Domain Generation Algorithm (DGA) Detection

by

Shubhangi Upadhyay

Bachelors of Computer Science, UPTU, 2016

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

Masters of Computer Science

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s):	Ali Ghorbani, Ph.D, Faculty of Computer Science
Examining Board:	Suprio Ray, Ph.D, Faculty of Computer Science, Chair
	Arash Habibi, Ph.D, Faculty of Computer Science, Chair
	Donglei Du, PhD, Faculty of Management, UNB

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2020

Abstract

Domain name plays a crucial role today, as it is designed for humans to refer the access point they need and there are certain characteristics that every domain name has which justifies their existence. A technique was developed to algorithmically generate domain names with the idea to solve the problem of designing domain names manually. DGAs are immune to static prevention methods like blacklisting and sinkholing. Attackers deploy highly sophisticated tactics to compromise end-user systems to gain control as a target for malware to spread. There have been multiple attempts made using lexical feature analysis, domain query responses by blacklist or sinkholing, and some of these techniques have been really efficient as well. In this research the idea to design a framework to detect DGAs even in real network traffic, using features studied from legitimate domain names in static and real traffic, by considering feature extraction as the key of the framework we propose. The detection process consists of detection, prediction and classification attaining a maximum accuracy of 99% even without using neural networks or deep learning techniques.

Dedication

I would like to dedicate my work to my family and their support that they have showed all across my masters. Also I would like to dedicate my thesis work to all my colleagues at CIC lab.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Dr. Ali Ghorbani, for his guidance, encouragement and constant support through this thesis. I also acknowledge the help and support of my friends and colleagues at the Canadian Institute for CyberSecurity, who always helped me keep my spirits high during hard times.

Table of Contents

Abstract ii

Dedication iii

Acknowledgments iv

Table of Contents v

List of Tables viii

List of Figures ix

Abbreviations x

1 Introduction 1

1.1 Botnets 5

1.1.1 Evolution of Botnet 8

1.2 Summary of Contribution 10

1.3 Thesis Organization 11

2 Literature Review 13

2.1 Related work 13

2.2 DGA Family Examples 18

2.2.1 DGA generated Domain Names 19

2.2.2 Taxonomy of DGA Families 19

2.2.2.1 Taxonomy based on topology . . . 19
2.2.2.2 Taxonomy based on algorithm . . . 20
2.3 Kraken . . . 21
2.3.1 Pseudo Random Number Generator . . . 21
2.3.2 Seeding . . . 21
2.3.3 Domain Names . . . 22
2.4 Conflicker . . . 22
2.5 Torpig . . . 23
2.6 Suppobox . . . 24
2.7 Concluding Remarks . . . 25
2.7.1 DGA Life-Cycle . . . 26
2.7.2 Control and Application . . . 26
2.7.3 Spreading and Injection . . . 27
2.8 Problems with DGA Detection . . . 28
2.8.1 Requirements for DGA Detecton . . . 28

3 Proposed Framework **30**
3.1 Overview . . . 30
3.1.1 Data Gathering . . . 31
3.1.1.1 Considerations . . . 32
3.1.2 Feature Extraction . . . 34
3.1.3 Analysis . . . 36
3.2 Algorithms and Equations . . . 38
3.2.1 Lexical Features . . . 38
3.2.2 Web Scrapping . . . 41

4 Implementation **45**
4.1 Libraries and configuration . . . 45

4.2 Model View 46
4.3 Module View 48
4.3.1 Module 1: Data Gathering 48
4.3.2 Module 2: Feature Extraction 50
4.3.3 Module 3: Analysis 52
4.4 Behavioral View 53
4.4.1 Pre-Processing Dataset 54
4.5 Concluding Remark 58

5 Experiments and Results **59**
5.1 Dataset and labeling 59
5.1.1 Dataset Sampling 60
5.1.2 Processing Dataset: Detection 60
5.2 Model Selection 61
5.3 Results 61
5.3.1 Detection 62
5.3.2 Prediction 63
5.3.3 Clustering 65
5.4 Concluding Remarks 67

6 Conclusion and Future Work **69**
6.1 Conclusion 69
6.1.1 Importance of Feature Extraction 70
6.2 Future Work 71
6.2.1 Types of future work 71

Bibliography **79**

Vita

List of Tables

2.1 Taxonomy of DGA Family based on Algorithm Type 21

3.1 Dataset Class 33

3.2 Algorithmically Generated Domain Names 34

3.3 List of Distinctive features 36

3.4 Linguistic Feature Formulas 39

4.1 List of Python Library in Feature Extraction Phase 47

4.2 List of Python Library for Analysis Phase 48

5.1 Description of Dataset 60

5.2 Detection Results 63

5.3 Prediction Results 66

List of Figures

2.1 DGA life cycle 26

3.1 DGA Detection Framework Overview 31

4.1 Local Configuration 46
4.2 Data Flow Diagram of Model 49
4.3 Data Gathering 50
4.4 Feature Extraction 51
4.5 Analysis 52
4.6 Behavioral View of Framework 54
4.7 Replacing Missing Value by Most Frequent Value 56
4.8 Categorical Transformation to Numeric Transformation 58

5.1 Feature extraction by Gini Index 64
5.2 Feature extraction by Entropy 65
5.3 Number of clusters by elbow method 66
5.4 DBSCAN Clustering Algorithm 67
5.5 Datset Extension 67

Nomenclature or Abbreviations

DNS	Domain Name Service
DGAs	Domain Generation Algorithms
PRNG	Pseudo Random Generator
AGDs	Algorithmic Generated Domains
TLDs	Top-Level Domains
FQDN	Fully Qualified Domain Name
MLR	Multiple Linear Regression
DFD	Data Flow Diagram
PCA	Principal Component Analysis
SDLC	Software Development Life Cycle

Chapter 1

Introduction

It is commonly known that malware try to compromise networks by establishing communication with the command and control center(C2C). On establishing this channel malware can proceed to steal information from the infected computer or server[52]. Over time this has become one of the main platforms for cyber criminals to execute spams, steal data, host phishing websites etc.[32]. Domain generation algorithms (DGA) are algorithms seen in various families of malware that are used to periodically generate a large number of domain names that can be used as rendezvous points with their command and control servers. The large number of potential rendezvous points makes it difficult for law enforcement to effectively shut down botnets, since infected computers will attempt to contact some of these domain names every day to receive updates or commands. The use of public-key cryptography in malware code makes it unfeasible for law enforcement and other actors to mimic commands from the malware controllers as some worms will automatically reject any updates not signed by the malware controllers. The Domain generating algorithm (DGA) technique is in use because malware that depends on a fixed domain or IP address is quickly blocked, which then hinders attack operations[56]. So, rather than bringing out a new version of the malware or setting everything up again at a new server,

the malware switches to a new domain at regular intervals to undergo the desired process undetected. Domain Generating Algorithm is the best technique in use by cybercriminals to prevent their servers from being blacklisted or taken down. The algorithm produces random domain names. The idea is that two machines using the same algorithm will contact the same domain at a given time, so they will be able to exchange information or fetch instructions. DGAs provide a method for controllers of botnets to dynamically produce a large number of random domain names and select a small subset for actual command and control use. This approach makes blacklisting ineffective. Being able to detect algorithmically generated domains automatically becomes, thus, a vital problem.

It is common for botnets to use domain fluxing in order to circumvent domain blacklisting[44]. In domain fluxing, an infected machine periodically uses a seed and a domain-generation algorithm to automatically generate a large set of domain names, and performs a Domain Name Service (DNS) query to discover the domains' IP addresses [12]. The seed may be time-independent, or may depend on the current time or other time-varying public information such as top news feeds [6]. There are different techniques by which DGA are generated and used, as there is an algorithm behind it, therefore the algorithm requires a seed as an input to generate domain names based on the seed. Therefore different families of malware generate variety of DGAs depending on the seed. The most frequently used DGAs are time-dependent or random character based, but in recent times it has been observed that there is an advancement in how adversaries are generating domain names, which includes English language based DGAs, Hashed DGAs, or Phishing DGAs (similar to legitimate ones)[14],[13],[22],[17].

The trends in an algorithmic name generation have been evident by many malware families like Conficker, Suppobox and Kraken (generates English domain names), Cycbot, Murofet which rely on DNS service and are very famous for generating a

extensive list of unique domain names. The general methodology of any DGA is using a deterministic pseudo-random generator (PRNG) to generate a list of candidate domain names. The seed of a PRNG can be the current date, some magic numbers, an exchange rate, etc. This random generator can be a single uniform distribution generator, e.g. use a combination of bitshift, xor, divide, multiply, and modulo operations to generate a string sequence as the domain name (such as in Conficker, Ramnit, and others); it can also be a rule generator, which selects from some knowledge base (such as in Suppobox). There are many bots that use P2P protocols to remove single point failure, therefore DGAs are still alive and in wide use. There are the DGAs that generate typosquat names of the famous and frequently used domain names i.e changing the domain name by adding, deleting a bit or two still making it look similar to the legitimate domain names, also DGAs are not limited to a specific type, since they are generated algorithmically there can be a number of ways to seed them that include hashed domain names, time-dependent, English, typosuqat domain names. There are always the way to craft more advanced version of the DGAs that are used across without being detected. Therefore, the need to develop a system which can prove trustworthy in terms of detection and prediction of DGAs under the set of any malware family.

Therefore the need to perform a comprehensive measurement study of the DGA landscape is necessary which could analyse DGA-based malware families and variants and also present a taxonomy for DGAs and use it to characterize and compare the properties of the studied families. One way can be by reimplementing the algorithms, pre-compute all possible domains they generate, covering the majority of known and active DGAs. Then, by study the registration status of over 18 million DGA domains and show that corresponding malware families and related campaigns can be reliably identified by pre-computing future DGA domains. Also this approach will give insights into botmasters' strategies regarding domain registration

and identify several pitfalls in previous takedown efforts of DGA-based botnets. To the challenging nature of the problem of detecting Dictionary-AGDs based solely on the domain name string itself, one often resorts to other information such as the IP address of the source [24], or information about the time when the domain was sent to the DNS server . This kind of information can be expensive to acquire, or due to privacy concerns, it might just not be available. Moreover, detecting an AGD based solely on the domain allows for inline, real-time blocking of such domain at the DNS server level – a highly desirable feature. Another existing approach to detect Dictionary-AGDs is to reverse engineer the malware, extracting the list of words in the dictionary and using this list to identify domains that are generated by the malware. This process is labor-intensive and time-consuming, making it unsuitable to detect new Dictionary-based DGA malware as soon as it emerges.

Little or no attention has been given in the literature to the problem that we address in this paper: detecting Dictionary-based DGAs purely based on the domain name string, and without reverse engineering the malware. A notable recent exception is the work by Lison and Mavroeidis [28] who constructed a deep learning-based DGA detection model that can detect Dictionary-AGDs generated from a "familiar" dictionary. Familiar in this context means that a large number of Dictionary-AGDs stemming from the same dictionary are assumed to be available as examples to train the model. Once trained, the model can detect previously unseen Dictionary-AGDs provided that they originate from a dictionary that has already been seen during training time. In practice, it is natural for hackers to change the dictionary in a Dictionary-based DGA, leaving the problem of detecting Dictionary-AGDs largely unresolved.

1.1 Botnets

As the fundamental characteristics of a botnet are shared with other pieces of malware such as worms and trojan horses, it is worth mentioning the original famous piece of malware named the Morris Worm. Created by university student Robert Morris, and let loose on the early Internet in November, 1988, the Morris Worm was reported to have infected 6,000 UNIX machines, which accounted for roughly ten percent of the Internet at that time.

Originally designed to target and exploit insecure UNIX machines to spread awareness of the dangers of lack security, it had the unintended side effect of infecting machines multiple times and slowing them down enough to be practically unusable until disinfected. The spread of the worm caused monetary losses guessed to be at least $100,000 as machines and networks were taken offline to be fixed, as well as prompting the creation of the Computer Emergency Response Team (CERT) Coordination Center, which is dedicated to research and counter cyber threats to the Internet. It wasn't until a decade later that the first pieces of malware that used IRC channel 3 communication for commands began to become widespread. Sub7 is a famous example of a trojan virus that used IRC to communicate to the attacker. Spread through infected software, Sub7 would install a backdoor into the target computer once run that opened an IRC channel to the attacker to notify them of the target infection. From there, the hacker could access files, steal credentials, add additional malicious code, or whatever else the attacker wished to do with the privilege they gained on the machine. Pretty Park is another example of malware from that time that used IRC for commands. As a worm, the virus would self-propagate via email by attaching a malicious .exe file to the body of the mail it sent. Once a user would execute the file, it would install itself on the host system, hide itself by modifying registry files and hiding its process from the task list, install a backdoor, and run two internet applications. The first application would be to attempt to connect

to an IRC channel that would allow the malicious actor to monitor infected devices. The second application scanned email address books on the victim, and sent further infected emails to anyone on the list to propagate itself further. These two examples of early malware would be precursors to more advanced botnet technology.

One of the first botnets that became publicly noticed was a botnet used to send phishing emails to users of Earthlink, an Internet service provider that also offers services such as email and web hosting. The author of the virus stole an estimated $3 million from stolen credit cards and other financial information, and managed to send 1.25 billion phishing emails within a year, which accounted for over 25% of the total emails worldwide sent by 2001. The financial impact to the company was estimated to be around $4.1 million. This showed the potential gains for hackers that successfully exploited botnets, as well as the damage they could do to companies and individuals.

By the mid 2000s botnets began to grow more advanced in multiple aspects, such as for developing peer to peer (P2P) communication abilities and advanced stealth techniques, as well as widening their scope of attack. The Zeus botnet is a prime example of these advancements. First detected in July 2007 and spreading quickly from there, the Zeus botnet was mainly used to steal credentials and financial information, but also has the capability to run arbitrary code on infected machine developed its capabilities. To date, Zeus and its derivatives have caused estimated losses in the hundreds of millions for businesses across the world. The mid 2000s also saw a rise in a number of botnets dedicated to botnets spread through email spam. The Grum, Storm, and Cutwail botnets alone made up a majority of spam emails sent from 2007-2010.

