INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Passive DNS Analytics

Priya Singh - a46383
Manindra Rana - a46249

Thesis presented to the School of Technology and Management in the scope of the Bachelor's in Informatics Engineering.

Supervisors:
Prof. Tiago Pedrosa
Prof. Rui Pedro Lopes

This document does not include the suggestions made by the board.

Bragança
2022-2023

Passive DNS Analytics

Priya Singh - a46383
Manindra Rana - a46249

Thesis presented to the School of Technology and Management in the scope of the Bachelor's in Informatics Engineering.

Supervisors:
Prof. Tiago Pedrosa
Prof. Rui Pedro Lopes

This document does not include the suggestions made by the board.

Bragança
2022-2023

Dedication

We would like to dedicate this report to our families, whose unwavering support and encouragement have been our pillars of strength throughout this journey. We also dedicate this report to our mentors and teachers, whose wisdom and expertise have shaped our knowledge and inspired our passion for learning. Your mentorship has been invaluable, and we are thankful for the guidance you have provided.

Finally, this report is dedicated to all those who share a curiosity for knowledge and a commitment to making a positive impact in their fields.

Acknowledgment

We extend our heartfelt thanks to the D4 project team for their pivotal role in providing access to valuable resources and data.

We are also indebted to CIRCL (Computer Incident Response Center Luxembourg) for their generous contribution to our project. Their data resources and expertise were instrumental in our research endeavors, and we acknowledge their vital role in making this project possible.

We would like to express our sincere gratitude to Prof. Tiago Pedrosa , Prof. Rui Pedro Lopes, and for their support. Their encouragement was instrumental in achieving a goal.

We would also like to thank Instituto Politécnico de Bragança for providing resources.Their contributions made this project possible.

Finally, we would like to acknowledge friends and families for their continuous efforts.

This project would not have been possible without the support of all of those who have contributed. Thank you.

Abstract

In this work, we dive into the world of Passive DNS analysis and its critical role in bolstering cybersecurity. Passive DNS is a technique that records and stores data about DNS resolutions, offering us valuable insights into internet activities and potential security risks.

Our discussion covers key aspects of Passive DNS, including how it works, what it does, and why it's so essential for keeping an eye on DNS records. We explore the nuts and bolts of setting up Passive DNS servers, the nitty-gritty of analyzing DNS records, and how we use something called the Common Output Format (COF) to make sure our data is structured consistently.

One of the standout features is the integration of the D4 platform. This includes the D4 Go client, which acts like a watchful sensor, constantly capturing DNS records and sending them to the D4 server. The D4 platform is like our control center for collecting and dissecting network data, with a specific focus on Passive DNS info.

We take a closer look at the architecture and how everything flows in the D4 setup. The star of the show is the Passive DNS analyzer, equipped with a full-fledged Passive DNS server for storing and retrieving DNS records. The use of COF ensures that all our data is presented in a consistent format, making it easier for different parts of the system to work together seamlessly. To supercharge our project, we tap into the CIRCL Passive DNS database to get our hands on historical Passive DNS records. These records are like treasure troves for building datasets and models, especially for machine learning. They become crucial in our mission to spot malicious domains and enhance cybersecurity.
Keywords: Cybersecurity, DNS, Passive DNS, Machine Learning

Contents

List of Tables

List of Figures

Acronyms

A Address record.

AAA Authentication, Authorization and Accounting.

API Application Programming Interface.

CNAME Canonical Name.

HTTP HyperText Transfer Protocol.

IP Internet Protocol.

IPB Instituto Politécnico de Bragança.

KNN K-Nearest Neighbors.

ML Machine Learning.

NS Name Server.

REST Representational State Transfer.

SSH Secure Shell.

SUDO Superuser Do.

SVM Support Vector Machine.

URL Uniform Resource Locator.

VPN Virtual Private Network.

Chapter 1

Introduction

When we look around today's world, we can see that we are more reliant on technology than ever before. The benefits of this trend range from near-instant Internet access to modern conveniences made possible by smart home automation technology and concepts like the Internet of Things.
With so much good that technology provides, it's easy to forget that potential threats lurk behind every device and platform. Nonetheless, cyber security threats posed by modern technology are a real threat, despite society's optimistic view of modern advances.
In recent years, the number and variety of cyber-attacks and malware samples have increased significantly, making it extremely difficult for security analysts and forensic investigators to detect and defend against such security attacks. To address this issue, researchers defined "Threat Intelligence" as "the collection, assessment, and application of data regarding security threats, threat actors, exploits, malware, vulnerabilities, and compromise indicators."

1.1 Objectives

The main objective of this project is to investigate, comprehend and explore Passive DNS and its applicability in cybersecurity. To accomplish this, the following working methodology was chosen:

1. Research about passive DNS and its applicability in cybersecurity intelligence.
2. Research methods for passive DNS analytics.
3. Identify interesting scenario to support the deploy of the solution for gather the information on resolving time on the network and the storing, management, and analytics components.

4. Definition of methodology for data analysis
5. Start developing the solution
6. Test and analysis.

1.2 Document Structure

The report is organized with the following structure:

1. Chapter 2 consists of literature review with contents; Cybersecurity intelligence, the domain name system, DNS threats/attacks and passive DNS.
2. Chapter 3 consists of approaches defined for the passive DNS replication.
3. Chapter 4 consists the description of the implementation, highlighting the most important aspects, the difficulties and the technical solutions that were followed.
4. Chapter 5 describes the tests that were developed to check if the project fulfills the objectives and solves the problem described.
5. Chapter 6 consists of the conclusion providing the single view to the work developed.

Chapter 2

Literature Review

In this chapter, we delve into the world of passive DNS data analysis and its critical role in cybersecurity intelligence. We also explore the alarming landscape of DNS attacks, which pose significant threats to organizations. These attacks, such as DNS amplification, spoofing, tunneling, fast flux, and hijacking, have become increasingly prevalent, costing organizations millions of dollars and jeopardizing data security.

2.1 Cybersecurity Intelligence

The study of how we protect devices and services from malicious actors such as hackers, spammers, and cybercriminals is known as cybersecurity. While some cybersecurity components are designed to be the first to attack, most professionals today are more concerned with determining the best way to defend all assets from computers and smartphones to networks and databases against attacks. Cisco Systems defines cybersecurity as "the practice of protecting systems, networks, and programs from digital attacks." These cyberattacks are typically intended to gain access to, alter, or destroy sensitive data, extort money from users, or disrupt normal business processes."

There are 7 different types of cybersecurity:

1. **Application Security**
   Application security is the implementation of various defenses in an organization's software and services to protect against a wide range of threats. To reduce the possibility of unauthorized access or modification of application resources, cybersecurity experts must write secure code, design secure application architectures, implement robust data input validation, and perform other tasks.

2. **Cloud Security**

Cloud security is concerned with developing secure cloud architectures and applications for businesses that use cloud service providers such as Amazon Web Services, Google, Azure, Rackspace, and others.

3. **Mobile Security**
   Nowadays, as more people rely on mobile devices, mobile security is becoming increasingly important. This subdomain of the Cybersecurity safeguards organizational and personal data stored on mobile devices such as tablets, cell phones, and laptops against various threats such as unauthorized access, device loss or theft, malware, viruses, and so on. Furthermore, mobile security employs authentication and education to help increase security.

4. **Endpoint Security**
   Endpoint security allows the businesses to protect end-user devices such as desktops and laptops by implementing data and network security controls, advanced threat prevention such as anti-phishing and anti-ransomware, and forensic technologies such as endpoint detection and response (EDR) solutions.

5. **IOT Security**
   While the use of Internet of Things (IoT) devices increases productivity, it also exposes organizations to new cyber threats. IoT security protects these devices through device discovery and classification, auto-segmentation to control network activities, and the use of IPS as a virtual patch to prevent exploits against vulnerable IoT devices. In some cases, the device's firmware can be supplemented with small agents to prevent exploits and runtime attacks.

6. **Network Security**
   Network security refers to the hardware and software mechanisms that safeguard the network and infrastructure against disruptions, unauthorized access, and other abuses. Effective network security safeguards organizational assets against a wide range of threats, both internal and external to the organization.

7. **Zero Trust**
   Zero trust employs a more granular approach to security, safeguarding individual resources through a combination of micro-segmentation, monitoring, and role-based access control enforcement.

Cyber intelligence is the backbone of integrated security frameworks, where the accuracy and credibility of information collected about potential threats can have a significant impact on the ability to counter or respond to such threats.

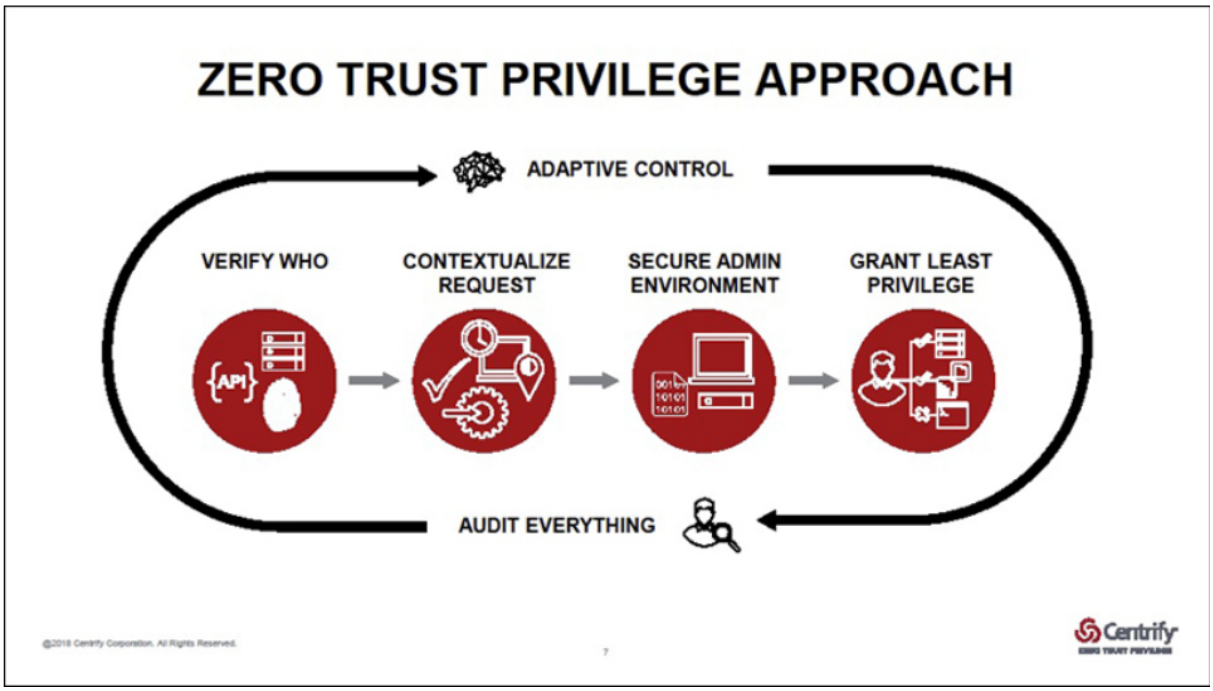

Figure 2.1: Zero Trust Approach
12

Hackers and attackers, as well as private and public cyber intelligence teams, use the Internet and public resources to learn about threats and vulnerabilities.In 2016, alone, 135 high-risk zero-day exploits were discovered in Adobe products, 76 in Microsoft products, and 50 in Apple products.

The Cyber Intelligence is classified into three levels:

2.1.1 Strategic Cyber Intelligence

Strategic Cyber Intelligence is Cyber intelligence that is strategic. These are the organizations' high-level goals for combating threats or malicious behavior. It is critical to identify threats, sources, main goals, and potential outcomes at this level.

2.1.2 Operational Cyber Intelligence

More operational information about threats is targeted at this level. For example, a cyber intelligence team will try to gather the following information about each potential threat: the capabilities and resources that attackers have or will need to achieve their strategic goals. The cyber intelligence team will also attempt to predict the cyber attacker's entry

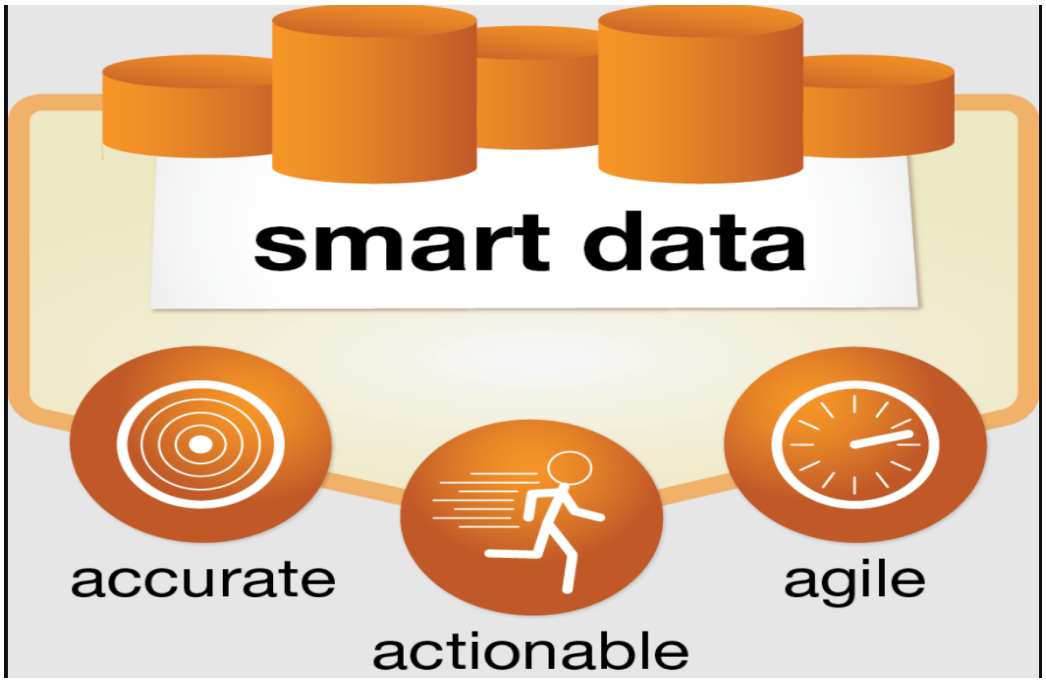

Figure 2.2: Smart Data
13

targets, intrusion, and propagation methods, and so on that will be required to carry out the attack in order to further their strategic goals.

2.1.3 Tactical or Technical Cyber Intelligence

At this level, knowledge will be related to the types of real-time methods and tools (software) they will use, as well as the potential counter or response mechanisms from the defender.

Sources of the Cyber Intelligence / Collection Capabilities

There are currently several types or sources of cyber intelligence, also known as collection capabilities or intelligence gathering disciplines. This list is constantly expanding vertically and horizontally.

