A Genetic-Algorithm-Based Solution for HTTP-Based Malware Signature Generation

by

Amir Pourafshar

Bachelor of Software Engineering, Amirkabir University of Technology (Tehran Polytechnic), 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor: Ali A. Ghorbani, PhD, Faculty of Computer Science
Examining Board: Michael W. Fleming, PhD, Faculty of Computer Science, Chair
Rodney Cooper, PhD, Faculty of Computer Science
Brent Petersen, PhD, Department of Electrical and Computer Engineering

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

February, 2014

©Amir Pourafshar, 2014
Abstract

The rise in prevalence of malwares has become the most serious threat to Internet security. In order to minimize the devastating impact of this threat many malware detection strategies and systems have been developed, in recent years. This thesis presents a novel malware signature generation and evolution system to detect never-before-seen malwares. We focus on automatic generation of evolved signatures for HTTP-based malware traces based on features and the structure of currently known malwares. The idea is that we can evolve signatures of known malwares to predict the structure of future malware traces since they usually inherit some of their characteristics and structure from their predecessors.

We implemented a proof-of-concept version of our proposed evolutionary signature generation system. Datasets of malicious and legitimate network traffic have been used to evaluate the proposed system. Results from performed experiments show the system’s ability in detecting an acceptable portion of new, unknown malware samples while maintaining a low false alarm
rate. Using the base and evolved signatures together increased the average detection rate of the unknown malicious traces from 38.4% to 50.8%. This improvement happens while the average false positive rate of the evolved signature sets is $2.7 \times 10^{-3}$. 
Dedication

To the sources of true love, my parents.
I would like to express my gratitude and appreciation to my supervisor, Dr. Ali Ghorbani for his guidance and support. I would also like to sincerely thank all my colleagues at the Information Security Centre of Excellence for providing a productive research environment. My heartfelt thanks goes to my dear friends who have encouraged, entertained, cajoled, and supported me through this road.
Table of Contents

Abstract ii

Dedication iv

Acknowledgments v

Table of Contents viii

List of Tables ix

List of Figures 1

1 Introduction 2

1.1 Summary of Contributions 4

1.2 Thesis Organization 5

2 Related Works 7

2.1 Malware Signature Generation 7

2.1.1 Introduction 7

2.1.2 Signature-Based Detection Techniques 9
2.2 An Introduction to Genetic Algorithm ............................................. 20
  2.2.1 A brief History of Evolutionary Computation ..................... 21
  2.2.2 Elements of Genetic Algorithm ........................................... 23
  2.2.3 How Genetic Algorithm Works ........................................... 25
  2.2.4 Encoding Methods (Representation) ................................. 30
    2.2.4.1 Binary Encoding .................................................. 30
    2.2.4.2 Characters and Real Numbers Encoding ...................... 32
  2.2.5 Selection Methods ....................................................... 32
  2.2.6 Genetic Operators ....................................................... 37
    2.2.6.1 Crossover ......................................................... 37
    2.2.6.2 Mutation .......................................................... 39
  2.3 Malware Detection Using Genetic Algorithms ....................... 42
  2.4 Concluding Remarks ......................................................... 46

3 A Genetic-Algorithm-Based Solution for HTTP-Based Mal-
    ware Signature Generation .................................................. 49
  3.1 Overview ........................................................................... 49
  3.2 Network Traffic Analysis ................................................... 52
    3.2.1 HTTP Extractor ........................................................ 53
    3.2.2 Pcap Features Extractor .............................................. 55
  3.3 Malware Clustering and Signature Generation ...................... 57
    3.3.1 Clustering Malware Traces ........................................... 58
    3.3.2 Generating HTTP-based Signatures ............................... 59
List of Tables

4.1 HTTP request clustering and signature generating results . . . 76
4.2 Signature detection rate on current and future malicious HTTP
    requests . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
4.3 Signature false positive rate on legitimate HTTP requests . . . 80
4.4 Signatures average false positive rate and average detection
    rate of undetected malware traces vs. number of iterations . . 82
4.5 Evolved signatures detection rate on current and future mali-
    cious HTTP requests . . . . . . . . . . . . . . . . . . . . . . . 83
4.6 Detection rate improvement by the evolved signatures . . . . 85
4.7 Evolved signatures false positive rate on legitimate HTTP re-
    quests, and signature evolution time . . . . . . . . . . . . . . 85

List of Figures

2.1 A classification of malware detection techniques [33] . . . . . 9
2.2 Architecture of an Autograph Monitor [39] ................. 13
2.3 High-level overview of Honeycombs signature creation algo-
    rithm [43] .................................................................. 16
2.4 Overview of our HTTP-based behavioral malware clustering
    system [58] ................................................................. 18
2.5 Representation of schema $H = (\ast,\ast,\ast)$ .................. 26
2.6 The algorithm finds a fit plane (i.e. $H = (1,\ast,\ast)$) out of the
    initial hyperplane .................................................... 27
2.7 The algorithm approaches a better solution (i.e. $H = (1,\ast,1)$) 28
2.8 The algorithm reaches the optimum solution (i.e. $H = (1,\ast,1)$)
    after a number of generations .................................. 28
2.9 Phenotype to Genotype Representation .......................... 31
2.10 Roulette wheel selection example ............................... 34
2.11 One-point crossover operator ................................. 37
2.12 Three-point crossover operator ............................... 39
2.13 Arithmetic crossover operator ............................... 40
2.14 Random order changing mutation operator ............... 40
2.15 Permutation mutation operator .............................. 41
2.16 Reverse mutation operator ................................. 41
2.17 The classifier system learning and decision process with feed-
    back from the credit assignment and genetic algorithm [31] .. 45
2.18 Architecture of applying GA into intrusion detection [47] .. 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Overview of the proposed evolutionary HTTP-based signature generator</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Data flow in the Internet protocol suite model [7]</td>
<td>54</td>
</tr>
<tr>
<td>3.3</td>
<td>Structure of an HTTP request</td>
<td>55</td>
</tr>
<tr>
<td>3.4</td>
<td>Structure of an HTTP request line</td>
<td>57</td>
</tr>
<tr>
<td>3.5</td>
<td>Example of HTTP-based network signature</td>
<td>61</td>
</tr>
<tr>
<td>3.6</td>
<td>Example of string alignment for two strings to find the common subsequence</td>
<td>62</td>
</tr>
<tr>
<td>3.7</td>
<td>Example of signature generation process</td>
<td>65</td>
</tr>
<tr>
<td>3.8</td>
<td>Chromosome representation of a signature</td>
<td>67</td>
</tr>
<tr>
<td>3.9</td>
<td>Example of one-point crossover operation on two signature chromosomes</td>
<td>70</td>
</tr>
<tr>
<td>3.10</td>
<td>Flowchart of the proposed genetic algorithm</td>
<td>73</td>
</tr>
<tr>
<td>4.1</td>
<td>Signatures average false positive rate vs. number of iterations</td>
<td>82</td>
</tr>
<tr>
<td>4.2</td>
<td>Signatures average detection rate of undetected malware traces vs. number of iterations</td>
<td>83</td>
</tr>
<tr>
<td>4.3</td>
<td>The correlation between crossover and mutation ratios with average detection rate</td>
<td>86</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Currently, our lives significantly depend on services provided on the Internet. While this dependency is rapidly increasing, the number and possibility of Internet threads and online crimes grows as well. Therefore, protecting information systems and crucial online resources against network malware is one of the major challenges for organizations and enterprises which use or provide the online services. Among these network malware, HTTP-based malware are widely responsible for attacks on the internet. Nowadays, malware writers mostly use executable packing [30] and other code obfuscation techniques to generate a large number of polymorphic variants of the same malware. On the other hand, there are traditional signature-based anti-virus tools mainly based on a static analysis of the code of malicious software (a.k.a. malware), and malware signatures are usually represented by a fixed set of byte sequences in malware executable files [25].
The main drawbacks of these kinds of approaches are that AV companies have a hard time keeping their signature databases up to date, and their AV scanners often suffer from a high rate of false negatives [56][20]. Although it is not difficult to create many polymorphic variants of a given malware sample, executing different variants of the same malware will result in similar malicious activities. Based on this fact, there exist some anti-viruses which extract signatures out of groups of malware variants, taking similarities in their malicious behavior into account. It is easier to write generic behavioral signatures as opposed to traditional AV signatures that may be used to detect future malware variants with low false positives and false negatives [58].

Not only the majority legitimate traffic of a network is HTTP generated by user browsing and autonomous programs [66], but also the vast majority of today’s malware and botnets take advantage of HTTP-based network communications for committing their malicious activities or command-and-control (C&C) purposes [65]. For instance, according to [58], about 75% of malware samples having network activities produce HTTP traces. Also, [38] states that spam botnets mostly use HTTP to maintain the command and control (C&C) channel to their servers.

A network level analysis of HTTP requests is required in order to detect the presence of the current HTTP-based malware and predict the appearance of new variations of these malware in the future. In fact, we need an automatic signature generator engine that is able to analyze known malware samples
and automatically identify unknown ones. Obviously, generated signatures should offer a low false alarm ratio since it saves a lot of time on the network administrator side.

1.1 Summary of Contributions

The main purpose of this thesis is to develop a novel framework for offline generation and evolution of HTTP-based network signatures. The contributions of this thesis can be summarized as follows:

- An offline framework is proposed for extracting string-based signatures of HTTP-based malware traces based on their statistical and structural similarities. The proposed framework provides a means of analyzing network traffic and extracting request lines of malicious HTTP requests.

- We propose a novel network signature evolution algorithm based on the genetic algorithm. In fact, our framework measures the ability and effectiveness of an evolutionary algorithm in evolving network signatures in order to predict the structure of future malware traces.

- Experiments show that the proposed system enables accurate and efficient automatic generation and evolution of network-level malware signatures which are able to identify never-before-seen malware samples.
• We implemented a proof-of-concept version of our signature generation and evolution framework and performed experiments with 5 months worth of malware samples captured in the wild. The results of our experiments shows that our framework not only provides a relatively high detection rate of unknown malware traces, but also guarantees a significantly low false alarm rate.

1.2 Thesis Organization

The rest of thesis is organized as follows:

In Chapter 2, some of the related works in malware detection are reviewed. Firstly, different proposed techniques in the literature are briefly explained, and their advantages and limitations are discussed. Then, we briefly describe how the genetic algorithms work, and define different elements, operators, and functions of the genetic algorithms. Finally, some of the few related works which employ evolutionary strategies, specifically genetic algorithms, to improve malware detection abilities are reviewed.

Chapter 3 provides details of the proposed framework for HTTP-based signature generation and evolution. First, the Network Traffic Analysis component is described. Secondly, we present the malware clustering and signature generating algorithms. Next, development of a new evolutionary algorithm for enhancing generated network signatures is presented.
Chapter 4 describes implementation details of the framework and experiments we conducted for evaluating the proposed system. These experiments include measuring the detection rate of matching generated signatures over the current and future malware traces. The normal traffic of the ISCX dataset is also used in order to measure the false positive rates.

Finally, the conclusions and some suggestions for future work are provided in Chapter 5.
Chapter 2

Related Works

2.1 Malware Signature Generation

2.1.1 Introduction

Malware or malicious software refers to a wide class of software that are used to provide unauthorized accesses, interrupt computer’s intended functions, delete, insert, modify, or steal data. These types of meddling in system’s expected operations may result in consequences ranging from system damages, exploits, data lost, or sensitive information theft to human death, all under the radar. In order to perform malicious activities, malware either take advantage of a certain number of legitimate system privileges, or exploit application or system vulnerabilities. The second type of malware are probably more complex ones since authors have to first find and exploit system’s security holes. Therefore, malware authors are computer security
experts since they should have a broad knowledge over computer systems and networks to be able to detect weak points. Thus, that is why computer malware grow as computer technologies grow.

The rise in the use of network malware has become the most serious threat to Internet security [41]. In order to disable the escalation of malware threats, the Internet security community has developed a large number of malware detection strategies and systems, in the recent years. We can divide different malware detection techniques into 3 major categories including Anomaly-based Detection, Specification-based Detection, and Signature-based Detection techniques [33]. [33] examines 45 different malware detection techniques and provides a classification of these techniques. As you see in Figure 2.1, this work depicts the relationship between different types of these techniques. Further, they also divide each category into static, dynamic, or hybrid, based on how the technique gathers information to detect malware.

The most prevalent approaches applied for detecting malware are pattern-based signature generating methods. Security researchers and anti-virus firms show interest in these sorts of approaches since speed and simplicity are two important features of such techniques. Despite the fact that signature-based approaches look promising for detecting already known malware, they suffer from lack of performance to detect never-before-seen samples [58].
In the following section, we describe related research works that employ signature-based scanning methods to identify malware. Then, we briefly review the genetic algorithm and describe its steps and operators. Finally, in Section 2.3, we discuss the applications of genetic algorithms in intrusion and malware detection systems.