Researchers also saw new techniques being used in botnets like Storm, such as the central CC server being replaced by a P2P model, and remote control servers being hidden by ever changing IP addresses, making tracking and dismantling the botnet

much more difficult. As botnets began to grow in size, they also began to see use in Distributed denial-of-service attacks(DDOS), meant to limit or stop access to web services by overwhelming the target with requests, rendering them incapable of actual communication. With the rise of online advertising in the past decade, with 2019 looking to have spending on digital advertising surpassing that of traditional methods [47], hackers began to use botnets to commit ad fraud. Ad fraud is when either human or non-human actors click on or otherwise interact with advertisements in order to gain malicious actors money from advertisers without actually intending to purchase or genuinely interact with the advertised business or product. Advanced botnets such as Methbot have developed techniques to mimic human interaction with advertisements to avoid detection from ad providers, and 5 are estimated to cause losses to advertisers at a minimum of $3 million dollars a day just from Methbot alone [37]. With the continual rise of online advertising revenue, it is likely that botnets will continue to take advantage of this trend. Other botnets have begun using victims resources in order to mine cryptocurrency. The Smominru botnet is estimated to make thousands of dollars a week from using bots they have accumulated to mine various cryptocurrencies.

With the rise of the Internet of Things (IoT) in recent years, the number of devices connected to the Internet has skyrocketed, with an estimated addition of 1.1 billion new devices in 2018 alone [30]. Many of these devices are inherently insecure, with default credentials or unpatchable vulnerabilities, making them easy targets for botnets. The Mirai botnet that was behind the massive attack on Dyn in 2016 was a botnet built with IoT devices, and is estimated to have had 600,000 bots at its peak, and was able to DDOS Dyn with a peak rate of 1.2 Tbps.

1.1.1 Evolution of Botnet

To receive their task, botnets connect to a command and control server (CC). A basic problem in the deployment of botnets that malware authors (botmasters) face is that they need to find a way so that the distributed clients can find and connect to the CC server, and although the address is known to anyone in possession of the malware the channel should not be easily disrupted by network owners or law enforcement. This is for example the case with the most simple way of linking clients to the CC server by means of a static and hard-coded IP address, which could trivially be blacklisted in firewalls or taken out of circulation by a hosting provider on request from the authorities.

In order to be able to dynamically update the location of the CC infrastructure, botmasters came up with multiple approaches to dynamically locate the command and control server. One observed approach uses publicly available websites to distribute information about commands to be executed, or ways to find the actual CC infrastructure. As it is to be assumed, that these public websites will not be taken down by authorities, botmasters only have to rely on a lack of moderation by those sites in order to stay active for long periods. As soon as the webmaster of a used website notices these malicious access patterns, these botnets can be efficiently sink holed. Another common approach to dynamically control infected hosts in a botnet is to create a peer-to-peer (P2P) network with no clear central authority. While these kind of networks are less effective for a botmaster, than a setup with a clear reachable central authority, as using a P2P network may result in significant delays for finding other infected peers and distributing commands to all nodes in the network, these type of botnets can often be taken down by inserting a majority of simulated peers, also called a "Sybil attack"[21], which will simply not relay commands [35, 49].

Unlike taking down a central botnet authority, this does however require substantial amounts of work by researchers, instead of relying on network operators or law

enforcement to blacklist a single address on the internet. In order to prevent being susceptible to take downs by such a Sybil attack, some botnets have moved to use the Bitcoin blockchain [42] as command and control data storage. Due to the size of computing power of this network, it is infeasible to apply a Sybil attack on this network. A minor drawback of this approach for botmasters is, that any command sent to nodes costs money, and researchers can easily monitor all commands issued without running the malware itself, as the data of the blockchain is publicly available. A final method, which this thesis will discuss in depth, are botnets finding the control server based on domain names. Although a domain can also be seized by law enforcement, the process takes significantly longer, as it may require cross-border actions. To further limit this angle, botnet owners are also not relying on a single static domain name in their client software, which after being seized would leave the entire system inoperable, but rather dynamically generate domain names on the fly. The domains at which the CC server is contacted are computed using a domain-generation algorithm (DGA) based on the current time or some other information publicly available across all hosts, each only valid for a short amount of time. In addition to the short validity, DGAs frequently generate also hundreds of candidate domains per time interval. Infected hosts look up all candidate values to find the domain that was actually registered by the botnet owner, which from a defense perspective makes this mechanism very difficult and costly to suppress, as registering, seizing or sink holing thousands of domain names per day is often too administratively complex, costly or not scalable.

Current DGAs generate domain names by concatenating random letters and morphemes. This has the clear advantage that the resulting domain names are with very high likelihood available, but has the drawback that they leave in the network a characteristic trace of a series of unsuccessful look ups (NXDomains) for domain names, each with a high entropy name such as yfewtvnpdk.info or rwyoehbkhdhb.info that

is unlikely requested by normal users. A variation to this, applies this same technique of generating domain names, to generating user names on websites that allow publishing arbitrary data (e.g. twitter.com and github.com). In practice, malware employing this technique has not been observed, yet.

1.2 Summary of Contribution

The contribution of this research to the develop a new framework to detect, predict and cluster domain generating algorithms (DGAs). The main contributions of this research can be summarized as follows:

- Analyzed the DGAs of 43 malware family and variants. Using multiple identified seeds, we enumerate all possible domains generated by those algorithms, covering the majority of known and active DGAs.
- Identified new features to detect all the DGAs including English, time-dependent, typosquat, hashed etc. domain names.
- Investigated the performance of the developed and in use methods to detect and predict DGAs and proved the new framework developed in this research is better than the existing solutions.
- Considering the malware family that produce DGAs which are English word or those which are typosquat to be detected is also the biggest asset to the research.
- Released a fully functional framework that could detect the DGAs of any malware family type, also with the satisfying test results on static and real network traffic, by performing test on large diverse dataset.
- Compared the result of the new framework under differently extracted dataset with the same feature and process providing differential results, that makes the

framework trustable source by achieving the accuracy of 99% and false positive upto 0.0002% .

- The full fledged dataset is generated that can be used for future research and will also provide a web service to check domains for potential DGA identity.

1.3 Thesis Organization

The remainder of this document is organized in the following fashion:

- Chapter 2 Literature review, is the introduction of the work done in this field until now, and also this section is explaining the relevant few examples of the DGA families that are been famous and are still changing forms to exploit the security realm. Also this chapter gives the detail on how DGAs type and the way it is generated.

- Chapter 3 is about explaining the proposed framework while giving a brief on each module of the framework. Also this chapter comprises of the section which gives details on all the algorithms and equations used in the research.

- Chapter 4 gives details on implementation techniques, the configurations and library used in developing the proposed framework also it describe model view of the framework which is described using data flow diagram. In module view section of this chapter describes each module in detail and also it's sub components. In the last section it unravel the complexities of the whole model in behavioral view section.

- Chapter 5 is about the experiments and results that validated the proposed framework for DGAs analysis in terms if detection prediction and classification. Also this chapter tells the classification of the dataset and how it is formed.

- Chapter 6 is the last chapter and is about conclusion and future work.

Chapter 2

Literature Review

The domains being queried in a network can be acquired vialogs of Domain Name Service(DNS) servers or a monitoring server. One can build a system to analyze each queried domain and give a verdict, either malicious or benign, that guides further actions such as communication block or remediation. There have been multiple work done in this field by using approaches that involve machine learning, rule based learning, deep learning and even reverse engineering. DGA is one of the most effective and most popular tools in the attackers' toolbox. It is being used by a variety of malware families to hide the location of their CC servers, and by that maintain the robustness of the botnet. At the same time, DGAs leave a substantial footprint in the DNS traffic.

2.1 Related work

Blacklists were one of the first actions taken by the security community to address the problem of malicious domains. These continuously updated lists serve as databases for known malicious entities. One of the main advantages of black- lists is that they provide the benefit of lookup efficiency and precision. However, after the deployment of DGAs by recent malware, domain blacklisting became an ineffective technique for

disrupting communications between infected machines and CC centers. In the last few years, a combination of network and lexical features has been used to detect DGA-based domain names. The Domain Name Server (DNS) protocol [33] is responsible for locating Internet domain names and translating them into Internet IP addresses. The DNS protocol specfies the structure of how request and response messages are built and transferred, including fields with information other than the requested domain name itself, such as the message type, destination IP addresses, or Time-To-Live (TTL), location amongst others as well. Albeit DGAs with full P2P, C2C topology are on the rise, DNS is still abused in multiple ways by cybercriminals. The idea behind the techniques of using IP address, Whois information and lexical features has been used by many researchers [31].

In 2011, Bilge et al. published EXPOSURE [6], later improved [7]. EXPOSURE is a detection system of malicious domains that employs passive DNS analysis techniques based on 15 composed features, organized in 4 groups: time-based features, DNS answer-based features, TTL-based features, and domain name-based features. Most of these features must be extracted from real-time (or recently captured) DNS traffic, since they rely on volatile data that may change over time. These features were selected after a manual analysis process, targeting well-known malware behaviors. In terms of lexical features, they focused on two values: the percentage of numerical characters and the percentage of the longest meaningful substring length. These features were selected in order to detect DGA-based domain names. However, the approach lacked a full lexical analysis for domain names. EXPOSURE achieved 98.4% successful detection with around 1% of false positives, using a J48 decision tree (Witten, Frank, Hall, 2011). [55] detected domain fluxes using Kullback-Leibler, edit distance and Jaccard index metrics. Since Yadav's method has the best detection accuracy among published results, we use Yadav's detection techniques to evaluate our DGAs. Antonakakis et al. introduced Pleiades [4]. Their approach focused

on failed DNS requests, since a DGA generates hundreds to thousands of domain names, but only a few of them are successfully resolved. Pleiades clusters those failed domains with lists of previously known DGA-based and legitimate domains.

To distinguish between benign and DGA-generated names, researchers have considered multiple theories, including classification of different features.Raghuram et al.[41] built a probability model of white listed domain names and used it to distinguish between benign and algorithmically generated domain names (AGDs).Schiavoni et al. [45] developed a series of unsupervised classifiers to identify AGDs and their domain generation algorithms (DGAs). Wei-wei and Qian [54] detected AGDs by analyzing morphemes in the strings of domain names. The idea was that human-generated domains and AGDs would have different features. Barabosch et al. [5] classified DGAs based on time dependency and causality. They automatically extract DGAs from malware binaries. Crawford and Aycock [14] proposed a DGA called Kwyjibo which generates pronounceable domain names using a Markov process. Gasser [17] proposed a random word generator for pronounceable passwords. They used units that consist of one or two letters and rules defining the unit's usage to create pronounceable syllables and then concatenated generated syllables to form a word. Yadav et al.

Schiavoni et al. proposed Phoenix in Schiavoni, Maggi, Cavallaro, and Zanero (2014)[46], a layered system that detects DGA-based domain names and classifies them based on the IP addresses resolved. Their approach relies on two features widely used in linguistics: meaningful characters ratio and N-gram normality score. These features provide a measurement of meaning and pronounceability, as a means to detect randomly generated strings. They used a well-known list of non-malicious domain names (namely, Alexa's list) to calculate the probabilistic distribution for legitimate domains based on those linguistic features. The Mahalanobis distance and an empirical threshold were finally used to classify a suspicious domain name as

a DGA-based domain name. Later on, they clustered those malicious domain names together with well-known DGA-based domains to classify them as belonging to a specific malware family.

Finally, da Luz in 2014 [15] used a combination of 18 lexical features (such as N-gram statistics, Shannon entropy, length, number of vowels per consonant, among others) and 17 network features (such as TTL statistics and number of IP subnetworks, to name a couple) extracted from a (passive) DNS system. His approach is based on some of the lexical features used by Antonakakis et al.[4], although most of them were simplified to reduce dimensionality and to improve speed. These features were used with two different datasets and three different machine learning models (namely, k-Nearest Neighbors, Decision Trees, and Random Forests). The best results were achieved with Random Forests.With the intention of building a simple DGA classifier based on domain names only, Mowbray and Hagen [14] proposed a DGA detection classifier based solely on URL length distributions. The approach allows the detection of a DGA at the end of the first time slot during which the first infected machine is used for malicious queries. However, their approach is effective for only a limited set of DGA families. All methods described above rely on the extraction of predefined, human engineered lexical features from the domain name string. Recently, several works have proposed DGA detection models based on deep learning techniques that learn features automatically, thereby offering the potential to bypass the human effort of feature engineering [10,16,20,22,23]. Deep learning approaches for DGA detection based on convolutional neural networks (CNNs) and long short term memory networks (LSTM) achieve a predictive performance that is at par with or better than the methods based on human defined lexical features, provided that enough training data is available. Note that although all the previous works produce very good results (in terms of detection accuracy), they rely strongly on volatile information that may change over time. For instance, IP addresses in a

DNS response can differ at any given moment from those obtained with the same requests a few weeks earlier. In such cases, it is very unlikely that experiments can be reproduced obtaining similar results since the datasets are not captured at the same time interval. Therefore, it is difficult to train a machine learning model with algorithmically generated domain names that are inactive today or that cannot be artificially generated, since real DNS responses cannot be obtained.

All methods described above rely on the extraction of predefined, human engineered lexical features from the domain name string. Recently, several works have proposed DGA detection models based on deep learning techniques that learn features automatically, thereby offering the potential to bypass the human effort of feature engineering [28]. Deep learning approaches for DGA detection based on convolutional neural networks (CNNs) and long short term memory networks (LSTM) achieve a predictive performance that is at par with or better than the methods based on human defined lexical features, provided that enough training data is available.

The DGA detection methods that we have described so far in this section all use the domain name string itself, sometimes combined with some side information like IP-based features. All these methods have been proposed and studied in the context of traditional DGA detection. Traditional DGAs produce domain names that appear more random looking than usual benign domain names, even to human observers. This substantial difference between the domain name strings created by traditional DGAs vs. those created by humans is the underlying reason for the success of the DGA detection methods described above. A newer generation of DGA algorithms, the so-called Dictionary-based DGAs, attempt to evade the traditional DGA detection methods by producing domain names that look more similar to the ones created by humans. To this end, they concatenate words from a dictionary.

Since catching Dictionary-AGDs based on the domain name string itself is challenging, it is natural to look at side information instead. An interesting approach in this

regard is the work of Krishnan et al.[24] who followed the insight of Antonakakis et al.[4] that infected machines tend to generate DNS queries that result in non-existent (NX) responses. Krishnan et al.[24] applied sequential hypothesis tests and focused on NX traffic patterns of individual hosts to identify infected machines. More recently, Abbink and Doerr [3] proposed to detect DGAs based on sudden rises and declines of popularity of domain names in large networks.

