MALICIOUS USER INPUT DETECTION ON WEB-BASED ATTACKS WITH THE NEGATIVE SELECTION ALGORITHM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA MERT KARATAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CYBER SECURITY

DECEMBER 2019

Approval of the thesis:

MALICIOUS USER INPUT DETECTION ON WEB-BASED ATTACKS WITH THE NEGATIVE SELECTION ALGORITHM

submitted by **MUSTAFA MERT KARATAŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Cyber Security Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, Graduate School of **Informatics** ________________

Assoc. Prof. Dr. Aysu Betin Can
Head of Department, **Cyber Security** ________________

Assist. Prof. Dr. Aybar Can Acar
Supervisor, **Health Informatics, METU** ________________

Examining Committee Members:

Assoc. Prof. Dr. Cengiz Acartürk
Cognitive Science, METU ________________

Assist. Prof. Dr. Aybar Can Acar
Health Informatics, METU ________________

Assoc. Prof. Dr. Aysu Betin Can
Information Systems, METU ________________

Assist. Prof. Dr. Murat Perit Çakır
Cognitive Science, METU ________________

Assoc. Prof. Dr. Hacer Karacan
Computer Engineering, Gazi University ________________

Date:

Name,Surname:	Mustafa Mert Karataş
Signature :	

ABSTRACT

MALICIOUS USER INPUT DETECTION ON WEB-BASED ATTACKS WITH THE NEGATIVE SELECTION ALGORITHM

Karataş, Mustafa Mert
M.S., Department of Cyber Security
Supervisor: Assist. Prof. Dr. Aybar Can Acar

December 2019, 54 pages

In the cyber security domain, detection and prevention of intrusions is a crucial task. Intrusion attempts exploiting vulnerabilities in an organization's servers or applications may lead to devastating consequences. The malicious actor may obtain sensitive information from the application, seize database records or take over the servers completely. While protecting web applications/services, discrimination of legitimate user inputs from malicious payloads must be done.

Taking inspiration from the Human Immune System (HIS), numerous research studies have been conducted, where the HIS' behavior while protecting the body from the malicious pathogens is applied to the problem of intrusion detection. The T-cell is one of the lymphocytes that form the human immune system. The study of Artificial Immune Systems (AIS), applies the self/non-self discrimination of T-cells to computational discrimination problems. The ability to discriminate self (safe) from non-self (malicious) is used for the detection of any malicious activity in a computer, or a computer network. The AIS model of interest in this thesis is Negative Selection. Negative Selection Algorithm is applied to detect malicious user input that is submitted in HTTP GET parameters. Detection is done through detector strings with varying lengths. Detectors are constructed with randomly chosen n-grams generated from the training dataset. The number of n-grams required to form a detector is sampled from the Poisson distribution. Detection rates, number of attempts needed for generating a single detector, average detection rates for each detector, the lengths of the detectors and the number of detectors that can be generated over a course of time are calculated and presented.

Keywords: artificial immunology, negative selection, web based attacks

ÖZ

WEB TABANLI SALDIRILARDA ZARARLI KULLANICI GİRDİLERİNİN NEGATİF SEÇİLİM ALGORİTMASI İLE TESPİTİ

Karataş, Mustafa Mert
Yüksek Lisans, Siber Güvenlik Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Aybar Can Acar

Aralık 2019 , 54 sayfa

Siber güvenlik alanı içerisinde saldırıları tespit etme ve önleme için bir çok çalışma yapılmaktadır. Bir kurumun sunucularında veya uygulamalarında var olan bir açıklığın istismar edilmesi yıkıcı sonuçlar doğurabilmektedir. Saldırgan bir uygulamanın kullanıcılarının kişisel verilerine erişim sağlayabilir, veri tabanı kayıtlarını çalabilir veya sunucuları tamamen ele geçirebilir. Web uygulamalarının başarılı olarak korunması için gelen kullanıcı girdilerinin geçerli veya zararlı olarak ayırımının yapılması gerekmektedir. İnsan Bağışıklık Sisteminden(IBS) ilham alınarak bir çok arıştırma yapılmış ve IBS'nin patojenlerden korunmak için gösterdiği davranış saldırı tespit sistemlerine uyarlanmıştır.

T-Hücreleri bağışıklık sistemindeki lenfosit türlerinden biridir. Vücut içerisinde T-Hücreleri oluşturulurken kullanılan Negatif Seçilim süreci Yapay Bağışıklık Sistemi (YBS) içerisinde tanımlanmıştır. T-Hücrelerinin yeteneklerinden olan kendi/yabancı ayrımı, bilgisayar ortamında veya bir bilgisayar ağında bulunan anormal durumların tespit edilmesi için yararlı görülmüş ve bu alan üzerinde çalışmalar yürütülmüştür. Bu tez, HTTP GET trafiği üzerinde taşınan ve kullanıcının istemci tarafından gönderdiği parametrelerdeki zararlı aktivitelerin tespit edilmesi için Negatif Seçilim Algoritması'nı kullanmaktadır. Tespit işlemi çeşitli uzunluklardaki dizge değerleri kullanılarak yapılmaktadır. Bu dizgeler oluşturulurken eğitim veri setinden oluşturulmuş n-gram dizgeleri kullanılmaktadır. Kaç n-gram dizgesinin aynı anda kullanılacağının belirlenmesi için Poisson Dağılımı kullanılmıştır. Tespit oranları, her bir tespit edici dizgenin oluşturulabilmesi için gerekli deneme sayıları, oluşturulan dizgelerin uzunlukları, dizgilerin bireysel tespit başarımları ve belirli bir zaman içerisinde kaç dizgenin oluşturulabileceği çalışma içerisinde gözlemlenmiş ve sunulmuştur.

Anahtar Kelimeler: yapay bağışıklık, negatif seçilim, web tabanlı saldırılar

To my family

ACKNOWLEDGMENTS

I would like to thank to my advisor Asst. Prof. Aybar Can Acar. His door was always open for me and he directed me until the end with his wisdom.

I would like to thank my beloved wife, Gözde, she has never lost her trust in me and gave her support from the start.

I must express my gratitude to my parents, Rabia and Hakan, for their never-ending support throughout my life. I could not be who I am today without their care. I am also lucky to have my brother, Bahacan, to whom I wish to achieve further success. Every accomplishment I have, would not be possible without them.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AIS	Artificial Immune System
HIS	Human Immune System
MHC	Major Histocompatibility Complex
NSA	Negative Selection Algorithm
RCB	Recursive Coordinate Bisection
HTTP	Hyper Text Tranfer Protocol
SQL	Structured Query Language
XSS	Cross-Site Scripting
IDS	Intrusion Detection System
DBMS	Database Management System
CSS	Cascading Style Sheets
HTML	Hyper Text Markup Language
kNN	k-Nearest Neighbours
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Artificial Immune Systems (AIS) [1, 2] are modeled from the structure and behavior of the human immune system. Protection from pathogens that trespass to the body is done by the immune system [3]. The immune system consists of two sub-systems; the innate and adaptive immune Systems. These two sub-systems complement each other for the protection of the human body. The innate immune system contains general protections. Physical barriers such as the skin are part of the innate immune system. It is gained at birth and is common to every human. Lymphocytes are part of the adaptive immune system. They detect and eliminate intruding pathogens. The generation and training of lymphocytes is the responsibility of the adaptive system. The success of a lymphocyte is based on its ability to distinguish self from non-self. A lymphocyte's success is tested during its generation phase and if it fails it is eliminated by the immune system.

As the human immune system protects the body from any malevolent outsiders, an Artificial Immune System (AIS) is constructed to protect a digital system or a network. To successfully protect a digital asset, the segregation of normal behavior from anomalies must be performed. This problem is analogous to the self/non-self discrimination of the lymphocytes.

A T-cell is a type of lymphocyte embodied in the adaptive immune system. They are generated in the thymus located between the lungs. T-cells undergo two processes while being created, positive selection [4] and negative selection [5]. Positive selection is the process where the ability of the generated T-cell's receptors to bind to an MHC (Major Histocompatibility Complex) is developed. MHC is a form that presents antigens to the T-cells. If a T-cell can not bind any MHC, it will not be able to bind to any antigens, it is recognized as unfit and eliminated directly. T-cells that pass the positive selection phase are subjected to negative selection.

Negative selection gives T-cells the ability to distinguish self from non-self. Self are the elements of the body that are benign and non-self stands for pathogens that are foreign. In negative selection, if a T-cell reacts to a self sample, it is eliminated directly. The Negative Selection Algorithm (NSA) is based on this behavior of the human immune system.

NSA is useful for classifying target data as normal (self) and anomalous (non-self). The algorithm generates objects with desired attributes called "detectors" which are the equivalent of T-cells. Later, these are tested against the data samples that do not contain any anomalies. If the detectors do not match any, they are considered fit and is marked to be used for detection. If they match any of these ‘self’ samples, they are eliminated.

Matching done in the testing phase of the NSA calculates the affinity between the randomly generated candidate detectors and normal (self) data. An affinity higher than a predetermined threshold is not desired, and will eliminate the detector.

Different distance algorithms can be used for the affinity calculation. The choice of the distance algorithm depends on the problem that is trying to be solved. Since the representation of the detectors may vary (bit, string or real-valued) the distance algorithm varies as well. Examples are Euclidean, r-chunk, r-contiguous, rcb, Hamming and Levenshtein distance [6, 7] algorithms.

Generation method of a candidate detector, the length of the generated detector and the distance algorithm chosen play an important role in detection success. It is desirable that the generated detector set covers the non-self space as much as possible. Representation of self and non-self spaces can be seen in Figure 1.1.

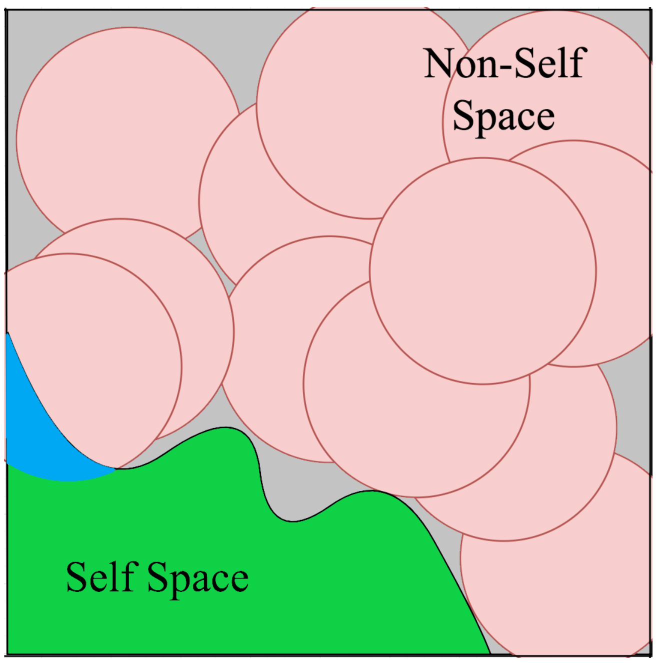

Figure 1.1: Representation of Self Space.

In the detection phase;

- If the data to be tested falls in the non-self space, it is true positive. (Area colored in red)
- If the data to be tested falls in the self space, it is true negative. (Area colored in green)

- If the data to be tested falls into an area that is not covered by the detector set, it is a false negative. (Area colored in grey)

- If the detector's distance to the self space is not far enough, meaning that the detectors coverage area intersects self space, then it is a false positive. (Area colored in blue)

In this study, an anomaly detection algorithm is proposed for user-supplied input that is obtained from HTTP GET parameters [8]. The proposed algorithm is based on the negative selection idea. The dataset used for training and testing the generated detector set contains real-world examples of a benign web application traffic. Malicious traffic in the dataset is formed with the output traffic of the popular penetration testing tools that create SQL Injection [9], XSS (Cross-Site Scripting)[10][11], Command Injection [12][11] and Path Traversal[12][11] attack payloads.