2.1.2 Signature-Based Detection Techniques

Traditionally, security experts used to create malware signatures manually. Not only the manual methods lack enough precision but also they are significantly slow. Today that IT community is facing the tremendous impact of malicious network activities, generating high quality signatures is a major challenge for network security related firms. To come up with a

![Figure 2.1: A classification of malware detection techniques [33]](image-url)

---

9
solution, researchers have proposed many different automatic signature generation frameworks and methods.

Modern anti-spyware and anti-virus tools offer a variety of methods to identify malware, including heuristic-based detection [11], behavioural detection [35], and signature-based scanning [2]. Signature-based techniques create a pattern based on the malware’s activities in order to identify the malware traces later. Anti-virus software which use this approach create a database of these malware patterns as a history of detected malware. These behavioural patterns representing malicious activities of malware are called signatures.

Ilgun et al. in [34] represent attacks as state transition diagrams. They monitor and gather data generated while an attack is happening. After a preprocessing phase to convert the data to a well-formed structure, they analyze this data with the state transition diagrams. These diagrams are employed to determine what conditions and events are required for a penetration to take place. Thus, in this work, the sequence of consecutive events triggered by an already known malware in a system is considered as a signature to identify same or similar malicious activities in the future.

In this work, they implement a rule based expert system for detecting the penetrations, called the state transition analysis tool (STAT). Authors believe that one of the advantages of their system over typical rule based de-
tectors is its capability to detect variations of the same malware. In addition, STAT can also identify cooperated penetrations since it keeps a list of users who have triggered events it monitors. They also mention their system’s weakness in handling legitimate access by attackers using valid credentials. These credentials could be stolen from authorized system users. Their system is only capable of capturing and tracing events that cause visible state changes to the system.

Singh et al, propose a content sifting algorithm for automatic extraction of unique content signatures of unknown malware [68]. They state that their proposed system, called Earlybird, analyzes packets’ payloads with an acceptable memory and processing power requirement since a prototype of Earlybird has been able to process around 200Mbps of live traffic. Therefore, the authors are proposing a real-time malware detection engine. In this work, specific sequences of strings in the network traffic content is the signature they try to generate.

Since they look into network traffic content to identify invariant portions of worm payload, Earlybird is not able to identify polymorphic malware because they encrypt their payload in each generation. Moreover, metamorphic malware have the capability to mutate operational orders but keep the result of its overall malicious activity the same. Although Earlybird has been able to identify and extract signatures for all known worms at the time of this research, malware designers are still able to get around its monitoring system
using common IDS evasion techniques [60].

Autograph [39] is a similar automated malware signature generation system. As you see in Figure 2.2, it consists of two main components. The first one is responsible for identifying, picking and passing the suspicious flows to second component which is the signature generator engine. After shrinking the volume of incoming network traffic, Autograph ranks content according to its prevalence but, in order to decrease the number of generated signatures, it only generates signatures for pools of malicious flows not for every single suspicious content. The authors have tested Autograph on offline traffic of real DMZ\(^1\) traces. They claim that Autograph is able to generate accurate and unique signature sets with a high detection rate and significantly low false positives. Using a port-scanning flow classifier to identify suspicious flows, Autograph can detect newly released worm signatures before infection of a large number of vulnerable nodes.

Although they achieved acceptable results on their initial evaluations of offline traces, there are two major limitations in this work. The first limitation is that currently Autograph takes offline traffic as input data while performance measures would drop in an online setting. Secondly, since they target specific ports and protocols in this system, the malware traces in their testbed are not as diverse as traces in the wild. This limitation can definitely

\(^1\)“In computer security, a DMZ or Demilitarized Zone is a physical or logical subnetwork that contains and exposes an organization’s external-facing services to a larger and untrusted network, usually the Internet” [3].
affect true positive and false positive rates if they feed the system with actual network traffic. However, in order to cover both limitations, they are currently evaluating a single Autograph monitor with an online feed containing more diverse traces and protocol (port) workloads.

While Autograph extracts specific signatures out of pools of malicious flows, [24] proposes a behavioural approach to categorize malicious behaviour signatures into three different categories. By monitoring incoming and outgoing network traffic on a single host, their system is able to detect four major signatures. One of these characteristic patterns of malware behaviour in network traffic is sending similar data from one machine to the next that is called alpha-in and alpha-out. This signature is based on a simple idea that states worms usually send out an invariant portion of the data flow they receive in order to spread to other nodes over the network. However, there are legitimate applications that represent such behaviour, for example file

![Figure 2.2: Architecture of an Autograph Monitor [39]](image-url)
sharing services receive and send similar contents that are files. Therefore, this case can result in false alarms in this category.

Receiving and sending resembling content by worms is not always the case. Therefore, to partially cover the remaining types of malware behaviours, the authors propose another sort of behavioural signature called *fanout*. It considers tree-like propagation and the reconnaissance of worms as a fingerprint. This behaviour is observed once a worm tries to propagate itself to another node by using a chain or tree of infected nodes as a propagation bridge. Thus, *fanout* inspects infected hosts and the hierarchy of infections to find out which node has infected the current node. Then this signature limits the number of descendants of a node in the hierarchy tree. This threshold is based on four tree metrics including the number of descendants in the tree, the depth of the tree, the required time-window to access a specific level of the tree, and the average branching factor. This approach is limited in that it needs overall network monitoring and processing that requires a lot of resources and processing time as well.

Moreover, they consider changing a server into a client as the last form of behavioural signature. When an infected host connects to a new target, the target node behaves like a server. Once the target is compromised, it connects to other nodes to propagate itself by exploiting the same vulnerability. Therefore, now it is acting as a client. We also observe such behaviour in peer-to-peer communications. Despite the fact that many modern worms employ
sophisticated peer-to-peer command-and-control (C&C) mechanisms based on protocols such as Kademlia [21], this signature also covers a large number of benign peer-to-peer communications. Therefore, the false positive rate can drastically increase because using peer-to-peer protocols is widespread, due to popular file sharing and communication protocols, like Bittorrent and Skype [10][17].

[43] presents Honeycomb that is an automatic NIDS signature generator system. This system applies a pattern-matching method to network traffic captured by a honeypot system. It applies the longest common substring (LCS) algorithm to match similarities in the invariant portions of packet payloads. Figure 2.3 depicts a high-level overview of the Honeycombs signature creation algorithm. The authors mention that the system is capable of producing high quality signatures without any specific knowledge hardcoded into the system. However, a drawback of this system is that generated signatures may fail to identify polymorphic malware payloads since signatures are all continuous strings, whereas polymorphic malware are able to mutate their content while keeping the same malicious functionality.

To overcome the problem of polymorphic malware mutations, Newsome et al. developed Polygraph [55]. Instead of looking for single substring signatures in payloads, they investigate different string signature classes used to identify polymorphic worms. The three signature classes they propose include: (1) Conjunction signatures: “A signature that consists of a set of
tokens, and matches a payload if all tokens in the set are found in it, in any order.”. (2) Token-subsequence signatures: “A signature that consists of an ordered set of tokens.”, and Bayes signatures: “A signature that consists of a set of tokens, each of which is associated with a score, and an overall threshold”.

Despite the common belief in the network security community that content-based signatures are not practical anymore because of the appearance of polymorphic malware, Polygraph is proposed to show that applying new signature extraction approaches can still help content-based signatures to provide more robust and promising results. The other contribution of this work is auto-

Figure 2.3: High-level overview of Honeycombs signature creation algorithm [43]
matic signature generation for polymorphic malware. Automatic signature
generation with low false positive rate is considered as a valuable contribu-
tion in the field of network security since it not only decreases the need for
human expertise to inspect malware traces and extract signatures but also
significantly increases the velocity of mitigating new malware variations [53].
There are two major limitations in this work. First is that its clustering
algorithm is greedy and computational cost increases by an increase in the
volume of malicious traces generated by different types of malware. The
second issue is with extracting signature for malware which use payload en-
cryption in different network levels and leave very small portions of packets
unencrypted. In this case Polygraph is not able to extract unique signatures
out of the small unencrypted content. However, they mention it is able to
identify specific malware traces like the Slapper worm [1] during the initial
TCP handshake phase, meaning that it is not encrypted yet.

Perdisci et al. propose a network-level behavioural malware clustering and
signature generating method that targets HTTP-based botnet C&C detect-
ton [58]. Their system includes two clustering components: coarse-grained,
and fine-grained clustering (See Figure 2.4). During the coarse-grained clus-
tering process, they cluster malware samples based on a few statistical fea-
tures. This step roughly puts resembling malware samples into the same
buckets. Since finding structural similarities between all pairs of HTTP re-
quests is computationally expensive, they apply the fine-grained clustering
method to HTTP traces of malware samples in each cluster. In this phase they use a single-linkage hierarchical clustering algorithm [36] and define a distance function based on HTTP requests structural similarities. In order to maintain simplicity and lower computational cost, they only look into the request line, which consists of a request method and a request path [45], of each HTTP request to extract its structure. After obtaining fine-grained clusters, a cluster merging method is used to combine close small clusters to create more generic clusters. Finally, they employ a Token-Subsequences algorithm implemented in Polygraph [55] to extract pattern-based signatures out of each malicious HTTP request pool.

![Figure 2.4: Overview of our HTTP-based behavioral malware clustering system [58]](image)

The main objective of this approach is extracting more generic and high quality signatures by only inspecting malicious HTTP requests. Hence, it first inspects the bottom levels of the clustering hierarchy to obtain more specific and distinguishing signatures. Subsequently, cluster merging and signature pruning methods are employed in order to maintain signature gen-
erality whereas the aim is to summarize the behaviour of malware families, instead of individual malware samples. Perdisci et al are interested in these sort of generic signatures since they want each of the generated signature sets to be able to generalize to new, never-before-seen malware variations. To represent this capability, given malware sample traces of a month, they generate corresponding HTTP-based signatures then match them over HTTP traffic traces generated by malware samples in the following months. Choosing this approach is due to the fact that different variations of the malware detected during the current month will appear in the following months.

They evaluated their system on a malware dataset consisting of malware samples they have collected in a period of six consecutive months. The evaluation results shows that the signature set generated for one month not only identifies malware traces in the current month’s dataset but also matches with a considerable number of the malicious HTTP traffic traces generated in the following months that are considered as never-before-seen traces. At the same time, they report a significantly low false positive rate. Generating less false positive prone signatures is an advantage of this work over similar automated signature generator methods. Maintaining a low false positive rate is a concern because the more generic signature we generate, the higher the possibility of overlap with normal traffic.

The main limitation of this work is dealing with encrypted HTTP queries i.e. HTTPS requests. However, employing HTTPS protocol by malware is
not prevalent whereas many enterprise networks and anti-spyware systems only accept HTTPS traffic from certified and trusted servers [73][72]. The other drawback of this approach is that it depends on analyzing interesting actions made by malware in a limited time window, for example a 5 minutes window. Interesting actions here mean any HTTP-based communication of the captured malware with other hosts. This approach is limited in that case if malicious activities of a malware do not get triggered immediately after running the malware sample and not even in the specified time window. This characteristic is very prevalent among botnet malware. Botnet slaves usually spend most of their running time in a dormant mode [54] and wait for the master bot to send off commands. Hence, this approach may only try to capture HTTP traces in the malware’s silence period.

2.2 An Introduction to Genetic Algorithm

The evolutionary algorithm is a class of optimization methods that simulate the process of natural evolution. Evolutionary computing comprises genetic algorithms, genetic programming, evolutionary programming, evolutionary strategy, and classifier systems [12]. It is also a member of a group of methods known as meta-heuristics. This set of techniques includes simulated annealing, tabu search, the ant colony algorithm, the bee algorithm, particle swarm optimization, artificial immune systems, and distributed reinforcement learning. Genetic Algorithms were proposed by John Holland in
the 1960s [32] and were developed by Holland, his students, and colleagues at the University of Michigan in the 1960s and the 1970s [28].

2.2.1 A brief History of Evolutionary Computation

In over 50 decades, many researchers have examined evolutionary systems with the idea that they can be used as optimization tools for Engineering problems. To date, there has been considerable progress in providing evolutionary solutions for problems such as the Travelling Salesman problem, integer programming problems, and multi-objective optimization problems. [44][19][48][22]. All of these systems are inspired based on evolving a population of candidate solutions for a problem to find the optimum ones.