Regarding the development of a classifier that can label a given domain name in real time as benign or malicious, solely based on the domain name string itself, there has been some initial success with deep learning approaches for catching Dictionary-AGDs [28].

To overcome these issues, in this paper we propose a detection technique based on a purely lexical analysis of the domain names. Domain names, unlike other fields within DNS packets, never change over time, and thus they are suitable for training a machine learning model with DGA-based domain names that are currently inactive. Furthermore, our approach allows other researchers to reproduce our work, verify our results, and use them as a baseline for future research. Thus, our work seeks to maintain the good results from previous works, but with a more restricted set of characteristics.

2.2 DGA Family Examples

This chapter talks about the taxonomy of DGA families. It also briefly describes the domain generation algorithms used in the four popular malware families, i.e Kraken, Conflicker, Torpig and Suppobox.

2.2.1 DGA generated Domain Names

A malware with DGA algorithmically generates a large number of possible botnet server domain names. This technique makes it difficult to retrieve botnet command and control (C2) information from malware binary by reverse engineering. The large number of potential domain names also makes it difficult for law enforcement agencies (LEA) to identify and neutralize them. The malware tries to connect to the generated domain names and eventually finds an active one to receive malware updates and commands.

2.2.2 Taxonomy of DGA Families

In this section we discuss the taxonomy of the DGA families based on the topology of network and C2 servers and based on the DGA algorithm.

2.2.2.1 Taxonomy based on topology

Based on the topology of the network and distribution of botnet C2 servers, botnet can be classified as followings[26]:

- Centralized Topology: a small number of HTTP or IRC servers working as C2 servers.
- Decentralized Topology: peer to peer network where every bot can be a C2 server.
- Locomotive Topology: command and control entities as moving over time.

Usually, Centralized and Locomotive botnets address the C2 servers using domain name system (DNS). Centralized topology of botnet uses a small number C2 servers, generally hard-coded in the malware executable and could be discovered by analysis of the malware executable which can lead to neutralization of the C2 servers of the

botnet. The bots (infected hosts) can still exist but in the absence of the C2 servers they can not be harmful.

2.2.2.2 Taxonomy based on algorithm

Botnet owners, in order to avoid hard-coding the C2 servers, use DGA algorithms to dynamically generate possible domain names for C2 servers. This technique is called domain fluxing and it increases resilience of botnet against take-downs by increasing the flexibility of the C2 server addressing layer. The domain fluxing bots periodically generate a large number of possible domain names for the target C2 servers until a correct one is found. For example, Conficker.C generates 150-200 existing domain names in the 500 DNS queries made everyday. Domain fluxing bots generate domain names using pseudo-random approach [5].

The seeding of pseudo-random generation can be time dependent or independent. Hence we can classify the DGA family into following four classes [5]:

- TID-DGA: The DGA uses a static seed for pseudo-random algorithm and generates the same set of domain names every time it is executed. Example: Kraken botnet used a TID-DGA [43].

- TDD-DGA: This class of DGAs is the time dependent and deterministic algorithm. It uses a deterministic algorithm and seed hence domain names can be easily computed for a given seed. However the seed is changed based on time.

  Example: The Conficker.c worm [25].

- TDN-DGA: The algorithm used in DGA is time dependent and non deterministic. The anticipation of seed is infeasible and hence precomputation of domain names.[49]

  Example: Torpig botnet uses popular trending topics of Twitter as part of the seed[5] for its pseudo-random name generator.

- TIN-DGA: The algorithm used for domain name generation is time independent and non deterministic. Since algorithm is non deterministic and time independent, guessing or precomputing domain names is infeasible. At the same time, due to the randomness of generated domain names, the probability of generating a correct domain name decreases drastically. [5]

Table 2.1: Taxonomy of DGA Family based on Algorithm Type

DGA Type	**Time Dependent**	**Deterministic**	**Example**
TID-DGA	no	yes	Kraken, Suppobox
TDD-DGA	yes	yes	Conficker, Torpig
TDN-DGA	yes	no	Newer Conflicker (D)
TIN-DGA	no	no	not seen

2.3 Kraken

Till April 2008, Kraken was considered the world's largest botnet with over 400,000 bots in at least 50 of Fortune 500 companies. It was primarily used for spams. Kraken uses DGA which is originally time independent i.e. a specific sample will at all times generate the same domain names. The later version of Kraken used a time dependent DGA to avoid generating same domain every time.

2.3.1 Pseudo Random Number Generator

Kraken uses a linear congruential generator (LCG) as pseudo random number generator (PRNG) in DGA.

$$r_{n+1} = 1103515245 * r_n + 12435(mod 2^{31})$$

2.3.2 Seeding

DGA uses two values to set the seed of PRNG used in algorithm:

- A running counter initialized to zero. The counter is incremented in steps of one in version-1 and one or two in version-2. The increment in version-2 depends on the outcome of the DNS response for the domain. Boolean indicating whether or not a list of hard-coded IPs could be contacted.

2.3.3 Domain Names

Kraken DGA uses PRNG to generate domain names. Calls to the PRNG determine the characters of the random domain. Random numbers generated are mapped to a-z and every character is treated equally likely in both version of the Kraken. The randomly generated domain name is appended to base domain. The based domains used in the DGA are as below:

- Version 1: Version 1 of Kraken uses free DNS providers.

  i.e. dyndns.org, yi.org, dynserv.com, mooo.com

- Version 2: Version 2 of Kraken uses top level domains.

  i.e. com, net, tv, and cc

2.4 Conflicker

Conflicker is a computer worm which targets Microsoft Windows operating system and it was fist discovered in 2008. It forms a botnet by compromising the admin account using dictionary based attacks on passwords. It infected millions of computers including government, business and home computers in over 190 countries.
Conflicker uses DGA for domain fluxing. There are five known variants of conflicker worm.

- Conflicker:A

it generates a list of 250 domain names every day over five top level domains (TLDs). It uses pseudo-random generator seeded with the current date to generated the domain names for C2 servers.

- Conflicker:B

  it generates a list of 250 domain names every day over five top level domains (TLDs). It uses pseudo-random generator seeded with the current date to generated the domain names for C2 servers.

- Conflicker:C

  Variant C creates a named pipe, over which it can push URLs for downloadable payloads to other infected hosts on a local area network.

- Conflicker:D

  Variant D creates an ad-hoc peer-to-peer network of botnet. It generates daily a pool of 50,000 short domain names with 4-9 characters across 110 TLDs, from which it randomly chooses 500 to attempt for that day.

- Conflicker:E

  Variants E of conflicker creates an ad-hoc peer-to-peer network to push and pull payloads over the wider Internet.

2.5 Torpig

Torpig[49] uses time depended and deterministic algorithm for domain name generation. It uses current date and a numerical parameter to seed the algorithm. After seeding, DGA generates a weekly domain name, i.e. dw, based on the current week and the year and is independent of the current day. The generated weekly domain

name remains same for the entire week. Now the generated domain name is appended with a number of top level domains (.com, .net and .biz). The computed domains are access in order as following: dw.com, dw.net, and dw.biz to reach the C2 servers.

If attempt to connect to C2 servers fails using all the three weekly domain names then Torpig DGA computes a domain name daily, say dd, which in addition depends on the current day. Now the new domain, dd is generated each day. The generated daily domain names are accessed in order, dd.com is tried first, with fallbacks to dd.net and dd.biz to reach the C2 servers.

If, again, the attempt to connect to C2 servers fails using all the three daily domain names, Torpig DGA tries to contact the domains hard-coded in its configuration file (e.g., rikora.com, pinakola.com, and flippibi.com). The DGA used in Torpig is deterministic; i.e., once the current date is determined, all bots generate the same list of domains, in the same order.

2.6 Suppobox

Suppobox uses a combinational approach that has used time depended and deterministic algorithm to generate different classes of DGAs including hashed (SHA and MD5), also the new version of suppobox generate English word based DGAs similar to kwyjibo [14]. Traditional DGA algorithms usually start from random seeds and produce domains that are distinctly different from common benign domains. They appear more "random looking", such as, for example, the domain sgxyfixkhuark.co.uk generated by the malware Cryptolocker. Traditional DGAs are detected with techniques that leverage the distribution of characters in the domain, either through human engineered lexical features or through training deep neural networks. Lately, a newer generation of DGA algorithms has been observed. This

new kind of DGA makes detection by the techniques mentioned above much harder, namely by producing domains that are similar to the ones created by a human. Dictionary-based DGAs generate domains by concatenating words from a dictionary. For example, the malware Suppobox [18], a known Dictionary-based DGA, produces domains such as: heavenshake.net, heavenshare.net and leadershare.net [39]

There are two types of Supoobox malware family classification available namely:

- Suppobox.A

  It uses combination of two words from the provided wordlist and generates 254 domains per day on an average. Also the tld used by this version is (net). It is a time dependent DGA.

  Example: sharmainewestbrook.net, tablethirteen.net, childrencatch.net

- Suppobox.B

  It uses combination of two words from the wordlist and generates 783 domains per day on any average, also the tld used by this version is (net, ru). And it is also a time dependent DGA.

  Example: stephaniebernadine.ru, arivenice.ru, thinkgoodbye.net

2.7 Concluding Remarks

Domain Generation Algorithms (DGAs) is a hot topic today, there have been plenty of research done on this topic as well. The life cycle of a DGA is where it is important to recognise it and stop it from growing or proceeding to the next phase of the life cycle.

2.7.1 DGA Life-Cycle

The need to detect DGAs in time is of utmost importance, DGAs follow a similar set of steps throughout their existence. The sets can be characterized as a life cycle. Figure 2.1 illustrates the generic life cycle of a DGA. Our understanding of the life cycle can improve our ability to both detect and respond to this threat by developing a well detailed taxonomy including all different instances corresponding to each phase.

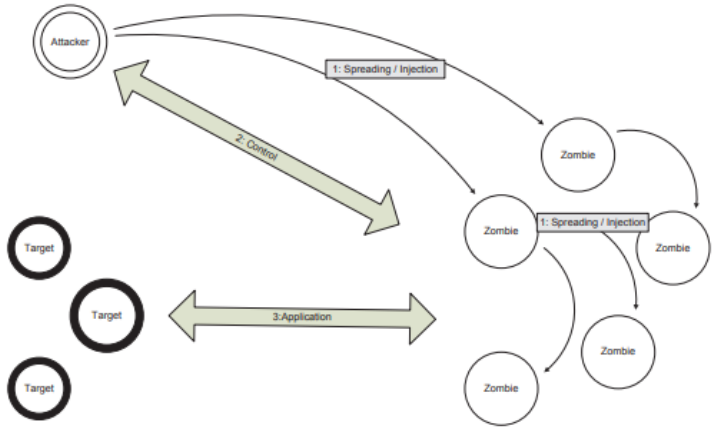

Figure 2.1: DGA life cycle

2.7.2 Control and Application

It is evident that malware try to compromise networks by establishing communication with the command and control center(C2C). Once the communication channel is established then malware can proceed to steal information from the infected computer or server [52]. It is better to detect the DGAs at the earliest stage, before establishing the communication channel. On establishing the passage for themselves the next step is to spread and infect the system to be attacked, this is preparation for attack. Over time this has become one of the main platforms for cyber criminals to execute spams, steal data, host phishing websites etc.[32]. The DGA technique

is in use because malware that depends on a fixed domain or IP address is quickly blocked, which then hinders operations[56], So, rather than bringing out a new version of the malware or setting everything up again at a new server, the malware switches to a new domain at regular intervals. Domain Generating Algorithms are in use by cybercriminals to prevent their servers from being blacklisted or taken down. The algorithm produces random looking domain names. The idea is that two machines using the same algorithm will contact the same domain at a given time, so they will be able to exchange information or fetch instructions.

2.7.3 Spreading and Injection

DGAs habitually recruits new zombie machines using similar approaches as those for other malwares. These malwares spread and inject themselves into systems in different ways, which are classified in the above methods: Distribution of malicious emails: One of the methods that DGAs use to compromise new hosts is through social engineering and distribution of malicious emails. This also includes the distribution of emails and malicious codes in social networks. According to the Sophos security report[5], malware and spam rose 70% in 2010 on social networks in comparison to 2009. In a common scenario, a DGA may distribute email messages with malware attached, or perhaps an embedded link to a malware binary located elsewhere. Social engineering techniques are used to trick computer users into executing the malware, which leads to the compromise of hosts. DGA family uses these methods to infect the machine, and subsequently propagate.
Therefore even though there have been a lot of research done in this field but there is still need to consider certain conditions which makes adversaries more craftier.

1. Now a days DGAs are not just the random number or any random string, there are certain malware families who generate English DGAs, not English but pronounceable.

2. Also the scope is not limited on producing English domains, but there are malware families which generate typosquat of the legitimate domain names available or are in the list of most frequently used.

Today considering the craft of adversary it is impossible to detect DGAs in their early stage by the means of static methods like blacklisting or sinkholing. The need of the era states that there should be a way to detect DGAs in their dynamics so as to tract them in network as efficiently as possible.

2.8 Problems with DGA Detection

From the previous section, it can be seen that current DGA detection technology makes strong assumptions about the methods of operations of DGAs. Assuming that botmasters will, in the near future, cease to use obvious randomly generated domains with a fixed periodicity, a gap of knowledge with current detection mechanisms that work with the only assumption that DGAs use the domain name system (DNS) and create new domain names for use in the botnet very often (on a daily basis) can be noticed. This thesis tries to design a novel approach to detect DGAs active in a network, and therefore fill this gap of knowledge. Apart from the assumption that future DGAs will try to circumvent current detection methods, designing a system that does this now is necessary, as there might already exist malware that is currently undetected.

2.8.1 Requirements for DGA Detecton

Before creating a new DGA detection mechanism, clear requirements for the functionality of such a system should be defined. From literature, common requirements set by network operators for detection of malicious activities on their network can be deduced. Assuming future DGAs will no longer rely on fixed periodicities of DNS

traffic or using fully randomized domain names on a per character basis, which immediately stands out to a human observer, the goal is to create a detection scheme that also follows these requirements:

- High true positive rate for detection, in order to catch all malicious activity.
- Low false positive rate, to avoid flooding network operators with non-malicious data.
- Fast detection speed, to allow network operators to quarantine infected hosts swiftly.
- Correct clustering of infected hosts, in order to give insights about subsets of the network infected with the same malware. This allows network operators to easily backtrack how malware entered a system.