Open-source intelligence (OSINT)

A cyber intelligent team should learn how to collect data points and turn them into actionable intelligence that can prevent target attacks. They must learn to recognize, repel, or neutralize targeted intelligence gathering against organizational assets. OSINT includes

information gathered from freely available public sources, whether free or subscription-based, online, or offline. OSINT can include many sub-categories such as:

- Classical Media: Newspapers, magazines, radio, and television channels.
- Social Media Intelligence (SOCMINT): Blogs, discussion groups, Facebook, Twitter, YouTube, etc.
- Communication Intelligence (COMINT)
- Internet public websites and sources
- Measurement and signature intelligence (MASINT)
- Search engines (e.g. Google, Yahoo)

Deep or dark web intelligence:
These are web pages, documents, and so on that are not indexed by major search engines and/or cannot be read or accessed using traditional methods. The public or visible web is much smaller in percentage terms than the deep web.

Deep web can be divided into the following categories: Dynamic web pages, Blocked sites, Unlinked sites, Private sites, Non-HTML or Scripted content, and Limited or local access networks or content that is not publicly accessible via the Internet are all examples of restricted content.

These include web pages, documents, and other resources that are accessed anonymously (e.g via TOR browsers) and are frequently used for criminal purposes.

1. **Dark web or net**The dark web has evolved into a meeting place for hacking communities, allowing cyber criminals to discuss, offer, and sell new and emerging exploits (e.g zero-day vulnerabilities or exploits). A popular example of a darknet website is the zero-day forum (The website link is constantly varies, for example, http://qzbkwswfv5k2oj5d.onion.link/, http://msydqstlz2kzerdg.onion/). Public websites, not just dark websites, can be used as effective hacking or attacking tools. Websites such as Shodan (https://www.shodan.io/), Zomeye (https://www.zoomeye.org), and https://www.go4expert.com/ , for example, can provide attackers with a wealth of information about candidate targets, as well as very good introduction details to begin further investigations and analysis.
2. **Signal Intelligence (SIGINT)**It can also include GEOINT (geospatial intelligence, such as images taken from aircraft or satellites) and MASINT (military intelligence) (measurement and signature intelligence, e.g, radar data).

2.2 The Domain Namespace

Domain names are used to index DNS's distributed database. Each domain name is essentially nothing more than a path in a large, inverted tree known as the domain namespace. The hierarchical structure of the tree, as shown in below figure, is similar to the structure of the Unix filesystem at the top of the tree, there is a single root. This is the case in the Unix filesystem. The root directory is denoted by a forward slash (/). DNS simply refers to it as " "DNS tree, like a filesystem, can branch in any number of ways at each intersection point, or node. The tree's depth is limited to 127 levels a limitation we are not aware of most likely to succeed.

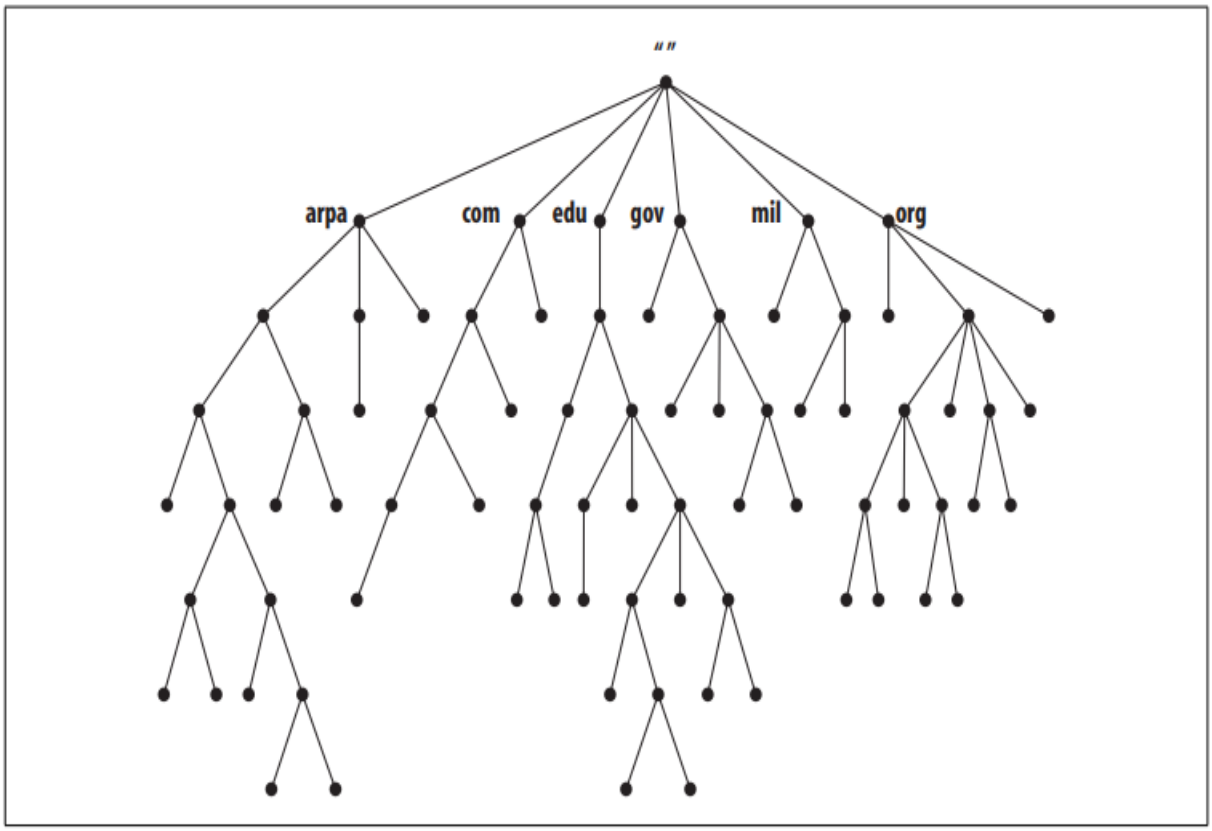

Figure 2.3: Structure of the DNS namespace
1

2.2.1 Domain Names

Each node in the tree has a text label without the dots of up to 63 characters. The root is given a null (zero-length) label. The full domain name of any node in the tree is the

label sequence from that node to the root. Domain names are always read from the node up the tree ("up"), with dots separating the names in the path.

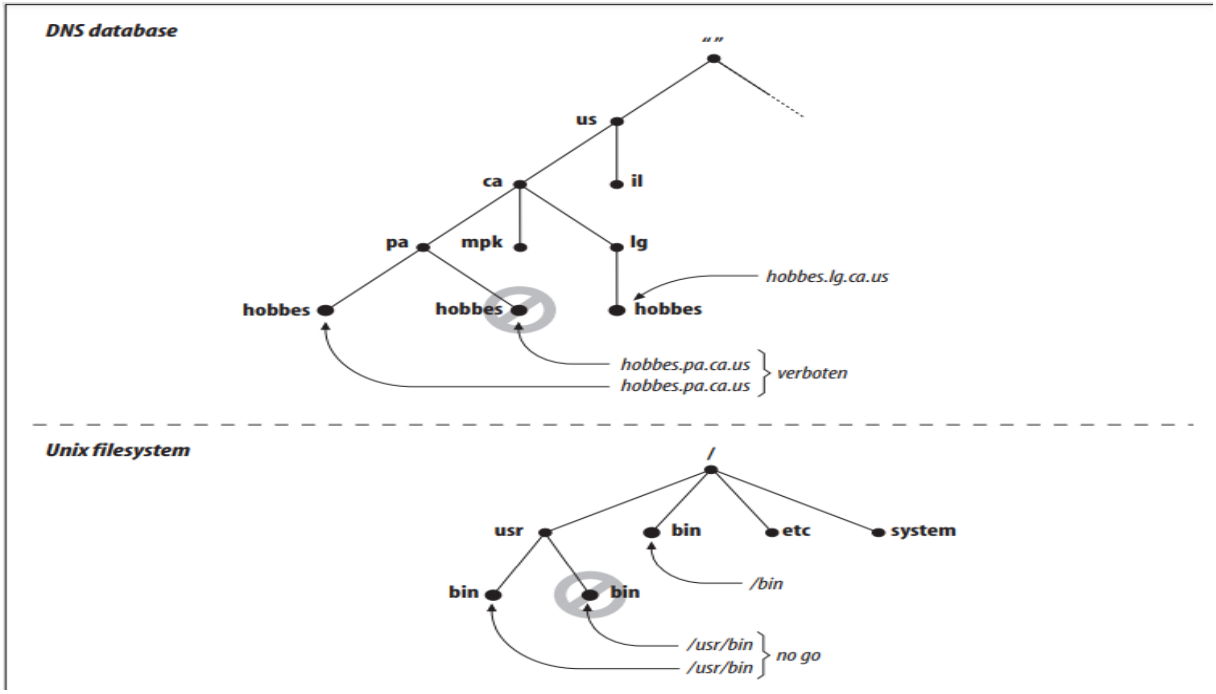

Figure 2.4: Uniqueness in the domain and path names
1

As shown in Figure, just as there cannot be two hobbes.pa.ca.us nodes in the namespace, there cannot be two /usr/bin directories. However, we can have both a hobbes.pa.ca.us node and a hobbes.lg.ca.us node, just as we can have a /bin directory and a /usr/bin directory.

2.2.2 The Internet Domain Namespaces

The existing Internet domain namespace, on the other hand, has some self-imposed structure. Domain names, particularly in the upper-level domains, abide to certain conventions (not rules, really, because they can be and have been broken). These customs help to keep domain names from appearing completely random. Understanding these traditions is extremely useful when attempting to decipher a domain name.

Top Level Domains

1. Generic Top Level Domains (gTLDs)
   The original top-level domains divided the Internet domain namespace organizationally into seven domains:

   com : Commercial organizations, such as Hewlett-Packard (hp.com), Sun Microsystems (sun.com), and IBM (ibm.com).

   edu : Educational organizations, such as U.C. Berkeley (berkeley.edu) and Purdue University (purdue.edu).

   gov : Government organizations, such as NASA (nasa.gov) and the National Science Foundation (nsf.gov)

   mil : Military organizations such as the United States Army (army.mil) and Navy (navy.mil)
   Previously, organizations such as NSFNET (nsf.net) and UUNET provided network infrastructure (uu.net). However, since 1996, net has been open to any commercial organization, just as com is.

   org : Previously non-profit organizations, such as the Electronic Frontier Foundation (eff.org). However, restrictions on org, like net, were lifted in 1996.

   int : NATO and other international organizations (nato.int).

   During the ARPAnet's transition from host tables to DNS another top-level domain called arpa was created. All ARPAnet hosts had hostnames beginning with arpa, making them easy to find. They had later moved into different subdomains of the organizational top-level domains.

2. Country Code Top level Domains (ccTLDs)
   To accommodate the growing internationalization of the Internet, the designers of the Internet namespace made concessions. Rather than requiring all top-level domains to describe organizational affiliation, they decided to allow geographical designations as well.

   To correspond to individual countries, new top-level domains were reserved (but not necessarily created). Their domain names adhered to an existing international standard known as ISO3166. * ISO3166 standardizes two-letter abbreviations for every country on the planet.

New top-level Domains

To accommodate the rapid expansion of the Internet and the need for more domain name "space," the Internet Corporation for Assigned Names and Numbers, or ICANN, established seven new generic top-level domains in late 2000. Some of the New (gTLDS) are :

aero : Sponsored; for the aeronautical industry,

biz : Generic

coop : Sponsored; for cooperatives ,Generic; for individuals

pro : Generic; for professionals

2.2.3 Delegation

The main goal of the design of the Domain Name System was to decentralize administration, and this is achieved through the Delegation.

Delegating domains is similar to delegating tasks at work. A manager may divide a large project into smaller tasks and assign responsibility for each to different employees. Similarly, an organization that manages a domain can subdivide it.

Each subdomain can be delegated to other organizations, which means that one organization is responsible for all data in that subdomain. It has complete control over the data and can even divide and delegate its subdomain. The parent domain retains only pointers to the subdomain's data sources in order to refer queriers there.

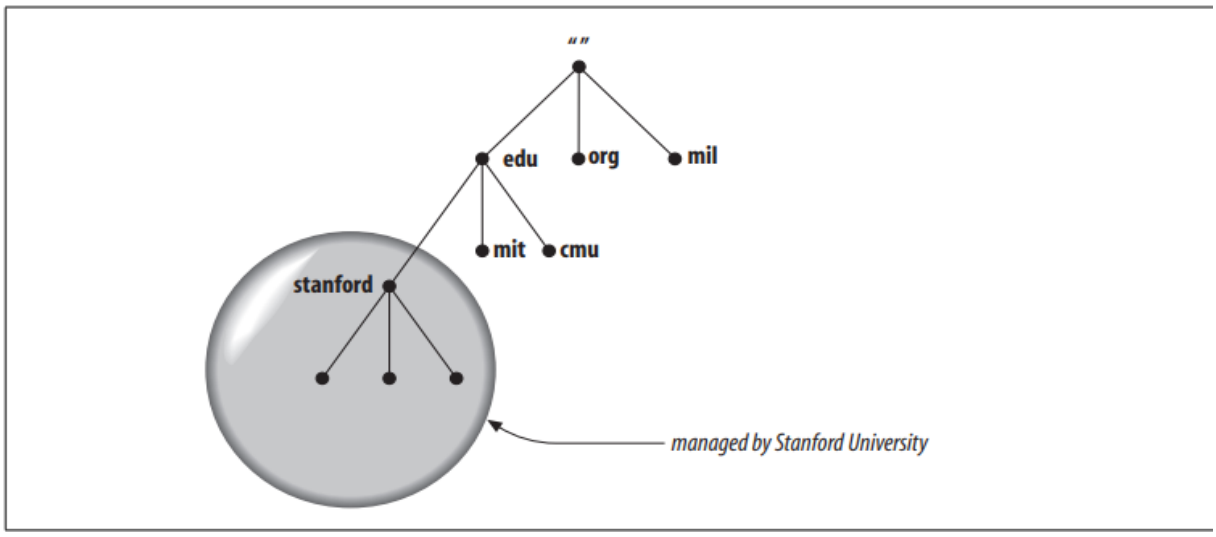

Figure 2.5: stanford.edu delegated to the Standford University

1

The domain stanford.edu, for example, is delegated to Stanford's network administrators/ who runs the university networks.

2.2.4 DNS Attacks/ Threats

DNS attacks, in which bad actors exploit vulnerabilities in the DNS Internet protocol, are extremely common—and expensive

DNS's role is to convert the term we type into a search box (known as the human-readable name) into the corresponding string of numbers (IP address) that our device requires to access a website or send an email. Attacks on these vital systems can be extremely damaging.