In 1960s, Reckenberg proposed evolutionary strategies and used it to optimize real parameters in designing airplanes’ aerodynamic components. This idea was later extended by Schoeffel. Evolutionary strategies was an active field of research that developed independently from genetic algorithms. However, today, we can hardly differentiate between these two fields. Hence, they are recognized all together under the title of evolutionary computation.

Fogel, Walsh, and Owens expanded evolutionary programming in which candidate solutions for a specific problem are represented as a finite-state machine [26][27]. Many other researchers since the 1950s extended evolutionary algorithms for optimization problems and machine learning improvements
Unlike evolutionary strategies and evolutionary programming, Holland’s main goal was not designing algorithms for solving specific problems. Actually, he studied the adaptability phenomena the way it happens in nature, and proposed a solution to transfer natural adaptation mechanisms to the machinery computing domain. Holland’s Genetic Algorithm is an approach in which a population of chromosomes - that are strings of zeros and ones in Holland’s approach - generate a new population using the concept of natural selection and operators adopted from genetic science.

Each chromosome consists of a certain number of genes each representing an specific feature. Genetic operators include crossover, mutation, and inversion. The selection function chooses chromosomes that are allowed for reproduction from the current population. Naturally elite chromosomes produce more offsprings compared to unfit ones. The crossover operator exchanges two or more subsections of parent chromosomes whereas the mutation operator applies some sort of random modifications in a chromosome. Somehow similar to mutation, the inversion operator inverses the order of certain sections of the chromosome.

Introducing an algorithm based on population production using crossover, mutation, and inversion is Holland’s main innovation. It should be mentioned that the primitive evolutionary strategies proposed by Reckenberg consisted
of a twosome population including parent and child produced by parents mutation. Holland was the first person who tried to propose a framework for evolutionary computations. To date, this theoretic foundation still forms the basis of other theoretical researches done based on genetic algorithms.

In recent years, many interactions have happened between researchers studying various evolutionary approaches and the distinguishing borders between genetic algorithms, evolutionary strategy, and evolutionary programming have disappeared. Instead, these algorithms have been altered to a unified concept. Researchers often use the term “genetic algorithm” to address concepts much further than Hollands primitive idea.

2.2.2 Elements of Genetic Algorithm

As we discussed previously, there is no precise and comprehensive definition for genetic algorithm to distinguish a genetic algorithm from other evolutionary computation approaches that has been accepted by the community of evolutionary computation. Nevertheless, all of the approaches called genetic algorithms are at least common in the following elements:

1. A population of chromosomes
2. Selection based on fitness
3. Crossover in order to produce new offspring
4. Random mutation of new offspring (Nowadays, inversion operator - fourth operator of Holland’s genetic algorithm - is less used and its advantages, if any exist, are not well recognized yet.

In genetic algorithms, chromosomes can be in the form of a string of features (genes). Each locus in a chromosome can be assigned to 0 or 1 (albeit, later we discuss that there are other representation methods available). Each chromosome is considered as a point in the search space of candidate solutions. The genetic algorithm processes a population of chromosomes and eventually replaces this population with a new one. The genetic algorithm requires a fitness function as a metric to compare pairs of chromosomes. Each chromosome’s fitness depends on the way it solves the problem.

There exist various search approaches in the field of Artificial Intelligence. Usually, general approaches that can be employed in a wide range of different problems are considered as *weak* approaches. On the other hand, *strong* approaches are designed for specific problems. All of these methods such as Hill Climbing [51], Simulated Annealing [42], and other general search methods perform based on following steps:

1. Generate a collection of initial solutions

2. Evaluate these solutions using a fitness metric

3. Decide which solutions to keep and which ones to drop
4. Apply further modifications to the surviving candidates using operators

In the following section we specifically discuss how genetic algorithms work.

2.2.3 How Genetic Algorithm Works

Although genetic algorithms can be simply described and converted to a computer program (see Algorithm 1), their behaviour is complicated and there are many questions to discuss regarding their functionality and the problems they can solve. Much research work has been done on the theoretic aspect of genetic algorithms [15][70][62].

Algorithm 1 Genetic Algorithm

initialize population
evaluate population
while termination-criteria is not satisfied do
 select parents for reproduction
 perform recombination and mutation
 evaluate
end while

In the traditional theoretical genetic algorithm first proposed by Holland, the assumption is that genetic algorithms identify, emphasize, and produce proper building blocks of good solutions. In other words, good solutions are constructed from good building blocks.

Holland proposed Schemas theory in order to formulate the idea of building blocks. A schema is a collection of bit strings that is represented in a template
consisting of 0s, 1s, and asterisks for those that have non-determined state.

For example, \( H = (1****1) \) represents all 6-bit strings that start and end with 1. \( H \) represents schemas. Schemas demonstrate hyperplanes\(^2\) with different dimensions. Assuming that we have a chromosome with a length of 3, then at the beginning of the algorithm \( H = (***) \) represents a schema that demonstrates a hyperplane. We see the geometric representation of this example in Figure 2.5. In fact, \( H \) can cover all of the possible combinations of 0s and 1s that construct this hyperplane.

![Figure 2.5: Representation of schema \( H = (***) \)](image)

In the genetic algorithm, gradually during generations, schemas having better average fitness are selected. Thus, asterisks gradually convert to zero.

\(^2\)“In geometry, as a plane has one less dimension than space, a hyperplane is a subspace of one dimension less than its ambient space” [6]
or one and the algorithm approach an optimum point. Assume that in the previous example, after one generation pass, the schema $H = (1, *, *)$ has had better average fitness (this average is obtained by combining fitness of all of the possible states). Figure 2.6 depicts all of the possible combinations of this schema that construct the $ABCD$ plate.

![Figure 2.6](image)

Figure 2.6: The algorithm finds a fit plane (i.e. $H = (1, *, *)$) out of the initial hyperplane

Continuing this evolution process, the algorithm improves the fit plane it has found. Assume average fitness of schema $H = (1, 0, 1)$ is higher than others therefore we consider the line $AD$ in the plane $ABCD$ as a better solution (See Figure 2.7).

Eventually, when the algorithm reaches the optimum solution, the schema is completed and all of the asterisks convert to 0s or 1s. Considering the
above example, the algorithm reaches a solution such as $H = (1,1,1)$ that represents point $A$ in Figure 2.8.

Figure 2.7: The algorithm approaches a better solution (i.e. $H = (1,*,1)$)

Figure 2.8: The algorithm reaches the optimum solution (i.e. $H = (1,*,1)$) after a number of generations
This is an abstract example that visually describes how genetic algorithms work in optimization problems based on the building blocks hypothesis. In fact genetic algorithms find proper building blocks that are sub-strings of schemas. It is worth mentioning that every possible subset of a bit string of length of L cannot represent a schema. Actually, many of these subsets cannot be a schema. There are $2^L$ possible bit strings of length L. Therefore, there exist $2^{2^L}$ possible subsets of these strings, but obviously there are only $3^L$ possible schemas. The main idea of traditional genetic algorithms is that schemas are implicitly the building blocks that genetic algorithms effectively process using selection, crossover, and mutation operators.

Some of the advantages of a genetic algorithm are as follows:

- the concept is easy to understand
- modular, modules are adaptable for different applications
- supports multi-objective optimization
- easy to exploit previous or alternate solutions
- there is always an answer
- answer gets better with time
- ability to scan a vast solution set quickly
- bad initial population (e.g. not so good signatures in our work) do not influence the end solution negatively
is useful and efficient when the search space is large, complex, or poorly understood

A genetic algorithm can be one of the search candidates when someone wishes to find a solution in a huge multi-dimensional search space.

2.2.4 Encoding Methods (Representation)

One of the most effective factors in the performance of genetic algorithms is the representation of the problem. Meaning that the problem’s characteristics have to be comprehensibly defined to the algorithm.

2.2.4.1 Binary Encoding

Binary encoding (i.e. bit strings) is the most prevalent encoding method and there are several reasons for this preference. The first reason is a historic one since Holland and his students used this method for the first time. The genetic algorithm theory is based on the assumption that the length and order of binary codes is fixed.

This theory can be extended to be used by non-binary encoding methods. Holland justifies using binary codes despite the fact that binary coding is not natural and not practical for many problems such as weight adjustment in artificial neural networks [52][46].
Using successive dividing in order to convert real numbers to binary bits causes poor performance of the operators. There are several methods to map a real number to a binary number. The most prevalent approach is the following.

If we want to represent a given real number \( z \in [x, y] \subseteq \mathbb{R} \) as a binary number \( \{a_1, a_2, ..., a_L\} \in \{0, 1\}^L \), we need the following one-to-one function (see Figure 2.9):

\[
\Gamma : \{0, 1\}^L \rightarrow [x, y]
\]

\[
\Gamma(a_1, a_2, ..., a_L) = x + \frac{y-x}{2^L-1} \left( \sum_{j=0}^{L-1} a_{L-j} \cdot 2^j \right) \in [x, y]
\]

where \( x \) is the smallest real number in the range and \( y \) is the biggest one. Each \( a_i \) is one bit of the corresponding binary representation of length \( L \). \( a_1 \) is the most significant bit and \( a_L \) is the least significant bit.

Figure 2.9: Phenotype to Genotype Representation
2.2.4.2 Characters and Real Numbers Encoding

In many of the genetic algorithm applications, it is more natural to use characters or real numbers to create chromosomes. For example, the Traveling Salesman problem or any other graph related problem can be represented by labelling the graph’s edges by characters. Moreover, real numbers can be used to represent weights in Artificial Neural Networks. Albeit, problem representation is not an easy job indeed and often it is as difficult as solving the problem itself.

Using real number encoding, genetic algorithms cannot perform effectively to solve problems including continuous optimizations whereas the Hamming distance\(^3\) between binary representations of two integers is too large. This problem is called the Hamming Cliff. Although Gray coding\(^4\) is a solution to get around Hamming cliff problem, in a large number of applications, researchers prefer to use real numbers for problem representation since it is conceptually closer to problem’s nature.

2.2.5 Selection Methods

After finding a proper approach to representing a problem, we have to decide how the selection process should be done. Selection refers to the pro-

\(^3\)“In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.”[5]

\(^4\)“Gray code is a binary numeral system where two successive values differ in only one bit.”[4]
cess that chooses individuals in a population for reproductions, and specifies how many offspring they should produce. In fact, the purpose of selection is to emphasize elite individuals in a population hoping that their offspring are better fitted. Similar to encoding methods, there are several selection approaches as well. However, saying which method is more suitable depends on other parameters and the structure of the problem.

Selection Based on Roulette Wheel and Random Sampling

In Holland’s genetic algorithm a selection method related to fitness is employed in which the selection probability of an individual is the individual’s fitness divided by the population’s average fitness. The most prevalent approach in implementing this selection method is using Roulette Wheel. This approach assigns a portion of a roulette wheel to each individual proportional to its fitness score. The wheel spins \( N \) times (\( N \) is the number of individuals in a generation) and after each spin one individual is selected as a parent for reproduction.

For instance, assume that we have 3 chromosomes A, B, and C in the population. Using the roulette wheel selection approach, a sector of the roulette wheel is assigned to each individual proportional to its fitness (see Figure 2.10). In order to calculate the probability of the selection of an individual, we have to divide its fitness score by summing all of the individual fitness scores. In the following example, A’s probability to be selected is 50%,
and for B and C, 17% and 33%, respectively.

Figure 2.10: Roulette wheel selection example

From the statistical point of view, this randomized selection method results in a reproduction probability proportional to the fitness value for each individual. However, since we usually have a relatively small population for a genetic algorithm, the actual number of offspring of an individual is usually different from its probability. For example, in the worst case scenario, the weakest individual gets selected after each roulette wheel spin. In order to reduce this difference, James Baker proposes a similar approach called Stochastic Universal Sampling (SUS) [13]. Instead of N times spinning of the roulette wheel in order to select N parents, SUS uses N equally spaced pointers to select N parents. Albeit, this method does not resolve the main problem with fitness proportionate selection. Typically, at the beginning of a search process, the fitness variance of the population is high and there are a small number of individuals that have higher fitness score than others. With fitness proportionate selection, fit individuals and their offsprings rapidly grow in the population. This prevents the genetic algorithm from
performing a more exhaustive search. This phenomenon is called *premature convergence* meaning that fitness proportionate selection tends to get faster results rather than searching the search space for better solutions.

