration on Table 3.

4.2 Experimenting with Y_m and Pf as constants.

i. Assume N_{r0} is independent of $N_{s,}$ and if Y_m and pf are fixed values then the detector set should not grow dependently with Self-set. This implies that fewer numbers of detectors are likely to protect a larger number of self-sets.

ii. Exponential growth of N_{r0} and N_s can only occur if N_R, pf, and Y_m are fixed to some specific values. The difficulties being that the value become unusable in an environment of PC computers. The benefits of such a large detector is that it is apparently difficult for an intruder to modify self-elements and a detector without being noticed.

iii. The price of exponential growth is a completely secure system in a large computer environment and minimal or no protection at all on a smaller machine due to resources constraints. When space is used up then speed is affected negatively.

4.3 Experimenting with Theoretical N_{R0} and Experimental N_{R0}

We experimented with a fixed Nr for more than 500 trials The N_R for this experiment was fixed to 45 and the theoretical Pf was calculated using the equation $Pf=(1-Y_m)^{N_R}$.

N_s	Y_m	Pf	N_R	Experimental N_{ro}	Theory N_{ro}	Experiments Pf		
8	0.125	0.002456758	45	139.9012049	130.9628416	0.367879441		m=2
16	0.0625	0.054790774	45	130.5001719	126.3781785	0.367879441		l=32
22	0.045454545	0.123267227	45	128.159127	125.2238372	0.367879441		Γ =8
30	0.033333333	0.217497086	45	126.5467656	124.425736	0.367879441		N_R =45
38	0.026315789	0.301172792	45	125.6324235	123.972018	0.367879441		Pf =0.5
46	0.02173913	0.371929558	45	125.0434809	123.6793339	0.367879441		
54	0.018518519	0.431216073	45	124.6324718	123.4748723	0.367879441		
64	0.015625	0.492295534	45	124.2655963	123.2922232	0.367879441		
74	0.013513514	0.54212735	45	123.9992919	123.1595589	0.367879441		
82	0.012195122	0.575708302	45	123.8336157	123.0769885	0.367879441		
90	0.011111111	0.60483564	45	123.6977379	123.0092485	0.367879441		Pf =(1-Y_m) power of Nr
98	0.010204082	0.630310636	45	123.5842825	122.9526727	0.367879441		Y_m =I/Ns
106	0.009433962	0.652761484	45	123.4881225	122.9047111	0.367879441		
114	0.00877193	0.67268485	45	123.405583	122.8635357	0.367879441		
122	0.008196721	0.690477107	45	123.3339617	122.8278012	0.367879441		
130	0.007692308	0.706457613	45	123.2712264	122.7964961	0.367879441		
138	0.007246377	0.720886071	45	123.2158201	122.7688449	0.367879441		
146	0.006849315	0.733975545	45	123.1665294	122.7442431	0.367879441		

Fig 7 is illustrating the protection capabilities of detectors protecting N_s of 146 and when N_R is kept constant.

The full detail of the equation being as down listed (i) to (vi)

(i) $Pf=(1-Y_m)^{N_R}$

(ii) $Y_m=1/N_s$

(iii) $N_R=45$

(iv) The alphabet composition being m=2 for binary {0,1}, ⌊=32, and Γ=8

(v) N_s = Random generated strings all round

(vi) The Random strings were generated until valid binary N_R strings detectors were confirmed.

4.3.1 Observed Results

There was a closer agreement between theories and practical in that for the 138 self-strings they could be protected by a mere 45 strings with a failure rate of 73%. 73% is quite a substantial failure that brings out the infeasibility potential of negative selection when applied to a smaller number of self-string. Though befits are realized when the repertoires are allowed to grow exponentially to protect even smaller self. Ratios of experimental and theoretical as described in this experiment show minor differences; otherwise they all produce a modest number of actual detectors N_R