During the implementation, contained in chapter 4, results will be presented in such a way that these requirements can be evaluated, in comparison to the current state-of-the-art of DGA detection.

Chapter 3

Proposed Framework

The aim of this chapter is to bring familiarity to the approach that is necessary to be considered, where the framework extract differentiating features from the static data available by studying the Alexa 1M [2] domain name dataset and using those selected features in detecting , predicting and clustering the DGA and non-DGA distinctively. Therefore the research in mainly focused on feature selection and extraction from the very basic level, using those feature sets in a detection module in which supervised learning algorithm is used i.e. random forest. To predict the non labelled dataset which for verifying the data extracted from real network traffic, decision tree C4.5 is used to extract features that are valuable to make prediction considering gini index and also entropy, then using both lists of features in multiple linear regression to predict the data. While the last section of the framework also do clustering using unsupervised machine learning algorithm i.e. Kmeans and DBScan to visually classify the result.

3.1 Overview

Figure 3.1 shows the overall architecture of the proposed framework, while it consists of three different section: Data Gathering, Feature Extraction and Analysis which

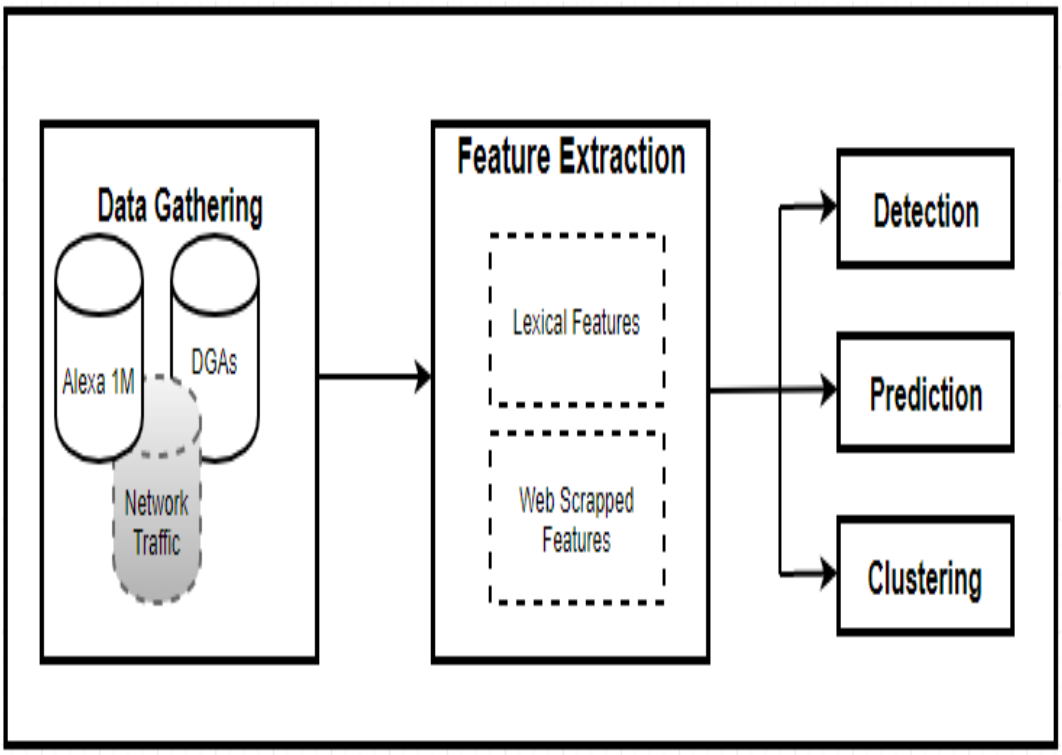

Figure 3.1: DGA Detection Framework Overview

itself consists of Detection, Prediction and Clustering.

3.1.1 Data Gathering

This is one of the basic steps to test any framework or model; therefore this makes the step vital in any research. In our scenario the lack of best practices and interest in sharing data between academia and business communities to safe guard the realm of computer security is the key fact to consider. Inspite, this circumstance we have tried not to limit this research on the static data, we have also demonstrated the result on the network traffic.

There are two aspects of our data consisting of two classes; one of them is the clean legitimate domain names and the second class is the algorithmically generated domain names. Therefore we have considered different families of malware which are famous for generating domain names on a large scale. The distinguishing factor

in gathering data here is considering the advancement of the adversary's pattern to generate and attack the network; therefore, we have considered the families of malware that produce the English domain names which are not a limit to the scope of our research. It does include a broad spectrum of English domain names based on various factors like meaningful, pronounceable, random string, looking like a clean domain name.

In this thesis, first step is to build a dataset of clean domains using the Alexa (1M) record that consists of the names of all the most visited and top websites (Domain names). The list contains 1 million data entries that are worth considering reliable and clean. As for the malicious domain names we have collected the information from various repositories provided by the trusted antivirus sources like netlab360 [1]. The dataset formation consists of collection of various forms of data present in different websites and GitHub. There are two classes of data in our dataset: Clean and DGAs. For clean dataset we are using Alexa 1M records for the domain names and for unclean data we have used multiple DGA families present in GitHub [19]. We have the dataset consisting of more than 25 families of DGAs, we aim at experimenting with different families to evaluate our framework better.

We have sampled the dataset in ascending order starting from 2k records to 1M, with the increase in the number of samples size, the accuracy of the framework also increases.

3.1.1.1 Considerations

In our methodology several assumptions have been made, which are also verified and proved by the results of our framework analysis and evaluation.

1. Now a days, Malware families like Suppobox and Kraken try to bluff by using the English DGAs; therefore, there is a need to address these DGAs in time. Here we consider a scenario to test all the domain names if they are present in

Table 3.1: Dataset Class

Clean Domain	***category***
alexa 1m	clean
DGAs	
Malware Famlies	
Banjori	random(med)
chinad	random (med)
corebot	random(small)
dircrypt	no numbers(med)
alureon	hash (med)
fobber	no numbers (small)
kraken	random (sub dom)
lockey	no number(med)
murofet	no number(med)
mydoom	no number (small)
necurs	no number (med)
newgoz	random (med)
nymaim	random/ English
padcrypt	no number (med)
pitou	no number (small)
pizd	English (small)
proslikefan	no number (small)
pushdo	no number (small)
pykspa	no number(small)
qadars	random(small)
qakbot	no number (med)
qsnatch	random (small)
ramdo	no number (med)
ramnit	no number(med)
ranbyus	no number (med)
reconyc	random (small)
shiotob	random (med)
simda	no number (small)
sisron	no number (small)
suppobox	English (med)
symmi	no number (med)
tempedreve	no number (small)
vawtrak	no number (small)

an (US/UK) English dictionary.

2. We also check if they are meaningful, pronounceable or random string on the basis of matrix consisting of [adverb, noun, pronoun, adjectives] etc.

3. Now since the adversaries have all the resources to get more and more advanced in terms of producing domain names that are similar to clean domain names list, we also consider generating the typosquat names of all the domain names that undergo feature extraction process.

4. We have also considered a checking hash as there are families of malware that generate hashed DGAs.

In our research we have tried to consider all the possible ways a DGA can generate and use malicious domain names.

Table 3.2: Algorithmically Generated Domain Names

Sample	**English Realm**		
	Meaningful	***Pronunceable***	***Random String***
Zoneeplaye.com	yes*	yes	no
huaze.org	no	yes*	no
kierdwees	no	yes	yes*

*Express 95 per cent concentration

3.1.2 Feature Extraction

To evaluate any approach we need to have a good set of features that have distinctive qualities to specify different classes in a dataset. The aim of our model to distinguish between legitimate domain names and algorithmically generated domain names. We have considered a different approach to extract features for our dataset; here we actually study and extract features from the clean domain names and use these to detect algorithmically generated domain names. Considering feature extraction as

the important aspect of every research we have used a novel approach to study our data and then processed them to extract important features that prove viable for the purpose of this research, using a clean record set provided by the Alexa website, best feature is extracted in Table 3.3.

We have basically divided our feature selection into 2 categories. Each category is different and consists of the features that proves to be important in results. We categorized the feature extraction process in 2 dimensions:

1. **Lexical Features:** For this lexical analysis an analyzer was built that could extract the basic feature from the domain name in terms of representation and demonstration.

   It is further categorized into *Linguistic features* where the features like length of the domain, subdomains+TLDs, number to alphabet ratio, special characters, hash, number of dot like features were extracted. The other feature category is *Human Engineered* where features are explicit advantage to classify if the domain name hold any credibility of personifying English in terms of being meaningful, pronounceable, random string, match ratio (percentage of the string present in clean domain list)[8].

2. **Web Scrapped Features:** This is the different level of feature extraction that makes the idea novel. Here the domain names undergo two approaches, namely *URL segmentation* and *Deep Web features*. In URL segmentation the domain name is executed as a Google search query request and the first 50 URL results are captured for the further processing of the hit counts. Hit counts is a process of extraction of the place where the domain name exists and stores it as a count. It is evident that if the domain name is present in the netloc (fully qualified domain name of url), then it has a greater possibility of being a clean domain name. In the Deep Web feature extraction process, the

program tries to find the available domain names which are registered and has the queried domain name in it and stores the counts returned; also it calculates the Shannon entropy and Lavenshtein distance array and returns the average to the feature set. Also we have designed a fuzzer for this phase where the fuzzer tries to find the typosquat domain names for each queried domain name. It is observed by the results that for each queried clean domain name there were a lot of typosquat versions available which also worked as asset to the study.

Table 3.3: List of Distinctive features

Lexical analyser
Lingustic Feature
Average Distance (Number to Alphabet)
Human engineered Features
Meaningful, pronounceability, Randomness of string
N-char, match
Web Scrapping
URL Segmentation
Top 50 results of URL (google search)
Deep Web Features
Available Domain names, Edit distance, Shannon Entropy
Registered Typosquat Domain Names

3.1.3 Analysis

This section gives a brief introduction to the final segment of the proposed framework, where the section is mainly divided into three phases and also uses three different machine learning approaches to validate the result of each phase and forms a complete analysis model. Each phase consists training model and then testing model by using simple machine learning algorithms. When considering DGA detection as a research topic, important thing to note is that there has been multiple research done in this field of study by using machine learning, deep learning and also rule-based learning approach. Therefore the framework here is mainly using the simpler machine learning

algorithm to prove the feature extraction as the main reason for the improvement in analysis step.

1. Detection Phase:

   It is at utmost importance to recognise these DGAs at the earliest phase of their life-cycle as to stop them to spread and infect the system. Therefore the importance of detection phase holds the high value itself. For this phase supervised machine learning algorithm i.e. random forest is used to detect the DGAs with maximum accuracy. It has been observed by the researches conducted on this topic has been extensive and have used deep learning techniques, where as this framework help to achieve better accuracy and lesser false positive only by using random forest machine learning technique.

2. Prediction Phase:

   Successful detection of DGAs can avoid the network attack all at once if incorporated properly. Therefore the need to address this also holds importance, to deal with this the framework uses two machine learning techniques decision tree C4.5 and multiple linear regression, where using decision tree the decision making features are extracted from the dataset and are then fed to the multiple linear regression model which later uses these features to predict on to an unknown data sample given at the time to testing.

3. Clustering Phase:

   It is the phase that completes the process of analysis, and this phase uses unsupervised machine learning model kmeans and dbscan to follow a structure of prediction by graphical representation to display the result. The framework uses kmeans elbow method on the dataset to get the number of clusters that should be formed, which is then verified by using dbscan method at the end. Broadly speaking, clustering can be divided into two subgroups :

- Hard Clustering: In hard clustering, each data point either belongs to a cluster completely or not.
- Soft Clustering: In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned.

Soft clustering is used in this approach.

3.2 Algorithms and Equations

It is clear that feature extraction is probably the most important aspect of this research and there have been several programs written to extract the features out of a domain name (clean or algorithmic), which consists of a series of algorithms and equations. The research is divided into two phases where each phase itself consists of 2 more categories, making 4 sub-divisions in total. There is a road-map which covers basic features to a valuable features of both the categories and it is important to consider every aspect of the feature to detect and predict the result with maximum accuracy. Some of the features are general and used in previous research but there are a lot of novel features introduced in this proposed research, which were never considered before. In this section they will be introduced.

3.2.1 Lexical Features

is the primary aspect of any feature related to studying word or name. Therefore considering linguistic features in important. We have categorized our linguistic section into 2 categories namely *lexical and human engineered. Lexical features* are the type of feature to begin with, they are easily detectable and also easy to extract. Linguistic Features are a series of 9 features including length of domain name, number of dots, special character, top-level domains (tlds), different ratio(number to length,

alphabet to length) and hash. Here the distinctive feature is *distance between number to alphabet* as it is a new feature in to this research in the linguistic feature section. Let D be the set of all clean legitimate domain names $\{d_1$, d_2,.., $d_i\}$ from Alexa 1M record set of most visited websites and also consider r_i as the ratio of number of digits n_i to alphabet a_i along with *r(avg)* is calculated as fraction of summation of the number to alphabet ratio r_i in the total set D. Therefore that makes γ as the distance between *r(avg)* and r_i.

$$\gamma = r(avg) - r_i \tag{3.1}$$

There is the higher chance of the legitimate domain names to have the positive distance, whereas there are multiple families of malware that generates the random domain name with the mix of alphabet and number which are distant in terms of the number to alphabet ratio. This is the feature where the mean of the number to alphabet ratio is stored and for every queried domain number alphabet ratio is calculated and then the distance between them is extracted and used as a feature. Also, while considering Table 3.4 d_l is calculated by removing tlds t_i as the length of domain, in special character referred in 3.3 d_s subtraction of summation of numbers n, alphabet$a$ and dots d_o if present in the queried domain name. Also, consider the number of dots d_o as calculated by removing summation of numbers n, alphabet$a$ and special characters d_s from fully qualified domain name (FQDN).

Table 3.4: Linguistic Feature Formulas

Length of the Domain	$\mathrm{d}_l = d - t_i (3.2)$
Special character	$\mathrm{d}_s = d_l - \sum(an, d_o) (3.3)$
Number of Dots	$\mathrm{d}_o = FQDN - \sum(an, d_s) (3.4)$

Human engineered features add a different approach then the previous work done in this field, as it gives the signature of all malware families that generate English language DGAs. In 2008 Kwyjibo [14] was built that generated clean pronounceable

domain names; later this concept was used by the adversaries and exploited for their own benefit. Therefore there is a need of different solution to tackle this problem from the root. Human Engineered features gives the solution to ever existing problem that is detecting the English domain name, not English yet pronounceable domain names [34]. The features in this aspect also include *uni-gram*[10] in Algorithm 1; we have only considered uni-gram not bi-gram or tri-gram, extracted by calculating the ratio of characters present in queried domain name to Alexa record set. Here the Alexa 1M record set is used where all the records are concatenated to one string S_a and then distributed in 1,2,3 character dictionary format, where the occurrence of each character is stored next to the character itself; the concept is to check the occurrence of the queried domain in the list and use it as a feature, then percentage of the existence of that character in Alexa string is calculate as P_u.