**Rank Selection**

Rank selection is a repetitive method proposed by Baker [14] in order to prevent the premature convergence problem. This approach ranks individuals of a population based on their fitness. The probability of selecting an individual depends on its rank among others instead of its absolute fitness score. Therefore, fitness value does not directly affect the selection process. This helps prevent premature convergence since using the absolute fitness score as a selection criteria may force the algorithm to converge to a local optimum while it hasn’t had a chance to search many untouched areas of the search space yet. Sometimes we need to know that how much fitter an individual is compared to its closest competitor. In rank selection, the algorithm chooses individuals with a higher degree of freedom when fitness variance is higher in the population. On the other hand, selection become more restricted once the population’s fitness variance decreases.

Baker’s linear method for ranking is as follows:
Individuals in the population get ranked ascending from 1 to N. The expected value of \( i^{th} \) individual at time \( t \) is calculated as following:

\[
\text{ExpVal}(i, t) = \text{Min} + (\text{Max} - \text{Min}) \cdot \frac{\text{rank}(i, t) - 1}{N - 1}
\]

\( \text{Min} \) is the expected value of the first rank individual and obviously \( \text{Max} \) is the expected value of \( N^{th} \) individual.

**Tournament Selection**

Many fitness proportionate selection methods require two steps for each generation. The first step is to calculate the average fitness and the second step is to find the expected value for each individual. In addition to these steps, rank selection needs a ranking step that is often time consuming. Tournament selection is similar to rank selection in terms of individual choosing method, but it is computationally more efficient. Tournament selection algorithm randomly (or even based on fitness) selects \( q \) chromosomes from the population. Then, in each iteration, it selects the fittest chromosome out of these \( q \) chromosomes for reproduction. It runs this procedure \( N \) times to select \( N \) parents.

**Competition Selection**

This method is implemented in many forms. The most prevalent one randomly selects two chromosomes after calculating fitness and expected values.
Then, it generates a random number between zero and one. We pick the fitter chromosome if this number is less than an arbitrary constant (usually 0.75), and the weaker chromosome otherwise.

### 2.2.6 Genetic Operators

After choosing representation and selection methods, the third concern is how to use genetic operators. This decision mostly depends on the encoding strategy. Here we describe common genetic operators.

#### 2.2.6.1 Crossover

As in Figure 2.11, the simplest form of crossover is one-point crossover. This method randomly specifies one point in each of the parent chromosomes and exchanges substrings between parent chromosomes. In fact, two building blocks of a schema are combined together. However, this type of crossover is not able to combine all of the possible schemas.

![Figure 2.11: One-point crossover operator](image)

Figure 2.11: One-point crossover operator
In other crossover operator techniques the number of crossover points are different or even this number is selected randomly. The other important item to specify is crossover rate that is the probability of applying crossover for each selected point. Experimentally, this rate is usually between 0.5 and 0.8. Determining crossover operator depends on the problem’s characteristics and other of the algorithm’s parameters. All in all, the employed crossover operator has to be commensurate with the encoding method since some encodings require their own specific crossover methods. For example, in the travelling salesman problem, a valid sequence of cities should be maintained after applying crossover.

Here we describe different variants of crossover operator:

- **One-Point Crossover** chooses a single point in length of the parents chromosomes. It divides each chromosome into two sections then replaces these sections between chromosomes. (See Figure 2.11)

- **Two-point crossover** randomly selects two point in each chromosome and divides chromosomes into 3 sections then replaces the middle sections.

- **n-point crossover** is in fact the generalized form of the previous methods. If we consider a chromosome as an array, we can select n points (obviously n has to be smaller than the length of the array) and then produce children by sequencing different sections of parents’ chromo-
Figure 2.12 depicts a three-point crossover operator.

![Figure 2.12: Three-point crossover operator](image)

- **Uniform crossover** first randomly specifies the number of point at which crossover should take place. Positions of these points are randomly chosen. Then, it exchanges pointed genes between two parent chromosomes to produce children.

- **Arithmetic crossover** is suitable for real number representations. If \( X \) and \( Y \) are parents chromosomes then \( Z = \alpha X + (1 - \alpha)Y, \ 0 \leq \alpha \leq 1 \) and \( Z = (1 - \alpha)X + \alpha Y, \ 0 \leq \alpha \leq 1 \) are children chromosomes. For example if \( \alpha = 0.4 \), two parent chromosomes in Figure 2.13 transform to children ones as you see in this figure.

### 2.2.6.2 Mutation

In the primitive theories of evolutionary strategy, random mutation is the only cause of change in individuals. As we previously mentioned, in genetic
algorithms, mutation is used to maintain genetic diversity in a population of chromosomes. It randomly alters a number of gene values in a chromosome. In fact, it gives the algorithm an opportunity to jump onto untouched areas in the search space. Mutation takes place during the reproduction process according to a user-defined mutation rate. A higher mutation rate increases the variance in each generation and prevents the algorithm from converging. Usually, this rate should be low but its certain value depends on the problem’s characteristics and the representation method.

Here we describe different types of mutation operator:

- **Adjacent order changing mutation** randomly select two adjacent genes in the chromosome and then switches their position.

- **Random order changing mutation**, as in Figure 2.14, picks two genes in the chromosome and then replaces them.

![Arithmetic crossover operator](image1)

![Random order changing mutation operator](image2)
• **Permutation mutation** randomly selects a sub-sequence of the chromosome and then randomly permute genes in the sub-sequence. Figure 2.15 demonstrate this operator.

```
1 2 3 4 5 6 7 8  \rightarrow 1 2 6 5 3 4 7 8
```

Figure 2.15: Permutation mutation operator

• **Reverse mutation**, similar to permutation mutation, randomly selects a sub-sequence of the chromosome but it reverses the order of this sub-sequence of genes. In fact, it applies a reverser operator on a sub-sequent of a chromosome. (see Figure 2.16)

```
1 2 3 4 5 6 7 8  \rightarrow 1 2 6 5 4 3 7 8
```

Figure 2.16: Reverse mutation operator

These are all common genetic mutation techniques used in genetic algorithms. However, there are many other mutation methods proposed by researchers that are designed to solve specific problems.
2.3 Malware Detection Using Genetic Algorithms

As we discussed previously in Section 2.1, a large number of malware detector systems are proposed to identify a wide variety of worms, viruses, and spyware. The most prevalent and simplest methods are signature-based ones. Signature-based detection techniques need human expertise to investigate traces of captured malware and monitor their behaviour in order to create new signatures and updates signatures database. The signature-based detector then matches every single network trace with the stored signatures to flag any similar pattern. Although this approach performs promising in detecting already recognized malware, it limits in identifying new malware in the early stage of their life.

Using machine learning techniques such as neural networks [71], data mining [63], and rule learning [64] are considered as a remedy for this shortfall. Evolutionary algorithms are also employed in rule-based and signature-based systems as either the core algorithm of a detector system or an auxiliary optimizer to enhance the main algorithm.

Kim et al [40] propose a detection mechanism focusing on script malware. They address the malware polymorphism problem and use a dependency graph analysis approach to represent each script malware. Then, they transform polymorphic malware detection to the problem of finding maximum
subgraph isomorphism. “Since subgraph isomorphism is an NP-hard problem, a GA is appropriate”. Thus, they use genetic algorithm as an axillary component for subgraph isomorphism to improve detection accuracy and decrease the computational cost. Although genetic algorithm reduces the cost of solving the NP-hard problem, it still needs a relatively large amount of processing time compared to traditional signature-based detectors.

The network intrusion detection approach proposed by Shon et al [67] is an example of detection systems that use genetic algorithm as an optimizer. They first use a genetic algorithm to choose proper fields of traffic packets for analysis. Using selected fields, they use the Support Vector Machine (SVM) to classify incoming network traces. Their experiment shows that SVM provides excellent performance using a genetic algorithm for feature selection.

Similarly, [69] employs a GA-based feature selection method in order to provide a decision tree classifier with a set of proper features. They used the KDDCUP 99 data set [8] to train and test the decision tree classifiers. This optimization in feature selection phase results in an increase in detection rate and a decrease in false positive rate. Choosing a small subset of more effective features rather than using all available features helps decision trees performing faster and more efficient.
Unlike [69], Xia et al [75] employ a genetic algorithm as the main classifier to classify network behaviours into normal and abnormal behaviours. They use information theory to select a minimal subset of the most important features in order to reduce the size of inbound traffic. They select appropriate features for each single type of attack to obtain higher detection rate on detecting specific malware. They applied their approach to KDDCUP 99 dataset and reported detection rate of 99.25% and false positive rate of 1.66%.

The proposed detection system in [31] consists of three main components: A classifier system that generates rules sensitive to input messages, a credit assignment algorithm that evaluates the strength of each generated rule, and a genetic algorithm that modifies the rule set (see Figure 2.17). The genetic algorithm component removes weak rules that are the ones resulting in more incorrect predictions which cause a higher false positive rate. Instead, it reproduces and adds new rules by mating strong rules. In fact, the credit assignment algorithm plays the role of a fitness function for the genetic algorithm.

In order to improve the effectiveness of IDSs, [59] proposes a Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis (GASSATA) tool. GASSATA first collects system logs from audited hosts then extracts attacks’ traces and the sequence of events associated with each attack. Then, it employs a genetic algorithm to search for combinations of known attacks by monitoring the events stream. This technique doesn’t give the reason for
a detected attack.

Sanaiee et al [61] propose PAGELS, a parallel genetic local search algorithm, in order to generate fuzzy rules to identify malicious behaviours. In this system, fuzzy rules of the form “if condition then prediction” represent individuals in the population. In order to boost the genetic algorithm’s performance PAGELS divides the global population into several smaller populations; then each subpopulation is evolved independently using a genetic local search procedure. The authors report a higher detection rate, lower false alarm rate, and shorter training time as the advantages of their parallel learning framework compared to several classification algorithms.

Figure 2.17: The classifier system learning and decision process with feedback from the credit assignment and genetic algorithm [31]
The system proposed in [47] evolves simple rules for network traffic of the form of “if condition then act”. Here condition refers to match a between suspicious network traffic features. The act field refers to the proper action the system should take once the condition is satisfied. This system applies a genetic algorithm to a rule set of this kind in order to generate novel rules that only match anomalous traces. Figure 2.18 depicts the architecture of this system. The author concludes that this implementation considers both temporal and spatial features of network connections for creating IDS rules that helps detect complex suspicious network behaviours.

Figure 2.18: Architecture of applying GA into intrusion detection [47]

2.4 Concluding Remarks

Network malware is defined as a malicious software or piece of code that propagates through computer networks and performs unauthorized activities on the infected nodes. In recent years, the Internet security community has developed a large number of malware detection strategies and systems in order to disable the escalation of malware threats. Among different malware detection strategies, security experts and anti-virus firms are interested in
employing signature-based ones since they are simpler and faster comparing to most of anomaly-based and specification-based detection techniques.

As was discussed in Section 2.1, most of the signature-based detection systems either consider malware’s specific behaviours as its signature or look into contents of the network traffic generated by the malware in order to extract specific patterns from invariant portion of payloads. Although both behavioural and content-based approaches provide promising results in identifying already known and analyzed malware, they are usually limited in detecting unknown and polymorphic variations of malware. Moreover, payload encryption limits content-based signature generators due to the fact that many recent malware families put multiple levels of encryption on packet contents. However, some of the recent works try to employ malware grouping and signature pruning techniques in order to generate more generic signatures to cover the general behaviour of malware families while maintaining a low false alarm rate. Thus, generating high quality signatures to identify newly emerged malware is a challenge in the field of malware analysis and detection. It’s essential to detect new variations of malware shortly after their presence in order to reduce the damage cost caused by their malicious activities.

In order to improve malware detectors ability to identify never-before-seen malware, some researchers employ evolutionary algorithms. In the previous section we briefly reviewed applications of evolutionary algorithms, specifi-
cally genetic algorithms, in the field of malware detection. In this section, we went over related research works that employ genetic algorithms as either the core algorithm of their system or an auxiliary optimizer to help the main algorithm. In some of these proposed approaches genetic algorithm is used to optimize feature selection phase to provide a minimal feature set for the main method that can be a classification algorithm such as decision tree or SVM.

There are also few proposed frameworks that include a genetic algorithm in the signature generation process. A genetic algorithm is able to evolve generated signatures in order to improve a system’s capability to predict never-before-seen malwares’ fingerprints. Overall, almost all of these approaches apply genetic algorithms to the traditional rule-based signature generator methods. None of these works use genetic algorithm to evolve string-based signatures that are extracted from the captured malicious network traffic. Another shortfall of these approaches is that they mostly train and test on the same dataset and do not consider temporal polymorphic characteristics of malware.
Chapter 3

A Genetic-Algorithm-Based Solution for HTTP-Based Malware Signature Generation

3.1 Overview

As we discussed in Chapter 2, many Network Intrusion Detection Systems (NIDS) tend to take advantage of signature-based detector engines. However, generating novel high quality signatures for malware in the early stage of their life is indeed a challenge in the fields of network security and malware analysis. In this chapter, we propose a genetic-algorithm-based framework for generating network signatures for HTTP-based malware. The main goal of this system is to generate signatures for groups of malware that have resem-
bling interactions with the Web based on their common network behaviours, then to evolve these signatures to identify the presence of new variations of the malware. At the same time, we try to maintain a low false positive rate.