The generation of candidate detectors are done by dividing the harmless dataset into n-gram strings and choosing a number of elements from this set. The number of n-grams used to create the detector is determined by the Poisson Probability Distribution [13]. The output of the Poisson Probability and the n-gram length gives the length of the detector string. To calculate the affinity between self data and the detector, the Levenshtein Distance [6] is used. Detection success tables for different affinity thresholds, n-gram lengths, and detector lengths are presented accompanied by the graphs of the mean numbers of candidate detector trials to successfully generate a single detector, execution times of the detectors and hit rates per detector with changes in the mentioned parameters. The goal of the study is to find the n-gram length, affinity threshold and detector length triple that results in the best detection rates.

1.2 Contributions

This study applies the Negative Selection Algorithm to detect attack inputs that are constructed to exploit web applications or services. Since web attacks we intend to detect use string-based payloads, the detectors that are generated by the algorithm are strings as well. Generating these detectors by combining n-grams which are sampled using the Poisson Distribution and comparing them with the HTTP traffic for the stated problem is a novel experiment as far as we know.

The study provides generation metrics and detection results to determine the optimal n-gram length, the total detector length and affinity threshold values that give the highest true positive and true negative rates. Furthermore, iterations needed in order to create a useful detector, time consumption when applying a detector to new data and estimated real-time behavior are presented and debated.

1.3 Assumptions

The method used in this study detects attacks at the user input fields of a web applications or services. In the detector generation phase, like the human immune system while creating the T-cell lymphocytes, must run with a dataset that contains only normal data. Any anomaly in the dataset may lead the algorithm to generate detectors that will cause false positives or false negatives since the detectors will learn the anomaly as normal behavior. Any appliances that will use this study shall have a dataset of the application or service to be protected or may use real-time traffic to generate detectors if the current network flow guarantees it has only legitimate traffic.

The asset that is to be protected by the proposed method is assumed to not have any other vulnerabilities that makes it liable to other attack types. Since the detectors are generated from, and guard, the HTTP parameters, it is not possible to detect other network activities.

1.4 Limitations

The proposed method in this study works only with HTTP parameters. In the real world examples of an application, an HTTP parameter may contain nearly static values as well as highly varying ones. User id, page id, the filename (image, pdf, CSS document, etc.) are the examples for nearly static parameters. User-supplied text (e.g. comments for a social media entry, feedback for the website owner, etc.), session-id or encrypted cookie values are examples for parameters with high variety. Generating detectors from nearly static values is easier compared to generating from parameters with many different character combinations. It will take more time for negative selection to generate a detector successfully. Furthermore, if a special character is used frequently in the values of a parameter, negative selection will omit that character since the use of that character inside a detector will cause the affinity threshold to be exceeded. This may cause lower true positive rates if an attack uses that character in its payloads.

The limitation stated above can be mitigated to a certain extent by generating different detector sets for different parameters. This will cause the algorithm to find a combination that will not exceed the affinity threshold and catch attack payloads that utilize that combination. In this study, separation of parameters for different detector sets was kept out of scope.

Another limitation of the proposed method is that it will fail to detect attacks that are conducted by legitimate application parameters. To give an example, authorization flows can not be detected by this algorithm. In an attempt to exploit an authorization flow vulnerability, the attacker will send a parameter value to the server that another authorized user uses in his requests. Since the authorized users use that parameter value, it is recognized as a normal value, not an anomaly. In order to detect this kind of attack, user behavior and action history must be known and tracked, which the proposed method is not capable of.

1.5 The Outline of the Thesis

In this thesis, there will be four chapters except for this chapter. The second chapter will present background information about the attacks that are conducted through HTTP parameters, an overview of the human immune System and of artificial immune Systems; as well as an examination of related work. The third chapter will describe the proposed method in detail. The fourth chapter contains the results of testing this algorithm on real data. The fifth chapter will evaluate these results. Finally, the sixth chapter will present the conclusions.

CHAPTER 2

BACKGROUND

There are four parts in this chapter.

The first part gives information about the HTTP protocol [8] and type of attacks conducted by the malicious actors. Although there are numerous web based attacks, the chapter will only give detailed information for those that can be detected by the proposed method in this thesis.

The second part gives a general insight on the human immune system. The mentioned properties of the immune system in this part have inspired researchers to build the models and algorithms used in Artificial Immune Systems.

The third part presents information about Artificial Immune Systems. The Negative Selection Algorithm which forms basis for this thesis is explained in this part.

The fourth part states the work and studies carried out by the researchers in the Artificial Immune Systems domain.

2.1 Attacks on Web Servers and Applications

Web applications, web services and majority of the mobile applications require a web server in order to operate. Web servers are responsible for distributing and storing objects and documents with the guidance of the business logic that is implemented in the backend code they run internally.

Web applications and services generally get user input to accomplish their desired operations. These inputs are treated as parameters in the server side functions and are transferred in HTTP requests. There are several HTTP methods but the mostly used methods are GET and POST. In HTTP GET, parameters are sent as an extension of the request URL. These parameters can be observed in the URL bar of the web browser and can be altered by the user easily. HTTP POST parameters are located as headers in HTTP requests. To make changes on the parameters found in POST requests requires alteration of the client-side code or a non-transparent proxy server that the traffic will pass over. Although it demands additional labor to alter a POST request, it is relatively easy for an attacker to modify the parameters in both HTTP methods.

Unless there are filtering and validation mechanisms applied to the input received from the client-side, a web application as well as its clients face several threats. A vulnerability in the server-side code may cause the web server to be breached or sensitive information that is stored to be seized by the attackers. The most convenient way to exploit a vulnerability in the server-side application or service is to alter the HTTP parameters in a way that is not normally expected by the system.

Absence of proper precautions taken in the server-side code may lead to the conduction of many attacks. Examples of these attacks are;

- SQL Injection Attack
- XSS
- Path Traversal
- Command Injection
- Local/Remote File Inclusion
- Authorization Flows

Proposed algorithm aims to detect examples of the first four attack types.

2.1.1 SQL Injection Attack

Web applications and services form a bridge between the client-side application and -if not contacted directly- the database server. The database server may hold sensitive information about the users and application specific data. Although there exists several Data Base Management System (DBMS) implementations, the Structured Query Language (SQL) [14] is used common to most of them and is commonly used to query the data that is stored.

Services that run on the server-side obtain a parameter from the client-side and perform queries on the database by constructing SQL statements. Later, the database server responds with a result that will be used by the same service. These SQL statements may be constructed to insert new data or to view, modify, delete existing data.

SQL Injection is a vulnerability that arises when the user-supplied or any client-side parameter is directly added to the SQL statement without proper filtering causing changes in the original statement. [**?**] Changed statement will cause actions on the database that are normally not intended by the program.

2.1.2 XSS

Several web technologies are used on the client-side of a web application. These technologies are developed to enhance client-side user experience and in some cases reduce the work load of the server-side. They are mainly derived from HTML (Hyper Text Markup Language) [15], CSS (Cascading Style Sheets) [16] and JavaScript [**?**].

HTML is used to create the semantic outline of a web page for the user to interact with. CSS consists of the styling statements for the visual presentation of the elements on the page. It handles the properties of an element on the page which can be size measures, colors, rotations etc. JavaScript is the programming language for the client-side environment. It has control over both the HTML elements and the CSS attributes. It has networking capabilities for sending HTTP requests.

Cross-Site Scripting is a vulnerability that arises from the lack of filtering and encoding operations applied to the user supplied data. Since JavaScript is not a compiled language and is interpreted on page load, if the attacker finds a parameter that can be directly inserted into the HTML document without proper handling, he can execute arbitrary code on the client. This may lead to the hijacking of the user session, obtaining sensitive information of the affected users and unauthorized access to the resources that the attacker can not access with his own privileges.

2.1.3 Path Traversal

Web applications need different resources in order to execute properly. HTML, CSS, Javascript, image and text files are examples of these resources.

A web application requests these resources upon page load. The web server responds by presenting the requested resource. The obtained resource is later used inside the web page either for viewing or expanding the functionality of the page.

The file can be requested with its absolute path, relative path or an object identifier that is assigned to it. Absolute path is the file's location on the server with full directory path. Relative path stands for the file's location beginning from the directory of the web application's root folder. Object identifiers are predetermined identification codes that map to an absolute or relative path. They are generally stored in a database on the server.

A Path Traversal vulnerability is when arbitrary files on the system, including operating system files, can be obtained by altering the parameter used for stating the absolute/relative path of the normally expected files. An attacker may exploit this vulnerability to gain access to operating system, or configuration and storage files that reside on the web server. The obtained information from these files can lead to the malicious actor gaining complete control over the system, even at the operating system level.

The vulnerability is caused by improper validation of the input data as well as server misconfiguration.

2.1.4 Command Injection

Web servers are run on the operating system of the server machine. Both the web server and web applications may need to interact with operating system functions while running. For web applications, while there is a chance to use the server-side programming language's capabilities, it may be required to directly call off-the-shelf programs that are installed or commands of the operating system.

A command injection vulnerability arises when the web application fails to filter and validate the user input and adds it directly to the command that will be executed on the operating system. Exploitation of the vulnerability grants partial if not complete control over the system by the attacker. All of the explained and targeted web based attacks by this study are the product of improper handling of the user supplied text input. Lack of protection mechanisms can arise from the carelessness of the developer and server administrator.

2.2 Human Immune System

The Human Immune System (HIS) [3] has to protect itself from the pathogens that enter the body. Pathogens are microorganisms that are harmful to the host. Viruses and bacteria are two examples. They can reproduce, harm body cells or consume the resources that are normally for the nutrition of the body cells. The pathogens can affect a part of the body in nonfatal manner or can be lethal. The immune systems aims to eradicate these pathogens.

For eradication, the immune system must detect and identify the encountered organisms. Identification must lead to classification because different pathogens may need different measures. Detection and elimination of the recognised pathogen is done through complex yet effective processes.

The immune system can be divided into two subsystems: the innate and adaptive immune systems. These system differ from each other by their methods and elements.

2.2.1 Innate Immune System

The innate immune system forms the first layer of defense against the invading pathogens. It does not have the ability to adapt itself during the human life span. Since the cells of the innate immune system are not subjected to acclimation, they can not learn from the previous encounters.

The skin is the initial layer protection for the body. The pathogens need to pass this layer to have contact with the inner cells of the body. The cells in the innate system are capable of recognizing the pathogens. Their responses are general, rather than specific to the pathogen that is infecting the body. These responses are to create an environment that is not suitable for the foreigners' survival and/or for their reproduction. Increasing the body temperature (i.e. fever) is one of the acts of the innate system when a pathogen is identified.

2.2.2 Adaptive Immune System

When the innate immune system's countermeasures against the invading pathogens are insufficient, adaptive immune system takes the responsibility. Some pathogens do have the ability to endure unsuitable surviving conditions. This makes the immune system take responses that are specific for the microbe. The cells in the adaptive immune system are able to react to distinct pathogens. Unlike the innate immune system, the adaptive immune system has memory. When a pathogen is recognized and successfully eliminated, the immune system keeps the pathogen's markers in its memory. Whenever the same pathogen is detected in the body, response to it is almost immediate compared to the first encounter.

B- and T-cells are the lymphocytes of the adaptive immune system. Actions taken by these cells are different when they encounter and detect a foreign organism. B-cells work together with the phagocyte cells which devour and eliminate the pathogens. Upon identification of a pathogen, B-cells excrete antibodies. An antibody is a type of protein that binds to the surface of a pathogen. This makes the reproduction of the organism difficult and consumption by the phagocyte cells easier or possible. T-cells, on the other hand, kill the contacted the pathogen directly if it is detected as an outsider.

Both B- and T-cells patrol the human body. Whenever they detect a pathogen, they act in the way they are designed to. These lymphocytes do not require a population to exist near in order to fight the invaders. Additionally, they are not controlled by a central mechanism.

2.2.3 Self/Non-Self Discrimination

Self/Non-Self Discrimination (SND) is one of the most important aspects of the immune system. The ability to segregate the body cells from the pathogens is crucial for success in protecting the human body. The recognition is done through receptors on the cellular level.

Every immune cell in the body has receptors over its cell surface. These receptors create a chemical bond between the cell and the encountering pathogen's epitopes. The structure of the receptor is three-dimensional. It is expected for certain epitopes to have a structure complementary to that of the immune cell's receptor. The similarity is refered as *affinity.* The higher the affinity between receptors and epitopes, higher the probability of binding.