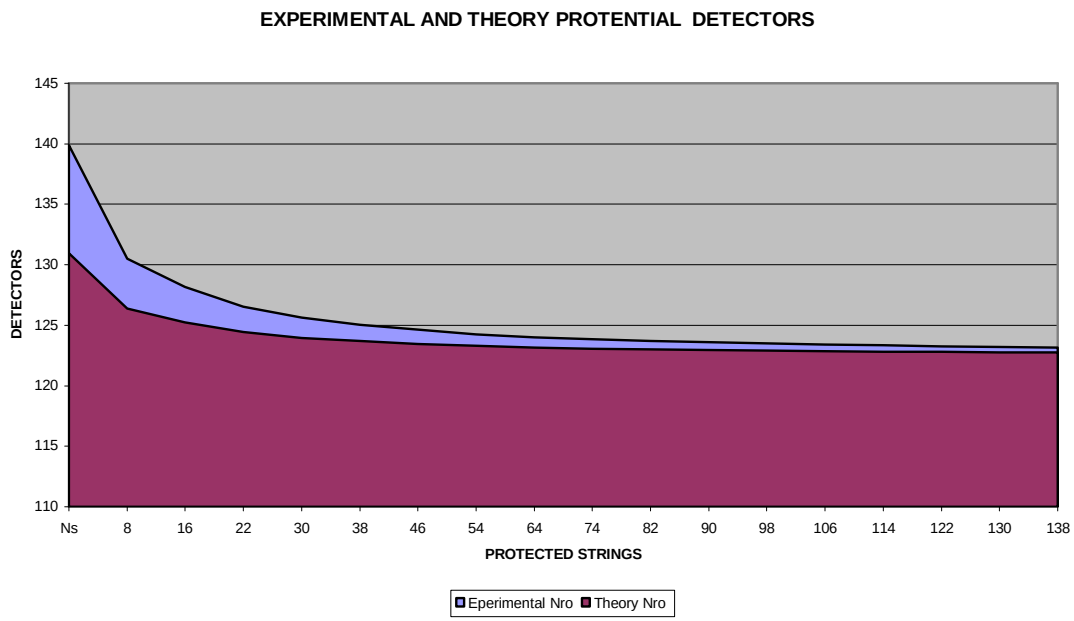


Fig 8 shows the relationship of potential detectors against actual detectors before and after censoring

4.4 Discussions

The algorithm used and presented in this thesis take its inspiration from the generation of T cells in the immune system. The T-cells are capable of identifying over 98% of foreign elements entering or circulating the body. The capabilities that allow the cells to recognised the cells is a chemical mask called receptors. These receptors are made of pseudo-random genetic process.

T-cells whose receptors recognise self-molecules are not allowed to leave the thymus where they are destroyed, and only the fittest cells that pass the fitness requirements are allowed to form the basis of our immune protection defence system.

Our algorithm works similarly to the system described above, generating detectors randomly, and eliminating (censoring) the one that detect self. We use binary strings as the model of our antigens, since the computer, system's basis of string recognition is made out of the binary strings.

The binary models have been used in studying several different aspects of immune systems. The emphasis however is that the binary strings oversimplify the complexities of chemistry of antibodies recognition system.

We in this thesis have experimented with mostly theoretical aspirants of the concepts of matching and recognizing of intruder in our systems. Various testing of upper and lower bounds were made and several interesting observations were made. To keep the number of holes low $N_s <= 2^r$. Number of holes can be controlled by formula $N_t <= Pf*2^{l}$, where N_t is the number of holes, Value of Γ may have to be chosen again many times before an acceptable probability is obtained. There is no rule of a thump for choosing Γ and l.

Sampling, especially done at random, brings non-determinism in the process, which can yield different solutions on different runs, even if the model remains the same. As compared to the, linear, non-linear and integer models included deterministic methods, as they yield similar answers if the initial input value in the run is the same.

Somewhat guessed values have to be put in place for a genetic algorithm to terminate, otherwise the algorithm never knows for certain when to stop, Besides the length of time, or the number of iterations or specific solutions sets, that one wishes to explore.

The inclusion of so many solutions, all representing either an elegant solution or a weak solution make it a very difficult task for an optimum solution to be arrived at.

4.5 Conclusion and Recommendations

The researches by various authors have shown how possible it was to use the negative selection solution algorithm to solve the security concerns of the discipline of

Computer science. Most of the work by these authors concentrated mainly at producing the right size and right detector set. We have described a general method for distinguishing self from non-self in the context of computational systems, and this research have illustrated infeasibility and feasibilities of the system in generating and matching antibodies (virus). The major computational difficulty is manifested in the generation of the first potential detectors; almost an exponential explosion is created to protect self-set. We have shown that the ratio of detectors is independent of the protected set. This illustrates that a genetic algorithm is useful in smaller and larger environments, where some monitoring of computers is required.

As the length of string matching grow the algorithm performance works at optimum level. The major future work should concentrate on how to eliminate holes that appear resilience despite the mathematical rigor being used to derive the censoring equations. Another area of future investigation is to derive a complementary set of methods of detection within the biological context, by trying to understand most of the complexities that contribute to process of immune defense in the human body.

Annexes

A.1 Introduction

The test on data collected from hornet site www.honeynet.org, was used, to simulate a live virus environment. The shell script Randomly generated data, generated by the Unix alpha server digital Unix environment was used as well. Data in random form from random.org was used as well as test data.