In order to achieve this goal, we first group malware’s HTTP traces based on some of their statistical features using the $X$-means clustering algorithm. Then, using a method similar to the Token-Subsequent algorithm proposed in [55], we group malicious traces and generate signatures. Finally, we apply an adapted genetic algorithm to the signature set in order to evolve signatures. Evolved signatures are supposed to identify never-before-seen malware traces in the future, in a monitored network.

Figure 3.1 shows the architecture of the proposed evolutionary signature generator system. The proposed system consists of three main components:

- **Network Traffic Analysis**: This component takes malicious network traffic generated by each malware (pcap files) as its input. Then, it extracts HTTP requests from these files. Next, it measures features such as the number of GET requests, the number of POST requests, average number of parameters, etc., for each malware sample that generates at least one HTTP request during the running time window.

- **Signature Extraction**: Using statistical features of each malware sample obtained in the previous component, this component employs an $X$-means clustering algorithm in order to group similar malware traces.
Then, the Signature Generator engine creates string-based signatures for groups of structurally similar HTTP requests.

- **Signature Evolution:** This component takes known malware signatures as its input and applies our proposed version of a genetic algorithm in order to create a new generation of signatures. In fact, the new generation of signatures inherits proper building blocks of signature from the prior population of signatures. To maintain a low false alarm rate, we use a dataset of normal network traffic for evaluating generated signatures.

![Figure 3.1: Overview of the proposed evolutionary HTTP-based signature generator](image)

The combination of Signature Extraction and Signature Evolution components creates a two-step signature generator and evaluator system. The Signature Extractor part tries to generate a minimal set of signatures to cover
all of the malware traces whereas the Signature Evolution job is to mix and match the effective subsections of the generated signatures to create generic signatures while controlling the false positives rate.

There are a couple of input parameters for the algorithms we use. We use these parameters to tune-up the signature generation and evolution engines. For example, a genetic algorithm performs more efficient by using a certain combination of parameters such as mutation rate, crossover rate, and size of population.

3.2 Network Traffic Analysis

As we discussed previously, a vast majority of malware requires a network connection to be able to not only perform malicious activities but also to update and propagate themselves on the network. Today, the World Wide Web (WWW or W3) is the broadest network of interlinked hypertext documents accessible thorough the internet. The HTTP protocol traditionally forms the fundamental basis of application level communication channels in the World Wide Web. It provides a communication protocol for every type of applications to transfer data (the hypertext documents) to other nodes. Malicious software is not an exception. Therefore, since the introduction of this protocol, many malware that use HTTP as their communication protocol have appeared on the Web. Despite the fact that many other transmission
protocols for different network levels are available, HTTP-based malware has became more prevalent in the recent years.

For example, while the majority of older botnet frameworks used to employ the Internet Relay Chat (IRC) protocol for communications between their Command and Control (C&C) servers and the infected clients, many of the newer ones choose HTTP requests and responses with encrypted content as their communication channel [29][38][23]. In addition, about 86% of malware samples in our own database have sent at least one HTTP request to an external server, during their running time window.

### 3.2.1 HTTP Extractor

This component is responsible to parse pcap (packet capture) files, that contain network traffic generated by each malware sample, and extract HTTP requests. Assume $S = \{s_i\}_{i=1..n}$ is a set of malware samples and $T(s_i)$ is the set of HTTP requests captured by running a malware sample $s_i \in S$ in the time window $T$. This component extracts $T(s_i)$ for each malware sample from its captured malicious traces which are collected in a pcap file. A standard pcap file contains packet information for different network levels. As Figure 3.2 shows, according to the Internet protocol suite or TCP/IP stack [18], the application layer places on top of the transport layer. Application layer is where the HTTP protocol functions. This component parses network packets down to the Internet layer in order to extract HTTP re-
quests from the application layer and the associated source and destination IPs from the Internet layer.

Figure 3.2: Data flow in the Internet protocol suite model [7]

In order to enhance the packet analysis process we use a tool from an open-source library for network traffic capture called *jNetPcap* [16]. For each captured HTTP request in each pcap file, we extract its request line plus some other information such as the pcap file name, source IP, destination IP, date, header length, content length, request method, request path, and host name. For each pcap file, that contains network traffic generated by each malware, we extract such HTTP request details tuples and insert them all
into a database. According to [45], Figure 3.3 shows the structure of an HTTP request and how one can drill in to access the request line.

Figure 3.3: Structure of an HTTP request

3.2.2 Pcap Features Extractor

This component provides the HTTP Request Clustering algorithm with the necessary information about each pcap file. Therefore, it takes malicious HTTP requests details which are extracted and stored in the previous section as input and calculates the required statistical features. Then, it stores tuples of these features including number of HTTP requests, number of GET requests, number of POST requests, average length of requests paths, average number of request parameters, average content length of POST requests, and average length of requests headers. These features are some of the statistical features suggested for a similar clustering case in [58].
In fact, this component summarizes and represents statistical features of each pcap file as a vector of numbers. This summarization reduces the computational cost of the clustering algorithm since it only needs to measure the distances between two short vectors of numbers, which is not computationally expensive compared to clustering process of HTTP requests instead of pcap files.

Figure 3.4 presents the structure of an HTTP request line (without considering HTTP Version part). As we see, each request line starts with the request method that can be GET, POST, PUT, DELETE, etc. Among all possible request methods, we only consider the number of POST requests and the number of GET requests as two of the statistical features for each malware sample since there are a very few number of requests with other request methods such as HEADER, PUT, DELETE, etc. These types of requests constitute approximately 1% of all of the requests in our database. Even though we do not consider the count of these types of requests as the statistical features, we still keep them associated to the malware sample since they have other valuable information in their path and parameters that will be used later in the signature extraction component. The average length of requests paths and the average number of request parameters are two of the other features we extract by parsing request lines. As illustrated in Figure 3.4, the request parameters are separated from the request path by a question mark (?). Each parameter-value pair is also delimited by an
ampersand (\&) from the next one.

![Figure 3.4: Structure of an HTTP request line](image)

3.3 Malware Clustering and Signature Generation

The related works in Chapter 2 mostly propose system-level or content-based signature generator systems while our proposed framework specifically focuses on malicious HTTP traffic traces and mainly on the request line of each malicious HTTP request. Different attackers may have used the same Command and Control (C&C) toolkit while owning different servers. Due to this fact, this component aims to group together malware variants that communicate with different web servers but their HTTP traces are strongly similar in terms of their structure and number of requests. Then, it automatically generates network-level signatures for identifying the malicious network traces.

The Signature Extraction component consists of two parts, the first one is the HTTP request clustering component that groups similar malware traces...
to feed to the second component which is responsible for extracting string-
based signatures from the clustered malware traces.

3.3.1 Clustering Malware Traces

This component queries the feature vectors that have already been calculated by the feature extractor component. We first normalize these feature vectors since we don’t want the differences in the range of different features to give some of the features more influence on the distance function, over the other ones. Therefore, we normalize all numbers in the given dataset of vectors to the range of [0, 1].

After the normalization process, we divide feature vectors representing malware samples into smaller groups using the X-means clustering algorithm [57]. The main motivation for this choice is that X-means does not require users to specify the exact number of clusters. Therefore, it’s appropriate for clustering samples which form an arbitrary number of clusters, such as our case, where we do not know the number of the similar groups of malware samples in our dataset.

Some of the advantages of the X-means clustering algorithm over K-means are: a) It does not require specifying a fixed number of clusters; b) It is efficiently scalable, and c) It is relatively less prone to local minima. In fact, X-means tries to efficiently estimate the $K$ parameter of the K-means
algorithm. In the K-means algorithm, $K$ is a user specified parameter that
determines the number of clusters in the space of cluster locations. Actually,
X-means goal is to optimize $K$ using Baysian Information Criterion (BIC)
or Akaike Information Criterion (AIC) [57]. Formally, given a set of feature
vectors $V = \{v_i\}_{i=1..n}$ extracted in the previous component, X-means first
guesses optimum number of existing clusters, say $x$. Then it aims to partition
the $n$ feature vectors into $x$ clusters ($x \leq n$) $C = \{C_1, C_2, ..., C_x\}$ while
minimizing sum of squares in each cluster:

$$\arg\min_C \sum_{i=1}^{x} \sum_{v_j \in C_i} ||v_j - \mu_i||^2$$

where $\mu_i$ is the mean of vectors in each $V_j$.

### 3.3.2 Generating HTTP-based Signatures

Up to this point, we extracted HTTP requests out of the network traffic
generated by each malware sample. Then, we calculated statistical fea-
tures of the malware samples and stored the feature vectors that represent
HTTP-based statistical characteristics of malware samples. Afterwards, we
normalized these vectors and clustered statistically resembling malware sam-
ples using X-means clustering algorithm. Now we have clusters of similar
malware samples, each of which containing details of the HTTP requests
generated by each malware.
Having the clusters, the signature generator component aims to aggregate all of the malicious HTTP traces generated by malware samples in each cluster and generate a set of string-based signatures to cover all of the malware traces in the cluster. Let $C_i$ be a cluster of malware samples obtained from the previous section, and $T_i = \{T(m_k^{(i)})\}_{k=1..c_i}$ the set of malware traces (i.e. HTTP request details) captured by executing each malware sample in the cluster $C_i$. Our goal is to generate the set of network signatures $S_i = \{s_k\}_{k=1..n}$. To cover all of the malware traces in $T_i$, we make sure each of them matches at least with one of the signatures in the $S_i$ collection.

Each signature in the signature set $S_i$ is made up of a method part followed by a regular expression string which is request URI (Uniform Resource Identifier). The method part in each signature is one of the conventional HTTP request methods, for example POST, GET, HEADER, etc. The regular expression part is a pattern to match the URI part of the similar HTTP queries. The URI itself consists of a request path and request parameters (See Figure 3.4). The regular expression part of a signature is made up of one or more tokens. We define a contiguous sequence of characters as a token. Actually, each token in the regular expression part of network signatures represents one of the invariant parts in the request line of the HTTP requests generated by resembling malware.

Figure 3.5 shows an example of HTTP-based network signature that matches HTTP request lines similar to the one mentioned in Figure 3.4. As repre-
sented in this example, $t_1$, $t_2$, $t_3$, and $t_4$ are the sequence of tokens (i.e. invariant parts) of request URI portion of a network signature in the form of “method $t_1*t_2*t_3*t_4$”. Asterisks (*) in a signature are delimiters that can be replaced with any arbitrary string in an actual HTTP request line.

In order to generate these type of signatures, existing HTTP requests in a cluster get alphabetically sorted by requests URIs, in an ascending order. This sorting helps to put samples which have similar starting URIs close to each other. Then, we iterate over these sorted samples and generate signatures for similar enough groups of adjacent URIs. We show how to extract signature from two samples, and then we describe how to employ an algorithm similar to *token-subsequences* implemented in [55] to extract token-subsequence signature from any number of samples.

A subsequence of two strings (e.g. URIs in our case) is a sequence of characters which occur in the same order in both strings. As opposed to a common substring of two strings, the characters are not necessarily the consecutive ones. For example, let “???unb????iscx???” and “???unb??iscx???” be the two strings we want to extract the common subsequence from, assuming

```
Method
POST /the/request/path/act*.ht?code=*&target=root&dw=8
```

Figure 3.5: Example of HTTP-based network signature
that each “?” can be any other character. Keeping the order of characters, if these strings are aligned in order to maximize the number of matching characters by inserting blank spaces to fill the empty spaces, the result looks like Figure 3.6. Therefore, the longest common subsequence is “unbiscx”. Since we are interested in a regular expression format, we implemented an algorithm to extract the longest common subsequence of two strings and put an asterisk between each two consecutive substrings. In the previous example, the longest common subsequence “unbiscx” transforms to a regular expression such as “*unb*iscx*” by replacing groups of adjacent question marks and dashes with a single asterisk.

Figure 3.6: Example of string alignment for two strings to find the common subsequence

Finding the longest common subsequence (LCS) of an arbitrary number of strings is an NP-hard problem [49]. But, for two strings of $n$ and $m$ characters, the dynamic programing version of the algorithm that solves the problem in $O(n \times m)$ time and space have been implemented.