The immune cell will react only if bound. The reaction of the cell depends on the type of the cell. If the cell is a B-Cell it reacts with Somatic Hypermutation in which the cell will divide continously leading more B-Cells with similar, if not identical, receptors. B-Cells that bind to the pathogen also release antibodies against the foreigner. T-Cells, destroy the pathogen directly.

The immune system expects the immune cells to not react to or and not to bind to self cells. Especially T-cells need to undergo the negative selection process when they are being created in the thymus.

2.2.4 Negative Selection of T-Cells

The thymus, which is an organ of the human body is located between the lungs. It is responsible for the creation and maturation of the T-Cells. Generating a T-Cell consists of two phases; Positive Selection [4] and Negative Selection [5]. Only the cells that pass both selection processes are allowed to leave the thymus.

Positive Selection is the process where the thymus controls a freshly created T-Cell's ability to bind to MHC (Major Histocompatibility Complex). MHC is responsible for presenting the antigen by peptide fragments that will bind to the receptors on the surface of the cell. If a T-Cell can not bind with MHC, it has not have the ability to bind to a epitope of a pathogen. Therefore, the incapable T-Cell is eliminated by the thymus to be replaced by another generation.

T-Cells that successfully pass the Positive Selection are exposed to the Negative Selection process. Thymus includes most of the bodies self proteins. They are subjected to non-mature T-Cells. If a T-Cell's receptor forms a bond with the self sample, T-Cell is activated and upon detection it is not allowed to exist and eliminated.

The receptors on the T-Cells that passes the Negative Selection have low affinity with the self samples of the body meaning that it will not react to any body cells. This also means that if the T-Cell react to a peptide or epitope, it is a non-self sample because the cell would have been eliminated in the thymus otherwise.

2.2.5 Properties of The Human Immune System

Mentioned attributes above makes it possible to list and clarify the below properties.

- Self/Non-Self Discrimination: Lymphocytes in the immune system are able to differentiate the body cells from the foreign pathogens. They do not react to the self cells and harm the body.
- Memorization: After the first encounter and elimination of a pathogen, information about it is remembered by the immune system. This makes it possible to react immediately when the same pathogen tries to invade the body.
- Distributed Structure: There is no central control over the immune system or the lymphocytes in it.
- Diversity: The adaptive immune system is able to react to different pathogens due to its mutation capabilities.
- Specificity: Receptors on the lymphocytes and other detector cells does not exactly bind to a specific antigen. They look for a rate of similarity. This makes the detectors to cover a wider range of pathogens.

2.3 Artificial Immune Systems

HIS (Human Immune System) and its properties forms the base inspiration for AIS (Artificial Immune Systems). Generation, detection and memory functions of HIS as well as its distributed structure attracted many researchers over the years. Analysis of HIS processes made it possible to model and develop algorithms for this subject. While HIS protects human body from the pathogens that are harmful for health, it is thought that AIS can protect a computer environment from the anomalies and malicious activities conducted by the cyber criminals. Appliance of HIS in this manner is covered by the (IDS) Intrusion Detection Systems. It can be inferred that the subject of AIS does exist for creating IDS' with the inspiration of the nature's attitude within the field of immunology.

HIS inspired algorithms are used to solve below problems in cyber security.

- Malicious Process Detection
- Anomaly Detection
- Intrusion Detection
- Flood Attack Detection
- Fraud Detection

In AIS, the lymphocytes of the biological immune system are named as detectors. The detectors in the AIS can be represented several ways that are specific for the problem that is trying to be solved. Detectors are often binary arrays, strings or real-valued representations. Real-valued representations stand for geometrical shapes that tries to cover non-self space.

Chosen representation must be applicable to the application data or the representation of the detectors must be compatible with the data. This requirement is senseful since the detector and the data will be compared and expected to match using a comparison algorithm. These algorithms are to calculate the affinity between the detector, self and non-self samples. Numerous distance and matching algorithms are used for mentioned purpose. Below are some algorithms that are used in the previous researches in the field;

- Euclidean Distance
- r-chunk
- r-contiguous
- Levenshtein Distance

Matching ratio between the detector and a sample is called the affinity. In HIS, affinity stands for the surface similarity of the immune cell receptor and the epitopes or peptides. Affinity is used to mark the traffic or the data to be tested to be an anomaly or normality.

Below are the main algorithms and models used by the AIS applications. Negative Selection Algorithm is described in detail later in this part of the chapter.

- Negative Selection Algorithm
- Clonal Selection Algorithm
- Immune Network Theory
- Danger Theory

2.3.1 Negative Selection Algorithm

T-Cells that are produced in the thymus are subjected to the negative selection process. In the process it is expected by the immune system that the candidate T-Cell to not react to a self sample of the body. T-Cells that are produced by the end of the process are released into the veins and lymph for detection of the pathogens.

In the Negative Selection Algorithm the same process is carried in the digital environment. The algorithm can be divided into two procedures; generation of the detectors and the detection of non-self samples.

To simulate the process in the thymus with the self proteins and peptides, a dataset containing only benign data is needed. The detectors that are randomly generated are subjected to every element of the training dataset. If a match occurs between the detector and the data item, the detector is discarded and a new detector is generated that is anticipated to pass the exact same test. The matching is the analogy of the chemical bond established between the cell receptors and the epitopes.

After the generation of desired amount of detectors, they are subjected to real-time data. It is expected from the detectors to match anomalous traffic. If a detector succeeds to detect a malicious activity that can be classified as true positive, it is marked as a memory detector. The memory detectors last long in the system and any detection by these detectors are treated as anomalies that needs prevention, immediately. Likewise, if a detector fails to detect any anomaly in a given period of time, two inferences can be made; the detector represents an anomaly that rarely occurs or the detector represents a non-self sample that can not be observed by the system which can be out of scope or is not meaningful. Consequence of the failure of any detection, the detector ought to be eliminated from the system to be replaced by a recent one.

2.4 Related Works

Forrest and Perelson [5] laid the foundation of the Negative Selection Algorithm. The work is the first study to observe and implement the algorithm. They used the algorithm to detect computer viruses. They also studied the computational cost and detection probability later in the paper.

Gabrielli and Rigodanzo [17] proposed a system where they assigned states to the detectors. Different states of the detectors have different weights over the decision of marking the incoming traffic as normal or malicious. To identify a traffic as anomaly, the proposed system expects for several detectors to be stimulated. If the threshold can not be exceeded, then the traffic is treated as normal. The study has been carried out for web server attacks. The researches further deepened their study by examining the HTTP response packets of the suspected requests.

Danforth [18] have used the Negative Selection Algorithm to classify the incoming attack type in the HTTP traffic. The method proposed expands the algorithm by a polling system. Detectors are generated using the Negative Selection Algorithm and subjected to abnormal traffic. Detectors have the capability to classify the attack type(SQL Injection, XSS, Buffer Overflow, Information Gathering, Path Traversal). If the traffic detected as malicious, then the system counts each classification. Classification with the highest count names incoming attack type.

Fuyong Zhang and Ying Ma [19] proposed a method to detect malware in a system with the help of Negative Selection Algorithm and Positive Selection Algorithm. In the proposal, elements in the training dataset are divided into n-grams(3-gram and 4-gram). Permutations of the resulting set are the candidate detectors for the system. The detectors are respectively subjected to Positive and Negative Selection Algorithms.

Saleh [20] et al. have proposed a method to detect spam e-mails with the use of the Negative Selection Algorithm. E-mails in the dataset are labeled as spam(non-self) and ham(self). Each word in the e-mails is added to a database to form a token set by their occurrence frequency. The words and frequencies are later used in the detection phase to mark a sample e-mail as spam or ham. In their work, they have divided the e-mail dataset that is being used to create six different subsets. Each of the datasets is fed to the algorithm one after other and the success rates are calculated with each addition. It is concluded that with each new dataset the success rates increased -up to 98.5%- and they have also claimed that their model has higher rates than the similar models [20].

Hooks [21] et al. have compared two popular Artificial Immune System algorithms, which are the Negative Selection and the Clonal Selection. In their work, they have used the NSL-KDD dataset and divided network packets' features into three different groups that have 13, 22 and 39 distinct features. They have presented the detection accuracy rates and execution time for each experiment with the formed feature groups. Each experiment has been conducted with 25, 100 and 250 detectors. For the experiments that use Negative Selection, the best accuracy results are accomplished with 250 detectors and 39 features. However, this experiment has taken 2654 seconds to complete. Worst accuracy results for 250 detectors were when 13 features are used, resulting in 29.9% but with 890 seconds of execution time.

Below thesis studies are examined and background chapters are studied for better understanding of the Human and Artificial Immune Systems concepts and terminologies.

Igbe [22] has implemented the Negative Selection Algorithm and the Dendritic Cell Algorithm for intrusion detection in smart grids, detection of distributed denial of service attacks and insider threat detection.

Ataser [23] proposed a method to expand the non-self area coverage in his study. The thesis of the researcher expresses self samples with an additional calculated radii and determines self space with k-Nearest Neighbours [24] and Local Outlier Factor [25] algorithms. The detectors generated have real-valued representations.

CHAPTER 3

PROPOSED METHOD

The study applies an algorithm that detects abnormal user-supplied input in a web application or service which is not similar to the normal traffic. Segregation of the normal data from the malicious resembles the immune system's efforts to discriminate a pathogen from the body cells. The resemblance concludes to the analogy between the HIS [3] and the anomaly detection process in this thesis as stated in Table 3.1.

Table 3.1: The analogy between HIS and the anomaly detection process

Human Immune System	**Anomaly Detection in HTTP Data**
T-Cells	Detector strings
Pathogens	Attack payloads
Body cells	Normal data strings
Receptors	Matching algorithm
Affinity	Output of the matching algorithm
Epitopes and peptides	Divided attack payloads
Thymus	Detector generation algorithm
Veins and lymps	Application/network traffic

The study focuses on the creation of the first generation of the detectors and their generation and detection metrics. It is intended to create a detector set that has the ability to discriminate self samples from the non-self with considerable success rates from the first generation. The data gathered from the generation phase includes the attempts needed to create a detector that can successfully pass the Negative Selection, the length of the detector and the number of detectors that can be generated in a specified time window.

The generation algorithm is based on the Negative Selection Algorithm [5]. The effort of the detectors to gain the ability to discriminate normal data from the abnormal is found to be similar to the Negative Selection process that the T-Cells undergo in the thymus. Generation algorithm uses a training dataset consisted of only normal traffic to resemble the self proteins that the thymus presents to the candidate T-cells.

Detectors that successfully pass the Negative Selection are stored in a file. At the end of the generation phase, the detector set is applied to two different datasets for detection results. The first dataset consists of only normal application traffic. The traffic is similar but not identical to the training dataset. The second one contains only attack payloads.

Affinity measure in the generation and detection phases is calculated by the Levenshtein distance algorithm [6, 7]. The output of the Levenshtein distance, which is the number of the alterations needed for a given text to convert to another one, can have the maximum value of the longest parameter and minimum value of zero which means two strings are completely different. A percentage is calculated from the algorithm's output considering this behavior. This percentage gives the similarity between two strings. In the generation and detection phases, the similarity is expected to pass or fail a certain threshold.

3.1 Dataset

There are three datasets used in the detector generation phase and the detection phase. These datasets are the training and the test datasets. The generation phase uses one and the detection phase uses two datasets.

The training dataset has only normal application traffic that does not include any malicious activity. To resemble the behavior of the immune system, the training set is limited to the traffic with benign parameters. The dataset is used to generate detectors to be used in the detection phase.

For the detection phase, two test datasets are used. The first dataset only consists of normal traffic like the training dataset. The second dataset has attack payloads and does not include any normal traffic. The attack payloads had been generated by the penetration testing tools. The payloads are the ones of the SQL Injection Attack, XSS, Command Injection Attack and Path Traversal Attack. These attack types were mentioned in the Background Chapter. The first dataset is used to test the detectors' behavior against a benign data sample which is examined to calculate false positive and true negative rates. Likewise, the second dataset which contains the attack payloads is used to calculate the true positive and false negative rates.