Another feature that checks English or 'not English yet Pronounceable' or 'meaningful' is by using *wordnet.synset* in Python which checks if the word has a meaning in an English (US/UK) dictionary. The concept here is if the word has meaning then it will be an English word and also be meaningful if not then the word might be made up of a combination of meaningful words, or just be pronounceable or might not end up in any category, in which case it would be a random string. To check this we break the queried domain name to make the vector of adjectives, and based on this vector our parser decides if the word is noun, adverb, verb, pronoun etc. If the vector has a noun, that means the domain name might be meaningful; if the vector has verbs and adverbs it portrays that the word is pronounceable, and if vector space returns empty, then it is evident that the queried domain name is a random string in Algorithm 2.

Algorithm 1: Unigram Percentage Calculation

Input: S_a
Output: P_u
Result: u_n: Unichar
while *While(S_a !="")* **do**
  u_n = tokenize(S_a)
end
while u_n*!=0* **do**
  Calculate Percentage P_u
  P_u = fraction(u_n,S_a)
end

Algorithm 2: Check Domain English

Input: domain name (dom)
Output: English, Meaningful, Pronounceable, Random String
ad=[]
IF dom is in *wordnet.synsets*
return YES
ELSE
return NO
IFad[]!=0 **check:**
adverbs, nouns, pronouns,adjectives,verbs **If** ad[] has nouns,pronouns
return "Meaningful"
ELIF ad[] has verbs, adjectives
return "Pronounceable"
ELSE
return "Random String"
end

3.2.2 Web Scrapping

This approach gives the another aspect to the study, by providing different sort of features that prove essential later. The features used in web scrapping are considered unbeatable because these features give a distinct difference in DGA detection. It is a way by which extraction of the information takes place by querying a domain name from the web using Python. The idea behind this is getting hold of all the ways a domain name can be spoofed. Therefore querying the registered available domain name and also the typosqaut proved beneficial.

URL Segmentation is the process of counting of the occurrences of the domain name

Algorithm 3: Google Search Result Hit Count

Input: domain name (dom)
Output: hit counts
counters: netloc,path,query,fragment,zero
for (dom google search)
Store "50 url list"
check: dom in each URL
IF dom is in netloc
Increase netloc count by one.
ELIF dom is in path
Increase path count by one.
ELIF dom is in param or fragment
Increase fragment count by one.
ELIF dom is in query
Increase query count by one.
ELSE
Increase zero count by one.
end

in the returned URL results [23] [36]. The process works by first issuing the Google search request of the domain names as a query in algorithm 3, then our program fetches the first 50 URLs returned from the request. As the URL is basically divided into netloc, path, query, fragment and params, we also assigned a counter for each and then the queried name is checked for its presence in any of the places. If the domain is present in netloc address, then the hit count for it increases by one. Here we have also used zero counter as if the domain names do not have any hits in the parameter specified it will increase zero counter. It is evident by our research that zero counter proves to be a valuable asset to detect the DGA as the value of zero counter is highest when a DGA is queried.

Deep Web features are the extension to the work that we did in URL segmentation Algorithm 3, as the queried domain name here undergoes two different processes. As mentioned in Algorithm 5 first, the queried domain is checked if there are any registered domains that contain queried domain name as a part and return the array of all the such names. As for an example if Google is queried the array that is

Algorithm 5: Fuzzer implementation for typosquat

Input: domain name (dom)
Output: hit counts
counters: netloc,path,query,fragment,zero
foreach dom:
Generate fuzzer List
(addition, bitsquat, Homoglyph, Hyphenation, Insertion, Omission, Repetition, Replacement, Subdomain, Transposition, Vowel-swap, various)
foreach list
check: registered domains
store list of typosquat registered domains
return list
end

returned will have [google-eservice, googlepay, googlemedias]. This feature we called registered available domain name. **levenshtein distance and shannon entropy** is calculated using Python library i.e *python-levenshtein, SCipy* and the average is stored. Second is where the queried domain name undergoes a typosquat process[50]; this is the way any domain names is phished and made to look similar to the original existing domain name. Here we have considered 12 different types of typosqauts including bit-squatting, addition, omission, transposition, vowel-swap, repetition, homoglyph, replacement, subdomain, etc. and after generating these typosquats [20], the program checks all the registered domain names among them. It turns out to be the crucial thing to our research as there is a maximum number of such spoofed domain name for the clean domain list as they are most frequently visited websites, therefore phishing these domains is of greater benefit for the adversary. It is observed that DGAs would have the lesser probability to have any other phished registered domain name or available registered domain name. With this vision we conclude our data set with numerous novel features that have been extracted and used in our research.

It is evident that our approach of selecting the features of demonstrating the detection phase is different and has never been considered before. Therefore, to our

expectation the results were also not surprising to us. We have dedicated almost 85 per cent of the work to the feature extraction phase itself.

Chapter 4

Implementation

This chapter provides an architectural overview of the proposed system and also presents a number of different architectural views to depict different aspect of the system. To provide the overall understanding of the system we document three different views of the system: module view and behavioural view. Module view shows how the system is structured as a set of implementation units, and behavioural view shows how the modules interact together and then finally at the end there is a model view that depicts the overall system and how it works. The DFD diagrams are employed to illustrate different view of the system. The use of different diagrams depicts the system module view and the behavioural view of the system.

4.1 Libraries and configuration

The framework implementation of the DGA Detection is mostly done using Python scripts. It uses different libraries provided by the python organization for the feature extract and analysis phase, Apktool [29] to decompress and decode APKs, Natural Language Toolkit (NLTK) [30] and regular expression (re) to extract linguistic features. While extraction of deep web feature require different library from the basics which includes googlesearch, urllib, requests, dns resolver, dns exception and other

shown in Table (4.1 , 4.2). While in analysis phase different machine learning tasks including preprocessing, dimensionality reduction, training and testing phases are carried out through Scikit-learn [31] and tensorow [32] libraries. All experiments are conducted on a PC test machine equipped with Intel Core i7 Quad Core with 2.67 GHz processors and 8 GB memory running the Windows 10 (64 bit) operating system. Also virtual machine Windows 10 of same configuration was required to deploy feature extraction phase to generate and clean large dataset. The configuration path to python is shown in Figure 4.1.

```
[Paths]
home_dir: /Users
library_dir: /Library
system_dir: /System
macports_dir: /opt/local

[Frameworks]
Python: 3.2
path: ${Common:system_dir}/Library/Frameworks/

[Arthur]
nickname: Shubhangi
last_name: Upadhyay
my_dir: ${Common:home_dir}/thesis
python_dir: ${Frameworks:path}/Python/Versions/${Frameworks:Python}
```

Figure 4.1: Local Configuration

4.2 Model View

The model of the framework is simple and easy to understand, the framework is basically divided into three sections and each section in itself is sub-divided into subsections where each subsection is specifying its own purpose.

The data flow diagram in Figure 4.2 depicts the model view from basic. The first entity in the figure shows the data gathering phase where the data is collected from different websites and also the real network traffic is gathered. Feature extraction is mainly the key of this research and therefore consists of a number of processes for extracting features from the raw data and constructing a dataset for analysis phase.

Table 4.1: List of Python Library in Feature Extraction Phase

Library	**Purpose**	**Module**
NLTK wordnet	WordNet is a lexical database for the English language, which was created by Princeton, and is part of the NLTK corpus. WordNet alongside the NLTK module to find the meanings of words, synonyms, antonyms, and more.	Linguistic
re	This library provides regular expression matching operations similar to those found in Perl.	Linguistic
google-search	Run a Google search and fetch the individual results (full HTML and text contents). By default the result URLs are fetched eagerly when the search request is made with 10 parallel requests.	Web-Scrapping
urllib	urllib is a Python module that can be used for opening URLs. It defines functions and classes to help in URL actions. With Python you can also access and retrieve data from the internet like XML, HTML, JSON, etc. You can also use Python to work with this data directly	Web-Scrapping
requests	Requests allows you to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs, or to form-encode your PUT & POST data	Web-Scrapping
Scipy	SciPy, a scientific library for Python is an open source, BSD-licensed library for mathe -matics, science and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that, it should work with NumPy arrays.	Linguistic, Web-Scrapping
dns resolver	dnspython is a DNS toolkit for Python. It supports almost all record types. It can be used for queries, zone transfers, and dynamic updates. It supports TSIG authenticated messages and EDNS0	Web-Scrapping
time	time module provides a function for getting local time from the number of seconds elapsed since the epoch called localtime()	Linguistic, Web-Scrapping
textdistance	TextDistance – python library for comparing distance between two or more sequences by many algorithms calculates lavenshtien distance and shannon entropy.	Web-Scrapping

Table 4.2: List of Python Library for Analysis Phase

Library	Purpose	Module
Scikit-learn	Scikit-learn is an open source Python library that has powerful tools for data analysis and data mining. It's available under the BSD license	Analysis
Numpy	NumPy is the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object.	Analysis
Pandas	pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.	Analysis
Matplotlib	Python's matplotlib library, it fortun -ately takes far less than a thousand words of code to create a production -quality graphic.	Analysis

Analysis phase is also the processing phase where the constructed dataset is first cleaned or pre-porcessed to remove noise and then steps like detection, prediction and clustering is followed.

4.3 Module View

This section gives details of each module present in the framework. The basic picture was covered by model view in section above gives the idea that the proposed framework has three modules in total to detect DGAs. Each module of the has sub modules and this section will introduce them in detail.

4.3.1 Module 1: Data Gathering

The evaluation of anomalies using a synthetic evaluation context runs the risk that an algorithm's performance is heavily influenced by the rules and procedures used for

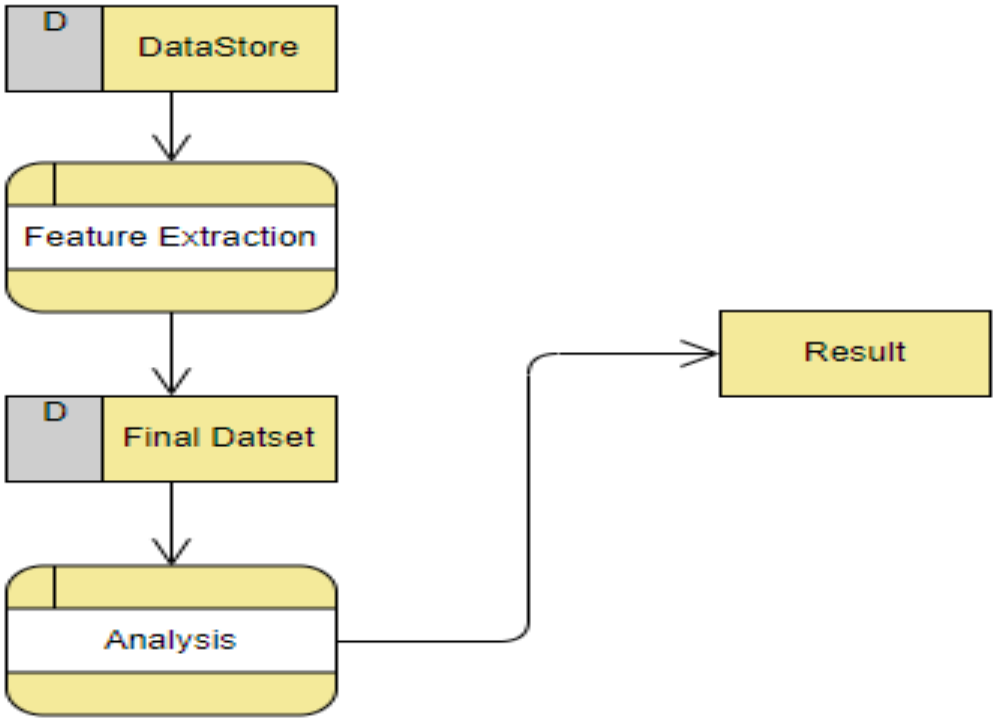

Figure 4.2: Data Flow Diagram of Model

creating the evaluation dataset. In order to create a realistic evaluation scenario, the DGA detection algorithms in this thesis are compared against an actual snapshot of DNS traffic obtained from a research network backbone. This section discusses the method used for collecting and anonymizing the dataset to safeguard the privacy of the network's users. Anonymizing the dataset is essential, as without anonymization detailed profiles could be created about specific individuals, without the affected network's users explicit consent, which would be a great breach of trust towards the user.

As discussed in Chapter 2, getting reliable data is not an easy task. There is the trend not to share network data of this form to safeguard the realm of cybersecurity and privacy to the individual, but the necessity to be ahead of the attackers today cannot remain secondary. The idea to develop a new reliable system needs to be tested under various failing conditions and then only can be accepted by the world.

Therefore the need to data to validate the framework in the most important thing. Based on Figure 4.3 the nature of the gathered data, it is classified under two classes:

- Static Class: This dataset is formed from the collection of data that is publicly available on different internet sources.
- Dynamic Class: This dataset is formed from the collection of data from the real network traffic.

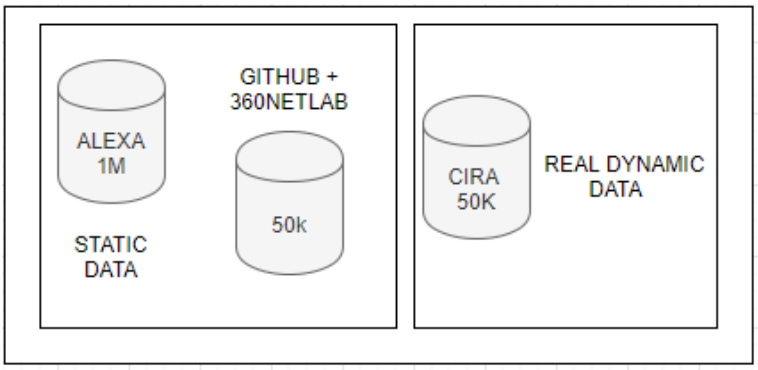

Figure 4.3: Data Gathering

Also the dataset is categorized into two different classes based on the categories:

- Clean Class: This is the category where the data consist of 1 M records and is fetched from the most reliable source, as these 1 million records are the domain names that are most frequently visit or accessed by a human.
- Unclean Class: This is the category where the data is a mix of all the possible DGAs generated by different malware families and also the real network traffic. More than 30 malware families generated DGAs are covered in this research giving a broad spectrum.