A shell script named Covert_to_as an ASCII to its equivalent binary or hex. We assumed the first character was our Γ-contiguous bit and it constituted the binary required. We picked any part like a port scan and converted the first letter of every one of its IP address in the file to a binary string and matched string from the file sample of known viruses, as many strings in as many file as required

A.2 Hornet Sample Data

Apr 16 02:45:37 lisa snort[7483]: IDS13/portmap-request-mountd: 200.190.13.181:1372 -> 172.16.1.107:111
Apr 16 07:17:06 lisa snort[7483]: IDS128/web-cgi-phf: 200.190.8.220:55220 -> 172.16.1.107:80
Apr 16 14:54:20 lisa snort[7483]: IDS171/Ping zeros: 24.201.15.148 -> 172.16.1.101
Apr 16 14:54:20 lisa snort[7483]: IDS171/Ping zeros: 24.201.15.148 -> 172.16.1.105
Apr 16 14:54:20 lisa snort[7483]: IDS171/Ping zeros: 24.201.15.148 -> 172.16.1.107
Apr 17 06:02:32 lisa snort[8255]: IDS198/SYN FIN Scan: 195.116.152.104:0 -> 172.16.1.101:111
Apr 17 06:02:32 lisa snort[8255]: IDS198/SYN FIN Scan: 195.116.152.104:0 -> 172.16.1.107:111
Apr 17 09:45:28 lisa snort[8255]: IDS198/SYN FIN Scan: 195.116.152.104:0 -> 172.16.1.105:111
Apr 19 08:00:19 lisa snort[3515]: IDS/DNS-version-query: 212.25.75.196:1723 -> 172.16.1.101:53

Apr 20 01:26:00 lisa snort[3515]: IDS212/dns-zone-transfer: 24.234.45.60:4075 -> 172.16.1.107:53
Apr 20 03:49:38 lisa snort[3515]: IDS/DNS-version-query: 216.123.23.5:4349 -> 172.16.1.101:53
Apr 20 03:49:39 lisa snort[3515]: IDS/DNS-version-query: 216.123.23.5:4350 -> 172.16.1.107:53
Apr 20 21:48:55 lisa snort[12353]: IDS246/large-icmp: 129.142.224.3 -> 172.16.1.107
Apr 20 21:48:55 lisa snort[12353]: IDS246/large-icmp: 129.142.224.3 -> 172.16.1.107
Apr 20 22:48:13 lisa snort[12632]: IDS159/Ping Microsoft Windows: 216.228.4.204 -> 172.16.1.101
Apr 20 22:48:13 lisa snort[12632]: IDS159/Ping Microsoft Windows: 216.228.4.204 -> 172.16.1.101
Apr 20 23:00:33 lisa snort[12657]: IDS171/Ping zeros: 216.228.4.133 -> 172.16.1.101
Apr 21 11:01:27 lisa snort[12777]: IDS/DNS-version-query: 207.236.55.76:4039 -> 172.16.1.101:53
Apr 21 11:01:28 lisa snort[12777]: IDS/DNS-version-query: 207.236.55.76:4044 -> 172.16.1.107:53
Apr 22 08:36:29 lisa snort[743]: IDS/DNS-version-query: 212.244.222.100:1368 -> 172.16.1.101:53
Apr 22 08:36:29 lisa snort[743]: IDS/DNS-version-query: 212.244.222.100

A.3 Shell Scripts

A.3.1 Covert_to_as Shell Script

Author: Hector Kapelewela

Purpose: Testing live virus

```
#!/bin/ksh
START=33 # FISRT OF PRINTABLE ASCII CHARACTERS IN DECIMAL.
END=125..#LAST OF PRINTABLE ASCII CHARACTERS IN DECIMAL

echo " DECIMAL HEX CHARACTER" # HEADER.
echo
i=33
#LIMIT=10

while [ "$i" -lt "$END" ]
do
 echo -n "$i "       # -n suppresses new line.
 #          ^        Space, to separate printed out numbers.
```

```
 i=`expr $i + 1`   # var0=$(($var0+1))  may be used.

echo $i | awk '{printf("   %3d       %2x         %c\n", $1, $1, $1)}'
done

exit 0
```

A.4 Description and functionalities

The basic Unix algorithms were written at each stage to suite the task .We take to it that the reader is familiar with the syntax used by Unix shell, simple awk rule and sed to appreciate fully the functions and the testing done to derive a practical qualification to our system.