3.2 Detector Generation Phase

Detector generation is the phase where candidate strings that are named as detectors are created and tested based on the Negative Selection Algorithm [5]. The program that is developed has three arguments as input and detector strings as output upon execution. The input parameters are n-gram length, Poisson lambda value, and affinity threshold.

The program begins execution by reading the training dataset from a file stored on the system. The dataset contains parameters extracted from the HTTP GET requests and has only normal/expected values. The program eliminates convergent values from the dataset and stores it into the program memory. The variable it is stored in is used to generate the n-gram string set.

Later in the execution, the dataset is divided into n-gram substrings. The usage of the n-gram method is the attempt to cover the non-self space as much as possible. It was mentioned in the earlier chapters that a detector may have a value that will not match any non-self samples. This can occur when the detectors identify strings as malicious while they are not suitable for an attack. N-gram strings are used to create a similarity between the samples and the detector to a certain level of extent.

The n-gram substrings of the original dataset are stored in a separate variable. This variable will be used by the algorithm to form a candidate detector by selecting a number of items from the list. The number of times a selection will be made from the set is determined by the Poisson Probability function that will take an integer as a parameter and outputs the selection count.

Detector length varies depending on the output of the Poisson probability function. This method is used to observe the possible detector lengths that can reach the affinity threshold that is given as a parameter to the program.

After forming a candidate detector by selecting and concatenating a number of elements from the n-gram string set, the detector is ready to be tested by the Negative Selection process. Every detector that is created is subjected to the process one by one.

In the Negative Selection process, a detector's affinity is calculated with every parameter in the benign dataset. The detector's affinity with every distinct element of the dataset is expected to be equal or less than the affinity threshold parameter given to the program. If it has a higher affinity with any element than the threshold, the detector is eliminated and the creation process for a new detector starts over.

An HTTP parameter can be shorter or longer in length compared to the detector string that is created by the given program parameters. If the affinity is calculated directly without any further operation, then the chances of obtaining an unfit detector rise. If the length difference between the detector string and the parameter that is being compared is high, the affinity ratio results low. If the difference is high enough, even if the detector string is a substring of the parameter, the affinity results low since there will be many additions to make to convert the detector string to the parameter [6]. It is concluded that the parameter must be divided into substrings with the same length as the detector string. After the split operation, the comparison is done between the substrings and the detector. If any of the substrings exceeds the affinity threshold, then the whole parameter is marked as self, therefore the detector is eliminated in order to be replaced by a new one.

Candidate detectors that pass the Negative Selection process are stored into a file and are used in the detection phase. The creation process is repeated until the desired amount of detectors is obtained.

Algorithm 1 Main frame of the generation algorithm

```
Input: n-gram length, $ng$
Input: Poisson lambda, $plambda$
Input: Affinity Threshold, $aff$
Output: Set of detector strings, $Ds$
  $self\_dataset$ = read self dataset from file
  $ngram\_dataset$ = split dataset as n-grams($self\_dataset, ng$)
  eliminate recurring strings ($ngram\_dataset$)
  $Ds$ = $generate\_detectors(ngram\_dataset, plambda, aff)$
  save detectors to file($Ds$)
```

Algorithm 2 Detector generation with the Negative Selection Algorithm

```
Input: Self dataset that the ngram_dataset is formed, $self\_dataset$
Input: Dataset formed of ngram, $ngram\_dataset$
Input: Poisson lambda, $plambda$
Input: Affinity Threshold, $aff$
Input: Desired number of detectors, $max\_detectors$
Output: Set of detector strings, $detector\_list$
  $detector\_list$ = create an empty list()
  while $length(detector\_list) < max\_detectors$ do
    number_of_selections = calculate detector length with Poisson($plambda$)
    candidate_detector = select a number of random n-grams($number\_of\_selections, ngram\_dataset$)
    candidate_detector = negative_selection($candidate\_detector, self\_dataset, aff$)

    if candidate detector survives NS then
      add candidate detector to $detector\_list$
    end if
  end while
  return $detector\_list$
```

Algorithm 3 Implementation of the Negative Selection Algorithm

```
Input: Self dataset, $self\_dataset$
Input: Detector to be tested, $candidate\_detector$
Input: Affinity Threshold, $aff$
Output: $candidate\_detector$ or $NULL$
  $detector\_length$ = length(candidate_detector)
  for each $set\ in\ self\_dataset$ do
    $self_substrings$ = split self into ngrams with length of detector_length
    for each $substring\ in\ self\_substrings$ do
      $affinity$ = calculate affinity with levenshtein_distance($detector, substring$)
      if $affinity$ > $aff$ then
        eliminate detector
        return $null$
      end if
    end for
  end for
  return $candidate\_detector$
```

Algorithm 4 Affinity calculation with Levenshtein Distance

```
Input: Detector, $det$
Input: Self sample, $self$
Output: Similarity percentage of $det$ and $self$
  $distance$ = levenshtein distance of $det$ and $self$
  if len($det$) > len($self$) then
    $longer\_string = det$
  else
    $longer\_string = self$
  end if
  $percentage$ = 100 - ($distance$ * 100 / len($longer_string$))
  return $percentage$
```

3.3 Detection Phase

The detection phase is the phase that applies the detectors that are generated in the generation phase to the test datasets to obtain the detection results. To calculate the success measurements, a detection program is developed. The program takes two parameters which are the affinity threshold and the test dataset's file name. The test dataset is divided into two datasets. One of them contains only normal and the other one contains only abnormal traffic. This is done to decrease the completion of the program's execution time and to calculate True Positive, False Positive, True Negative, and False Negative ratios more precisely.

The program reads the detectors from the storage file and the dataset given as the parameter. The variable that stores the HTTP request parameters is copied to another variable. When a parameter is detected as malicious, it is excluded from the second variable. This is done for the purpose of revealing the undetected parameter strings.

Every detector is applied to the dataset. The same problem mentioned in the generation phase with the difference of length between the detector string and the parameter applies with the detection phase as well. Based on the detector string's length, the parameter that is being compared with is divided into substrings with the same length as the detector. The affinity calculation is done with each substring. If the calculated value exceeds the threshold, unlike the generation phase, it is considered as a non-self sample. When the detection occurs in a substring of a parameter, then the whole string is treated as malicious/abnormal.

At the end of the execution, the program outputs hit counts for every detector, hit rates per detector which is the percentage of hit count over the total dataset's size, execution time consumed for each detector and the HTTP GET parameters that are not detected as malicious.

Algorithm 5 Detection algorithm

```
Input: Test dataset, $test\_set$
Input: Affinity Threshold, $aff$
Input: Detector set, $Ds$
Output: Hit count for each detectors in $Ds$ and undetected samples in $test\_set$
  $undetected\_samples$ = copy(test_set)
  for each $sample\ in\ test\_set$ do
    for each $detector\ in\ Ds$ do
      $substrings$ = csplit sample into n-grams with length of detector
      for each $substring\ in\ substrings$ do
        $affinity$ = calculate affinity with levenshtein_distance($detector$, $substring$)

        if $affinity > aff$ then
          increase hit count of detector
          undetected_samples.remove($sample$)
          $Break$
        end if
      end for
    end for
  end for
  return hit count for each detector, $undetected\_samples$
```

CHAPTER 4

EXPERIMENTS

In this study, the Negative Selection Algorithm [5] is implemented and metrics for both generation and detection phases are gathered. The information extracted from these metrics is presented in this section. The algorithm in this study uses n-gram strings for detector generation. The n-gram strings are obtained from the training dataset that contains only HTTP [8] GET parameters that are not malicious or abnormal. The number of n-gram strings to be concatenated is calculated with the Poisson Distribution Probability with several lambda values. The program developed is run with different n-gram string lengths, Poisson lambda values, and affinity thresholds.

The purpose of this study is to generate detector strings with n-gram substrings and to find which n-gram length gives the best result in both generation and detection phases. While the success of a detector is based on its detection rates, it is also sought that a detector must be generated within a reasonable amount of time, with minimum attempts in the Negative Selection Algorithm and has the least possible string length.

Several experiments have been performed with n-gram lengths of 2, 3, 4, 5 and Poisson lambda parameters of 3, 4, 5. The output of the Poisson Probability function gives the number of n-grams to be joined. The detector's size is the product of the n-gram's length and the output of the Poisson. If the longest n-gram is chosen and the Poisson Probability gives 5 for the number of n-gram selections, then the resulting string will have 25 characters in total. For different n-gram lengths, the generation algorithm will output detectors that differ in length. This feature that is added with Poisson is thought to increase detection rates by adding detectors with different lengths to the detector set.

In the earlier steps of this study, the minimum affinity threshold that allows detector generation in a reasonable amount of time has experimented. The experiment is done by slightly modifying the code for the detector generation phase. The modification made was to limit the number of attempts needed to successfully create a single detector. The number was defined as 100.000. It is observed that the program fails to generate a detector in the stated number of attempts if the threshold value is below 20 percent. This observation leads the study for using 20%, 21% and 22% as affinity threshold parameters.

To gather the metrics of the detector generation phase in the proposed algorithm, the generator program is executed for 24 hours. The execution is done in separate processes for different input parameters within a machine that runs with Intel Xeon CPU E5-2680 at 2.40GHz. The program generated as many detectors as possible in this time frame. For every detector following metrics are logged during the generation.

- The max affinity calculated in the negative selection process which can be for some detectors lower than the affinity threshold that the program is executed with.
- Number of attempts to finally procure the detector.
- Output of the Poisson Probability function which determines the number of n-gram selections to form the detector string.
- Detector's length in string size.
- N-gram, affinity threshold and Poisson lambda parameters that the program is executed with.

After the generation phase, the detectors that are created are stored in separate files based on their execution parameters. The stored detectors are applied to the test dataset in the detection phase. The phase uses a dataset that contains both normal and abnormal/malicious samples. The detectors are expected to detect only the malicious strings. Like the generation phase, several metrics are gathered in the detection phase as well. These metrics are logged for every distinct detector.

- Total time passed while the detector exercise over the whole dataset.
- Number of samples in the dataset that are recognized as malicious by the detector.
- Percentage of the remaining items in the dataset that are undetected by any detectors.

Information gathered in the detection phase is used to calculate the success rates of the detectors. These rates are;

- True Positive(TP): Percentage of the number of the detected malicious strings in the dataset over the number of the total number of malicious samples exist.
- True Negative(TN): Percentage of the number of undetected samples which are normal samples indeed over the total number of normal strings.
- False Negative(FN): Percentage of undetected malicious strings in the dataset over the number of total malicious samples.
- False Positive(FP): Percentage of normal samples that are marked as malicious over the total number of normal strings in the dataset.

With the information gathered in both phases, several tables and graphs are generated. The following table presents the number of attempts needed to generate a single detector that is formed with 2-grams.

Table 4.1: Number of generation attempts needed for detectors formed of 2-grams

Affinity Threshold	**λ=3**	**λ=4**	**λ=5**
20%	27736	5803	1876
21%	21821	5051	1334
22%	5612	1230	490

As Table 4.1 depicts, while the lambda and affinity threshold values increase, the number of attempts to generate a detector decreases. Higher lambda values cause longer detector strings. As the detector string length increases, the Levenshtein Distance Algorithm generates higher distance results because the samples in the dataset can be relatively shorter than the candidate detector. Higher distance results relieve the generation process and the detectors pass the Negative Selection easier. The same behavior occurs as the affinity threshold value increases. As the threshold of similarity between detector strings and the dataset samples increase a candidate detector's probability to overcome the Negative Selection process increases as well.

The same behavior can be observed by examining Table A.1, Table A.2, Table A.3 which can be found in the Appendix section of this document.

Figure 4.1, Figure A.1 and Figure A.2 introduce the distribution of the generated detectors' lengths based on the affinity threshold and the Poisson lambda value which determines the number of n-grams selection from the dataset to form a detector.