According to[IDC DNS Threat Report] (IDC, 2021), in survery of over 1,100 organizations in North America, Europe, and Asia Pacific in 2021, 87% had experienced DNS attacks. Each attack cost an average of $950,000 across all regions, and about $1 million for organizations in North America. The study also discovered a significant increase in data theft via DNS, with 26% of organizations reporting sensitive customer information stolen, up from 16% in 2020.

Common Types of DNS Attacks

DNS attacks pose a significant threat to the integrity and security of the DNS infrastructure. Understanding these attack methods is essential for implementing effective countermeasures. Here are some common types of DNS attacks:

1. **DNS Amplification Triggers DDoS Attacks:**

   DNS amplification is a type of distributed denial of service (DDoS) attack that exploits publicly accessible, open DNS servers to flood a target system with DNS response traffic. Attackers typically send DNS name lookup requests to open DNS servers, spoofing the target's address. When the DNS server sends the DNS record response, it is redirected to the target. Attackers request as much zone information as possible to maximize the amplification effect.

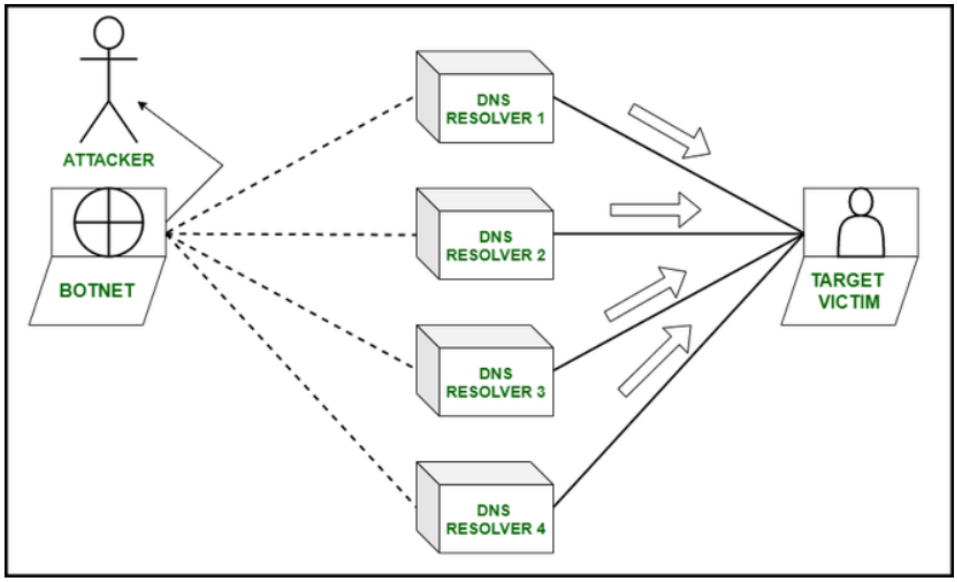

Figure 2.6: DNS Amplification Attack
14

2. **DNS Spoofing / Cache Poisoning:** DNS spoofing, also known as cache poisoning, allows attackers to take over DNS servers by exploiting vulnerabilities. They inject malicious data into DNS resolver cache systems, redirecting users to the attacker's sites for various malicious purposes. Attackers may steal personal information or data. DNS cache poisoning code is often distributed via spam or phishing emails that trick users into clicking malicious URLs.

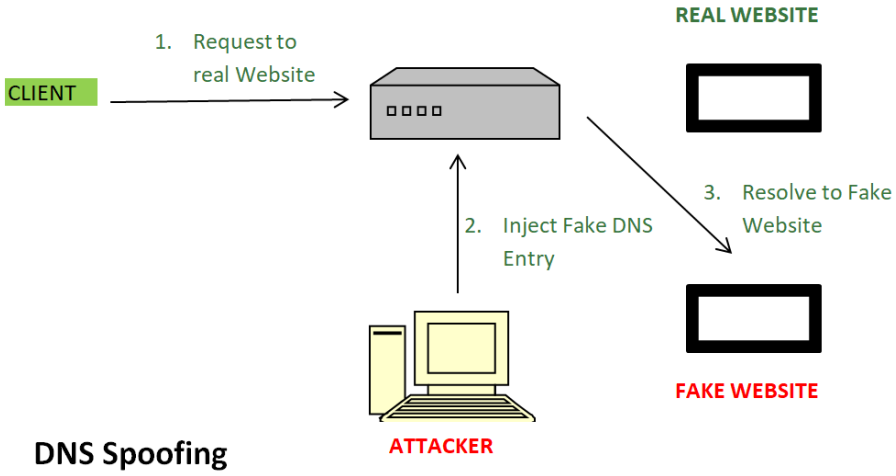

Figure 2.7: DNS Spoofing / Cache Poisoning
15

3. **DNS Tunneling:**

DNS tunneling is an old yet still prevalent DNS attack method that leverages the DNS protocol to tunnel malware and data via a client-server model. Attackers can control DNS servers, allowing them to bypass firewalls and exfiltrate data. This technique often relies on compromised systems' external network connectivity, granting access to internal DNS servers.

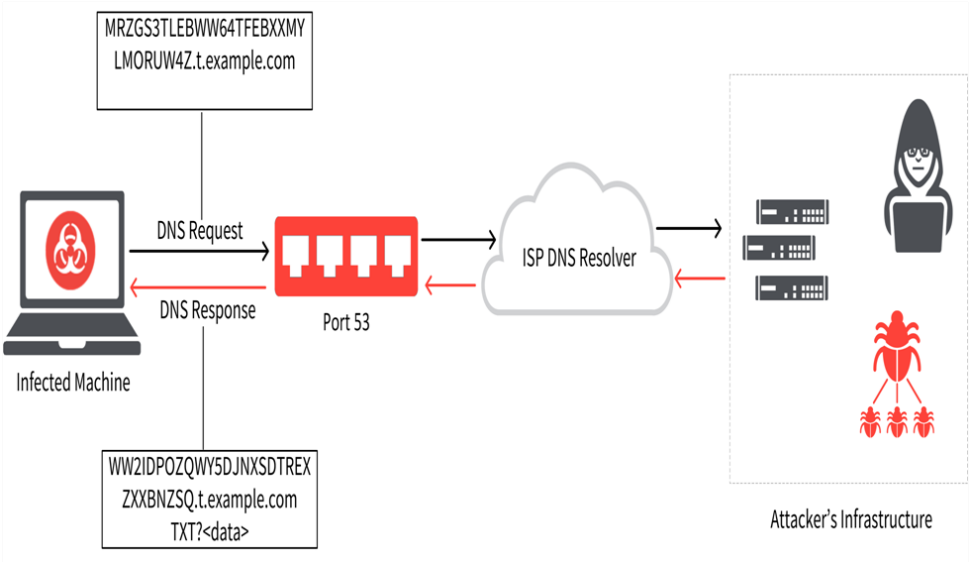

Figure 2.8: DNS Tunneling

16

4. **Fast Flux Evades Security Scans:**

Fast flux is a DNS evasion technique where attackers use botnets to hide phishing and malware activities from security scanners. It involves constantly changing IP addresses of compromised hosts acting as reverse proxies to the botnet master. This technique aims to associate a large number of IP addresses with a single legitimate domain, making detection challenging.

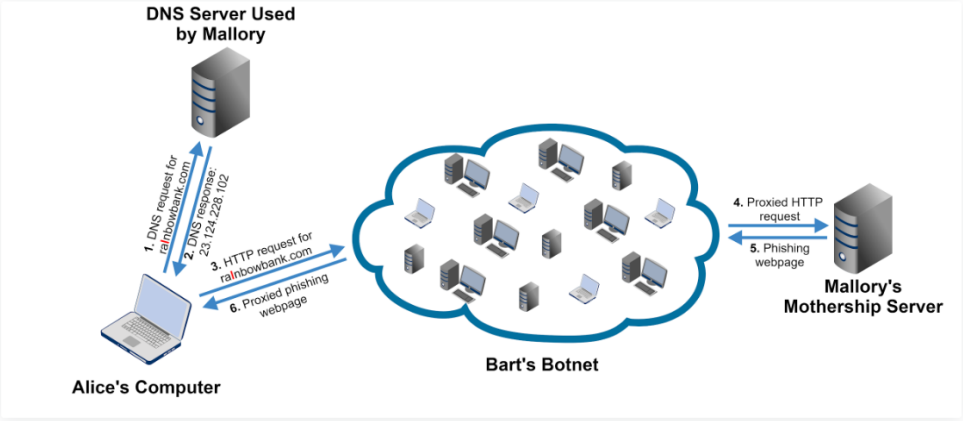

Figure 2.9: Fast Flux

17

5. **DNS Hijacking / Redirection:**

   DNS hijacking (or DNS redirection) occurs when malicious actors subvert DNS query resolution. They can do this by installing malware that alters a system's TCP/IP configuration to point to a rogue DNS server controlled by the attacker or by modifying the behavior of a trusted DNS server. These modifications are used for malicious purposes such as phishing.

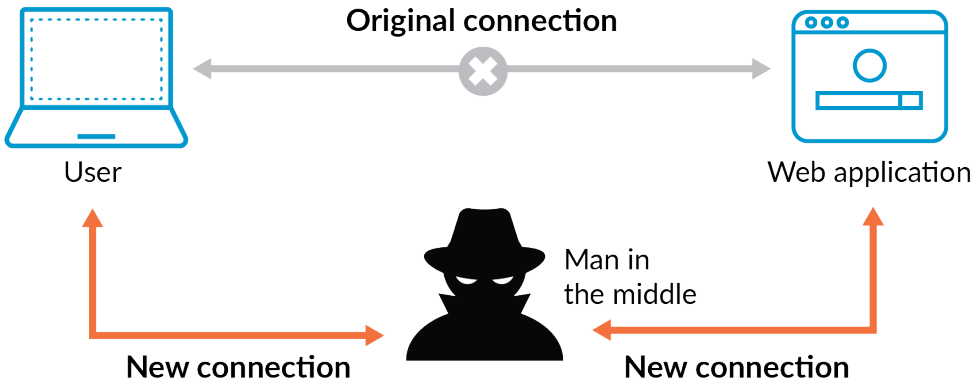

Figure 2.10: DNS Hijacking

18

Chapter 3

Solution For Passive Dns Data

3.1 Passive DNS (pDNS)

It is the reconstruction of DNS zone data through the recording and aggregation of live DNS queries and responses.

DNS messages exchanged between recursive DNS servers are captured and processed to produce DNS records. Passive DNS data could be collected without the involvement of zone administrators.

Furthermore, passive DNS data may be collected at various levels of the DNS name server hierarchy (e.g., **Root, TLD, and Authoritative name servers**), resulting in different perspectives on DNS data.

Passive DNS data might not represent the complete DNS structure as it only contains domain names that were actively queried through the Internet during the data collection period. Despite that, the aggregated DNS data form a big portion of the DNS traffic, which tends to be a reliable source of data.

Creating passive DNS replicas for research purposes necessitates advanced operations and a significant investment of time and money. Nonetheless, some researchers have released open-source frameworks for creating passive DNS databases to the community. Farsight Security also provides the Security Information Exchange framework (SIE), a secure framework for monitoring and analyzing DNS data in near real time.

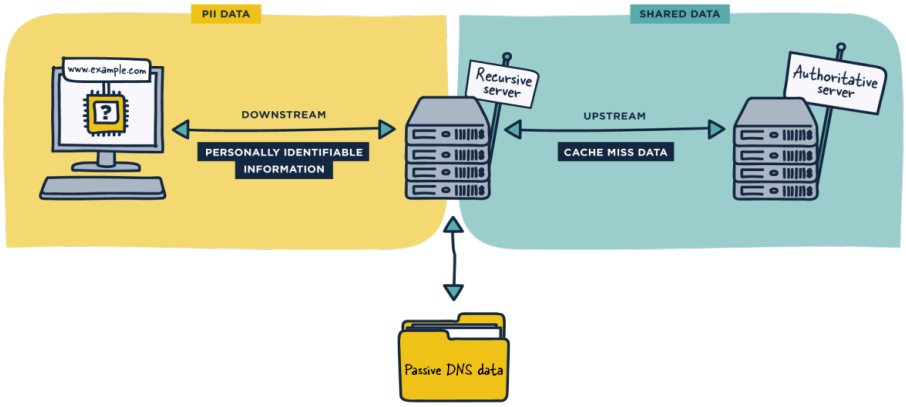

Figure 3.1: Passive DNS
19

3.2 Two Solutions Approached for collecting the Passive Dns Data

There are several ways for passive DNS replication to occur. A passive sensor can be used to sniff DNS traffic and record the responses. Alternatively, it can be added as a module to an existing network monitoring service, use it as a plugin with a name server, or extract data from previously saved network captures.

So we decided to use two different approaches:

1. D4 strategy way using the sensor. Adopting the approach outlined in the D4 Project's passive DNS tutorial Adopting the approach outlined in the [D4-Website] to collect passive DNS data. For better understanding the steps are done in below section 4.1

2. By using the [Circl Passive Dns Database] to collect the Pdns records from querying over 1 million domains . For better understanding the steps are also done in section 4.2

3.3 Understanding How the D4 Strategy Works

The D4 strategy aims to improve passive DNS collection diversity, enhance data sharing, simplify the collection process, and provide a distributed infrastructure for mixing and filtering DNS data sharing. The main components of the D4 strategy include:

1. **PassiveDNS**: This component captures DNS requests on an interface and pipes its standard output to `d4-goclient` (2).

2. **d4-goclient**: It encapsulates data from `stdin` (1) and sends it to the D4 server defined in the configuration file.

3. **D4-core server**: This server decapsulates D4 packets and pushes the content (PassiveDNS \n-separated records) to a list of D4 analyzer Redis queues.

4. **analyzer-d4-passivedns/bin/pdns-ingestion.py**: This script pops a specific D4 analyzer Redis queue and pushes the data into a Redis database that serves as the backend for the REST API.

5. **analyzer-d4-passivedns/bin/pdns-cof-server.py**: This script serves as the REST API, providing end-users access to the Passive DNS data.