In order to generate a signature that matches every URI in the group of similar sorted samples, we iteratively apply our longest common subsequence
extractor algorithm. First, it extracts the signature of the first two samples. Then, in each iteration, it extracts the longest common subsequence of the signature generated in the previous step, and the next sample.

Even though the clustering step gathers statistically similar malware samples into the same clusters, still there are smaller groups of structurally similar samples in each cluster, which do not share enough resemblance with the other small groups. In other words, the longest common subsequence of two samples with dissimilar structures may be too short for representing a specific family of malware. Therefore, the resulting short signature can match many different strings, that definitely increases the false positive rate. To avoid generating such signatures, before accepting the new generated signature by adding the next sample in each iteration, we compute a similarity score proposed in [74]. This score shows the percentage of obtained LCS length out of the total length of the input strings. Assume that the strings $s_1$ and $s_2$ have lengths $L_1$ and $L_2$, respectively, and their LCS has length $C$. The similarity score is calculated as $\frac{2\times C}{L_1+L_2}$. This function normalizes the score in the range of $[0..1]$, where 0 means $s_1$ and $s_2$ have nothing in common, and 1 means they are identical strings.

In addition, in order to avoid generating signatures which are prone to false positives, another criteria is defined that is minimum length of the longest token. Although we make sure that the similarity score of the grouped HTTP requests is above the threshold, we still may generate signatures with small
tokens which are more probable to match a non-malicious HTTP request and cause a false alarm. Therefore, after generating each new signature, the algorithm checks if the length of the longest token of the signature is greater than the threshold (e.g. 8 as will be described in Section 4.2). In other words, there should be at least one common substring in the two strings that is long enough to indicate that there is a meaningful similarity between the two. For example, in Figure 3.7, the longest token of the fourth signature is “/App/a”, which is not long enough.

Figure 3.7 illustrates the iterative process of signature generation procedure. The algorithm selects the first two samples and extracts their signature, $t_1t_2t_3t_4$. Then, it calculates the similarity score of the two strings based on their LCS (e.g. $t_1t_2t_3t_4$). If the score is above a certain threshold (e.g. 75% as will be describe in Section 4.2), the algorithm keeps the signature and continues the process by applying the algorithm to current signature and the next string. Once the similarity score drops under the threshold, we remove the undesired signature and put the last proper signature in the signature set for that cluster. For example, sample number 5 in Figure 3.7 does not have high enough similarity score with the third signature which is extracted from samples number 1 to 4. Then, the algorithm selects the next two samples and follows the same process.

Processing all of the sample HTTP requests in the cluster $C_i$ results in the signature set $S_i$ filled with network signatures that match all of the malware
traces in $T_i$.

### 3.4 Signature Evolution Using A Genetic Algorithm

Up to this point, the system has generated a set of signatures for each cluster of malicious HTTP traces. The signature evolution component combines all of the signatures from different clusters and processes this population of network signatures in order to generate an evolved generation of signatures. As malware families carry over their characteristics into the next generations of descendants, this component likewise transfers combinations of the important portions of current malware signatures to the next generation. Therefore, the new generation of signatures is able to predict the structure of new polymorphic malware variants.

![Figure 3.7: Example of signature generation process](image)

Figure 3.7: Example of signature generation process
To adapt the Genetic Algorithm on this problem, a definition for each of the necessary steps and functions as described in Section 2.2 is required. In the following we describe how to represent individuals and apply the genetic algorithm operators and functions to generate an evolved population of network signatures.

3.4.1 Chromosome Representation

As was discussed in the Section 2.2, chromosome representation is the first and most important step of employing the genetic algorithm to solve a problem. In our problem, the network signatures generated by the previous component are considered as the individuals which form the initial population. The generated network signatures are made up of a method part followed by a regular expression string. The regular expression itself consists of one or more tokens. For example a four-token signature looks like method t1*t2*t3*t4. Two types of genes that create chromosomes are defined; method gene, and token gene. A method gene can only take one of the conventional HTTP request methods such as GET, POST, HEADER, etc., as its value. Each token gene represents each of the tokens building a network signature. Figure 3.8 illustrates how we simply convert network signatures to chromosomes required for the genetic algorithm.

As opposed to the traditional chromosome representation methods mentioned in Section 2.2, we define genes which carry string values instead of
binary bits, real numbers, or characters. In fact, these tokens are considered as building blocks of our solution. The genetic algorithm arranges the building blocks in a proper order to obtain solutions that are string-based network signatures and are able to identify never-seen-before malware.

Obviously, our defined chromosomes do not have fixed lengths. The lengths of the chromosomes in the initial population are equal to the number of tokens of the corresponding signatures. Crossover function modifies chromosomes in the next generation, which changes the length of the chromosomes as a result.

### 3.4.2 Parameters, Functions and Genetic Operators

In this section, the parameters used to tune up the algorithm are described. The design of each genetic operator is given. Fitness function is an important part of the genetic algorithm. In fact, it helps the algorithm to produce good solutions and avoid the improper building blocks which cause bad solutions. Some of the parameters are the population size, crossover and mutation rate,

![Figure 3.8: Chromosome representation of a signature](image-url)
and the termination condition. Genetic operators include selection, crossover, and mutation.

**Fitness Function** is the only source of external knowledge to the genetic algorithm and determines how good a potential solution is. All of the individuals in a population are actually potential solutions. The fitness function is an evaluation metric to score each individual in order to make the algorithm able to compare reproduced individuals. The higher the fitness score, the more productive the individual. In our work, the goal is to generate network signatures to detect the evolved malware of the future with a low false positive rate. Therefore, the fitness function should express our expectation of such signatures based on our current knowledge. We know that future malware samples may have similar structure to current ones; then this knowledge is transferred to the next generation of signatures by considering the signature set of current malware as the initial population. In addition, we also recognize a signature as fit if it sets a lower number of false flags on normal network traffic. For valuing this feature in the next generation of signatures, the fitness score of each signature (i.e. chromosome) is defined as the inverse of its false positive rate on a set of normal HTTP requests. Therefore, signatures with high false alarm rates get lower scores, and lower chance of survival as a result.

**Termination Condition** is the criterion by which the genetic algorithm decides whether to continue searching or stop the algorithm. There may
be one or a combination of criteria that is checked after each iteration to see if it is time to stop. Since the proposed system is not searching for a specific perfect solution (e.g. a signature set that is able to identify all new malware trace with a zero false positive rate) and it is not aware of the actual detection rate for future malware, it cannot wait until the algorithm converges to the solution or the average fitness passes a certain threshold. Therefore, we decided to limit the number of the algorithm’s iterations based on the generation number. For example, we terminate the algorithm at the 12th iteration (the reason for choosing 12 as the threshold is discussed in Section 4.3) and consider the last generation of signatures as an acceptable solution. Later in the Chapter 4, it is described that how we adjust this limit on the number of generations to find the proper number while considering other parameters as well.

Selection Method is used to select survivors of the current generation which are parents of the next generation. Tournament Selection (q-tournament) is used to choose parents. As was discussed in the Section 2.2, the tournament selection method does not need to calculate the average fitness score of the population or sort the population. Therefore it’s computationally more efficient than some of the other methods. Moreover, q can be used as a parameter to adjust the selection pressure since for choosing each parent, it randomly picks q chromosomes then selects the fittest one as a parent. It repeats this process until all of the parents are selected. By choosing
a small $q$, we increase the chance of weaker individuals to be selected, and vice versa.

**Crossover** is the operator which plays a major role in the reproduction process. It combines portions of the parents chromosomes to produce children. We choose one-point crossover method for our algorithm. It divides the parent chromosomes into two sections then appends the second section of each parent to the first section of the other one. Thus, offsprings inherit some of the characteristics of both parents. Figure 3.9 illustrates the operation of the crossover operator on two signature chromosomes.

![Figure 3.9: Example of one-point crossover operation on two signature chromosomes](image)

**Mutation** helps to maintain the genetic diversity, meaning that it redirects parts of the population to untouched areas in the search space, in order to help the algorithm avoid local optimums and premature convergence. The adjacent order changing mutation method is implemented to reach this goal. As was mentioned in the Section 2.2, it randomly selects two adjacent genes in the chromosome and then switches their position. However, we do not
apply mutation to the first gene which is the method gene, because it de-
stroys the structure of the chromosome (network signature). Since during
the signature generation procedure the algorithm keeps the order of the to-
kens in each signature, this type of mutation helps the algorithm to slightly
rearrange this order to build new variations of signatures.

### 3.4.3 The Proposed Genetic Algorithm Procedure

Having had all of the elements, functions, and operators of the genetic
algorithm defined, we now describe the overall procedure of the proposed
genetic algorithm for evolving HTTP-based network signatures. The input
to this algorithm is the current malware signatures provided by the malware
clustering and signature generator component.

Applying the proposed chromosome representation, the algorithm converts
the network signatures set of malware samples of one month to a set of chro-
nosomes (individuals) which is considered as the initial population. Then,
it evaluates each individual based on the fitness function that assigns a score
to each individual according to its false positive rate on the normal network
traffic. Now that the individuals are scored, the algorithm is able to compare
them with each other and choose the parents. The q-tournament algorithm
is implemented to choose the parents for reproduction. In this step some
chromosomes may be selected more than once (elite individuals), and some
may not be selected at all (weak individuals). This elitism approach transfers
effective parts of signatures to the next generation. However, q-tournament also gives a chance to a weak individual, due to the fact that there may be some strong genes (tokens) in its chromosome, which will present their effect once these genes are mixed with other effective ones through recombination process.

In the recombination process, we pick pairs of the survived individuals and apply the described crossover operator if a random number from the range $[0,1]$ is lower than the crossover rate. In other words, offspring are only produced from a percentage (equal to the crossover rate) of the current population and transfer the rest untouched to the next generation. This is one of the approaches to control the convergence speed. Once the crossover process is done, the algorithm applies the mutation operator to a fraction of the produced offspring proportional to the mutation rate. Finally, we verify that the generated individual is a valid one, meaning that it is convertible to a valid network signature. For example, an HTTP method (e.g. GET) can not be in the middle part of network signature.

At this point, the algorithm completes production of the next generation. Then, it checks whether the termination condition is satisfied or not. The number of algorithm iterations is restricted by a limit set on the number of generations. Thus, we stop the algorithm and consider the last generation as final solution if the algorithm reaches that limit, otherwise, it goes over the same process again, taking the current generation as the initial popula-
tion. Figure 3.10 depicts the flowchart of the implemented genetic algorithm procedure.

Figure 3.10: Flowchart of the proposed genetic algorithm
3.5 Concluding Remarks

In this chapter, we proposed a three-component framework for generating evolved signatures to detect polymorphic variations of the current malware in the future. This offline system gets the detected malware traces of one month and analyzes the related network traffic. Then, it groups these traces and generates string-based signatures which are able to identify current malware traces. These signatures match HTTP request line of malicious HTTP requests. Since HTTP request line is a part of HTTP header, there is no need to perform computationally expensive payload inspections.

We transformed our problem into a genetic representation in order to take advantage of the genetic algorithm as a search heuristic. Hence, central to the framework is the novel evolutionary procedure that evolves this set of signatures in order to be able to predict the structures of malicious traces of the never-seen-before malware. At the same time, the system maintains a low false alarm rate on a normal network traffic.
Chapter 4

Test and Evaluation

In this chapter, the results of experiments performed for evaluating the proposed framework are reported. Two datasets are used to evaluate the proposed framework, one containing malicious network traffic for generating signatures, and the other one containing normal network traffic for finding false positive rate. We generate signatures from the malicious HTTP traces in each month then apply it to the network traffic of the same month, following months, and a set of normal HTTP traces in order to calculate and compare the detection and false positive rates.
4.1 Datasets

4.1.1 Malicious Network Traces

We used the same dataset of malicious network traces that has been used in [50], where the authors try to navigate and visualize the malware intelligence space. They have gathered 559,747 malware samples from May 2010 to April 2011 and captured the network traffic generated by each sample during the execution period. Therefore, all of the network traces (including HTTP traces) generated by each malware sample are in a pcap file. We use 5 months worth of malicious traffic of this dataset from November 2010 to March 2011 for our experiments. This period includes pcap files containing malicious network traffic generated by 5242 distinct malware samples. 4456 of these malware samples have generated at least one HTTP request during the running period. The second column in Table 4.1 shows the number of distinct malware samples collected in each month.