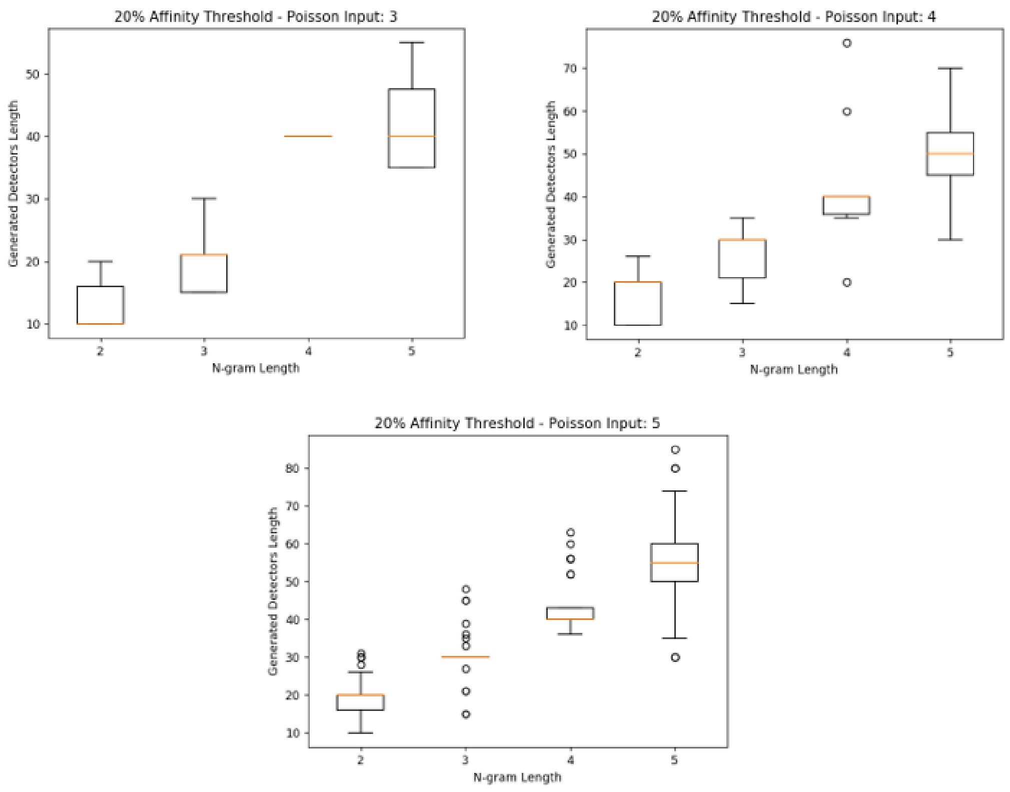

Figure 4.1: Detector length distribution (20% Affinity Threshold)

The string length variance that is achieved by the Poisson Distribution can be seen from the Figure 4.1, Figure A.1 and Figure A.2. When the results are analyzed further, it can be observed that there are only a few detectors that could be generated with the expected length which is determined by the Poisson lambda value. This situation occurred because of the Poisson lambda value, Levenshtein Distance Algorithm's behavior and the affinity threshold and it will be further analyzed in the Discussion section.

The following table presents the detection success rates of the detector sets generated with Poisson lambda 5 and 20% affinity threshold. This detector set is chosen over the others since the generation phase when run with lambda values of 5 resulted more detectors than the other executions. The Negative Selection Algorithm results better at the detection phase with lower affinity thresholds. Therefore, 20% affinity threshold value is chosen.

Table 4.2: Detection success rates of the detectors generated with affinity threshold = 20% and Poisson lambda = 5

N-gram	Detector Count	True Positive	True Negative
2	243	35,25	97,24
3	33	2,57	99,39
4	36	0,96	99,36
5	196	8,26	98,38

By examining the results that use different parameters presented at the Table 4.2, it can be observed that the success of the detector sets decreases as the number of detectors in a set decreases and the n-gram length increases.

If it is desired, the rest of the generation and detection results are presented in the Appendix section. Table A.4, A.5 and A.6 presents the detection results for each affinity thresholds, n-gram lengths and Poisson lambda values. The tables also show the number of detectors that could be generated in 24 hours of execution. This experiment is carried out to see the potential of the detectors in detecting anomalies when the program is freed to generate as many detectors as possible without any attempt limit in contrary to the experiment done to determine the minimum affinity thresholds achievable.

When the detection success results of the experiment shown in the Table A.4, A.5 and A.6 are analyzed, it is seen that the detection rates are not sufficient for anomaly detection. While the detector sets failed to approach the desired rates in True Positive metrics, the ability to discriminate self reflects on the True Negative rates. The reasons for this observation are further explained in the Discussions Chapter.

The tables are analyzed to conclude that the detector sets that are formed by 2-grams are better than the rest of the detector sets. The number of attempts to create a detector with 2-grams and the mean string length of those detectors are lower. When the Poisson lambda is treated constant, the higher the n-gram length, the lower the True Positive rates become. From observing this behavior, it is concluded that the n-gram strings must be chosen short. Therefore another experiment is carried out.

In the next experiment, 1-grams are tested against the problem. Detectors with 2-gram substrings are generated as well to compare the success rates of the 1-gram detectors. To make the experiment ideal, 300 detectors are generated for each Poisson lambda and affinity threshold combination. It was possible to create 300 detectors with 3-gram substrings by the Poisson lambda 5 and 21% affinity for further comparison. Other combinations could not be generated because of time limitations.

It is witnessed in the experiment that 1-gram detectors outperformed other detectors at both generation and detection phases in every affinity threshold. Furthermore, they are tested against lower affinity thresholds and 12% affinity is achieved. 2-gram and

3-gram detectors showed similar results to the former experiments.

At the following table, generation attempt count and length information of the detectors formed by 1-grams are depicted. Minimum, maximum, average and mode length of the detectors are presented to point to the correlation between the length of a detector and the Poisson lambda value which will be further discussed in the following section.

Table 4.3: Generation phase specifications of the detectors generated by 1-grams

	Min. Len	**Max. Len**	**Avg. Len**	**Mode Len.**	**Avg. Attempts**
Affinity = 12	9	18	9,4	9	117618
Affinity = 15	7	14	7,36	7	7684
Affinity = 17	6	13	6,82	6	1753
Affinity = 20	5	11	5,5	5	661

It can be observed from the Table 4.3 that it is easier to create a detector with 1-gram substrings than generating a detector formed by any other n-gram lengths. As the affinity threshold decreases, it becomes difficult for the generation phase to generate detectors.

At the following table detection rates for the generated detectors in the final experiment are shown. Each detector set contains 300 detectors. To compare 1-gram detectors with 2-gram detectors, detector sets that are generated with Poisson lambda 3 and affinity threshold 20% are chosen. Likewise, to add 3-gram detectors to the comparison, Poisson lambda 5 and affinity threshold 21% are chosen.

Table 4.4: Detection results for 300 detectors

Affinity	**N-gram**	**P. Lambda**	**True Positive**	**True Negative**	**Accuracy**
12	1	9	96,67	98,33	0,975
15	1	7	98,44	98,03	0,982
17	1	6	97,21	98,28	0,977
20	1	3	97,42	98,95	0,982
20	2	3	66,76	95,59	0,812
21	1	5	97,60	98,24	0,979
21	2	5	34,77	96,48	0,656
21	3	5	16,22	95,47	0,558

It can be observed from the Table 4.4 that 1-gram detector sets resulted better than the other detector sets that contain the same number of detectors. Complete data of the experiment with additional detector sets and success metrics can be examined from the Table A.7 and Table A.8 in the Appendix section of the document.

At the detection phase, every detector in the detector set examines the whole test dataset. The process takes a certain amount of time for every detector which is highly affected by the number of samples in the test dataset and the Levenshtein Distance Algorithm. Moreover, a detector can match with several samples from the test dataset. A match means that the detector recognized the sample as a malicious payload. The percentage of the detected sample count over the whole dataset is named Hit Rate. The following table depicts the average execution time per detector over the dataset and the average hit rate.

Table 4.5: Detection phase specifications of the detectors generated by 1-grams

	Avg. Hit Rate Per Detector	**Avg. Exec. Time Per Detector**
Affinity = 12	19,48	14423
Affinity = 15	14,85	12504
Affinity = 17	11,47	11516
Affinity = 20	9,00	8912

CHAPTER 5

DISCUSSION

In the experiments and results presented in the previous chapter, it is shown that detectors that are created using 1-grams perform better in anomaly detection using the Negative Selection Algorithm. If the n-gram length is increased, it results in lower detection rates as well as challenges in generating a single detector.

1-gram strings mean that the generation must be done with the character set of the training dataset. Generated detectors have the same character set with the dataset. If the generation is done with all the possible characters, regardless of the dataset's characters, it can result in False Positives. In the Negative Selection process of the Human Immune System, the T-Cells are only subjected to the self samples. If a character that is not in the dataset is added to the generation phase, the distance results of the Levenshtein algorithm will increase in the detection phase and the generated detector will result in farther distances with the strings that contain any anomalies. This calculation will lead to marking the normal string to be malicious. Thus it will increase the False Positive rates. For these considerations, the generation and detection phases are only done with the dataset's character set.

There are numerous self samples in the training dataset and malicious attack payloads in the training dataset. The strings in these datasets vary in length. Both in the generation and detection phases, these length differences are handled by the algorithm since it splits the dataset string to the substrings with the length of the detector. This operation is done for every detector. Thus, a detector is only compared with strings that have the same or shorter lengths. This approach avoids possible False Positives and False Negatives. The shorter the detector's length than the dataset's string, the farther they become in the Levenshtein Distance Algorithm. This deceives the detection algorithm to calculate proper results. A dataset string can be marked incorrectly, even if the detector string is its substring if the length difference is high enough. This is not an outlier case when the attack payloads are examined. While some payloads in the SQLi, XSS and Path Traversal attacks can be too long, it is known that increasing the attack payload's length is a technique used by the malicious actors to deceive the security mechanisms.

The situation explained in the paragraphs above with the lengths of the detectors and dataset strings also applies to the number of attempts to create a single detector. Poisson probability gives the number of n-gram string selection with the input value provided to the generation program. Resulting output and the product of the n-gram length gives the length of the detector to be formed. Later the detector will be tested against the training dataset and required to have a lower affinity than the determined threshold. If it exceeds the threshold value, then the whole process is repeated. This counts as the mentioned attempt. If the detector's length is too long for the strings in the training dataset, then the affinity decreases. This is seen as proper for the Negative Selection Algorithm and the detector passes the test successfully. The resulting detector fails to detect malicious activity since it is generated incorrectly. This condition decreases the number of attempts needed because it relieves the limitation of affinity in the Negative Selection. Moreover, the generation program can output more detectors in a given time period. The explained behavior can be observed from Table A.1, A.2 and A.3.

By analyzing the Figure 4.1, A.1 and A.2, it can be seen that the generated detector's n-gram selection count exceeds the Poisson lambda value and the detectors' lengths are longer than expected. Detector generation is made possible by the Poisson Distribution's outputs that are higher than the given input value. To take as an example, when the output of the n-gram = 2, Poisson lambda = 3 at the 20% affinity threshold is examined, the minimum length of the generated detector set is 10. This means 5 selections are done with 2-gram strings(Poisson Probability output is 5). If a detector could be generated by the input value of 3, as it is given to the program as a parameter, the resulting detector's length is expected to be 6. This is seen to be impossible by the algorithm. When the lengths of the detectors generated by 1-grams are observed, it is seen that the detector with the minimum length is 5 (Table 4.3). With the observations on the other detectors with different affinity thresholds, it is concluded that because of the Levenshtein Distance and the Negative Selection process, the expected detector's n-gram count can be found by dividing 100 with the affinity threshold. For the same example mentioned, the equation results 5 with the affinity threshold of 20%. This result holds the data displayed in the tables and the figures, 2-gram detectors had minimum lengths of 5 times 2 and 1-gram detectors resulted with 5 times 1. The reason behind the longer minimum lengths of the other detectors is that the detectors fail the Negative Selection process when they are formed with the number of the n-grams that the equation explained results.

It is also concluded from the study that longer detector lengths result in longer execution times. This obvious condition can be eluded by some detectors. The detection algorithm splits the dataset string to be compared with the detector into substrings that can have the maximum length of the detector's. If the affinity threshold is exceeded in the initial comparisons with the substrings, the algorithm marks the dataset string to be malicious and continues with the next one, leaving other substrings. Another detector may need to end all the substrings in every dataset strings, especially when the success of the detector is low. The time taken for the execution on the dataset for a single detector can be seen in Figure 5.1.