The architecture overview of the D4 strategy can be depicted as follows:

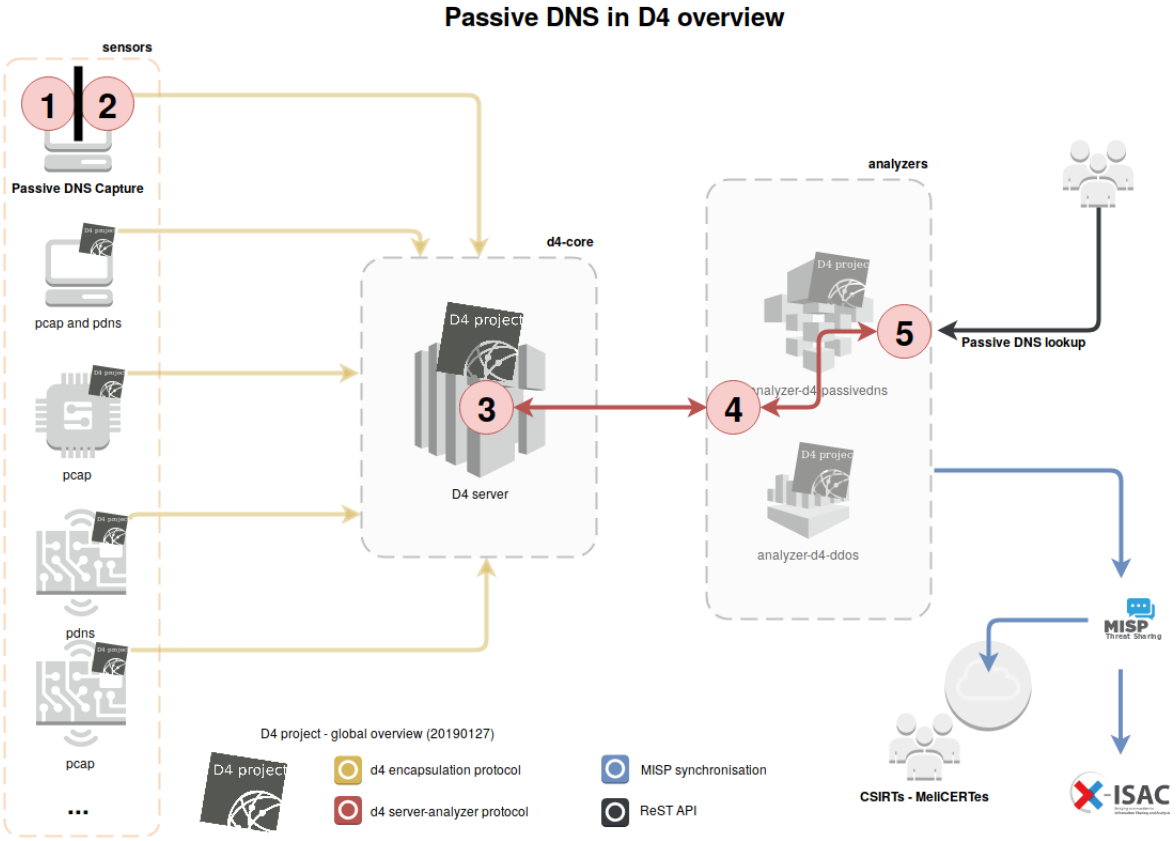

Figure 3.2: D4 Architecture Overview

Preparation:

To get started with D4-PassiveDNS,

Port Configuration:

The D4 setup requires several ports to be opened.

Table 3.1: Service Details

Service	Host IP	Host Port	Guest Port
D4 server - Admin Web Interface	127.0.0.1	7000	7000
D4 server - tls d4	127.0.0.1	4443	4443
D4 server - Passive DNS lookup	127.0.0.1	8400	8400

By understanding the D4 strategy and the architecture flow we are well-prepared to proceed with the steps for setting up our own Passive DNS instance using the D4 Project's approach.Later on we had implemented the steps on the Development 4.1.

3.4 Understanding the Second Solution of Collecting Passive DNS Using the CIRCL Database

In this section, we delve into the second approach of collecting Passive DNS data through the [Circl Passive Dns Database] and explore the process of accessing this valuable dataset.

CIRCL, the Computer Incident Response Center Luxembourg, provides an extensive Passive DNS database that offers insights into domain-to-IP mappings over time. The Passive DNS data is a result of the organization's continuous monitoring and data collection efforts. Researchers, analysts, and cybersecurity professionals can leverage this data to investigate historical relationships between domains and IP addresses, detect anomalies, and identify potential security threats.

3.4.1 Accessing the CIRCL Passive DNS Database

Gaining access to the [Circl Passive Dns Database] involves a straightforward process. Researchers and professionals interested in utilizing this dataset can request access from CIRCL by following established procedures. Once access is granted, users are provided with a unique username and password, enabling API access to the database. These credentials serve as the key to unlocking the wealth of historical DNS information within the CIRCL database.

3.5 Background

3.5.1 Ground Truth

In the context of passive DNS analysis, ground truth data is used to classify domain names as either benign or malicious where we get the idea from the [21] and [25]. There are several approaches to generating ground truth data for passive DNS analysis. One such approach is to use automated techniques that rely on publicly available domain name registration data . Another approach is to use passive DNS analysis techniques to detect malicious domains . Once the ground truth data has been generated, it can be used to train machine learning models that can accurately classify domain names as either benign or malicious.

3.5.2 Features Extraction

Table 3.2: An overview of domain features on basis of 25 26 27

Feature group #	**Feature Name**	**Description**
DNS Resolving Features	1	Number of distinct A records
	2	IP entropy of domain name
	3	Number of distinct NS records
	4	Similarity of NS domain name
Static Lexical Features	5	Length of domain name
	6	Max length of labels in subdomain
	7	Character entropy
	8	Number of numerical characters
	9	Ratio of numerical characters
	10	Max length of continuous numerical characters
	11	Max length of continuous alphabetic characters
	12	Max length of continuous same alphabetic characters
	13	Ratio of vowels
	14	Max length of continuous consonants

In the domain of passive DNS analysis, the process of feature extraction plays a pivotal role in understanding and classifying domain names as either benign or malicious. Feature

extraction involves the transformation of raw domain name data into a set of meaningful characteristics or features that can be leveraged for further analysis and classification.

The features extraction process is a critical step in the pipeline of analyzing passive DNS data, and it serves as the foundation for subsequent machine learning tasks. Here, we elucidate the various features extracted from the domain names and their significance:

1. **Number of distinct A records**: This feature counts the number of distinct A records associated with a domain name, where A records are linked to IPv4 addresses.

2. **IP entropy of domain name**: IP entropy measures the diversity of IP address suffixes associated with a domain name.

3. **Number of distinct NS records**: NS records pertain to name servers responsible for a domain. This feature counts the distinct NS records associated with a domain name.

4. **Similarity of NS domain names**: This metric quantifies the similarity or dissimilarity of name server records within a domain, helping identify patterns.

5. **Length of domain name**: This feature represents the length of a domain name, calculated by counting the number of characters in the name while excluding the dots separating subdomains. It provides insight into the overall length of the domain.

6. **Max length of labels in subdomain**: This feature identifies the maximum length of subdomain labels within a domain name. Longer labels may indicate a more complex subdomain structure.

7. **Character entropy**: Character entropy measures the diversity or randomness of characters within a domain name. It quantifies the unpredictability in character selection.

8. **Number of numerical characters**: This feature counts the total number of numerical characters within a domain name, which can be relevant in certain analyses.

9. **Ratio of numerical characters**: Calculated by dividing the number of numerical characters by the total domain length, this ratio indicates the prevalence of numeric values relative to the domain's length.

10. **Max length of continuous numerical characters**: This feature identifies the longest consecutive sequence of numerical characters within a domain name.

11. **Max length of continuous alphabetic characters**: Similar to the previous feature, this one identifies the longest consecutive sequence of alphabetic characters within a domain name.

12. **Max length of continuous same alphabetic character**: This feature identifies the longest continuous sequence of the same alphabetic character within a domain name.

13. **Ratio of vowels**: This ratio is calculated by counting vowels (AEIOUaeiou) and dividing by the total domain length, providing insight into linguistic aspects.

14. **Max length of continuous consonant characters**: Similar to features related to alphabetic characters, this one identifies the longest consecutive sequence of consonant

characters.

These extracted features provide the foundation for subsequent analysis and classification tasks in passive DNS analysis. By quantifying various aspects of domain names, they enable machine learning models to effectively distinguish between benign and malicious domains. Together with ground truth data, these features contribute to the development of accurate classification models.

3.5.3 Machine Learning

Machine learning is a branch of computer science that focuses on developing methods for systems to learn from data. The primary goal of most machine learning approaches is generalization: the ability to perform accurately on new and previously unseen examples based on what was learned from the training set.

Machine learning and data mining are two closely related scientific fields with one major difference: while machine learning uses known properties extracted from training sets, data mining focuses on extracting previously unknown features from training sets.

Machine learning algorithms are classified into three categories: supervised learning, unsupervised learning, and reinforcement learning. Each sample in the training data contains the corresponding target value in the case of supervised learning (labelled samples). The task is called classification if the goal is to assign a category (target) to each input sample from a finite set of categories.

A regression task, on the other hand, is one in which the goal is to obtain a description of one or more continuous variables. Unsupervised learning uses samples from the training set that do not have a target value and can be used for a variety of tasks like clustering, density estimation, and visualization. Finally, reinforcement learning is associated with tasks in which the goal is to devise a set of actions that allow for the maximization of some concept of reward.

Semi-supervised learning can also be achieved by combining supervised and unsupervised learning algorithms. The goal of such algorithms is not generalization, but rather understanding the structure of data from samples in the training set that do not have target values.

For the purposes of this study, we can use scikitlearn, a Python machine learning library. This library includes a variety of machine learning algorithms as well as a robust API for testing and analyzing results.

3.5.4 Algorithms Overview

In this study, we solve our problem using various supervised learning approaches, including **k-Nearest Neighbors (kNN), Decision Tree classifiers, Random Forests, Gaussian Naive Bayes (GNB), Linear Discriminant Analysis (LDA), Logistic Regression, and Support Vector Machines (SVM).**

k-Nearest Neighbors (kNN)

The k-Nearest Neighbors (kNN) algorithm is suitable for classification, regression, and clustering tasks. In our scenario, each sample in the training set is assigned a positive or negative category (or label). We will use the kNN algorithm as a classification task because our goal is to be able to classify unknown samples according to those categories. Furthermore, each sample in our training set contains the 36 feature values associated with a specific domain.

KNN works by calculating the distances between a query and all of the examples in the data, then selecting the number of examples (K) closest to the query and voting for the most frequent label (in the case of classification) or averaging the labels (in the case of regression).

Decision Trees

Decision Trees build models that can classify a given sample by extracting a set of decision rules from the feature sets of the samples in the training data. Every node in the tree can be thought of as an if-then-else decision node. The conditional test is applied to the entire set of features and their respective value ranges.

As a result, to classify a given sample, one begins by testing the feature value of the root node and proceeds down the tree, following the branches that correspond to the feature value. This process is then repeated for the sub-tree that begins at the new node and continues until it reaches a leaf node.

Random Forest

In addition, we can run the Random Forests algorithm. This algorithm generates several Decision Trees, each of which is trained on a subset of the entire training set. The samples for each training set are chosen using sampling with replacement (where a set may contain an element more than once).

Furthermore, at each node, only a random subset of features is available, in order to avoid that the trees become correlated. The result of predicting a given sample is then

given by the average prediction of the individual decision trees.

Gaussian Naive Bayes (GNB)

Gaussian Naive Bayes (GNB) is a probabilistic algorithm based on the Bayes' theorem. It assumes that features are normally distributed and independent within each class. GNB is often used for classification tasks, especially when dealing with continuous data.

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique that aims to find a linear combination of features that best separates different classes in the dataset. It's commonly used for classification tasks where the goal is to maximize the separation between classes.

Logistic Regression

Logistic Regression is a regression analysis method that is used for binary classification problems. It models the probability that a given sample belongs to a particular class based on a linear combination of the input features.

Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful algorithms used for both classification and regression tasks. They aim to find a hyperplane that best separates different classes in the dataset while maximizing the margin between classes.

These algorithms offer a diverse set of approaches to solving classification problems, and their performance will be evaluated comprehensively in this study.

3.6 Dataset Construction Architecture

Creating a robust dataset is a fundamental step in the realm of passive DNS analysis. This dataset serves as the cornerstone for training and evaluating machine learning models to discern between benign and malicious domain names. The construction of this dataset follows a meticulous process that ensures data integrity and relevance.

The journey begins with leveraging the CIRCL Passive DNS Database, a reputable source of historical passive DNS records. To compile our dataset, we initiate an orchestrated data collection procedure by interfacing with the CIRCL Passive DNS Database through its Application Programming Interface (API).

Our data collection endeavor focuses on the top 1 million domains, a representative sample of internet entities. Through the API, we extract the historical passive DNS records associated with each of these domains. These records are typically presented as a list of objects, which we meticulously parse and convert into a structured format, specifically, a list of dictionaries. This transformation enhances data accessibility and enables efficient extraction of pertinent information.

Once we have successfully extracted the relevant fields from the passive DNS records obtained from the CIRCL Passive DNS Database, we proceed to the next critical phase of dataset construction: feature extraction. As detailed earlier in this report, this process involves deriving meaningful features from domain names that contribute to our analysis, classification, and understanding of domain behavior.

Our dataset construction journey culminates with the integration of ground truth data. To label domains as either malicious or benign, we rely on ground truth data sources. These sources provide us with the essential context needed to categorize domains accurately.

In summary, our Dataset Construction Architecture is a multi-faceted process that encompasses data collection from the CIRCL Passive DNS Database via API, structured data transformation, feature extraction, and the integration of ground truth labels. The result is a meticulously crafted dataset that serves as the foundation for rigorous passive DNS analysis, enabling the development of accurate and reliable classification models.

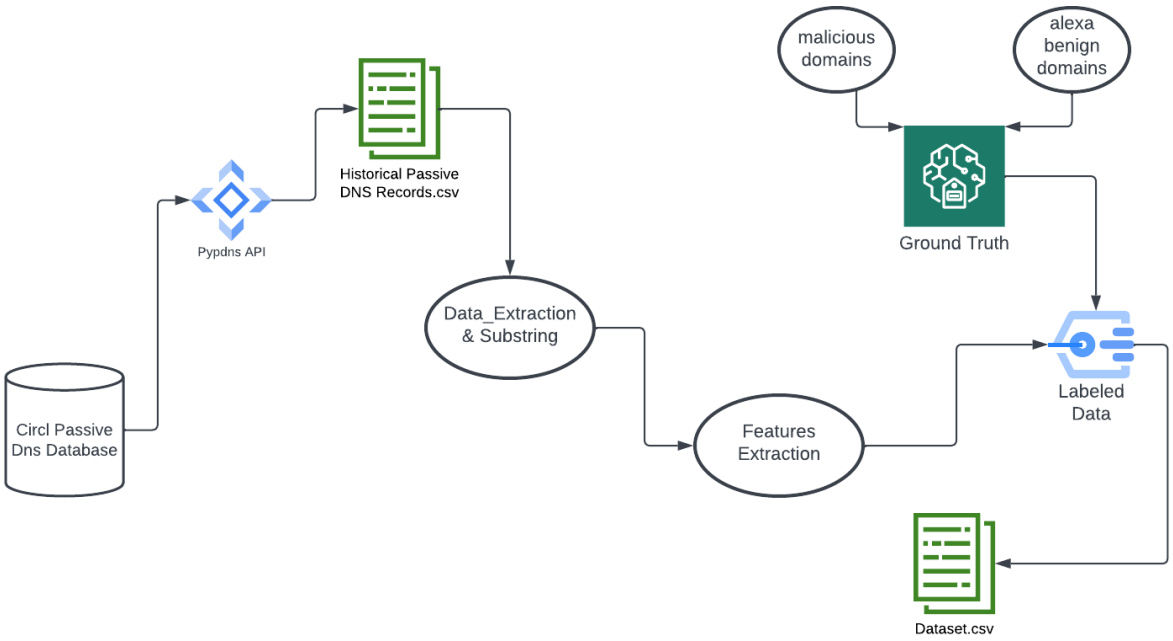

Figure 3.3: Dataset Architecture Overview

Chapter 4

Development

In this chapter, we should describe the implementation, highlighting the most important aspects, the difficulties and the technical solutions that were followed. In particular, if code from others was used (available as open-source), should be easily identified.