4.3.2 Module 2: Feature Extraction

The proposed method of detecting DGAs in networks relies on DNS traffic, therefore it is necessary to extract all relevant features from this traffic before the algorithm

can work on the data. Selecting right features can make a framework successful and therefore marks a very important place in any research for that matter. Also in this research feature extraction is the key. This section gives more details about the category of feature and the total number of features used in this research. There is a total of forty features used here covering the very basic feature to new features which were not considered before.

As Discussed in the previous chapter, an importance of feature extraction itself, it is categorised into two categories which is further classified into four sub-categories shown in Figure 4.4. The feature set displayed in the figure shows the entire list of feature that have been used in this research. Each feature from feach sub-category holds it's importance in the dataset.

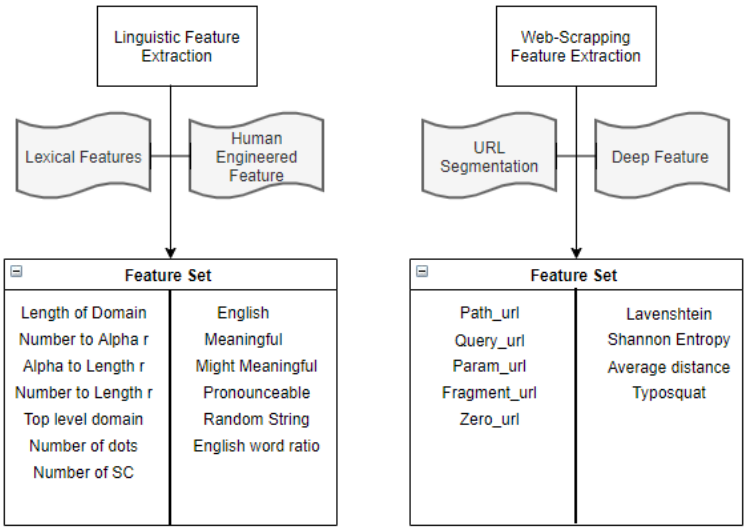

Figure 4.4: Feature Extraction

Going through this phase the raw record set collected in the data gathering module is now converted into the dataset that will be directly fed to the analysis phase for generating the relevant results.

4.3.3 Module 3: Analysis

This is the last module of the framework and also responsible for verifying the purpose of the proposed framework. For any research to be successful, it needs to be tested using multiple datasets. Analysis phase consists of three components namely detection, prediction and clustering. All these three phases use a different machine learning algorithms and produce different results. Detection, prediction and clustering are also interdependent as shown in Figure 4.5. Each phase of analysis is

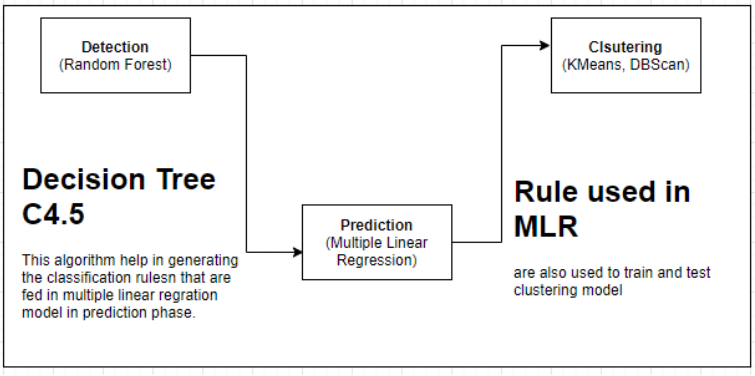

Figure 4.5: Analysis

meaningful and has its own purpose to satisfy as part of the proposed framework. The analysis phase is considered incomplete with any missing component than proposed.

1. Detection: uses random forest supervised machine learning algorithm to detect the DGAs in the dataset. Also in the phase the detection rules are extracted for using them in the prediction phase as well using decision tree C4.5 algorithm.

2. Prediction: uses the multiple linear regression model to predict the DGAs and clean domain names accurately, this model is trained and tested using the rules extracted from the C4.5 algorithm and then the rules are added or removed based on the regression error. Mostly the rule keep on changing with the size of the dataset. There are a the set of 7 rule used while testing the dataset of

2k records but with the increase the sample size the number of rules used for prediction are more as compares to the rules used for a smaller dataset.

3. Clustering: the same rule used in the prediction phase by multiple linear regression(MLR) are used by unsupervised machine learning algorithms in the clustering phase as well. These rules remain constant in both kmeans and dbscan.

also the analysis phase will be discussed more in details in the chapter 5 Experiments and Results.

4.4 Behavioral View

This section gives a full vision of the the proposed framework, how the components interacts and how the process in getting executed sequentially. The flow of execution of each component in a sequential manner is important and therefore the Figure 4.6 shows the behavior and response of each module against one another. While working with functions it is really important to know the order in which statements are executed this is called the flow of execution. Fortunately, Python is adept at keeping track of where it is, so each time a function completes, the program picks up where it left off in the function that called it. When it gets to the end of the program, it terminates.

Therefore it is necessary to have a sequential execution of all the modules and each each function in a program. In data collection phase there are three sets of raw data collected one is the record set of 1 million legitimate domain names for Alexa top visited websites, the other set is the collection of all the lists DGAs present on the internet and the last set of the collection of DGAs from real network traffic. The legitimate domain names are used with both the static data collection and also with the real data collection.

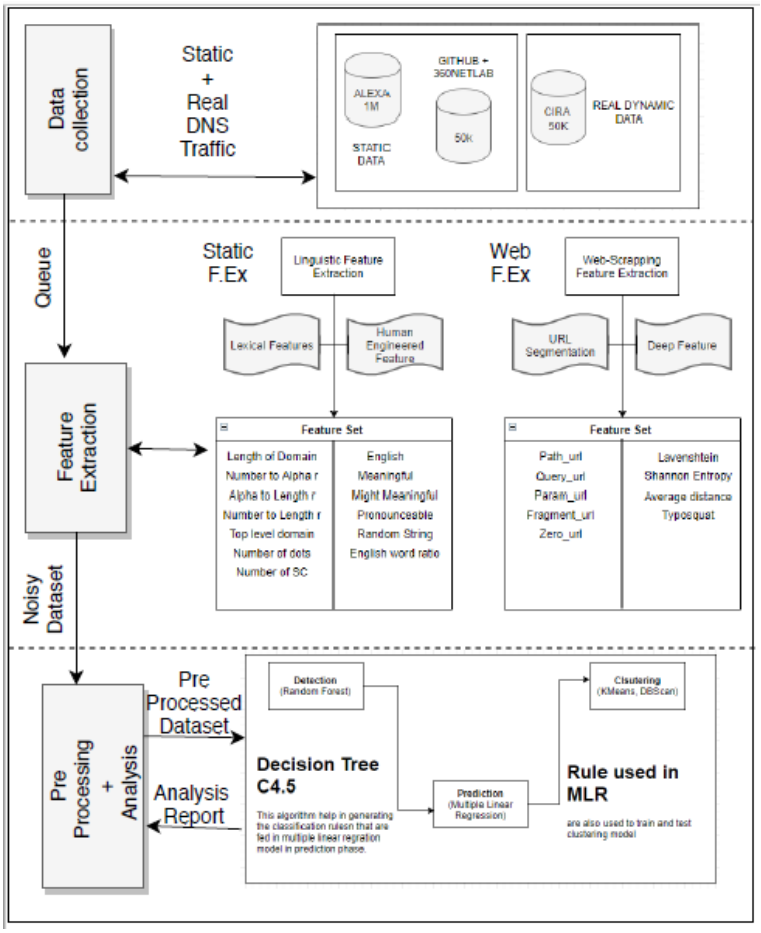

Figure 4.6: Behavioral View of Framework

Each domain names from the clean and DGA record set is queried using queue and hence leading to a new dataset. There raw record set is processed into the a new dataset by feature extraction phase. In feature extraction as discussed previously there are two type of extraction done, one is static feature extraction (static f.ex) which includes all the lexical features and the is web feature extraction (web f.ex) which includes the web scrapping feature extraction.

4.4.1 Pre-Processing Dataset

New dataset formed after feature extraction phase need to be pre processed this includes:

- Processing missing data:

  Data in real world are rarely clean and homogeneous. Data can either be missing during data extraction or collection. Missing values need to be handled because they reduce the quality for any of our performance metric. It can also lead to wrong prediction or classification and can also cause a high bias for any given model being used.Depending on data sources, missing data are identified differently. Pandas always identify missing values as NaN. However, unless the data has been pre-processed to a degree that an analyst will encounter missing values as NaN. Missing values can appear as a question mark (?) or a zero (0) or minus one (-1) or a blank. As a result, it is always important that a data scientist always performs exploratory data analysis(EDA) first before writing any machine learning algorithm. Handling missing data is important as many machine learning algorithms do not support data with missing values. There are multiple ways of pre processing the data:

  1. Imputation Using (Mean/Median) Values: This works by calculating the mean/median of the non-missing values in a column and then replacing the missing values within each column separately and independently from the others. It can only be used with numeric data.

  2. Imputation Using (Most Frequent) or (Zero/Constant) Values: Most Frequent is another statistical strategy to impute missing values and YES!! It works with categorical features (strings or numerical representations) by replacing missing data with the most frequent values within each column.

  3. Imputation Using k-NN: The k nearest neighbours is an algorithm that is used for simple classification. The algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in

the training set. This can be very useful in making predictions about the missing values by finding the k's closest neighbours to the observation with missing data and then imputing them based on the non-missing values in the neighbourhood.

4. Imputation Using Multivariate Imputation by Chained Equation (MICE): This type of imputation works by filling the missing data multiple times. Multiple Imputations (MIs) are much better than a single imputation as it measures the uncertainty of the missing values in a better way. The chained equations approach is also very flexible and can handle different variables of different data types (ie., continuous or binary) as well as complexities such as bounds or survey skip patterns.

5. Imputation Using Deep Learning (Datawig): This method works very well with categorical and non-numerical features. It is a library that learns Machine Learning models using Deep Neural Networks to impute missing values in a dataframe. It also supports both CPU and GPU for training.

The approach used here is imputation Using most frequent value, displayed in Figure 4.7 where the column with nan value is replaced with the most occurring value in the column.

```
# importing pandas
import pandas as pd
# making data frame from csv at url
df = pd.read_csv("path")
# Remove edge cases
ids = df.groupby('typosquats').preference_id.count()\
    .where(lambda x: x > 0).dropna().to_frame().reset_index()
ids.groupby('typosquats').preference_id.transform(lambda x: x.mode().iat[0])
```

Figure 4.7: Replacing Missing Value by Most Frequent Value

- Transforming the categorical data in the dataset.

  There are multiple ways to transform categorical data to numerical data.

1. LabelEncoder and OneHotEncoder: The labelEncoder and OneHotEncoder only works on categorical features. first to extract the categorial featuers using boolean mask.LabelEncoder converts each class under specified feature to a numerical value.

2. DictVectorizer: the LabelEncoder and OneHotEncoder usually need to be used together as two steps procedure. An more convenient way is using DictVectorizer which can achieve these two steps all at once. First, by converting the dataframe into a dictionary. This can be achieved by Pandas to_dict method. In this case we don't need to extract the categorical features, we can convert the whole dataframe into a dict. This is one advantage compared to LabelEncoder and OneHotEncoder and later by initiating DictVectorizer.

3. Get Dummies: Pandas get_dummies method is a very straight forward one step procedure to get the dummy variables for categorical features. The advantage is, it can directly apply it on the dataframe and the algorithm inside will recognize the categorical features and perform get dummies operation on it.

Here in this research this transformation is done using python str.get_dummies shown in Fig 4.8. Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analyzing data much easier.

Pandas str.get_dummies() is used to separate each string in the caller series at the passed separator. A data frame is returned with all the possible values after splitting every string. If the text value in original data frame at same index contains the string (column name/ splited values) then the value at that position is 1 otherwise, 0. Since this is a string operation, .str has to be prefixed

every time before calling this function. Otherwise, it will throw an error.

```
# importing pandas
import pandas as pd
# making data frame from csv at url
data = pd.read_csv("path")
# making dataframe using get_dummies()
dummies = data["English"].str.get_dummies(" ")
```

Figure 4.8: Categorical Transformation to Numeric Transformation

4.5 Concluding Remark

This section gives a detailed picture of the configuration and implementation details of the research, including the libraries used, configuration details of implementation language and the configured files. Furthermore this chapter gives the details of the proposed framework in three different views, including model view where the section is basically explaining the model as a whole unit, then module view which is telling about each section in a module. It basically shows the detail of each module and the subsections that each module comprise of, also explaining the specifics of the purpose of each component. Then the chapter further discusses about the behavioural view of the model. The behavior view not only explains the details of the component but also gives in-depth insight of the interact between each component and also with their sub-components.

Chapter 5

Experiments and Results

5.1 Dataset and labeling

The dataset formation consist of collection of various forms of data present in different websites and github. There are two classes of data in the dataset: Clean and DGAs. For clean dataset, Alexa 1M records for the domain names are used and for unclean data multiple DGA families present in github and various antivirus sources are considered. Here the dataset consists of more than 30 families of DGAs, aiming at experimenting with different families to evaluate the framework better.

The dataset is sampled in ascending order starting from 2k records to 1M, with the increase in the number of samples size, the accuracy or the framework also increases. There has been a lot work done in this field before and detection accuracy has reached to 98% precision rate [51] and more than 96% accuracy in DGA Botnet detection using random forest by [48] and 99% accuracy by EXPOSURE [7]. With the features we selected for our framework, we intend to achieve better prediction and accuracy.