Table 4.1: HTTP request clustering and signature generating results

<table>
<thead>
<tr>
<th>dataset</th>
<th>samples</th>
<th>clusters</th>
<th>requests</th>
<th>signatures</th>
<th>processing time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mal-2010-11</td>
<td>421</td>
<td>8</td>
<td>1648</td>
<td>761</td>
<td>3</td>
</tr>
<tr>
<td>Mal-2010-12</td>
<td>390</td>
<td>12</td>
<td>1278</td>
<td>749</td>
<td>3</td>
</tr>
<tr>
<td>Mal-2011-01</td>
<td>624</td>
<td>20</td>
<td>2165</td>
<td>1215</td>
<td>5</td>
</tr>
<tr>
<td>Mal-2011-02</td>
<td>2561</td>
<td>51</td>
<td>6174</td>
<td>3429</td>
<td>12</td>
</tr>
<tr>
<td>Mal-2011-03</td>
<td>460</td>
<td>49</td>
<td>1309</td>
<td>705</td>
<td>4</td>
</tr>
</tbody>
</table>
4.1.2 Normal Network traffic

A dataset of normal network traffic is required to calculate false positive rates in order to evaluate the quality of the generated network signatures. We use the ISCX dataset proposed by Shiravi et al [66]. From their dataset we selected one pcap file containing 17.3GB of normal network traffic (no malicious activities) generated on Friday, June 11, 2010. This file includes 98440 distinct HTTP requests. To calculate the false alarm rates the generated signatures are applied to a random set of 10000 normal HTTP requests of this dataset and the number of matches are counted.

4.2 Signature Generation

Before generating evolved signatures, we evaluate the malware signatures generated by the malware Signature Generation component of our framework. A proof-of-concept version of the HTTP-based signature generator component is implemented in Java. We set the threshold for the similarity score defined in Section 3.3.2 as \( \text{MIN\_SIMILARITY\_SCORE} = 0.75 \). Test results for different months show that this threshold provides an acceptable detection rate while maintaining a low false positive rate. We set the \textit{minimum longest token length} used as the other signature acceptance criteria to 8. The same thresholds at the detection time are used, meaning that this component only flags HTTP requests that match with a signature with a minimum similarity score of 0.75 and the matching result (i.e. a tokenized
LCS) contains at least one token with more than or equal to 8 characters. All the experiments were performed on a 2-core 2.66GHz Intel machine with 4GB of RAM, though we never used more than 1GB of RAM.

To evaluate the primitive generated signatures of each month, we apply them to malicious network traffic of the same month and the following months. Table 4.1 summarizes the results of the automatic signature generation process. For example, the first row of this table show that our malware dataset contains 421 malware samples each of which generate at least one HTTP request during the running time. This one month worth of HTTP-based malware contains a total of 1648 distinct HTTP requests. The clustering algorithm divides these samples and the corresponding HTTP requests into 8 clusters based on malwares statistical similarities. Number of signatures extracted from these clusters of malware traces totals 761. The last column shows that it takes 3 seconds to cluster samples and generate signatures for this month.

The matching method sets the thresholds for the minimum similarity score and minimum length of longest token to the same value used at the signature generation time, respectively $MIN\_SIMILARITY\_SCORE = 0.75$ and $MIN\_LONGEST\_TOKEN = 8$. Table 4.2 shows detection rates of sets of generated signatures for each month to identify new malware traces of the following months. This table illustrates how capable our signatures are to detect current and future malware traces. The following approach is
performed to measure detection rates of signatures generated by this component. Given a signature set \textit{Sig-2010-11}, we match it over sets of malicious HTTP requests generated by malware samples in the same month (i.e. Mal-2010-11) and the following months (i.e. Mal-2010-12, Mal-2011-01, etc.). The algorithm repeats the same process for the following months as well.

For example, according to Table 4.2, the signature set \textit{Sig-2010-11} is able to detect 99.45\% of malicious HTTP traces generated by malware samples collected in Mal-2010-11, in 121 seconds. It also detects 31.84\% of the future malicious traces in Mal-2010-12, 28.12\% of Mal-2011-01, and so on.

Table 4.2: Signature detection rate on current and future malicious HTTP requests

<table>
<thead>
<tr>
<th>dataset</th>
<th>Mal-2010-11 DR (%)</th>
<th>Mal-2010-12 DR (%)</th>
<th>Mal-2011-01 DR (%)</th>
<th>Mal-2011-02 DR (%)</th>
<th>Mal-2011-03 DR (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>signature</td>
<td>t (s)</td>
<td>t (s)</td>
<td>t (s)</td>
<td>t (s)</td>
<td>t (s)</td>
</tr>
<tr>
<td>Sig-2010-11</td>
<td>99.45</td>
<td>121</td>
<td>31.84</td>
<td>161</td>
<td>28.12</td>
</tr>
<tr>
<td>Sig-2010-12</td>
<td>-</td>
<td>-</td>
<td>99.68</td>
<td>70</td>
<td>27.11</td>
</tr>
<tr>
<td>Sig-2011-01</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>99.03</td>
</tr>
<tr>
<td>Sig-2011-02</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sig-2011-03</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Therefore, each generated set of signatures not only provides a significantly high detection rate on the current malicious traces but also is able to identify new, never-before-seen traces generated by malwares of the same families. This is due to the fact that the Signature Generation component tries to build generic signatures which represent common structure of a group of malware traces. In other words, having captured network activities of a group of
malwares, the proposed framework is able to generate network signatures to
detect the presence of new malware variations that inherit the structure of
the same malware family. The new malware traces resemble the structure of
their predecessors since malware authors usually use the same malware Web
tool-kits.

Table 4.2 also shows that the detection rate of generated signatures in most
cases decreases over time because malwares behaviours evolve. Therefore,
updating the signature set with newly generated signatures helps keeping
the detection rate of future malwares high over time.

Table 4.3: Signature false positive rate on legitimate HTTP requests

<table>
<thead>
<tr>
<th>FF rate</th>
<th>Sig-2010-11</th>
<th>Sig-2010-12</th>
<th>Sig-2011-01</th>
<th>Sig-2011-02</th>
<th>Sig-2011-03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>87</td>
<td>79</td>
<td>202</td>
<td>421</td>
<td>73</td>
</tr>
</tbody>
</table>

The parameters for the signature generation process are set so that to
maintain a relatively high detection rate and keep the false positive rate as
low as possible at the same time. The network signatures are also evaluated
on a random set of the normal HTTP traces of the ISCX dataset. Table 4.3
shows that generated signatures of each month cause a relatively small num-
ber of false alarms.
4.3 Signature Evolution

The signature evolution component employs a genetic algorithm to evolve the signature sets generated by the Signature Generation component in order to increase the detection rate of the future malware traces. It maintains a low false positive rate at the same time. We set the parameter \( q \) for the \( q \)-tournament selection method to 10. It was found that the best range for crossover and mutation probabilities are \([0.60, 0.85]\) and \([0.05, 0.15]\), respectively, for this dataset. No significant differences in the results were observed within these ranges. Hence, the fixed values of 0.7 for crossover rate and 0.1 for mutation rate are adopted.

The threshold on the number of generations is another input parameter for the genetic algorithm that is used as the termination criteria for our algorithm. To determine a proper limit on the number of iterations the proposed genetic algorithm is applied with different number of iterations on the signature set of each month (e.g. Sig-2010-11, Sig-2010-12, etc.) and the false positive rates are measured. We calculate the average false positive rate of evolved signature sets for each number of iterations. In addition, we measure the detection rate of these signatures over undetected malicious traces of the next corresponding month. Then, an average detection rate is calculated for each number of iterations. Table 4.4 summarizes the results of this experiment. Figure 4.1 illustrates that the algorithm provides a lower false positive rate around 8 iterations while Figure 4.2 shows that evolved
signatures of the 16th generation detect more undetected malicious traces on average. To keep a balance between false positive and detection rates, we terminate the genetic algorithm after 12 iterations and take the 12th generation of signatures as the output of the algorithm.

Table 4.4: Signatures average false positive rate and average detection rate of undetected malware traces vs. number of iterations

<table>
<thead>
<tr>
<th>Number of Iterations</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average FP rate</strong></td>
<td>$8.8 \times 10^{-3}$</td>
<td>$12.0 \times 10^{-3}$</td>
<td>$5.2 \times 10^{-3}$</td>
<td>$0.9 \times 10^{-3}$</td>
<td>$2.4 \times 10^{-3}$</td>
<td>$1.7 \times 10^{-3}$</td>
<td>$4.6 \times 10^{-3}$</td>
</tr>
<tr>
<td><strong>Average DR (%)</strong></td>
<td>9.32</td>
<td>8.28</td>
<td>11.66</td>
<td>19.42</td>
<td>24.74</td>
<td>11.93</td>
<td>3.48</td>
</tr>
</tbody>
</table>

Figure 4.1: Signatures average false positive rate vs. number of iterations

Almost the same procedures used for evaluating the base signatures of each month is used to evaluate the evolved signatures. The base signatures of each month are fed to the genetic algorithm; then it processes the current generation and generates evolved signatures. Instead of applying the evolved signatures to all of the malware traces of the current and following months, we first
match the base signature set over the malicious HTTP traffic traces and collect unmatched HTTP requests. In fact, these requests are the false negatives which have not been detected by the base signatures. Then, evolved signatures are matched over the set of undetected HTTP requests of each month. Table 4.5 summarizes the detection rate of evolved signatures matched over undetected malicious network traces.

Table 4.5: Evolved signatures detection rate on current and future malicious HTTP requests

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FN (%)</td>
<td>DR (%)</td>
<td>FN (%)</td>
<td>DR (%)</td>
<td>FN (%)</td>
<td>DR (%)</td>
</tr>
<tr>
<td>GA-Sig-2010-11</td>
<td>0.55</td>
<td>55.55</td>
<td>68.16</td>
<td>19.86</td>
<td>71.88</td>
</tr>
<tr>
<td>GA-Sig-2010-12</td>
<td>-</td>
<td>-</td>
<td>0.32</td>
<td>100.00</td>
<td>72.89</td>
</tr>
<tr>
<td>GA-Sig-2011-01</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.97</td>
</tr>
<tr>
<td>GA-Sig-2011-02</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GA-Sig-2011-03</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 4.2: Signatures average detection rate of undetected malware traces vs. number of iterations
Take the first row of Table 4.5 as an example. GA-Sig-2010-11 is an improved set of signatures obtained by evolving the base signature set of November 2010 (Sig-2010-11). The value in the FN column under Mal-2010-11 shows false negative rate of Sig-2010-11 signature set which contains the base signatures of malicious traces in Mal-2010-11. In fact, this rate is calculated as \( FP = 1 - DR \) where DR is the detection rate of Sig-2010-11 on Mal-2010-11 provided in Table 4.2. In other words, FN is the percentage of malicious traces of the month, which were not detected by the base signature set. The DR column in this table shows the detection rate of the evolved signatures matched over undetected malware traces. For example, the evolved signature set of November 2010 (GA-Sig-2010-11) is able to identify 19.86% of the 68.16% malicious traces of Mal-2010-12 that were not detected by the base signature set of November 2010 (Sig-2010-11). It mean that our the evolved signatures have improved our basic detection rate by 19.86% \( \times \) 68.16\% = 13.41\%, from 31.84\% (refer to first row and second column of Table 4.2) to 31.84\% + 13.41\% = 45.25\%. This is considered as a significant improvement in detecting never-before-seen malware traces by automatic enhancement of current signatures.
Table 4.6: Detection rate improvement by the evolved signatures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>signature</td>
<td>Base</td>
<td>Base+GA</td>
<td>Base</td>
<td>Base+GA</td>
<td>Base</td>
</tr>
<tr>
<td>DR (%)</td>
<td>DR (%)</td>
<td>DR (%)</td>
<td>DR (%)</td>
<td>DR (%)</td>
<td>DR (%)</td>
</tr>
<tr>
<td>2010-11</td>
<td>99.45</td>
<td>99.75</td>
<td>31.84</td>
<td>45.37</td>
<td>28.12</td>
</tr>
<tr>
<td>2010-12</td>
<td>-</td>
<td>-</td>
<td>99.68</td>
<td>100.00</td>
<td>27.11</td>
</tr>
<tr>
<td>2011-01</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>99.03</td>
</tr>
<tr>
<td>2011-02</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2011-03</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.6 represents how using evolved signatures improve the detection rate of the unknown malware traces. For example, the first row of results under Mal-2010-12 column shows that the base signatures generated in 2010-11 are able to detect 31.48% of malicious traces of the next month (i.e Mal-2010-12). Adding evolved signatures to this set increases the detection rate to 45.37%. Generally, using the base and evolved signatures together increases the average detection rate of the malicious traces of the next immediate month from \( \frac{31.84\% + 27.11\% + 32.55\% + 62.1\%}{4} \) = 38.40% by 12.48% to \( \frac{45.37\% + 37.86\% + 42.84\% + 77.45\%}{4} \) = 50.88%. This improvement happens while the average false positive rate of the evolved signature sets is \( 2.7 \times 10^{-3} \).