4.1 Installation of Sensor-Based Passive DNS tool and D4 tools

[passivedns] is a popular tool used to capture and record DNS traffic passively for further analysis and monitoring.In this section, we will outline the approach for setting up a sensor-based Passive DNS solution using the [passivedns] tool on Ubuntu.

4.1.1 Install Dependencies

Before installing the [passivedns] tool, make sure to have the necessary dependencies. Open a terminal and run the following command to install the required packages:

```
$ sudo apt-get install git-core binutils-dev libldns1 libldns-dev libpcap-dev libdate
```

The ‘libdate-simple-perl‘ package is also needed for ‘pdns2db.pl‘, which is used for processing the captured data.

4.1.2 Clone and Compile ‘passivedns‘

Once the dependencies are installed, proceed to clone the [passivedns] repository from GitHub and compile the tool. Execute the following commands in the terminal:

```
$ git clone git://github.com/gamelinux/passivedns.git
$ cd passivedns/
$ autoreconf --install
$ ./configure
$ make
```

The ‘autoreconf –install‘ command is used to generate the necessary build files for the compilation process.

4.1.3 Run `passivedns`

After successful compilation, we can now run the [passivedns] tool to start capturing DNS traffic passively. Use the following command:

```
$ sudo ./passivedns -i ens18 -l /dev/stdout
```

The `-i` option specifies the network interface from which we want to capture DNS traffic (replace `ens18` with the appropriate interface on our system). The `-l` option sets the log file to `/dev/stdout`, which means that the DNS traffic will be printed to the console.

For better usability and convenience, we can add the `passivedns` binary to the `$PATH` in `/usr/local/bin`, which is a directory in the system's default executable search path. This allows us to run `passivedns` from any location in the terminal without specifying its full path.

To add `passivedns` to the `$PATH` in `/usr/local/bin`, follow these steps:

```
$ sudo cp passivedns /usr/local/bin
$ sudo chmod +x /usr/local/bin/passivedns
```

With `passivedns` added to the `$PATH`, we can now run it from any directory without needing to specify its full path, as shown in Step 3.

By following these steps, we will have successfully set up and made `passivedns` readily available for passive DNS traffic capture and analysis on our system.

4.1.4 Analyze the Captured Data

Once ‘passivedns‘ is running, it will passively capture DNS traffic on the specified network interface. We can now analyze the captured data using various tools or scripts to gain insights into DNS queries and responses.

By following these steps, we will have successfully set up a sensor-based Passive DNS solution using the ‘passivedns‘ tool on Ubuntu. This will allow us to monitor DNS traffic passively and analyze it for various security and network analysis purposes.

The approach for setting up a sensor-based passive DNS solution involves deploying the[D4-core] server and using the provided *passivedns* sensor [d4-goclient] . The sensor utilizes Go programming language and has been thoroughly tested on Go version 1.13. The following steps outline the process of setting up our own Passive DNS sensor network and database using open-source software.

4.1.5 Installing D4-core Server

To begin, we need to install the [D4-core] server, which acts as a complete server for handling clients (sensors). This server is responsible for decapsulation of the D4 protocol, controlling sensor registrations, managing decoding protocols, and dispatching data to the appropriate decoders and analyzers.

To install the [D4-core] server, follow these steps:

1. Change the current working directory to the *server* folder.
2. Execute the *install_server.sh* script provided in the *server* directory to install the D4 server.
3. Create or add a PEM (Privacy Enhanced Mail) file in the *d4-core/server* directory. This file is crucial for secure communication.
4. Generate the PEM file and root certificate by running the *gen_root.sh* and *gen_cert.sh* scripts in the *gen_cert* directory.
5. After generating the certificates, go back to the root *server* directory.
6. Launch the D4 server using the *LAUNCH.sh* script with the ’-l’ option.

Upon successful launch, the D4 server web interface becomes accessible via *http://127.0.0.1:7000/*.

4.1.6 Updating Web Assets

If there is a need to update JavaScript libraries used in the web interface, follow these steps:

1. Change the current working directory to the *web* folder within the D4 server directory.

2. Run the *update_web.sh* script to update the JavaScript libraries.

By following these steps, the [D4-core] server will be installed and ready to handle clients (sensors) for passive DNS data collection. The sensor-based approach leverages the *passivedns* sensor [D4-goclient], making it possible to set up a comprehensive and efficient Passive DNS network using open-source components.

4.1.7 Installing d4-goclient Sensor

Next, we install the [D4-goclient] sensor, which is a D4 project client implementing the D4 encapsulation protocol. This sensor can be used on different targets and architectures to collect network capture, logs, and specific network monitoring data and send it back to a D4 server.

To install 'd4-goclient', follow these steps:

1. Fetch the 'd4-goclient' code and its dependencies by running the following command:

```
go get github.com/D4-project/d4-goclient
```

2. We make sure to have the required dependencies, which include 'golang' version 1.13 or higher (tested).

4.1.8 Configuration

After installing [D4-goclient], we need to configure the client to connect it to our D4 server and specify its behavior. The client's configuration can be stored in a folder containing the following files:

If `source` is set to `d4server` , two additional files are required:

- `redis_queue`: Redis queue in the form `analyzer:typeofqueue:queueuuid`, e.g., `analyzer:3:d42967c1-f7ad-464e-bbc7-4464c653d7a6`.

- `redis_d4`: Redis server location in the format `location:port/database`, e.g., `localhost:6385/2`.

Table 4.1: D4-goclient Parameters

Parameter	Description	Value
`destination`	Address and port of the receiving D4 server	127.0.0.1:4443
`source`	Where to look for input data	stdin
`snaplen`	D4 packet size	4096
`type`	Type of D4 packets sent	8
`uuid`	Sensor's unique identifier	Automatically provisioned
`key`	A Pre-Shared Key for authentication	"private key to change"
`version`	D4 protocol version	1

4.1.9 Running the d4-goclient Sensor

With the [D4-goclient] sensor properly configured, run the client to start collecting passive DNS data and sending it to the D4 server. Use the following command:

```
d4-goclient --config /path/to/config/folder
```

An Example to run passivedns and piping through the d4-goclient sensor with its config folder

```
$ sudo passivedns -i ens18 -l /dev/stdout | d4-goclient -c conf.sample/
```

The ‘d4-goclient‘ sensor will now capture the specified data and send it back to the D4 server for further analysis and processing.

By following these steps, we will have successfully installed and configured the ‘d4-goclient‘ sensor, enabling us to collect and transmit passive DNS data to the D4 server for analysis.

4.1.10 Cloning the analyzer-d4-passivedns

To perform advanced analysis on the Passive DNS data captured by the D4 network sensor, we will use the [analyzer-d4-passivedns] tool. This analyzer is designed to work with D4 sensors and can process data in passivedns CSV format (more formats to come) produced by D4 sensors. It can also process data independently from D4 using [COF] (Common Output Format) WebSocket streams.

Features:

- **Input Stream from D4 Servers:** The analyzer can be connected to one or more D4 servers to receive a stream of DNS records.
- **Input Stream from Websocket or File:** It can process a WebSocket stream (or a file stream) in NDJSON [COF] format.
- **Output API:** The analyzer provides a compliant Passive DNS ReST server that adheres to the Common Output Format.
- **Flexible Configuration:** The analyzer can be configured to collect the required records from DNS records based on specific criteria.

Requirements:

To run the [analyzer-d4-passivedns], we make sure to have the following requirements installed Python 3.8, Redis version >5.0 or kvrocks , Tornado and iptools.

Installation:

Redis:

To install Redis, run the following command:

```
$ ./install_server.sh
```

So we are choosing the redis way for it.
All the Python 3 code will be installed in a virtual environment (PDNSENV).
Kvrocks:
To install Kvrocks, execute the following command:

```
$ ./install_server_kvrocks.sh
```

All the Python 3 code will be installed in a virtual environment (PDNSENV).

Running:

Before starting the analyzer, ensure that the Redis server or Kvrocks server is running. Don't forget to set the DB directory in the `redis.conf` configuration. By default, the Redis for Passive DNS is running on TCP port 6400.

To start the Redis server:

```
$ ./redis/src/redis-server ./etc/redis.conf
```

Or to start the Kvrocks server:

```
$ ./kvrocks/src/kvrocks -c ./etc/kvrocks.conf
```

Next, start the Passive DNS COF server:

```
$ . ./PDNSENV/bin/activate
$ cd ./bin/
$ python3 ./pdns-cof-server.py
```

Feeding the Passive DNS Server:

There are two ways to feed data to the Passive DNS server. We can choose to combine multiple streams. A sample public [COF] stream is available from CIRCL with newly seen IPv6 addresses and DNS records.

Via COF WebSocket Stream: Start the importer:

```
$ python3 pdns-import-cof.py --websocket ws://crh.circl.lu:8888
```

Via D4: Configure and start the D4 analyzer:

```
$ cd ./etc
$ cp analyzer.conf.sample analyzer.conf
```

Edit the `analyzer.conf` to match the UUID of the analyzer queue from our D4 server. Here we have to manually add a analyzer queue with type 8 and sensor uuid

```
[global]
my-uuid = theonethatisgenerated-on-d4-web-interface
d4-server = 127.0.0.1:6380
# INFO|DEBUG
logging-level = INFO
[expiration]
16 = 24000
99 = 26000
[exclude]
substring=spamhaus.org,asn.cymru.com
```

Then we can start the analyzer, which will fetch the data from the analyzer, parse it, and populate the Passive DNS database:

```
$ . ./PDNSENV/bin/activate/
$ cd ./bin/
$ python3 pdns-ingestion.py
```

By following these steps, we will have successfully cloned and set up the [analyzer-d4-passivedns] tool to analyze the Passive DNS data captured by the D4 network sensor and populate the Passive DNS database for further analysis. After that we will have port 8400

opened on 127.0.0.1:8400 where we can manually make query by http://127.0.0.1:8400/query/google.com from our web-browser or

```
$ curl -s http://127.0.0.1:8400/query/google.com
OR
$ curl -s http://127.0.0.1:8400/query/2a02:250:0:8::53(it's the specific rdata)
```

4.1.11 Passive Dns Records from the D4-server

When collecting passive DNS records using D4, the records are obtained in the form of a Passive DNS CSV stream. This stream contains information about DNS queries and responses captured passively from network traffic. Each record in the stream represents a DNS event and includes various fields such as the timestamp of the first occurrence of the event (`time_first`), the timestamp of the last occurrence of the event (`time_last`), the number of times the event occurred (`count`), the type of DNS record (`rrtype`), the DNS record name (`rrname`), the data associated with the DNS record (`rdata`), and the origin URL (`origin`).

Once the Passive DNS CSV stream is collected and processed by the `analyzer-d4-passivedns`, the `pdns-ingestion.py` script is launched. This script resolves all the passive DNS records corresponding to the domains mentioned in the records. It fetches additional information related to the DNS records, such as IP addresses associated with the domain names, and stores the resolved data in a Redis database. This database serves as the backend for the passive DNS REST API.

As a result, after running the `pdns-ingestion.py` script, the passive DNS records are fully resolved, and the complete information about the DNS events, including timestamps, record types, record names, associated data, and origin URLs, becomes accessible through the passive DNS REST API. Users can query this API to access and retrieve specific information about the resolved DNS records.

The table provided below summarizes the fields present in the passive DNS records obtained from the D4 system, along with their descriptions. These fields play a crucial

role in providing comprehensive information about the DNS events captured passively, facilitating further analysis and investigation.

Table 4.2: Passive DNS Records

Field	Description
time_first	Time of the first occurrence
time_last	Time of the last occurrence
count	Number of occurrences
rrtype	Type of DNS record
rrname	DNS record name
rdata	Data associated with the DNS record
origin	Origin URL

4.2 Features Extraction

In this section, we discuss the functions used to extract various features from the DNS records dataset. These features provide valuable insights into the characteristics of domain names and their associated DNS records. The following function codes can be seen on appendix [F].

4.2.1 Loading the DNS Records

We begin by loading the DNS records data from a CSV file into a DataFrame, which serves as the basis for our feature extraction.

4.2.2 Function: `get_records(domain)`

This function retrieves DNS records for a given domain from the dataset. It filters the DataFrame to select records associated with the specified domain.

4.2.3 Function: `number_distinct_A_records(domain)`

This function calculates the number of distinct A (IPv4) records for a domain. It identifies A records and counts unique IP addresses associated with them.

4.2.4 Function: `distinct_A_records(domain)`

This function extracts distinct A (IPv4) records for a domain. It compiles a list of unique IP addresses associated with A records.

4.2.5 Function: `ip_entropy_domain_name(domain)`

This function calculates the IP entropy for a domain. It measures the diversity of IP address suffixes in the A records, providing insights into IP address distribution.

4.2.6 Function: `distinct_NS_records(domain)`

This function determines the distinct NS (Name Server) records for a domain. It identifies and returns unique NS records associated with the domain.

4.2.7 Function: `get_NS_records(domain)`

This function extracts NS (Name Server) records for a given domain. It retrieves and returns unique NS records associated with the domain.

4.2.8 Function: `number_distinct_NS_records(domain)`

This function calculates the number of distinct NS (Name Server) records for a domain. It counts unique NS records associated with the domain.

4.2.9 Function: `similarity_NS_domain_name(domain)`

This function computes the similarity of NS domain names. It calculates the edit distance between pairs of distinct NS records, providing a measure of similarity among them.

4.2.10 Additional Domain Characteristics

The following functions calculate various characteristics of domain names:

- `len_domain(domain)`: Determines the size of a domain name by removing dots.

- `max_len_labels_subdomain(domain)`: Calculates the maximum length of labels (subdomains) in a domain.
- `character_entropy(domain)`: Measures the character entropy of a domain, indicating the diversity of characters.
- `number_numerical_characters(domain)`: Counts the number of numerical characters in a domain.
- `ratio_numerical_characters(domain)`: Calculates the ratio of numerical characters to domain size.
- `max_len_cont_num_chars(domain)`: Computes the maximum size of continuous numerical characters in a domain.
- `max_len_cont_alpha_chars(domain)`: Determines the maximum size of continuous alphabetic characters in a domain.
- `max_len_cont_same_alpha_chars(domain)`: Calculates the maximum size of the same continuous alphabetic character in a domain.
- `ratio_vowels(domain)`: Computes the ratio of vowels in a domain.
- `max_length_continuous_consonants(domain)`: Calculates the maximum size of continuous consonant characters in a domain.