Table 5.1: Description of Dataset

Sample Size	Dataset Class	Description
1M	Alexa (Clean)	Most frequently Accessed Sites
60K	DGAs	30 famous Malware Family
50K	DGAs	Top 10 Malware Family in Dec 2019 - Jan 2020
30K	DGAs	Real Network Traffic

5.1.1 Dataset Sampling

There are three set of dataset considered to evaluate the result of this research two of the dataset consists of the static data i.e they are either algorithmically generated or downloaded from the reliable source that can be antivirus or github, and the second class of data is the collection of the domain names from the real network traffic.
As described in Table 5.1 there are three sets of data. It is important to use two different classes of static data as to test the proposed framework if it can be a reliable source to provide valuable results. The set 1 consist of the DGAs generated by the most known DGA families and set 2 consists of the DGAs generated by the most commonly known malware families for their attack world wide in the period of December 2019 to January 2020 (ZeuS, CryptoWall, CoinMiner, Kovter, Dridex, NanoCore, Cerber, Nemucod, Emotet, Gh0st) and set 3 comprises of the DGAs extracted from the network traffic.

5.1.2 Processing Dataset: Detection

This is the final step of the framework and hence holds importance to the research as well. This section will give a detailed introduction to the various algorithms used in pre-processing and detection technique. Processing any dataset depends on number of factors including the type of data and volume of dataset; also when the research is focused on DGAs, there has been much work done in this field using machine learning, deep learning and also rule based learning. The idea behind the design of our framework is mainly based on the 'features' that we use to detect clean and

algorithmically generated domain names.

5.2 Model Selection

Model selection is the process of choosing between different machine learning approaches - e.g. decision tree, logistic regression, etc - or choosing between different hyper parameters or sets of features for the same machine learning approach - e.g. deciding between the polynomial degrees/complexities for linear regression.The choice of the actual machine learning algorithm (e.g. Random forest or logistic regression) is less important - there may be a "best" algorithm for a particular problem, but often its performance is not much better than other well-performing approaches for that problem [9]. Since DGA detection is not a new topic, work done in this field includes all different types of machine learning algorithms. Here in this phase few different types of machine learning algorithms for detection, prevention and clustering are used.

5.3 Results

In this proposed approach the dataset is processed in ascending order, starting with 2K records in a dataset. It is evident by the results that with the increase in the number of records in the dataset the results also improve. While talking about the result of this framework, also there is a need to consider *Time Complexity*. With the increase in the size of the dataset the time for processing detection, using random forest also increases. When considering prediction and clustering the time required in processing do not have a dramatic increase. While processing 20k records, the random forest algorithm took 15 hours to complete. Nevertheless, with the remarkable result in accuracy, precision, false positive and false negative our approach is worth the time. The result of detection and prediction phase is measured in the

terms of accuracy, precision, false positive and false negative.

5.3.1 Detection

This section describes the results of the used approach to detects the DGAs in our dataset by the use of machine learning algorithm. The selection of machine learning algorithm is critical as it is one of the best approaches to deduce results using large volume of data. Also by using *Random forest* for detection part the approach is made simple and efficient as the Random Forest algorithm is one of the most popular methods or frameworks used by data scientists [27]. It is an ensemble learning method for classification and regression that constructs a number of decision trees at training time, outputting the class that is the mode of the class's output by individual tree.

This algorithm has been implemented in many languages; using *python3.x* version, using Sciket-learn, numpy, scipy, matplotlib libraries [38]. It is a *Supervised machine learning* approach based on ensemble learning, where different types of algorithms or the same type of algorithm are joined multiple times to make a powerful model. The sampled dataset has been tested using random forest model, with the result observed that the accuracy increases with the increase in number of the records the dataset. Starting with 2K sample, the division of the data is such that 20% test data is only DGAs and also while testing 20K sample set the test data mainly consists of DGAs. It is evident by the results in Table 5.2 that our framework works quite well in terms of detection of DGAs. There have been work with good results but our framework is advancement of all the work done with a good accuray by random forest.

Table 5.2: Detection Results

Random Forest Detection Results					
Static Data - SET 1					
Data Size	**Trees**	**Accuracy**	**Precision**	**False Positive**	**False Negative**
2K	1	98.813%	99%	0.54%	0.33%
	5	99.466%	99.23%	0.38%	0.20%
	10	99.822%	99.69%	0.13%	0.05%
20K	1	99.421%	99.32%	0.347%	0.231%
	5	99.918%	99.89%	0.054%	0.027%
	10	99.946%	99.93%	0.034%	0.010%
60K	1	99.874%	99.90%	0.048%	0.077%
	5	99.983%	99.99%	0.0003l%	0.0013%
	10	99.988%	99.99%	0.0002%	0.0003%
Real Traffic - SET 2					
30K	1	99.762%	99.76%	0.115%	0.122%
	5	99.993%	99.85%	0.0%	0.007%
	10	99.986%	99.96%	0.0%	0.014%
Static Data - SET 3					
50K	1	98.621%	99.78%	0.119%	0.133%
	5	99.793%	99.75%	0.0%	0.057%
	10	99.886%	99.86%	0.0%	0.024%

5.3.2 Prediction

This section of our work is based on the important features selected by the model (decision tree) to come up with a decision. With the use of C4.5 Decision tree algorithm to select the rules based on which shall predict the non labelled data and test proposed work on real traffic as well[16]. The key part here is on deciding the depth of the tree (levels of the tree) and also the index value as a creation. Here, gini index and entropy both are used to compare the features selected and use them in prediction presented in Figure 5.1, 5.2 respectively. Later by using multiple linear regression algorithm by using LinearRegression module of python3.x for prediction, as compared to random forest this phase is pretty fast at generating results, it has generated the table of result consisting of accuracy, false positive and false negative in just the matter of seconds. With the results in Table 5.3 it is quite visible the minute decrease in the accuracy with the increase in dataset size until 20k and on testing

the proposed work with 60k record set the accuracy increases again. The accuracy in this case depends on the features selected and with increase in the dataset the features selected by the decision tree also differs therefore, the accuracy differs as well.

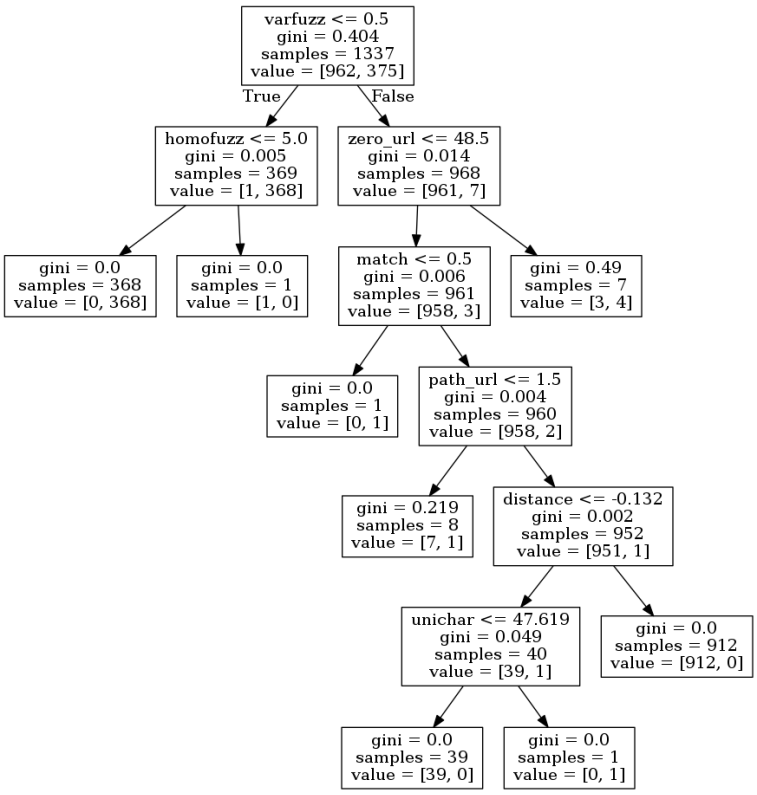

Figure 5.1: Feature extraction by Gini Index

Observations:

- With the increase in dataset the prediction accuracy decreases, even the percentage decrease in accuracy is negligible as compare to the record set values.
- With the Increase in dataset the features used for prediction also changes and even more features are considered.
- Therefore, it is important to use decision tree classifier in prediction as the feature selection is done by this machine learning algorithm, making prediction a faster process.

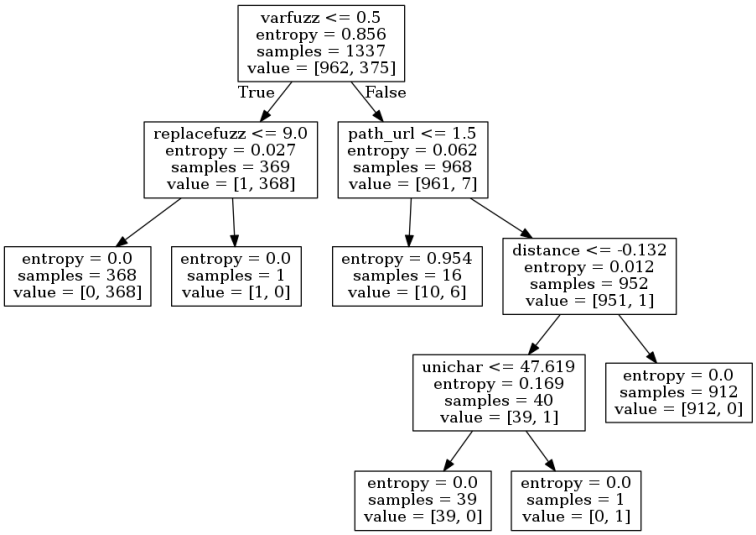

Figure 5.2: Feature extraction by Entropy

Using these rules extracted by the Decision tree algorithm in the prediction phase by the Multiple Linear Regression algorithm helps to justify the concept, by evaluating the correctness of the prediction which has to apply to non labelled dataset as well. Prediction is a very known concept like others machine learning concept that are used in general. Here, the selected features are among the distinctive features listed in the feature extraction section. These features include Average distance, Match, English ratio, Meaningful domain name from Lexical Analysis also pathURL, ZeroURL, Typosquats, Unichar from Web Scrapper. It provides the valuable side to our research as it helps in detecting even the DGAs from the non labelled dataset by prediction.

5.3.3 Clustering

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

This is a phase where marking a difference between two classes, DGAs and legitimate

Table 5.3: Prediction Results

Multiple Linear Regression			
Static Data - SET 1			
Data size	**Accuracy**	**False Positive**	**False Negative**
2K	99.112%	0.0%	0.217%
20K	98.775%	0.136%	0.918%
60K	98.981%	0.193%	0.513%
Real Traffic - SET 2			
30K	99.242%	0.072%	0.613%
Static Data - SET 3			
50K	99.642%	0.062%	0.513%

domain names is the result. The concept is to distinguish classes by the formation of clusters. For this purpose, the two different kinds of clustering algorithms that are *KMeans* and *DBSCAN* used. With the use of kmeans we were able to generate number of clusters that are required for clustering data, which itself shows a dent at 2 and also at 3, the Figure 5.3 is the output generated after processing 60k records, therefore to verify the cluster we have also used dbscan algorithm later. It is quite clear by the graph plotted by DBScan by generating 2 clusters. The x and y axis in this case are normalized PCA into two principal axis.

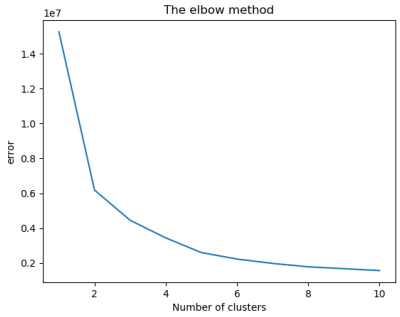

Figure 5.3: Number of clusters by elbow method

Also the motive behind using the DBSCAN clustering algorithm is to test the result by not mentioning number of clusters. Here the algorithm itself tries to figure out the categories calculating the difference between the features itself. As a result of the presence of a wide gap between these two classes, only 2 clusters are formed at

the end which is displayed in *Fig 5.4: DBCSAN Clustering Algorithm*, where red color is specifying legitimate domain names and green color specifies DGAs, the dots surrounding green cluster is the noisy classified data.

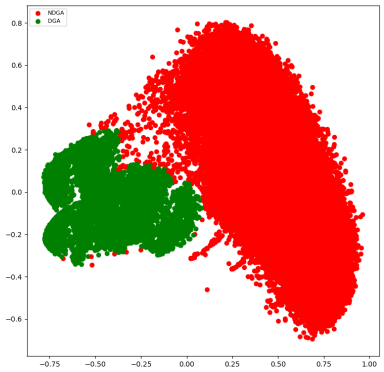

Figure 5.4: DBSCAN Clustering Algorithm

5.4 Concluding Remarks

It is clear from the results present in this chapter that analysis results are justifying the proposed framework. The connectivity of the modules together has made this proposed framework successful and reliable source to detect and predict the existing and even can work well on newly generated malware class like the top ten malware DGA families listed below:

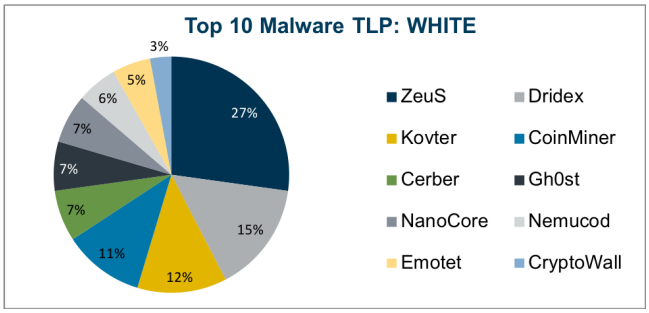

Figure 5.5: Datset Extension

The ZeuS, CryptoWall, and CoinMiner alerts account for activity within the mul-

tiple infection vector category for the month. Kovter, Dridex, NanoCore, Cerber, Nemucod, and Emotet drive malspam related infections for the month of January 2020. Gh0st is currently the only malware in the Top 10 whose primary initiation vector is being dropped by other malware. There is a high likelihood that malspam will continue to remain the preferred initiation vector for malware in the Top 10 these are used as the SET 3 (static Detection).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

There are techniques proposed in the past that can help to capture and detect malware generated domain names based on different features that might consist of static or dynamically captured data. The proposed methodology here, is to bring the best features possible which support the framework to never fail in detection and prediction of all the possible kinds of algorithmically generated domain names. The proposed model gives better results with the machine learning models used, even with the Random Forest. We propose a technique that is an amalgam of linguistic and web extracted features, which consist of the very novel feature set never considered before for DGA Detection purpose. The idea is to get a clear picture of domain names with their class of being in clean or unclean category. There has been a variety of work done in this domain and researchers have used all the types of approach to tackle the program of DGAs, but here our research is based on considering features from all the aspects to detect the DGAs with maximun accuracy. This research is basically the extension of the work done until now which makes the system hold the better ground. Domain Generating Algorithms are in use by cybercriminals to prevent

their servers from being blacklisted or taken down. The algorithm produces random looking domain names. The idea is that two machines using the same algorithm will contact the same domain at a given time, so they will be able to exchange information or fetch instructions.