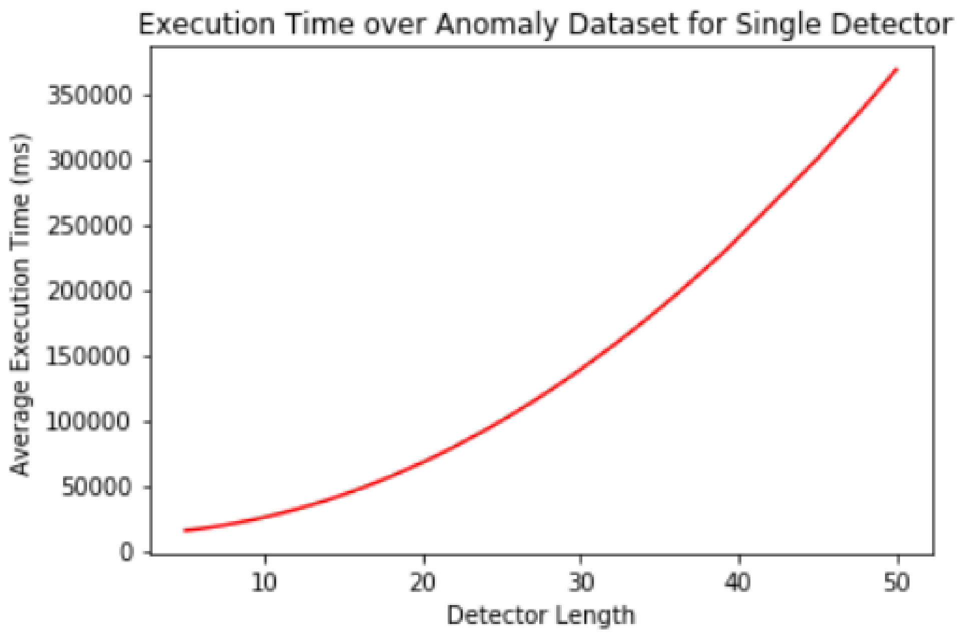

Figure 5.1: Execution time for a single detector over the dataset

From the experiments conducted with the 300 detectors formed with 1-grams, 2-grams and 3-grams for 21% affinity threshold and with the same amount of detectors with 1-grams and 2-grams for 20% affinity threshold, it can be seen from the Figure 5.2 and 5.3 that detectors generated from 1-gram strings cover the anomaly set more than the other detectors formed with 2-grams and 3-grams by using fewer amount of detectors. The line for 1-gram detectors has higher growth rates in both affinity thresholds. There is a need for a higher number of detectors for 2-gram and 3-gram detector sets to cover the same area. The growth rates in the line graphs also show the hit rates for the detectors. Each detector marks several strings on the dataset as malicious. The figures show the growth in the True Positive percentage created by these detectors.

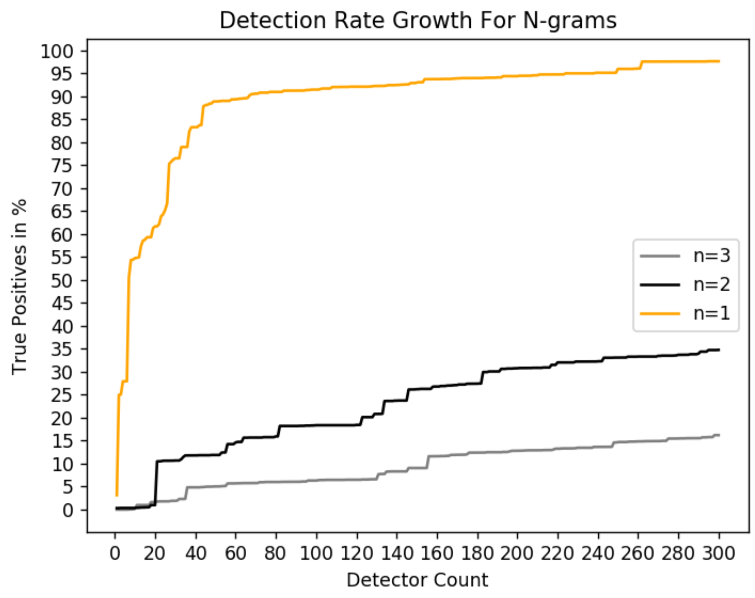

Figure 5.2: Detection rate growth for different n-grams (21% affinity threshold)

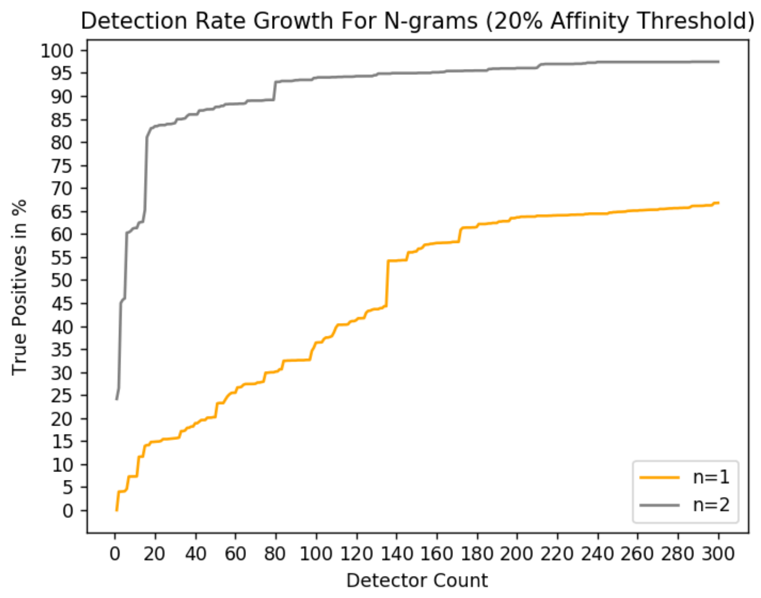

Figure 5.3: Detection rate growth for different n-grams (20% affinity threshold)

Figure 5.4 shows the hit rates of the detectors tested for 21% affinity threshold. Detectors generated by the 1-gram strings have ranging hit rates that are close to 70% for some detectors. This means that there is at least one detector that can detect 70% of the anomalies in the dataset only by itself. Hit rates for the detectors with 2-grams and 3-grams are much lower.

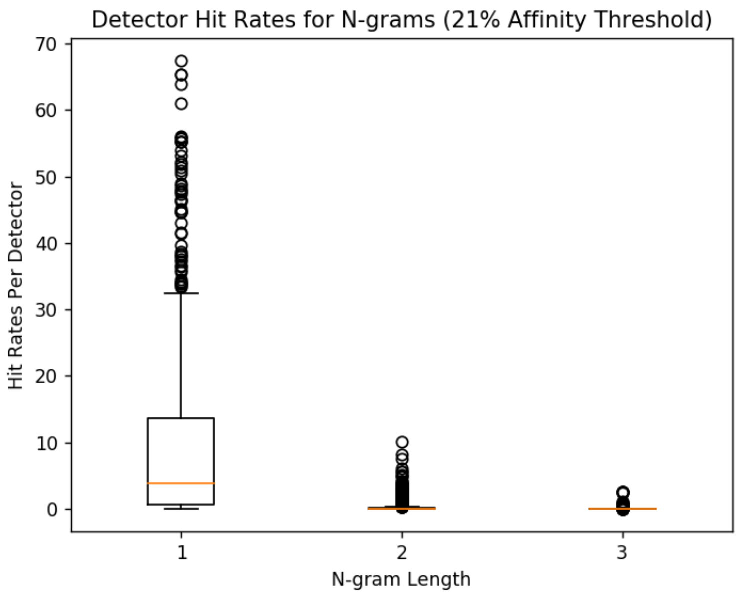

Figure 5.4: Detector hit rates for different n-grams (21% affinity threshold)

Figure A.3, Figure A.4 and Figure A.5 can be observed from the Appendix section. The figures presents hit rates for each n-gram length at 21% affinity threshold.

It is stated in the Results and Experiments Chapter that the affinity thresholds for detectors with 1-gram strings could be decreased to 12%. Figure 5.5 shows the line graph of True Positive rate growth for each detector set that is generated for distinct affinity thresholds. It can be seen from the figure that the initial 20 detectors for each set covers the same percentage and the difference from the affinity thresholds takes effect between 20. and 180. detectors. After the 200th detector of each set is applied to the dataset, the results become close to each other.

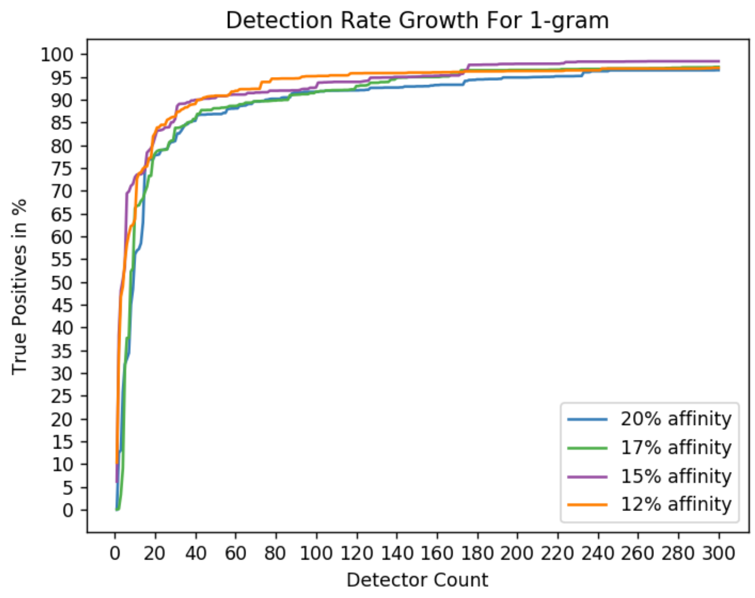

Figure 5.5: Detection rate growth for 1-gram

The experiments are conducted by Python [26] scripts that present the proof-of-concept. Each of the attack types targeted to be detected is conducted from the user input values. A malicious string-based payload that is submitted can harm other users or corrupt the server. To prevent a predicament, every parameter in every distinct HTTP request must be examined.

In real-world usage, the generation and the detection phases of the proposed method must be implemented distinctly. These two phases arise some challenges. Detectors generated from the request history must be validated periodically to support parameters with highly dynamic values. There may be new parameter values that have not been submitted before to the application/service. To give an example, this situation can occur if the parameter holds a new patient's data in a hospital information management system. To cover the fresh data that is just added to the system's database, a new detector must be generated and replaced with an old detector in the detector dataset. The detector that is being replaced must not have been activated by a suspicious request before which means that the detector covers a non-self space that has a low probability of occurrence. This concludes that the activation count for the detectors must be stored as well and detectors must be regenerated despite the data is static if they have not been activated for a given time frame.

Protecting parameters with dynamic data also leads to difficulties while generating detectors since parameters with diverse values may include most of the combinations of the overall character set. This restrains the generation phase to generate only a few distinct detectors and the attempts needed -generation time- highly increases. The affinity threshold value must be relieved to overcome the stated challenge. But by increasing the threshold value the false positive and false negative rates shall increase while the attempts needed to generate a single detector may decrease. Disability to generate the desired amount of detectors in an acceptable amount of time or having high false negative/positive rates when the affinity threshold is increased makes the proposed method unsuitable for attack detection.

The biggest challenge for the detection phase is that the parameters to be examined must be exposed to the previously generated detectors. Since affinity calculation with the Levenshtein distance algorithm requires several steps, every parameter-detector comparison takes an amount of time. The computing time may lead to latencies in the response time of the server to the requests if the real-world application of the proposed method is being used in-line. In the experiments section to this study, it is shown that to examine a dataset with 3920 parameter values with an average length of 89 took 8912 milliseconds for 300 detectors with an average length of 5. It can be calculated from the stated data that each parameter-detector comparison causes 2 milliseconds of latency. The latency value may drop if the parameter to be examined has values with shorter lengths and increase otherwise.