4.2.11 Storing Results

We create a list of unique domains from the dataset and use these functions to calculate the respective features for each domain. The results are stored in a DataFrame and saved to a new CSV file named "results.csv" for further analysis.

4.3 Collection of Malicious Domains

In this section, we describe the collection of malicious domains used in our analysis.

4.3.1 Malicious Domain Collection and Verification

To obtain a comprehensive list of malicious domains for our analysis, we utilized the Blocklist Project, which provides a repository of curated domain blocklists. Specifically, we accessed the repository at the following [blocklistproject]

By leveraging the Blocklist Project, we were able to access a substantial dataset of malicious domains. This dataset comprises 1,284,988 domains categorized into various malicious categories, including Abuse, Fraud, Drugs, Malware, Phishing, and Ransomware.

All the malicious domains from these categories were aggregated and saved into a single file named "mal_col.txt." This file serves as a valuable resource for our research, enabling us to examine and analyze the characteristics of these malicious domains in depth.

The malicious domain dataset is of particular significance for our analysis, as it constitutes a substantial portion of the historical Passive DNS records, totaling 176,495 domains, which we used for our research. The utilization of these malicious domains allows us to focus our analysis on identifying and categorizing malicious activities effectively.

To verify the presence of these malicious domains within our historical Passive DNS records, we employed a Python script. This script compared the domains in "mal_col.txt" with the domains in "completed_1m.csv," which contained the historical Passive DNS records. It found that 2,607 malicious domains matched with those in our dataset and we saved this matched output in new file called ["malicious_matchings_comp.txt"] But, we identified 1,015 malicious domains that matched when using the ["malicious_matchings_comp.txt"] file when compared to the unique filtered balanced data ("A" and "NS" records).

These matching malicious domains were saved in a separate file named ["malicious_matchings_comp
This verification process ensures that our dataset is accurately enriched with the identified malicious domains, providing a reliable foundation for our research and analysis.

4.4 Collection of Benign Domains

In this section, we describe the collection of benign domains used in our analysis.

Before proceeding to label the domains, we expanded our dataset to include both benign and malicious domains. We utilized the following here

[dga_domains_full.csv [11]]

This dataset contains both DGA (Domain Generation Algorithm) and legitimate Alexa benign domains. In our unique (non-repetitive) filtered dataset consisting of "A" and "NS" records, we found 11,735 legitimate/Alexa benign domains that matched this dataset.

4.5 Labeling the Domains

With our filtered dataset in place, we were poised to label the domains as either "malicious" or "benign." This comprehensive labeling process involved two distinct sources:

1. Malicious Domains: We utilized the "mal_col.txt" dataset, which contains a substantial collection of malicious domains. Any domain from our historical Passive DNS records found within this dataset was unequivocally labeled as "malicious."

2. Benign Domains: We expanded our dataset by comparing it with the [dga_domains_full.csv [11]] dataset, which includes both legitimate Alexa benign domains and DGA (Domain Generation Algorithm) domains. Initially, we identified legitimate Alexa benign domains from this dataset and marked them as "legitimate." Subsequently, these domains were further categorized as "benign" within the filtered dataset CSV.

In our unique (non-repetitive) filtered dataset, comprising "A" and "NS" records, we identified 11,735 domains categorized as "benign" through this process. Domains not present in either the malicious or benign datasets were marked as "unlabeled."

This comprehensive labeling process resulted in a dataset that includes domains categorized as "malicious," "benign," or "unlabeled." The labeled dataset is now ready for machine learning analysis, forming the foundation for our subsequent research and analysis, enabling us to apply various machine learning techniques to identify and classify domain activities effectively.

4.6 Dataset Construction

In this section, we elucidate the process of constructing our dataset for subsequent machine learning analysis. The construction of this dataset involves accessing the CIRCL Passive DNS database, employing the [PyPDNS] Python library for querying Passive DNS records, extracting relevant data, and preparing it for analysis.

4.6.1 Utilizing the PyPDNS Library

To access the CIRCL Passive DNS database programmatically, we developed a Python library called [PyPDNS]. This library seamlessly interfaces with the CIRCL Passive DNS API, allowing us to query Passive DNS records that adhere to the Passive DNS - Common Output Format. With PyPDNS, we can efficiently retrieve Passive DNS data for research and analysis.

4.6.2 Data Extraction and Transformation

Our data extraction process comprises multiple steps. We initially acquired a list of the top 1 million domains for analysis, provided in CSV format. Given the scale of this task and the need to manage a substantial number of domains, we adopted a domain list

splitting strategy. This strategy involved dividing the 1 million domains into smaller, more manageable segments of $250,000$ domains each. The division was executed using a Python script, which is detailed in appendix [B].

To ensure the smooth retrieval of Passive DNS records without exceeding query quotas, we implemented a cautious approach. We introduced a sleep interval of 3 seconds between each domain query for the first two Python scripts and reduced it to a 2-second delay for the subsequent two scripts, as documented in appendix [C]. This approach helped maintain compliance with query limits and minimized the risk of disruptions during the querying process.

4.6.3 Extraction of Passive DNS Records

After successfully obtaining the Passive DNS data for our list of domains, we were left with a dataset containing valuable information in a specific format. The data for each domain was presented as a string, encapsulating multiple PDNS (Passive DNS) records within square brackets. Each PDNS record includes various fields such as "rrname," "rrtype," "rdata," "time_first," and "time_last."

To extract and organize this data effectively, we developed a Python tool. This tool parses the PDNS records using substring operations and regular expressions, ultimately converting the data into a structured list of dictionaries. This list is composed of individual records, each associated with a domain, its corresponding PDNS information, and relevant timestamp details.

The extracted data was then saved into a CSV file named ["extracted_data_comp.csv"]. This file serves as a valuable resource containing the organized PDNS records, ready for further analysis. Which we can see it done on appendix [D].

4.6.4 Filtering Relevant Data

In preparation for feature extraction, we needed to filter the dataset to include only the records relevant to our analysis. Specifically, we focused on DNS records with "rrtype" values of either "A" (Address) or "NS" (Name Server). To achieve this, we created a Python tool that extracted and filtered these records, resulting in a new CSV file named ["filtered_data_comp.csv"].Which we can see it done on appendix [E].

4.6.5 Labeling the Dataset

With our filtered dataset in place, we were poised to label the domains as either "malicious" or "benign." For this purpose, we relied on the ["malicious_matchings_comp.txt"] dataset Which created using python tool can be seen on appendix [G], which contains a substantial collection of malicious domains.

The labeling process was straightforward: if a domain from our historical Passive DNS records was found within the ["malicious_matchings_comp.txt"] list, it was unequivocally labeled as "malicious." Conversely, domains not present in the malicious dataset were marked as "benign."

4.6.6 Balancing the Dataset

Following the filtering of relevant data and saving it as "[filtered_data_comp.csv]" we proceeded with feature extraction using the designated functions. The resulting dataset, named "[results.csv]", initially exhibited an imbalance in the distribution of labeled domains. Specifically, the number of benign domains significantly outnumbered the malicious ones.

To address this imbalance and ensure equitable representation for both malicious and benign domains, we developed a Python tool. This tool effectively balanced the dataset by randomly selecting malicious domains to match the count of benign domains. The resulting dataset, now with an equal number of malicious and benign domains with unique domains(non repetative), was saved as `"balanced_data.csv."` after it was labeled.

4.6.7 Dataset Ready for Analysis

With the labeling process complete, our dataset was primed and ready for machine learning analysis. It consists of historical Passive DNS records for a vast array of domains, each labeled as either "malicious" or "benign." This comprehensive dataset serves as the foundation for our subsequent research and analysis, enabling us to apply various machine learning techniques to identify and classify malicious domain activities effectively.

Chapter 5

Results and Discussion

In this chapter, we will showcase and analyze the outcomes and evaluations of our implementation. Following the presentation of each result, we will engage in a discussion to identify areas for potential improvement, if applicable.

5.1 DNS data collection

The collection of DNS data was facilitated through the utilization of the CIRCL Passive DNS Database, a highly regarded resource offering comprehensive insights into the historical domain-to-IP mappings. Our data collection process commenced with the utilization of an orchestrated procedure, involving interaction with the CIRCL Passive DNS Database's Application Programming Interface (API). Our specific focus was on gathering information related to the top **1 million** domains, by dividing it into manageable segments of **25000** domains. Leveraging the API, we systematically extracted the historical passive DNS records associated with each of these selected domains. Amongst the **1 million** domains queried, only **176495** domains historical passive DNS record were returned. In the figure [5.1], we can see the DNS queries that were collected monthly and in the figure [5.2]we can see DNS queries collected daily. Additionally, in figure [5.3], we can see the distribution on time first and time last stamps.

In order to collect the DNS data, we deployed the D4 sensor, specifically designed for the purpose of collecting passive DNS queries. Upon the successful completion of this implementation, our sensor effectively acquired the DNS query data. However, an important challenge emerged as these DNS data were obtained in a binary format, rendering them unsuitable for automatic storage in our database. This posed a significant obstacle in the seamless integration of the collected DNS data into our database system, necessitating further measures to transform and process the binary data for proper storage and

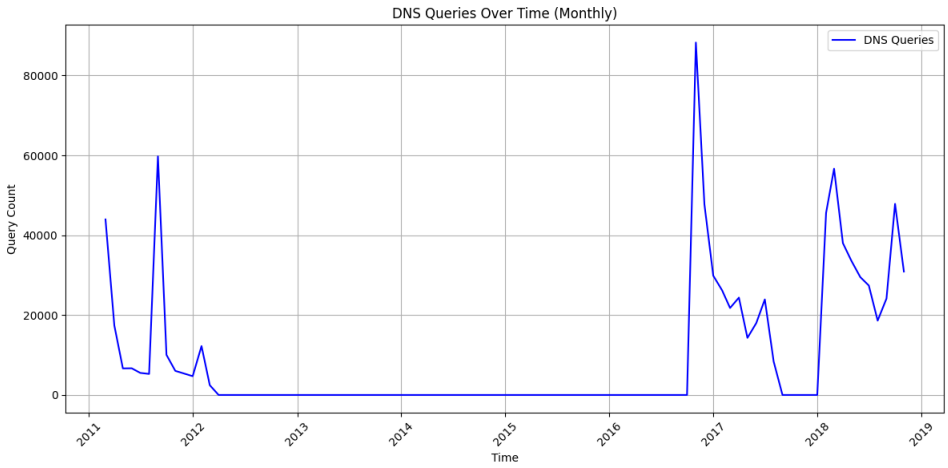

Figure 5.1: DNS queries monthly

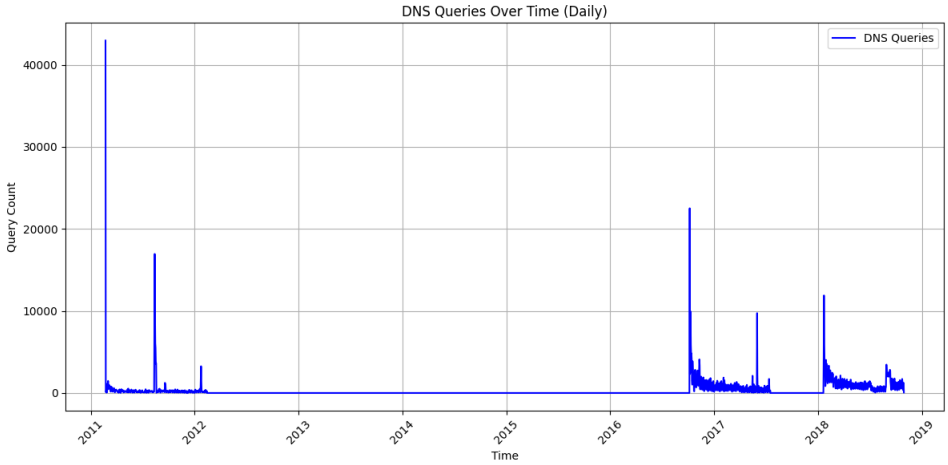

Figure 5.2: DNS queries daily

subsequent analysis. So, we followed the second approach for DNS data collection.

5.2 Domain Labelling

In the process of conducting our analysis, it was crucial to accurately classify domains as either "malicious" or "benign." This classification played a pivotal role in our feature extraction and subsequent analysis. To achieve this, we followed a systematic labeling process:

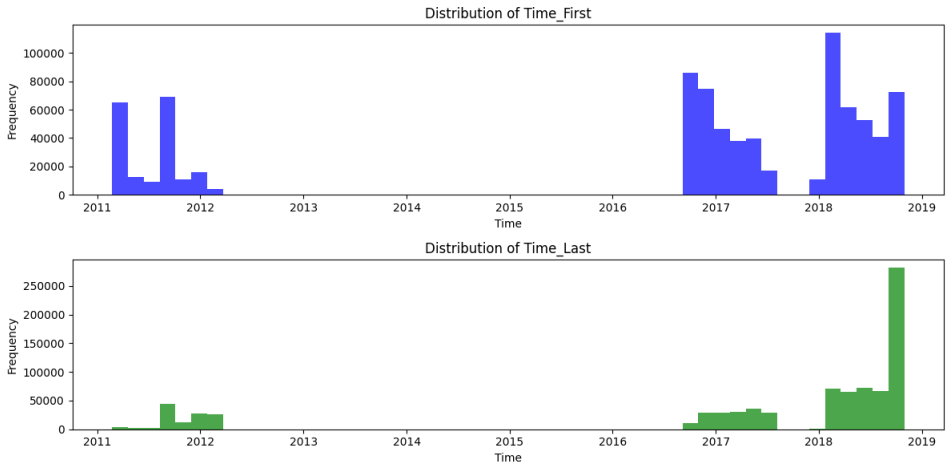

Figure 5.3: Distibution Time stamps

5.2.1 Filtering Relevant Records

Before starting the labeling process, we first needed to filter the dataset to include only the DNS records relevant to our analysis. Specifically, we focused on DNS records with "rrtype" values of either "A" (Address) or "NS" (Name Server). This step was essential to ensure that our subsequent analysis was based on the most pertinent data. In the figure [5.3] below, we can the visualize the distribution of DNS rescord types.

5.2.2 Feature extraction

This process encompassed the extraction of lexical, DNS, and time-based features, each contributing valuable insights into the domains under investigation. The visualized results of this feature extraction analysis can be found in the appendix [L].

5.2.3 Data Labelling

The labeling process was straightforward and binary in nature:

Malicious Domains

If a domain from our historical Passive DNS records was found within the "mal_col.txt" list, saved to different file called "malicious_matchings" This criterion allowed us to confidently identify domains with a known history of malicious activity.