A script **random.ksh** was used to generating the random detector initially N_{R0} and then $N_{s.}$ This script has a varying variable that is used to tune fine the size of the repertoires as they are produced by the scripts. The fact that our random number are smaller than 0, (0<x<0, where x is the random value obtained.) further manipulation is applied to convert it to whole numbers e.g. multiplying it by 10,100,1000 etc.

We present the algorithm **random.ksh** outline

```
#! /bin/ksh

  # $RANDOM returns a different random integer at each invocation.

  # Nominal range: 0 - 32767 (signed 16-bit integer).

  # Generation of probabilistic values


  MAXCOUNT=1000
  count=1
```

```
  echo
  echo "$MAXCOUNT random numbers:"
  echo "-----------------"
  while [ "$count" -le $MAXCOUNT ]        # Generate 1000
($MAXCOUNT) random integers.
  do
    number=$RANDOM
    echo $number
    let "count += 1"  # Increment count.
  done
  echo "-----------------"

  exit 0
```

The MAXCOUNT was used to control generation of the random strings as require for comparison performance using earlier defined equations.

A sed command is used to strip off the decimal fractional part, as when required.

We used the script

```
#!/bin/ksh

string=xxxx

echo "len($string)" | m4              # 4

echo "substr($string,4)" | m4         # A01

echo "eval(33 / 3)" | m4              # 33

exit 0
```

When we wanted a specific length and we simply wanted to reduce our strings either to octal words by mainly sub stringing a given string or dividing it by another number to reduce it.

Script used to convert and reconvert through bases known as **Base_Conv.ksh**

```
#!/bin/bash
NOARGS=65
bs=`basename "$0"`                    # Program name
VER=`echo '$Revision: 1.2 $' | cut -d' ' -f2`  # ==> VER=1.2
Usage () {
   echo "$bs - convert number to onther bases, $VER (stv '95)
usage: $bs [number ...]
If there is no number read from standard input.
A number may be
   binary (base 2)          starting with 0b (i.e. 0b1100)
   octal (base 8)           starting with 0  (i.e. 014)
   hexadecimal (base 16)    starting with 0x (i.e. 0xc)
   decimal                  otherwise (i.e. 12)" >&2
   exit $NOARGS
}   # ==> Function to print usage message.

Msg () {
   for i   # ==> in [list] missing.
   do echo "$bs: $i" >&2
   done
}

Fatal () { Msg "$@"; exit 66; }
```

```
PrintBases () {
  # Find base  number
  for i      # ==> in [list] missing...
  do         # ==> so operates on command line arg(s).
    case "$i" in
      0b*)              ibase=2;;       # binary
      0x*|[a-f]*|[A-F]*)  ibase=16;;      # hexadecimal
      0*)               ibase=8;;       # octal
      [1-9]*)           ibase=10;;      # decimal
      *)
        Msg "illegal number $i - ignored"
        continue;;
    esac

    # Remove prefix, convert hex digits to uppercase (bc needs this)
    number=`echo "$i" | sed -e 's:^0[bBxX]::' | tr '[a-f]' '[A-F]'`
    # ==> Uses ":" as sed separator, rather than "/".

    # Convert number to decimal
    dec=`echo "ibase=$ibase; $number" | bc`  # ==> 'bc' is calculator
utility.
    case "$dec" in
      [0-9]*)    ;;                  # number ok
      *)         continue;;          # error: ignore
    esac
```

```
      # Print all conversions in one line.
      # ==> 'here document' feeds command list to 'bc'.
      echo `bc <<!
         obase=16; "hex="; $dec
         obase=10; "dec="; $dec
         obase=8;  "oct="; $dec
         obase=2;  "bin="; $dec
!
   ` | sed -e 's: :    :g'

   done
}

while [ $# -gt 0 ]
# ==>  Is a "while loop" really necessary here,
# ==>+ since all the cases either break out of the loop
# ==>+ or terminate the script.
# ==> (Thanks, Paulo Marcel Coelho Aragao.)
do
   case "$1" in
      --)    shift; break;;
      -h)    Usage;;               # ==> Help message.
      -*)    Usage;;
       *)    break;;               # first number
```

```
   esac   # ==> More error checking for illegal input might be useful.
   shift
done

if [ $# -gt 0 ]
then
   PrintBases "$@"
else                          # read from stdin
   while read line
   do
      PrintBases $line
   done
fi
exit 0
```

We used grep, m4 macro processor, awk, sed command as required, however any other language or a Unix utility may be used whenever appropriate, and whenever the reader feels like.

Our scripts or programs were high performing due to a one linear nature, or as in Unix idiom as command prompt using the high performing Unix capabilities.