CHAPTER 6

CONCLUSION

The biological immune system is responsible to protect the body from the pathogens. A pathogen is a hazardous organism that can cause disease. The Human Immune System consists of two subsystems. Innate Immune System forms the first layer of protection. It reacts generally to the invaders by changing the environmental conditions in the body or in a specific part of the body to prevent the survival and reproduction of the microbe. The skin is one of the parts of the Innate Immune System. It has also have the capability to devour the detected pathogens. Adaptive Immune System is where special kinds of lymphocytes reside, B- and T-cells. These lymphocytes are created continuously in the human life span. Adaptive immunity has the ability to learn new types of pathogens as they are eliminated from the body. If the same pathogen tries to invade the body in the future, the reaction of the body is immediate.

Negative Selection of T-cells, in which the T-cells gain the capability to discriminate self from non-self, forms the basis structure of this study. In Negative Selection, the T-cells that react to self proteins are eliminated directly before they are sent to the veins and lymph. Only those who do not react to self is left after the process. When this process is applied to the computer security domain, every detector that does not match self and survives the Negative Selection can be thought to demonstrate an anomaly case. If a detector matches by a certain threshold any data in the web application traffic, it can be concluded that the traffic contains an abnormal data.

In this thesis, a Negative Selection Algorithm based method is proposed. The proposed method generates detectors to detect anomalies in web application traffic by examining the HTTP GET requests. The parameters in the HTTP GET requests are used by malicious actors to attack and exploit the vulnerabilities on a web application or service. These attacks commonly arise from the lack of input validation that must be done on the server-side of a software. Data about the generation phase of the detectors and detection phase are gathered during the execution of the implementation of the method. The data presented in the Experiments chapter.

This study focuses on the metrics of the first generation of the detector set that is created by the proposed method. Success results of the detectors in the forms of True Positive, False Positive, True Negative and False Negative are shared. Besides these results, the number of detector generation attempts needed to successfully create a single detector, the length distribution of the generated detectors, detection rates for each detector and the count of the generated detectors in a time window are also presented in this paper.

The generation of the detectors is done by combining n-gram strings with different lengths. Detection results when the n-gram length and affinity threshold changes are examined. The success of a detector is based on its individual detection rate, the ease to create a similar detector, its execution time over a given dataset and the string length of detectors. The study concludes that using 1-grams to create a detector is beneficial, rather than using 2-grams, 3-grams, 4-grams, and 5-grams. Detectors generated using 1-grams outperformed every other detector in every distinct affinity threshold. The criteria of success of a detector mentioned formerly best suits those detectors.

By the end of this thesis, it is deduced that, if Levenshtein distance is used for affinity calculation, 15% affinity threshold is required and average of 7 1-grams are used to form a detector, over 98% True Positive and True Negative rates can be achieved by using the Negative Selection Algorithm for detecting web-based attacks.

REFERENCES

[1] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary computation*, vol. 8, no. 4, pp. 443–473, 2000.

[2] J. Timmis, M. Neal, and J. Hunt, "An artificial immune system for data analysis," *Biosystems*, vol. 55, no. 1-3, pp. 143–150, 2000.

[3] C. A. Janeway, P. Travers, M. Walport, M. Shlomchik, *et al.*, *Immunobiology: the immune system in health and disease*, vol. 7. Current Biology London, 1996.

[4] K. A. Hogquist, S. C. Jameson, W. R. Heath, J. L. Howard, M. J. Bevan, and F. R. Carbone, "T cell receptor antagonist peptides induce positive selection," *Cell*, vol. 76, no. 1, pp. 17–27, 1994.

[5] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Proceedings of 1994 IEEE computer society symposium on research in security and privacy*, pp. 202–212, Ieee, 1994.

[6] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[7] J. B. Kruskal, "An overview of sequence comparison: Time warps, string edits, and macromolecules," *SIAM review*, vol. 25, no. 2, pp. 201–237, 1983.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[9] *SQLMap: Automatic SQL Injection and Database Takeover Tool*. `http://sqlmap.org/`.

[10] *Cross Site Scripting Scanner Vulnerability Confirmation*. `https://github.com/yehia-mamdouh/XSSYA`.

[11] *FuzzDB: Dictionary of attack patterns and primitives for black-box application fault injection and resource discovery.* `https://github.com/fuzzdb-project/fuzzdb`.

[12] *Vega Scanner.* `https://github.com/subgraph/Vega/wiki/Vega-Scanner`.

[13] J. G. Skellam, "A probability distribution derived from the binomial distribution by regarding the probability of success as variable between the sets of trials," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 10, no. 2, pp. 257–261, 1948.

[14] D. D. Chamberlin and R. F. Boyce, "Sequel: A structured english query language," in *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pp. 249–264, ACM, 1974.

[15] D. Raggett, "Html+ (hypertext markup language)," 1993.

[16] H. W. Lie and B. Bos, "Cascading style sheets, level 1," 1996.

[17] N. Gabrielli and M. Rigodanzo, "An artificial immune system for network intrusion. detection on a web server: First results," in *Proceedings of the 2nd Italian Workshop on Evolutionary Computation (GSICE 2006)*, 2006.

[18] M. Danforth, "Towards a classifying artificial immune system for web server attacks," in *2009 International Conference on Machine Learning and Applications*, pp. 523–527, IEEE, 2009.

[19] F. Zhang and Y. Ma, "Integrated negative selection algorithm and positive selection algorithm for malware detection," in *2016 International Conference on Progress in Informatics and Computing (PIC)*, pp. 605–609, IEEE, 2016.

[20] A. J. Saleh, A. Karim, B. Shanmugam, S. Azam, K. Kannoorpatti, M. Jonkman, and F. D. Boer, "An intelligent spam detection model based on artificial immune system," *Information*, vol. 10, no. 6, p. 209, 2019.

[21] D. Hooks, X. Yuan, K. Roy, A. Esterline, and J. Hernandez, "Applying artificial immune system for intrusion detection," in *2018 IEEE Fourth International*

Conference on Big Data Computing Service and Applications (BigDataService), pp. 287–292, IEEE, 2018.

[22] O. Igbe, *Artificial Immune System Based Approach to Cyber Attack Detection*. PhD thesis, The City College of New York, 2019.

[23] Z. Ataser, *Variable Shaped Detector: A Negative Selection Algorithm*. PhD thesis, Citeseer, 2013.

[24] T. M. Cover, P. Hart, *et al.*, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[25] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, pp. 93–104, ACM, 2000.

[26] *Python*. `https://www.python.org/`.

APPENDIX A

TABLES AND FIGURES OF THE GENERATION AND THE DETECTION PHASES

Table A.1: Average number of generation attempts (20% affinity threshold)

20% (Affinity Threshold)	λ=3	λ=4	λ=5
n-gram=2	27736	5803	1876
n-gram=3	285783	78302	20023
n-gram=4	1315057	57604	11076
n-gram=5	42719	5671	1236

Table A.2: Average number of generation attempts (21% affinity threshold)

21% (Affinity Threshold)	λ=3	λ=4	λ=5
n-gram=2	21821	5051	1334
n-gram=3	79878	13478	6562
n-gram=4	177771	12622	2760
n-gram=5	48085	3309	861

Table A.3: Average number of generation attempts (22% affinity threshold)

22% (Affinity Threshold)	λ=3	λ=4	λ=5
n-gram=2	5612	1230	490
n-gram=3	76174	12108	2130
n-gram=4	11210	1956	718
n-gram=5	21987	1299	234

Table A.4: Detection results for the detectors generated in 24 hours (20% affinity threshold)

Parameters	**TP**	**TN**	**Detector Count**
n-gram=2, Poisson Lambda=3	27,42	99,28	48
n-gram=2, Poisson Lambda=4	23,18	99,04	84
n-gram=2, Poisson Lambda=5	35,25	97,24	243
n-gram=3, Poisson Lambda=3	0,71	99,90	5
n-gram=3, Poisson Lambda=4	0,56	99,85	9
n-gram=3, Poisson Lambda=5	2,57	99,39	33
n-gram=4, Poisson Lambda=3	0,02	99,96	1
n-gram=4, Poisson Lambda=4	2,09	99,65	17
n-gram=4, Poisson Lambda=5	0,96	99,36	36
n-gram=5, Poisson Lambda=3	0,86	99,79	11
n-gram=5, Poisson Lambda=4	11,35	98,50	102
n-gram=5, Poisson Lambda=5	8,26	98,38	196

Table A.5: Detection results for the detectors generated in 24 hours (21% affinity threshold)

Parameters	**TP**	**TN**	**Detector Count**
n-gram=2, Poisson Lambda=3	50,40	98,6	104
n-gram=2, Poisson Lambda=4	41,45	97,35	178
n-gram=2, Poisson Lambda=5	13,64	98,95	121
n-gram=3, Poisson Lambda=3	2,21	99,42	30
n-gram=3, Poisson Lambda=4	1,96	99,37	29
n-gram=3, Poisson Lambda=5	2,21	99,65	29
n-gram=4, Poisson Lambda=3	6,93	99,87	8
n-gram=4, Poisson Lambda=4	0,81	99,67	21
n-gram=4, Poisson Lambda=5	1,30	99,34	61
n-gram=5, Poisson Lambda=3	1,58	99,71	24
n-gram=5, Poisson Lambda=4	7,14	99,39	62
n-gram=5, Poisson Lambda=5	19,13	96,94	389

Table A.6: Detection results for the detectors generated in 24 hours (22% affinity threshold)

Parameters	**TP**	**TN**	**Detector Count**
n-gram=2, Poisson Lambda=3	45,43	96,46	271
n-gram=2, Poisson Lambda=4	57,34	94,95	435
n-gram=2, Poisson Lambda=5	26,02	97,88	176
n-gram=3, Poisson Lambda=3	7,29	99,51	31
n-gram=3, Poisson Lambda=4	0,07	99,70	24
n-gram=3, Poisson Lambda=5	15,10	96,77	250
n-gram=4, Poisson Lambda=3	5,22	99,39	44
n-gram=4, Poisson Lambda=4	32,44	96,80	289
n-gram=4, Poisson Lambda=5	29,97	95,34	408
n-gram=5, Poisson Lambda=3	0,81	99,82	16
n-gram=5, Poisson Lambda=4	3,72	99,12	141
n-gram=5, Poisson Lambda=5	10,66	97,75	363

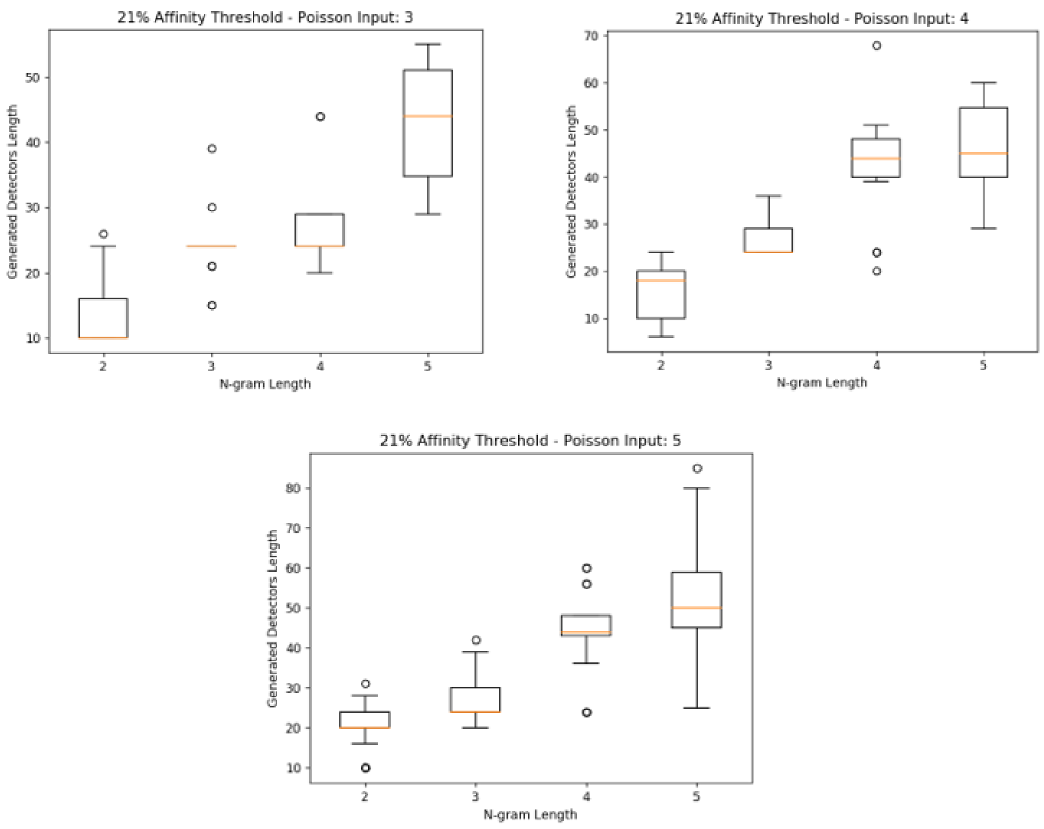

Figure A.1: Detector length distribution (21% affinity threshold)