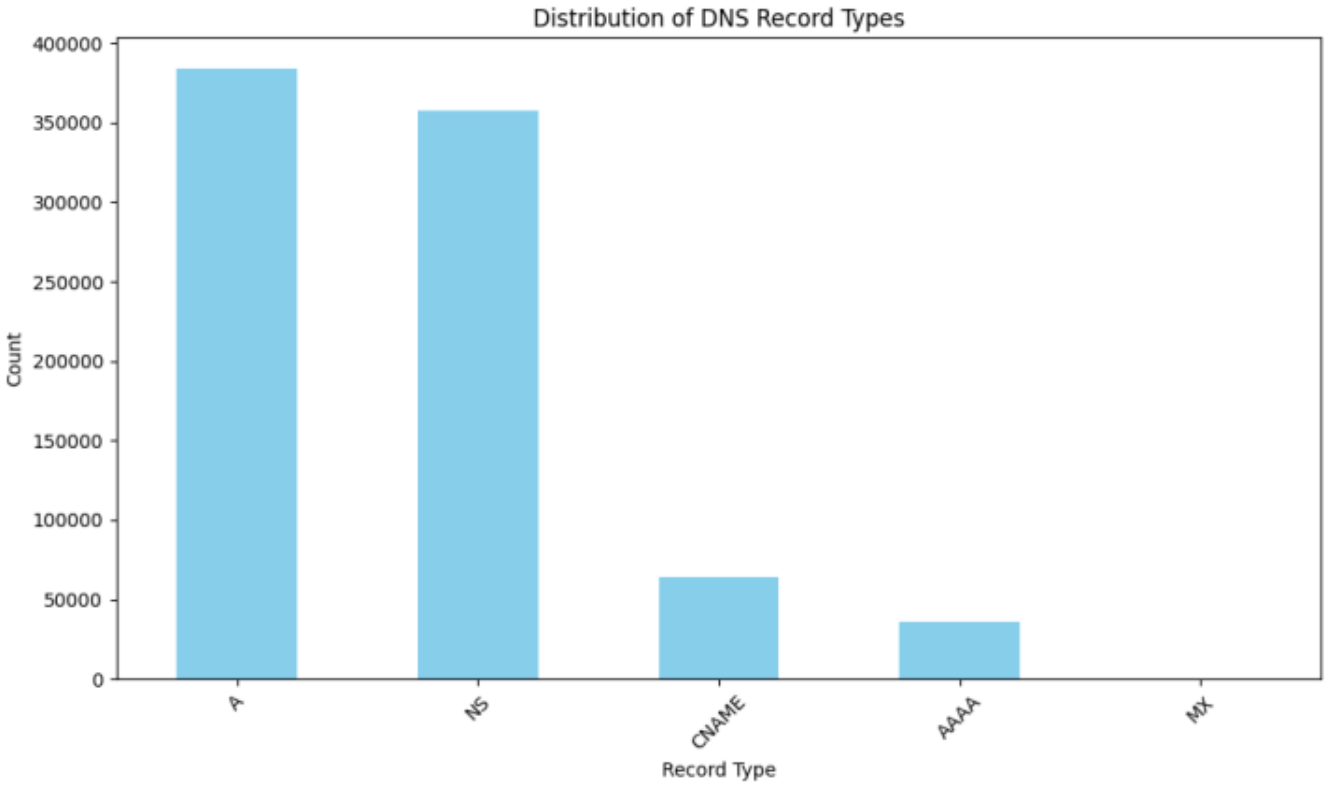

Figure 5.4: DNS records distribution

Benign Domains

Domains that were present as legit alexa domains "[dga_domains_full.csv [11]]" dataset were marked as "benign." This category encompassed domains with no known malicious associations.

Table 5.1: Labelled domain statistics

Dataset	Benign domains	Malicious domains
Unbalanced dataset	11735	1015
Balanced dataset	1015	1015

5.3 Machine learning algorithm results

The table [5.2] and figure [5.5] provides a table and visual representation of the accuracy results for each model, both on the training and test datasets with the size of 70% and 30% respectively.

K-Nearest Neighbors (K-NN)

The K-NN classifier achieved an accuracy of 0.75 on the training set, demonstrating its ability to fit the data well. However, on the test set, it achieved an accuracy of 0.60, indicating that it may not generalize as effectively to new, unseen data.

Gaussian Naive Bayes (GNB)

The GNB classifier showed an accuracy of 0.59 on the training set, suggesting that it learned the training data but may have limitations in generalizing to the test data. On the test set, it also achieved an accuracy of 0.60.

Linear Discriminant Analysis (LDA)

The LDA classifier exhibited an accuracy of 0.64 on the training set, indicating a good fit to the training data. However, on the test set, it achieved an accuracy of 0.61, suggesting some degree of overfitting and a slightly reduced performance in generalization.

Logistic Regression

The Logistic Regression classifier obtained an accuracy of 0.63 on the training set, signifying a reasonable fit to the training data. On the test set, it achieved an accuracy of 0.61, indicating consistent performance in generalization.

Support Vector Machine (SVM)

The SVM classifier displayed an accuracy of 0.62 on the training set, demonstrating its capability to fit the data. However, on the test set, it achieved an accuracy of 0.60, suggesting that it may not generalize as effectively to new, unseen data.

Random Forest

The Random Forest classifier showed remarkable performance on the training set, achieving a perfect accuracy score of 1.00. This could be indicative of overfitting, where the model essentially memorizes the training data. On the test set, it achieved an accuracy of 0.62, indicating some reduction in performance compared to the training set.

Each machine learning classifier exhibited varying levels of accuracy on both the training and test datasets. Further evaluation, including examining other metrics such as precision, recall, and F1-score, is essential to assess their overall performance and generalization ability. Additionally, considering the possibility of overfitting is important when interpreting the accuracy results, especially when perfect accuracy is observed on the training set.

Table [5.3] and figure [5.6] displays the comprehensive performance scores of the machine learning models on the balanced dataset.

In summary, the machine learning models, except for LDA, exhibit near-perfect accuracy on both the training and test datasets.

Table 5.2: Accuracy of the ML models

Classifiers	Training set	Test set
LDA classifier	64%	61%
KNN classifier	75%	60%
GNB classifier	59%	60%
Logistic regression classifier	63%	61%
SVM classifier	62%	60%
Random forest classifier	100%	62%

5.4 Classifier Performance Analysis

In assessing the performance of the RandomForest classifier, we rely on several key performance metrics based on the provided confusion matrix.

5.4.1 Accuracy

Accuracy serves as a comprehensive measure of the model's overall correctness in its predictions, taking into account both instances correctly classified as positive (True Positives) and negative (True Negatives), expressed as a ratio:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$

For the given confusion matrix:

$$\begin{bmatrix} 206 & 109 \\ 123 & 171 \end{bmatrix}$$

True Positives (TP) = 206, True Negatives (TN) = 171, False Positives (FP) = 109, False Negatives (FN) = 123

$$\text{Accuracy} = \frac{206 + 171}{206 + 171 + 109 + 123} \approx 0.6135 \ (61.35\%)$$

The RandomForest classifier achieved an accuracy of approximately 61.35%.

Table 5.3: Predictions

LDA prediction			
	precision	recall	f1-score
0.0	0.59	0.68	0.63
1.0	0.64	0.56	0.60
accuracy			0.61
macro avg	0.62	0.62	0.61
weighted avg	0.62	0.61	0.61

KNN prediction			
	precision	recall	f1-score
0.0	0.58	0.60	0.59
1.0	0.61	0.60	0.61
accuracy			0.60
macro avg	0.60	0.60	0.60
weighted avg	0.60	0.60	0.60

GNB prediction			
	precision	recall	f1-score
0.0	0.55	0.92	0.69
1.0	0.79	0.29	0.42
accuracy			0.59
macro avg	0.67	0.60	0.55
weighted avg	0.67	0.59	0.55

LOG prediction			
	precision	recall	f1-score
0.0	0.60	0.65	0.62
1.0	0.64	0.59	0.62
accuracy			0.62
macro avg	0.62	0.62	0.62
weighted avg	0.62	0.62	0.62

SVM prediction			
	precision	recall	f1-score
0.0	0.62	0.62	0.62
1.0	0.59	0.59	0.59
accuracy			0.60
macro avg	0.60	0.60	0.60
weighted avg	0.60	0.60	0.60

RF prediction			
	precision	recall	f1-score
0.0	0.63	0.65	0.64
1.0	0.61	0.58	0.60
accuracy			0.62
macro avg	0.62	0.62	0.62
weighted avg	0.62	0.62	0.62

5.4.2 Precision

Precision focuses on the accuracy of positive predictions made by the model. It is the ratio of True Positives to the total number of instances predicted as positive:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

In our context:

True Positives (TP) = 206, False Positives (FP) = 109

$$\text{Precision} = \frac{206}{206 + 109} \approx 0.6548 \ (65.48\%)$$

The RandomForest classifier exhibited a precision of approximately 65.48%.

5.4.3 Recall (Sensitivity)

Recall assesses the model's capability to correctly identify positive instances among all actual positive instances. It quantifies the ratio of True Positives to the total number of actual positives:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

In our scenario:

True Positives (TP) = 206, False Negatives (FN) = 123

$$\text{Recall} = \frac{206}{206 + 123} \approx 0.6265 \ (62.65\%)$$

The RandomForest classifier demonstrated a recall of approximately 62.65%.

5.4.4 Specificity

Specificity measures the model's ability to correctly identify negative instances among all actual negative instances. It is the ratio of True Negatives to the total number of actual negatives:

$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

In our context:

True Negatives (TN) = 171, False Positives (FP) = 109

$$\text{Specificity} = \frac{171}{171 + 109} \approx 0.6107 \ (61.07\%)$$

The RandomForest classifier also achieved a specificity of approximately 61.07%.

5.4.5 F1-Score

The F1-Score strikes a balance between precision and recall, providing a harmonious assessment of both metrics:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In our case:
Precision = 0.6548 (65.48%), Recall = 0.6265 (62.65%)

$$\text{F1-Score} = 2 \cdot \frac{0.6548 \cdot 0.6265}{0.6548 + 0.6265} \approx 0.6403 \ (64.03\%)$$

The RandomForest classifier achieved an F1-Score of approximately 64.03%, signifying a balanced performance between precision and recall.
In summary, the RandomForest classifier delivered a balanced performance, achieving moderate accuracy, precision, recall, specificity, and F1-Score. While it didn't reach exceptional levels of performance, it still provides valuable insights into the dataset and classification task.

5.4.6 ROC Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a classifier's performance across different threshold values for classification. It illustrates the trade-off between the True Positive Rate (Recall) and the False Positive Rate (1 - Specificity) as the decision threshold for classifying positive instances is varied.

The ROC curve is created by plotting the True Positive Rate (sensitivity) on the y-axis against the False Positive Rate (1 - specificity) on the x-axis at various threshold settings. Typically, the threshold values range from 0 to 1, and for each threshold, the corresponding True Positive Rate and False Positive Rate are calculated based on the classifier's predictions.

The importance of the ROC curve lies in its ability to provide a visual representation of a classifier's performance, especially in scenarios where the class distribution is imbalanced or when the choice of threshold significantly affects the model's behavior. A perfect classifier would have an ROC curve that passes through the upper-left corner of the plot, indicating a high True Positive Rate and a low False Positive Rate across all thresholds.

The Area Under the ROC Curve (AUC) is a single numeric value that summarizes the classifier's overall performance. AUC quantifies the model's ability to distinguish between positive and negative instances, regardless of the threshold chosen. Higher AUC values

(closer to 1) indicate better classifier performance, while random guessing corresponds to an AUC of 0.5.In short, the ROC curve and AUC are valuable tools for evaluating classifier performance, especially when dealing with classification tasks with imbalanced class distributions or when threshold tuning is critical.

5.5 Classifier Performance Visualizations

In this dedicated section, we embark on a journey to thoroughly analyze the performance of our classifiers through a rich array of visualizations. These visual representations are designed to provide you with a holistic understanding of how each classifier behaves and excels in the challenging task of domain classification.