A.6 Tools and Usage

We ran a comprehensive simulation using the Unix, utilities and we have presented our findings. In most cases quite representative with guidance and trends, however in very smaller occasions results were surprisingly different.

To work with test from hornet site, we had to convert the strings to Γ-contiguous standard matching bits. Hence the data was partially converted to a binary string

We had to convert random strings from bases to more familiar bases to enable us make comparisons, and reconvert to binary strings.

The thrust of our project is to simplify the art of protecting computers and develop a genetic solution to protect those machines, using a combination of very readily available tools as supporting algorithms.

The awk utility is versatile and we had to use it in some cases as a prompt command. Many algorithms were used in conjunction with the above.

Bibliography

Anderson, J.P. “Computer Security: (1980), Threat Monitoring and Surveillance.” James P. Anderson Co. (February 1980).

Ayara, M. J Timmis, L. de Lemos, de Castro, R and Duncan: (2002), Negative Selection: How to generate detectors. [Ayara Timis]

Corchado J. M., Alonzo L., and Fyfe (eds.) C, :(2002), SOCO-2002 In Artificial Neural Networks in Pattern Recognition, University of Paisley, UK, pp. 67-84.

Crosbie M, Spafford G. :(1996), Defending a Computer System using Autonomous Agents, Technical report No 95-022, 8th National Information Systems Security Conference Perdue University, 1996 [MARKC]

Dasgupta D, and Gonzalez F. :(2002), An Immunity Based Techniques to characterize intrusions in Computer Networks, IEEE

Dasgupta D, and Forrester S :(1996), Novelty Detection in time series data using ideas from immunology, in proceeding of the international conference on intelligent systems.

Dasgupta D and Forrest S :(1999) An anomaly Detection algorithm inspired by the immune system in: Artificial Immune systems and their applications, Springer_Verlang, Inc

De Castro, L. N. and Timmis, J. I. (2002), "Artificial Immune Systems: A Novel Paradigm to Pattern Recognition"

DeCastro L. N and J.Timmis :(2002) Artificial Immune Systems anew computational approach, London, UK

Denning D :(1986) An Intrusion Detection Model, (1986), http://citeseer.ist.psu.edu/crosbie96defending.html

Genetic Programming Home Page www.geneticprogramming.com [GHM]

Haeseleer P', Forest S, Helman P. (1997): An Immunological Approach to Change Detection: Algorithms, Analysis and Implications, University of New Mexico. [PATRICK D]

Helmer, G, Johnny S, Wong K, Honavar, V, Miller Les: (1998) Intelligent Agents for Intrusion Detection. Proceedings, IEEE Information Technology Conference, Syracuse, NY, September 1998, pp. 121-124; [GUGH]:

Hofmeyr S. and Forrest S. :(2000), Architecture for an Artificial Immune System Evolutionary Computation.

Juniper Networks, Inc Intrusion Detection and Prevention, White Paper part number: (2005) 200065-001. [JUN]

Know Your Enemy: (2001) Statistics Analysing the past predicting the future; www.honeynet.org, [HONN]

Koza J. R, Forest H. B. et al: (1999) Genetic Programming III, Morgan Kaufmann Publishers, Inc. [JRKOZA]

Koza J. R. Forest H. B. et al: (1994) Discovery of Rewritable Rules in Linden Mayer and system state transmission rules in cellular automata; via GP; Proceeding to Symposium on pattern matching, [RJKOZA2]

Luger G, F, (1997): A. Stubblefield: Artificial Intelligence-Structures and Strategies for Complex Problem Solving 3 Edition, Addison Wesley

Mutz D, Vigna G, Kemmere R: (2004) University of California, Reliable software group.

Rebecca B, Peter M :(2003), Intrusion Detection Systems; Special publication on Detection Systems, National Institute of Standards and Technology; Infidel, Inc, Scotts Valley, CA

Ruggett J, Bains W, (1992): Artificial Intelligence a-z; Chapman Publishers, UK.

Smith R., Forrest S, Perelson A.S: (1993) Searching for Diverse, cooperative Population with genetic algorithms. Evolutionary computations, 1(2): 127-149,

Stallings, W (2003): Cryptography and Network Security 3rd Edition, Publisher Prentice Hall Inc.

Stoll C. (1990), Cuckoo's Egg, Pocket Books, stoll@ocf.berkeley.edu

Tanebaum A. S :(2002) Computer Networks 4th Edition, Prentice Hall

Ware W. H: (1979) Security Controls for Computer Systems, R-609-1; Report of Defence Science Board Task Force on Computer Security.