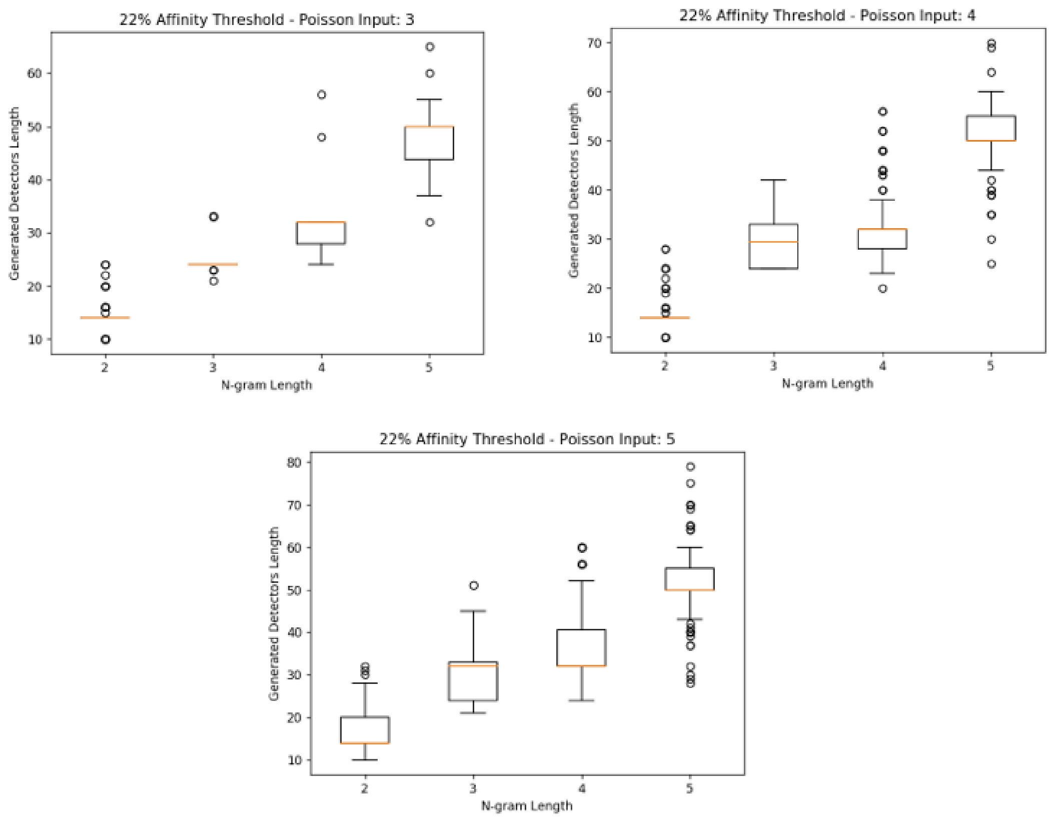

Figure A.2: Detector length distribution (22% affinity threshold)

Table A.7: Detection results for 300 detectors (complete)

Parameters	**Affinity**	**TP**	**TN**	**FP**	**FN**
n-gram=1, Poisson Lambda=9	12	96,67	98,33	1,67	3,33
n-gram=1, Poisson Lambda=7	15	98,44	98,03	1,97	1,56
n-gram=1, Poisson Lambda=6	17	97,21	98,28	1,72	2,79
n-gram=1, Poisson Lambda=3	20	97,42	98,95	1,05	2,58
n-gram=1, Poisson Lambda=4	20	96,32	98,80	1,20	3,68
n-gram=1, Poisson Lambda=5	20	96,47	98,70	1,30	3,53
n-gram=2, Poisson Lambda=3	20	66,76	95,59	4,41	33,24
n-gram=2, Poisson Lambda=4	20	64,28	95,82	4,18	35,72
n-gram=2, Poisson Lambda=5	20	33,87	96.68	3.31	66,13
n-gram=1, Poisson Lambda=3	21	97,01	98,72	1,28	2,98
n-gram=1, Poisson Lambda=4	21	96,55	98,92	1,08	3,45
n-gram=1, Poisson Lambda=5	21	97,60	98,24	1,76	2,40
n-gram=2, Poisson Lambda=3	21	64,38	96,01	3,98	35,62
n-gram=2, Poisson Lambda=4	21	48,82	96,20	3,80	51,18
n-gram=2, Poisson Lambda=5	21	34,77	96,48	3,52	65,23
n-gram=3, Poisson Lambda=5	21	16,22	95,47	4,53	83,78
n-gram=1, Poisson Lambda=3	22	97,11	98,89	1,11	2,89
n-gram=1, Poisson Lambda=4	22	97,44	98,72	1,28	2,56
n-gram=1, Poisson Lambda=5	22	96,55	98,49	1,51	3,45

Table A.8: Detection results evaluation for 300 detectors (complete)

Parameters	Affinity	Accuracy	Precision	Recall	F-Score
n-gram=1, Poisson Lambda=9	12	0,975	0,983	0,967	0,975
n-gram=1, Poisson Lambda=7	15	0,982	0,980	0,984	0,982
n-gram=1, Poisson Lambda=6	17	0,977	0,983	0,972	0,977
n-gram=1, Poisson Lambda=3	20	0,982	0,989	0,974	0,982
n-gram=1, Poisson Lambda=4	20	0,976	0,988	0,963	0,975
n-gram=1, Poisson Lambda=5	20	0,976	0,987	0,965	0,976
n-gram=2, Poisson Lambda=3	20	0,812	0,938	0,668	0,780
n-gram=2, Poisson Lambda=4	20	0,801	0,939	0,643	0,763
n-gram=2, Poisson Lambda=5	20	0,966	0,911	0,339	0,494
n-gram=1, Poisson Lambda=3	21	0,979	0,987	0,970	0,978
n-gram=1, Poisson Lambda=4	21	0,977	0,989	0,966	0,977
n-gram=1, Poisson Lambda=5	21	0,979	0,982	0,976	0,979
n-gram=2, Poisson Lambda=3	21	0,802	0,942	0,644	0,765
n-gram=2, Poisson Lambda=4	21	0,725	0,928	0,488	0,640
n-gram=2, Poisson Lambda=5	21	0,656	0,908	0,348	0,503
n-gram=3, Poisson Lambda=5	21	0,558	0,782	0,162	0,269
n-gram=1, Poisson Lambda=3	22	0,980	0,989	0,971	0,980
n-gram=1, Poisson Lambda=4	22	0,981	0,987	0,974	0,981
n-gram=1, Poisson Lambda=5	22	0,975	0,985	0,966	0,975

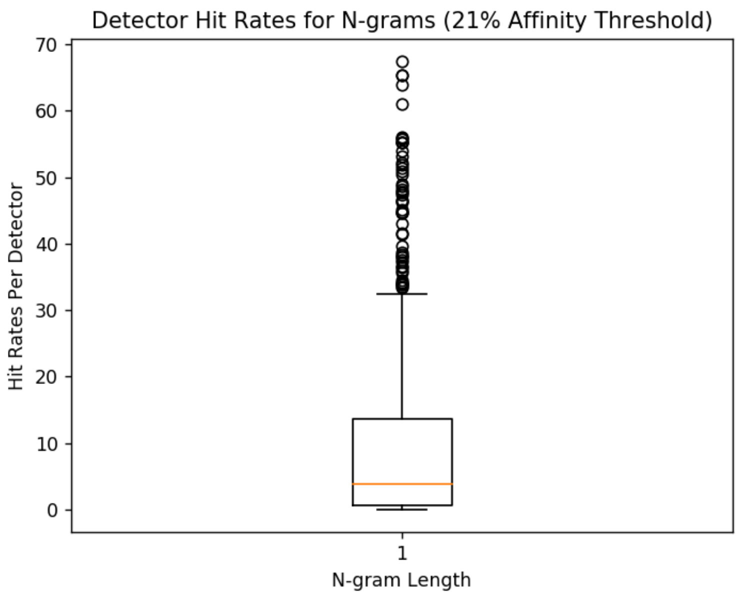

Figure A.3: Detector hit rates for 1-gram (21% affinity threshold)

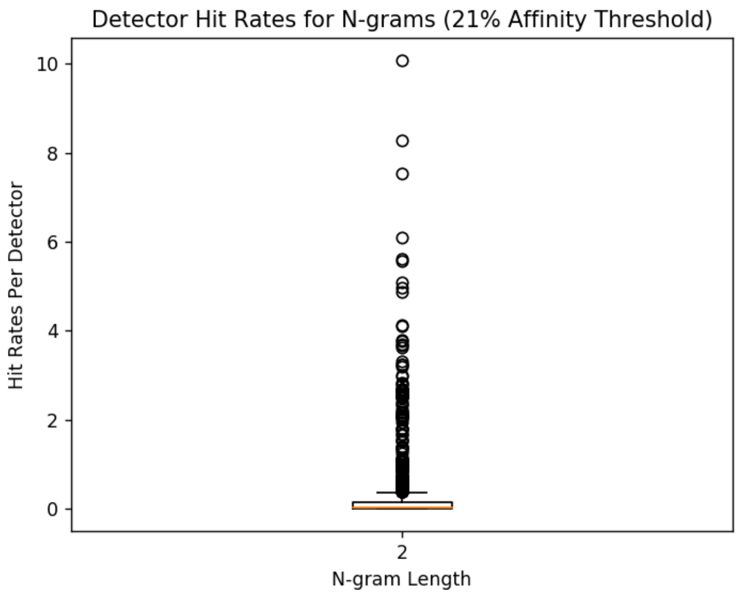

Figure A.4: Detector hit rates for 2-gram (21% affinity threshold)

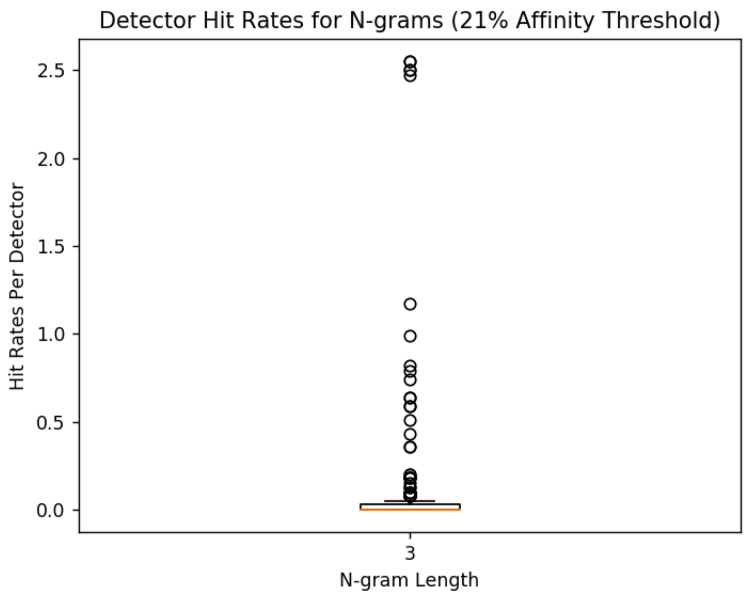

Figure A.5: Detector hit rates for 3-gram (21% affinity threshold)