Table 4.7: Evolved signatures false positive rate on legitimate HTTP requests, and signature evolution time

<table>
<thead>
<tr>
<th>FP rate</th>
<th>GA-Sig-2010-11</th>
<th>GA-Sig-2010-12</th>
<th>GA-Sig-2011-01</th>
<th>GA-Sig-2011-02</th>
<th>GA-Sig-2011-03</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA Time (min)</td>
<td>2.1 \times 10^{-3}</td>
<td>0</td>
<td>4.3 \times 10^{-3}</td>
<td>4.7 \times 10^{-3}</td>
<td>1.1 \times 10^{-3}</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>23</td>
<td>59</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>
We also measure false positive rate of the evolved signatures using the normal HTTP traffic of the ISCX dataset, in the same way that false alarms for the base signatures were measured. Table 4.7 represent the quality of evolved signatures of each month in terms of false positive rate. These false positive rates are comparable to the false alarm rates of the base signature sets from Table 4.3. Even in few cases the evolutionary process has improved the false positive rate. This is due to the fact that false positive rate is considered as a criteria to evaluate and score reproduced chromosomes (i.e. representation of network signatures). Therefore, false positive rate is always controlled in order to keep the algorithm away from the improper solutions.

Figure 4.3: The correlation between crossover and mutation ratios with average detection rate
To evaluate the correlation between crossover and mutation ratios with average detection rate of the undetected malware traces (i.e. the ones that were not detected by the base signatures), and to find the suitable crossover and mutation probabilities needed to obtain the acceptable solutions, the algorithm were conducted over 4 different months of malicious HTTP requests for 11 different crossover rates and 6 different different mutation rates. For each combination, we measure the average detection rate of the 4 months worth of evolved signatures matched over their next month. Crossover rate varied from 0.5 to 1 in increments of 0.05. Mutation rate varied from 0.05 to 0.25 in increments of 0.05. A three-dimensional plot of the average detection rate for different values of crossover rate and mutation rate is depicted in Figure 4.3. As shown in this figure, the algorithm works best with crossover ratios about 0.7 and mutation rates interval about 0.1.

4.4 Comparison With Related Works

As was discussed in Section 2.1, most of the signature-based malware detection systems either consider malwares specific behaviours as its signature or look into contents of the network traffic generated by the malware in order to extract specific patterns from invariant portion of payloads.

Ilgun et al. in [34] represent attacks as state transition diagrams. Their system, called STAT, is able to identify the sequence of consecutive events
triggered by an already known malware. This system is also able to detect variations of the same malware. Their approach requires system level analysis of malware samples while ours is based on network level communications. Hence, our proposed system can sit on the edge of a network to detect malicious traces.

Earlybird [68], Autograph [39], Honeycomb [43], and Polygraph [55] are similar automated malware signature generation systems. They all look into packet’s payloads to extract signatures based on invariant parts of the contents. Our work is different because we focus on the headers of the malicious HTTP requests. Therefore, even if the content in encrypted, the proposed system can still extract network signatures out of the HTTP headers. In addition, Compared with Autograph, we do not consider the specific TCP port and domain names used by the malware. While longest common subsequence algorithm is used in the Signature Extraction component of our framework, Honeycomb applies longest common substring algorithm to match similarities in the invariant portions of the packet payloads. Therefore, it may fail to identify polymorphic malware payloads since signatures are all continuous strings, whereas polymorphic malware are able to mutate their content while keeping the same malicious functionality. Although Polygraph covers this limitation, it still suffers from encrypted contents because it uses longest common subsequent algorithm to extract string-based signatures out of packet’s payloads.
The most similar system to our proposed framework is proposed by Perdisci et al. [58]. They cluster malicious HTTP queries and generate network signatures of each cluster. To generate high-quality generic signatures to detect future malware of the same families, they adopt a multistep clustering process which includes coarse-grained clustering of malware samples and fine-grained clustering of malicious HTTP requests. On the other hand, in this thesis we generate network signatures with high detection rate of the current month malware, after a similar coarse-grained clustering step. Then, instead of using a computationally expensive fine-grained clustering algorithm, an adopted genetic algorithm is applied to these signatures to produce evolved ones. They offer a comparable detection rate on never-before-seen malware, and a low false positive rate on a benign network traffic. However, the processing time for clustering and signature extraction steps is in the order of hours while signature generation and evolution process of our proposed system takes less than an hour to produce enhanced signatures. Albeit, the processing times also depend on size of the datasets.

Evolutionary algorithms are also employed in rule-based and signature-based systems as either the core algorithm of a detector system or an auxiliary optimizer to enhance the main algorithm. Proposed works in [40], [67], and [69] mainly use a genetic algorithm to optimize the core algorithm or the feature selection process. On the other hand, we take advantage of the genetic algorithm capabilities in the core of the proposed framework.
Genetic algorithms are employed either to classify network behaviours into normal and abnormal ones, or to evolve rule-based signatures in [75], [31], [61], and [47]. Our work is different because it applies a genetic algorithm to string-based network-level signatures.

4.5 Concluding Remarks

We evaluated our proposed framework in two steps. First, the Signature Generation component was fed with the malicious HTTP requests extracted from the pcap files. Once this component generates network signatures, we evaluate the quality of the signatures in terms of false positive rate on normal traffic of ISCX dataset, and detection rate on malicious HTTP traces of the future. Given malicious network traffic of a month, network traffic generated by malware in the following months are considered as future malicious traces. The results show that this component of our proposed framework is able to identify almost all of the malicious traces of the current month (i.e. learning dataset). It is also able to detect a portion of the malicious traffic of the following months (i.e. test dataset). Moreover, our experiments show that this component maintains a low false alarm rate. This is due to the fact that the defined thresholds on minimum similarity score and minimum length of the longest common token prevent the matching algorithm to match legitimate HTTP requests.
In the next step, we evaluated evolved signatures which are generated by the Signature Evolution component, using the same procedure. The evolved signature were applied to undetected malicious traces and reported the detection rates. According to the obtained results, the evolved signatures enhance the detection rate of the future malicious traces. This is based on the idea that one can evolve current signatures to predict structure of future malware traces since future malware are usually siblings or descendants of current malware and inherit some of their characteristics and structure.

The processing times of each procedure were also reported. The timing results show that signature generation process is relatively quick and finishes in couple of seconds. The experimental results with different sizes of datasets show that Signature Evolution and Signature Evaluation process roughly scale linearly with the size of the dataset.

Finally, a comparative analysis is provided to compare the proposed framework with the similar works. We described that how our proposed system is different from the others in terms of used signature generation approach and application of the genetic algorithms.
Chapter 5

Conclusions and Future Work

5.1 Conclusions

Malware detection is an active area of research in the Malware Analysis community to address the major problem of anti-malware software and systems in identifying the presence of new, never-before-seen malware before causing a major damage to the computer networks and systems. Since HTTP is the most prevalent communication protocol for many types of legitimate and malicious applications, it is necessary for malware detection systems to be able to discriminate malicious HTTP traces in order to identify HTTP-based malware samples. Early detection of new malware traces provides the security administrator with an early alert of the new types of network intrusion incidents. This helps with immediate detection and mitigation of the new malware.
This thesis presents a framework for offline analysis of malicious HTTP traffic. The proposed framework consists of three components. Malicious network traces are received by the Network Traffic Analysis component. Afterwards, the Signature Extraction component clusters malware samples based on their statistical features and generates network signatures for resembling malware traces based on structural similarities between HTTP request lines. Although these signatures are able to identify current and a portion of future malware traces, the Signature Evolution component evolves the generated signature set in order to improve its ability to identify never-before-seen malware in the future.

The idea of evolving network signatures is due to the fact that many malware authors use the same Web-based reusable platforms or kits for remote command of malware. In addition, malware developers usually reuse others code and customize it to use it in their new malware. Therefore, despite the fact that server names and addresses and functionality of these types of malware is different, they still share structural similarities in their communication phase. The proposed technique, identifies these similarities among known malware traces and generates improved signatures to detect resembling malware traces in the future.

According to the signature detection rate results provided in Chapter 4, extracted base signatures of each month are able to identify a portion of malware traces of the following months and evolved signatures are able to
detect a considerable fraction of the remaining undetected malicious HTTP requests. This early detection of new malware happens without any need of human expertise or interaction and only based on our current knowledge of already detected malware. While our proposed framework aims to generate more generic and stronger signatures to detect more malware samples, it also maintains the false positive rate as low as possible. Low false positive rate is an important factor for network intrusion detection systems since a high rate of false alarm makes the system unreliable and wastes a huge amount of expensive human and processing resources. In addition, this framework only needs network level information of a computer network comparing to system level malware detectors which require access to high level system resources of every single host in a network. Hence, generated network signatures can be easily translated into a format compatible with Snort IDS [9]. Therefore, the Signature Extraction component can be easily replaced with another signature generation engine that generates signatures in a compatible format. This is due to the fact that the Signature Evolution component works completely independent from the Signature Extraction component.

The proposed framework and algorithms were implemented and evaluated from several aspects. We evaluated the performance of our proposed framework in terms of detection ratio on future malware traces, false positive rate on the normal ISCX dataset, and processing times. The base signatures are able to detect almost 100% of the malicious traces of the current month and a
good fraction of malware of the following months, but the detection rate naturally decreases over time. The evolved signatures improve the detection rate by identifying undetected malware. Obviously, periodical signature updates help to maintain an acceptable detection ratio of never-before-seen malware. Generally, using the base and evolved signatures together increased the average detection rate of the malicious traces of the next immediate month from 38.40% by 12.48% to 50.88%. This improvement happens while the average false positive rate of the evolved signature sets is $2.7 \times 10^{-3}$. According to processing times, we see that signature generation process is relatively fast and detection time roughly scales linearly with the size of malicious dataset.

Compared to similar works, our proposed evolutionary framework offers a comparable detection rate of unknown malware traces while providing a significantly low false positive rate. In addition, none of the related works have employed genetic algorithm as the core algorithm to improve network-level signatures.

\section*{5.2 Future Work}

Some extensions to the proposed work are summarized in the following:
**Encrypted Communication Analysis**

Since our proposed system investigates HTTP headers to find structural similarities of HTTP request lines, encrypting payloads by malware do not affect our system. However, some malware employ HTTPS protocol as communication channel. The HTTPS protocol encrypts requests header as well. Therefore, the request line of the request header is not visible to a network-level monitor. Although many networks and browsers do not allow HTTPS traffic from uncertified servers many websites use self-signed public keys. Therefore, malware can use the same approach to get around the certification problem. Hence, identifying malicious self-certified servers can be an extension to this framework to add the ability of detecting malicious HTTPS traces.

**Genetic Algorithm Improvements**

In this thesis, a proof-of-concept version of proposed framework is implemented to investigate the possibility of using genetic algorithm to solve the problem of identifying never-before-seen malware traces. The genetic algorithm has many elements, operators, and functions which can significantly change the results. Since in our proposed system is not looking for a specific solution (i.e. a perfect signature set), one can improve the output results by some changes in the genetic algorithm such as proposing a better chromosome representation which better defines the problem, define different genetic
operators, and improve the fitness function in order to take more influential factors other than false positive rate into account.

**Replacing Signature Generation Component**

As was mentioned previously, the Signature Evolution component operates independent from the Signature Generation component. Hence, we can feed the genetic algorithm with network signatures generated by another string-based signature generator engine. In fact, any signature generator method can be replaces ours if it is able to provide the algorithm with network signatures in the defined format. This extension to our proposed framework can further evaluate and improve the functionality and performance of the proposed Signature Evolution engine.

**Online Traffic Analysis**

As discussed earlier, computational expenses of the signature generation and evolution linearly grows by increase in size of the dataset. Therefore, implementing an optimized online version of the framework is another extension of the system left to be done later. The online version can be tested and tuned in a live environment. Since the generated signatures are easily convertible to Snort rules, a Snort IDS can be used on the edge of a network to match signatures over the real world network traffic. We can further implement a feedback loop for tuning our signature generator engine.
References


[61] Mohammad Saniee Abadeh, Jafar Habibi, Zeynab Barzegar, and Muna Sergi, *A parallel genetic local search algorithm for intrusion detection in*


[67] Taeshik Shon, Jungtaek Seo, and Jongsub Moon, SVM approach with a genetic algorithm for network intrusion detection, Computer and In-

[68] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage, 


Vita

Amir Pourafshar

Bachelor of Computer Engineering
Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
2007-2011
Major: Software Engineering

Master of Computer Science
University of New Brunswick, Fredericton, NB, Canada
2011-2014