Inspite of all the work done by the researchers in prevention and detection of cyber attacks by deploying malwares, adversaries always try to be craftier to exploit vulnerabilities. Therefore, there is a continuous need to understand and update the system to make it less vulnerable to attacks. The approach towards feature extraction and using those feature to categorize the clean and unclean domains is the extension of the work done by any approach till now. As there is a distinguishing difference between the legitimate domain names and algorithmically generated ones. Therefore feature extraction is considered an important part to the proposed framework.

As part of bringing this system to a better state, it is important to detect and classify the families of malware rather just classifying them under unclean category.

6.1.1 Importance of Feature Extraction

Making feature extraction as the primary aspect of the research is giving this research a different direction where the need to understand how important it is to recognise and extract the feature that could possibly make a difference in the proposed research work. There are multiple work done where studying feature extraction is considered and there are ways the researchers have proposed different feature extraction approaches. It can be algebraic feature extraction [29], recognition feature extraction [40], wavelet [53], classification, robust, PCA feature extraction [11], etc. There is a proper study a researcher have to do before selecting the number of feature, the category of feature and also the approach he would use to extract feature therefore it is not an easy process. These factors while considering feature makes the proposed framework to fail or pass. Here the proposed system has passed with the flying

colors based on the type of feature, number of features and also depending the the approach to extract them from the raw data.

6.2 Future Work

The proposed system shows satisfactory results, but there is always a scope for improvement in every framework and also the need to stay up to date to make it less vulnerable to any exploit. That is the reason why every software needs to patch update. In software development life cycle (SDLC) a developer can easily understand the flaws in the system and starts preparing to solve them before deploying the final software or if its not major then it is considered for the patch update process. therefore the need to recognise what has to be done in future to make the system better is important and noted by any researcher.

6.2.1 Types of future work

There can be several directions of future work depending on ways that would be necessary in regards to work on them, few of which are mentioned as:

1. Design Based: There can be a structural issue that might have mage the design of any model complex or there can be ways to simplify the structure. It is important to try to make the design on the model as simple as possible to avoid future glitches.

2. Programming Based: The possibility of the any software to be exploited also depend on the programming language it is build in therefore. The software written in C are more prone to threats then software written using java, python.

3. Time Complexity Based: Also based on the complexity of the code and also the kind of programming language the time complexity varies. Python is a fast

and robust programming language but if there is a lot of computation involved and with the use of libraries the process can slower done, which increases the time complexity of the execution to complete.

4. Space Complexity Based: While working on a large set of dataset, it is important to consider the fact that storing that data and the reason of loading that data in memory (ram) will make the system slower.

Therefore there is the need to recognise what is the possibility of future work the research would have later. The problem on the proposed framework is time complexity due to calculation of the Features and dataset reduction. In the current implementation, all these similarities for all suspicious domains detected for the entirety of the input data are calculated. In practice, the amount of calculations can be reduced to by the introduction of white listing method as the extension of this method where the Alexa 1M record feature can be calculate once and kept and the domain with the same features can be added to the list directly only by determine the similarity between whitelist and suspicious record. Therefore the time complexity in the research be made much simpler if it implemented using the other programming language in feature extraction phase, because by using libraries in python it might have made the programming structure simpler but have increase the complexity by O(n). Also since there are ample features considered in this approach which makes it a time consuming process to extract features for all the domain names our future work would also include minimizing time consumption as an advancement to the proposed framework.

Bibliography

[1] *360 netlab*, https://data.netlab.360.com/dga/.

[2] *Alexa Top 1M*, https://www.alexa.com/topsites, Accessed: 2019-09-30.

[3] Jasper Abbink and Christian Doerr, *Popularity-based detection of domain generation algorithms*, Proceedings of the 12th International Conference on Availability, Reliability and Security, 2017, pp. 1–8.

[4] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon, *From throw-away traffic to bots: detecting the rise of dga-based malware*, Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12), 2012, pp. 491–506.

[5] Thomas Barabosch, Andre Wichmann, Felix Leder, and Elmar Gerhards-Padilla, *Automatic extraction of domain name generation algorithms from current malware*, Proc. NATO Symposium IST-111 on Information Assurance and Cyber Defense, Koblenz, Germany, 2012.

[6] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi, *Exposure: Finding malicious domains using passive dns analysis.*, Ndss, 2011, pp. 1–17.

[7] Leyla Bilge, Sevil Sen, Davide Balzarotti, Engin Kirda, and Christopher Kruegel, *Exposure: A passive dns analysis service to detect and report malicious*

domains, ACM Transactions on Information and System Security (TISSEC) **16** (2014), no. 4, 14.

[8] Alkan Borges, Udhayashankar Dhasarathan, Ankur Gupta, and Ramesh Manickam, *N-gram combination determination based on pronounceability*, September 24 2015, US Patent App. 14/282,529.

[9] Michael Bowles, *Machine learning in python: essential techniques for predictive analysis*, John Wiley & Sons, 2015.

[10] Aitao Chen, Yiping Zhou, Anne Zhang, and Gordon Sun, *Unigram language model for chinese word segmentation*, Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing, 2005.

[11] Anil Cheriyadat and L Mann Bruce, *Why principal component analysis is not an appropriate feature extraction method for hyperspectral data*, IGARSS 2003. 2003 IEEE International Geoscience and Remote Sensing Symposium. Proceedings (IEEE Cat. No. 03CH37477), vol. 6, IEEE, 2003, pp. 3420–3422.

[12] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim, *Botnet detection by monitoring group activities in dns traffic*, 7th IEEE International Conference on Computer and Information Technology (CIT 2007), IEEE, 2007, pp. 715–720.

[13] Debra L Cook, Vijay K Gurbani, and Michael Daniluk, *Phishwish: a stateless phishing filter using minimal rules*, International Conference on Financial Cryptography and Data Security, Springer, 2008, pp. 182–186.

[14] Heather Crawford and John Aycock, *Kwyjibo: automatic domain name generation*, Software: Practice and Experience **38** (2008), no. 14, 1561–1567.

[15] Pedro Marques da Luz, *Botnet detection using passive dns*, Radboud University: Nijmegen, The Netherlands (2014).

[16] Manoranjan Dash and Huan Liu, *Feature selection for classification*, Intelligent data analysis **1** (1997), no. 1-4, 131–156.

[17] Morrie Gasser, *A random word generator for pronounceable passwords*, Tech. report, MITRE CORP BEDFORD MA, 1975.

[18] Jason Geffner, *End-to-end analysis of a domain generating algorithm malware family*, Black Hat USA **2013** (2013).

[19] Inc. GitHub, $domain_generation_algorithms, https://github.com/baderj/domain_generation_al$

[20] Martin Grill, Jan Kohout, Martin Kopp, and Tomas Pevny, *Illegitimate typosquatting detection with internet protocol information*, November 26 2019, US Patent 10,491,614.

[21] R John, *Douceur*, The Sybil attack. Revised Papers from the First International Workshop on Peer-to-Peer Systems, 2002, pp. 251–260.

[22] Yogesh Joshi, Samir Saklikar, Debabrata Das, and Subir Saha, *Phishguard: a browser plug-in for protection from phishing*, 2008 2nd International Conference on Internet Multimedia Services Architecture and Applications, IEEE, 2008, pp. 1–6.

[23] Min-Yen Kan and Hoang Oanh Nguyen Thi, *Fast webpage classification using url features*, Proceedings of the 14th ACM international conference on Information and knowledge management, ACM, 2005, pp. 325–326.

[24] Srinivas Krishnan, Teryl Taylor, Fabian Monrose, and John McHugh, *Crossing the threshold: Detecting network malfeasance via sequential hypothesis testing*, 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2013, pp. 1–12.

[25] Felix Leder and Tillmann Werner, *Know your enemy: Containing conficker*, The Honeynet Project (2009).

[26] Felix Leder, Tillmann Werner, and Peter Martini, *Proactive botnet countermeasures: an offensive approach*, The Virtual Battlefield: Perspectives on Cyber Warfare **3** (2009), 211–225.

[27] Andy Liaw, Matthew Wiener, et al., *Classification and regression by randomforest*, R news **2** (2002), no. 3, 18–22.

[28] Pierre Lison and Vasileios Mavroeidis, *Automatic detection of malware-generated domains with recurrent neural models*, arXiv preprint arXiv:1709.07102 (2017).

[29] Ke Liu, Yong-Qing Cheng, and Jing-Yu Yang, *Algebraic feature extraction for image recognition based on an optimal discriminant criterion*, Pattern recognition **26** (1993), no. 6, 903–911.

[30] Knud Lasse Lueth, *State of the iot 2018: Number of iot devices now at 7b–market accelerating, 2018*, URL: https://iot-analytics. com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/(visited on 03/04/2019) (2018).

[31] D Kevin McGrath and Minaxi Gupta, *Behind phishing: An examination of phisher modi operandi.*, LEET **8** (2008), 4.

[32] Jason Milletary and CERT Coordination Center, *Technical trends in phishing attacks*, Retrieved December **1** (2005), no. 2007, 3–3.

[33] Paul V Mockapetris, *Domain names-concepts and facilities*, (1987).

[34] Aditya Anand Mugali, Andrew W Simpson, Scott King Walker, et al., *Pronounceable domain names*, December 22 2015, US Patent 9,218,334.

[35] Pratik Narang, Subhajit Ray, Chittaranjan Hota, and Venkat Venkatakrishnan, *Peershark: detecting peer-to-peer botnets by tracking conversations*, 2014 IEEE Security and Privacy Workshops, IEEE, 2014, pp. 108–115.

[36] Nicolas Nicolov and Franco Salvetti, *Efficient spam analysis for weblogs through url segmentation*, AMSTERDAM STUDIES IN THE THEORY AND HISTORY OF LINGUISTIC SCIENCE SERIES 4 **292** (2007), 125.

[37] White Ops, *The methbot operation*, 2016.

[38] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., *Scikit-learn: Machine learning in python*, Journal of machine learning research **12** (2011), no. Oct, 2825–2830.

[39] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla, *A comprehensive measurement study of domain generating malware*, 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 263–278.

[40] K Venkatesh Prasad and David G Stork, *Facial feature extraction method and apparatus for a neural network acoustic and visual speech recognition system*, October 21 1997, US Patent 5,680,481.

[41] Jayaram Raghuram, David J Miller, and George Kesidis, *Unsupervised, low latency anomaly detection of algorithmically generated domain names by generative probabilistic modeling*, Journal of advanced research **5** (2014), no. 4, 423–433.

[42] Fergal Reid and Martin Harrigan, *An analysis of anonymity in the bitcoin system*, Security and privacy in social networks, Springer, 2013, pp. 197–223.

[43] Paul Royal, *On the kraken and bobax botnets*, Whitepaper, Damball, Apr (2008).

[44] Kazumichi Sato, Keisuke Ishibashi, Tsuyoshi Toyono, Haruhisa Hasegawa, and Hideaki Yoshino, *Extending black domain name list by using co-occurrence relation between dns queries*, IEICE transactions on communications **95** (2012), no. 3, 794–802.

[45] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero, *Tracking and characterizing botnets using automatically generated domains*, arXiv preprint arXiv:1311.5612 (2013).

[46] ———, *Phoenix: Dga-based botnet tracking and intelligence*, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2014, pp. 192–211.

[47] Hamza Shaban, *Digital advertising to surpass print and tv for the first time, report says*, Washington Post, February **20** (2019), 2019.

[48] Jan Spooren, Davy Preuveneers, Lieven Desmet, Peter Janssen, and Wouter Joosen, *Detection of algorithmically generated domain names used by botnets: a dual arms race*, Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, ACM, 2019, pp. 1916–1923.

[49] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna, *Your botnet is my botnet: analysis of a botnet takeover*, Proceedings of the 16th ACM conference on Computer and communications security, 2009, pp. 635–647.

[50] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich, *The long "taile" of typosquatting domain names*, 23rd {USENIX} Security Symposium ({USENIX} Security 14), 2014, pp. 191–206.

[51] Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran, and Linh Giang Nguyen, *A lstm based framework for handling multiclass imbalance in dga botnet detection*, Neurocomputing **275** (2018), 2401–2413.

[52] George Tsakalidis, Kostas Vergidis, Michael Madas, and Maro Vlachopoulou, *Cybersecurity threats: a proposed system for assessing threat severity*, Proceedings of the the forth international conference on decision support system technology–ICDSST 2018, 2018.

[53] Murat Uyar, Selcuk Yildirim, and Muhsin Tunay Gencoglu, *An effective wavelet-based feature extraction method for classification of power quality disturbance signals*, Electric power systems Research **78** (2008), no. 10, 1747–1755.

[54] ZHANG Wei-wei and GJL Qian, *Detecting machine generated domain names based on morpheme features*, International Workshop on Cloud Computing and Information Security (CCIS 2013), 2013.

[55] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan, *Detecting algorithmically generated domain-flux attacks with dns traffic analysis*, IEEE/Acm Transactions on Networking **20** (2012), no. 5, 1663–1677.

[56] Yonglin Zhou, Qing-shan Li, Qidi Miao, and Kangbin Yim, *Dga-based botnet detection using dns traffic.*, J. Internet Serv. Inf. Secur. **3** (2013), no. 3/4, 116–123.

Vita

Candidate's full name: Shubhangi Upadhyay

University attended:
Master of Computer Science
University of New Brunswick
2018-2020

Bachelors of Technology (Computer Science)
Dr. APJ Abdul Kalam University
2012-2016

Publications:
Submitted: Shubbhangi Upadhyay, Ali Ghorbani, : Feature Extraction Approach to Unearth Domain Generating Algorithms (DGAs). In 5th Cyber Science and Technology Congress. IEEE, 2020.

Conference Presentations:
Submitted: Shubbhangi Upadhyay, Ali Ghorbani, : Feature Extraction Approach to Unearth Domain Generating Algorithms (DGAs). In 5th Cyber Science and Technology Congress. IEEE, 2020.