5.5.1 Accuracy of Different Classifiers

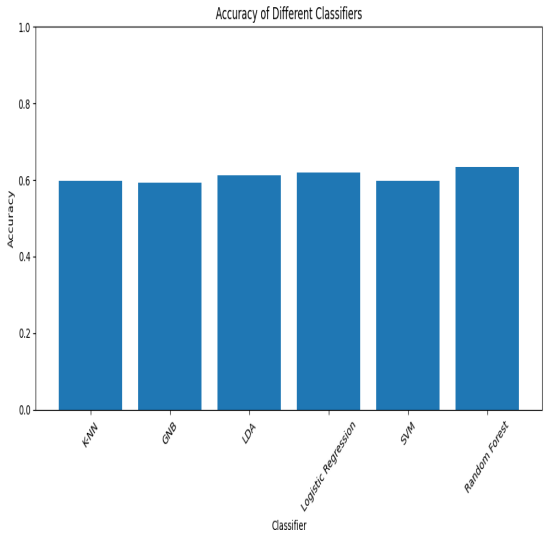

Figure 5.5: Accuracy of Different Classifiers

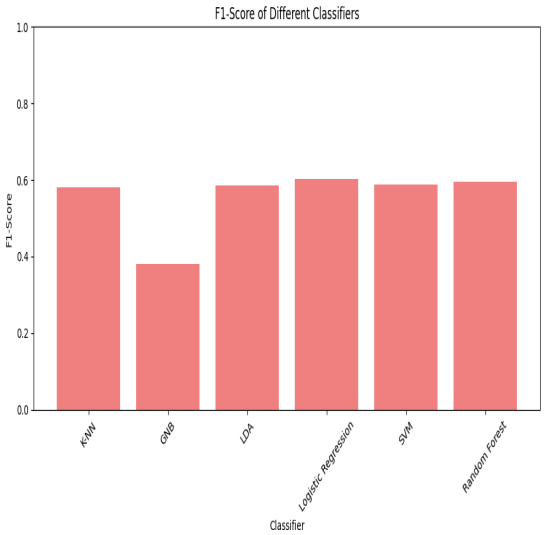

Figure 5.6: F1-Score of Different Classifiers

5.5.2 Confusion Matrix Heatmaps

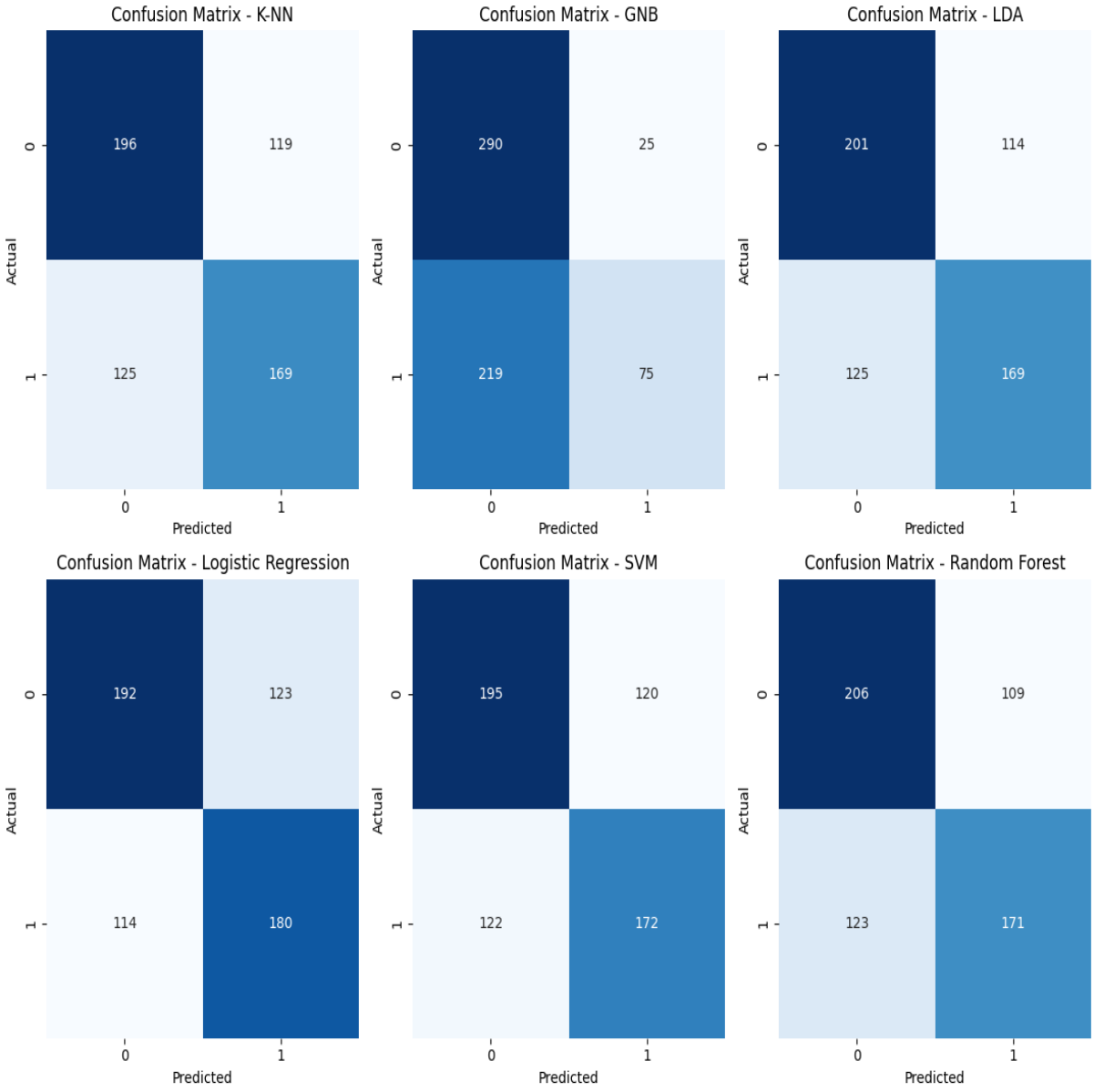

Figure 5.7: Confusion Matrix Heatmaps

5.5.3 Receiver Operating Characteristic (ROC) Curves

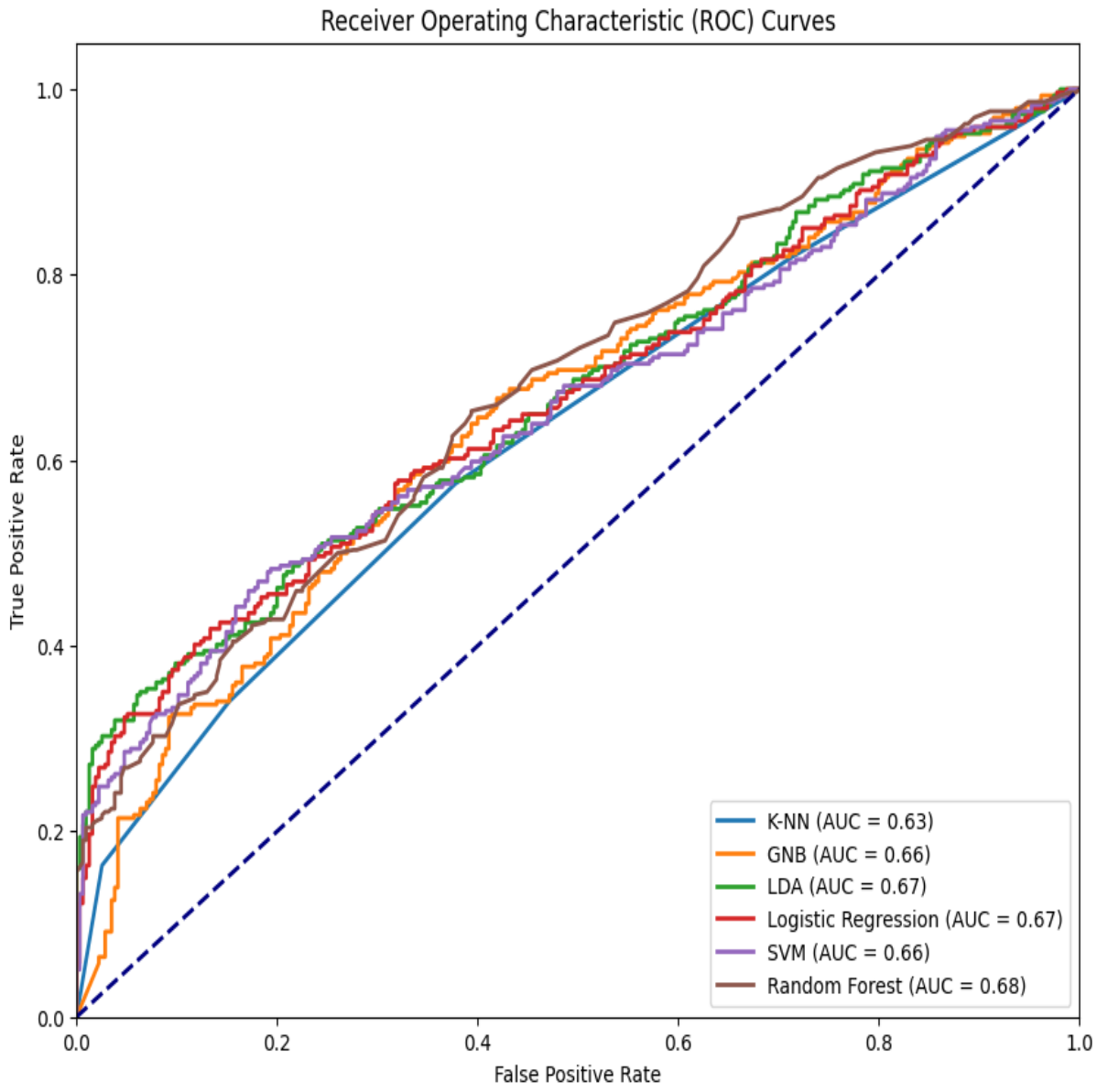

Figure 5.8: Receiver Operating Characteristic (ROC) Curves

5.5.4 Precision-Recall Curves

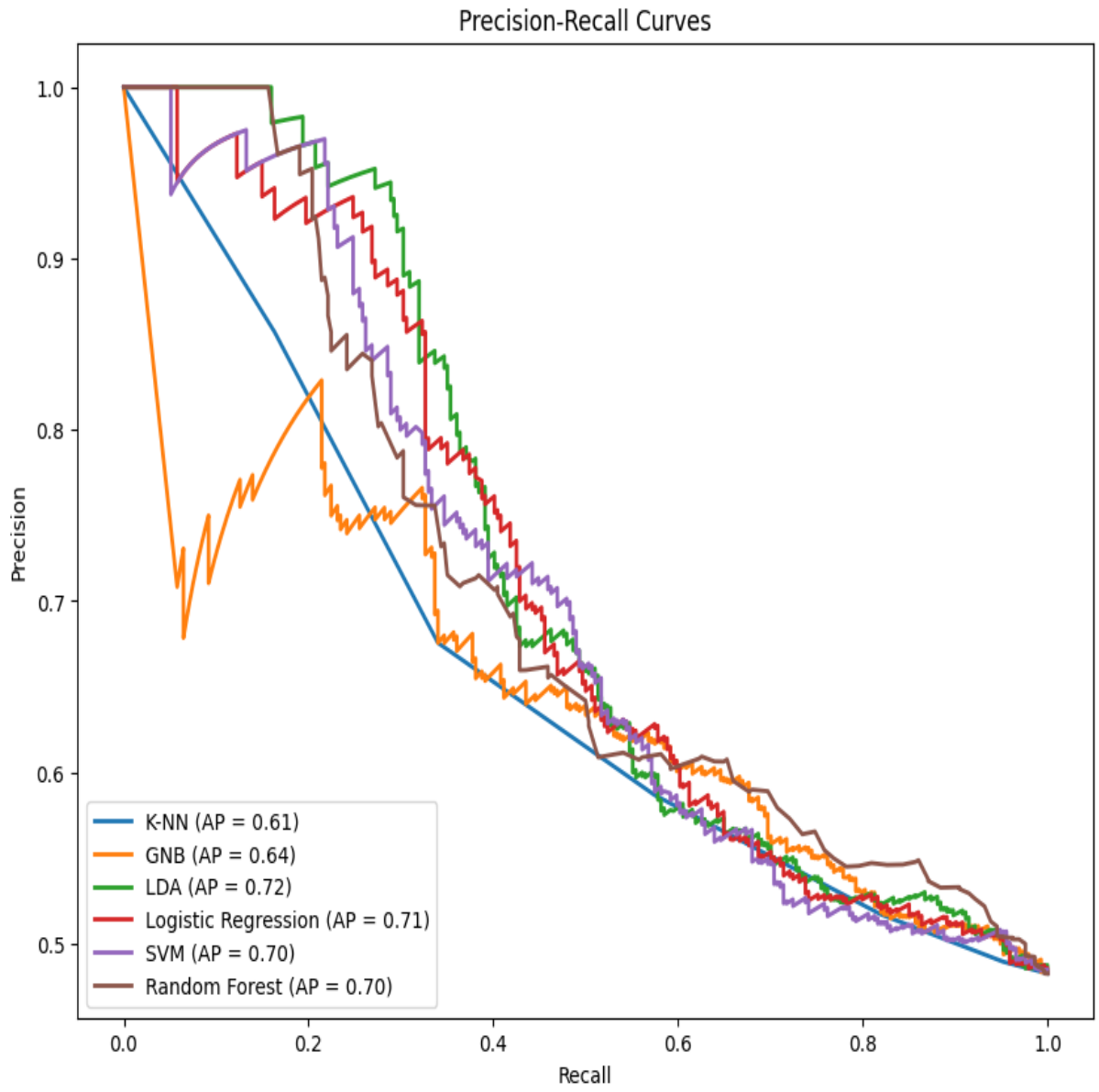

Figure 5.9: Precision-Recall Curves

5.6 Machine Learning Model Evaluation

In this section, we evaluate the performance of various machine learning models that are saved in the pickle file on the given dataset using key performance metrics and analyze their predictive capabilities for classifying domains as "benign" or "malicious."

1. **K-Nearest Neighbors (KNN)**: The K-Nearest Neighbors (KNN) classifier displays distinct behavior when classifying domains. It successfully predicts 9 out of 10 "malicious" domains accurately. However, when it comes to "benign" domains, KNN tends to misclassify 9 out of 10 as "malicious," indicating a lower accuracy for this class. This suggests that while KNN can effectively identify malicious domains, its performance with benign domains is less reliable.

2. **Support Vector Machine (SVM)**: The Support Vector Machine (SVM) classifier displays strong predictive performance. It accurately predicts all 10 malicious domains. However, when it comes to benign domains, it classifies all 10 of them as malicious.

3. **Linear Discriminant Analysis (LDA)**: Linear Discriminant Analysis (LDA) exhibits distinct performance in predicting domains, with varying outcomes for "benign" and "malicious" domains. Among the 10 benign domains, LDA accurately predicts one, but regrettably, it misclassifies the remaining nine as malicious. Conversely, for the 10 malicious domains, LDA successfully predicts nine of them accurately, but it erroneously classifies one as benign.

4. **Gaussian Naive Bayes (GNB)**: The Gaussian Naive Bayes (GNB) classifier exhibits accuracy in classifying domains. It correctly predicts "benign" domains with an accuracy of 1 out of 10, and for "malicious" domains, it accurately predicts 8 out of 10 while misclassifying 2 of them as "benign."

5. **Logistic Regression (LOGREG)**: The Logistic Regression (LOGREG) classifier demonstrates its ability to classify domains. It predicts 5 out of 10 "benign" domains accurately, but it also misclassifies 5 of them as "malicious." On the other hand, for "malicious" domains, it accurately predicts 1 out of 10 as malicious but misclassifies the remaining 9 as "benign."

6. **Random Forest**: The Random Forest classifier demonstrates strong predictive capabilities. It accurately predicts all 10 "malicious" domains. However, it tends to be overly cautious when classifying "benign" domains, as it predicts all 10 of them as "malicious."

5.6.1 Overall Analysis

In summary, the machine learning models employed for domain classification exhibit varying degrees of performance. Each model has its strengths and weaknesses:

1. **K-Nearest Neighbors (KNN)**: KNN shows effectiveness in predicting malicious domains but struggles with benign domains.

2. **Support Vector Machine (SVM)**: SVM excels in identifying malicious domains but misclassifies all benign domains.

3. **Linear Discriminant Analysis (LDA)**: LDA performs moderately well, accurately predicting some malicious domains but misclassifying others. Similarly, it has mixed results for benign domains.

4. **Gaussian Naive Bayes (GNB)**: GNB achieves reasonable accuracy, particularly in predicting malicious domains. However, it has some misclassifications for both classes.

5. **Logistic Regression (LOGREG)**: LOGREG demonstrates a cautious approach, accurately predicting some benign and malicious domains but misclassifying others.

6. **Random Forest**: Random Forest achieves high accuracy in predicting malicious domains but tends to be overly cautious, classifying benign domains as malicious.

The choice of the most suitable model depends on the specific requirements of the application. If avoiding false negatives (misclassifying malicious as benign) is crucial, GNB may be preferred. However, for applications where balanced accuracy across both classes is essential, Random Forest emerges as the top-performing choice.

Further fine-tuning and experimentation with different models and hyperparameters could provide additional insights into improving model performance and robustness.

Chapter 6

Conclusion and Future Work

In this chapter, we will provide a comprehensive final overview of the methodology's development and implementation. We will thoroughly discuss the results we have obtained, drawing meaningful insights from our findings. Additionally, we will outline potential areas for improvement and exploration in the future. This chapter serves as a crucial conclusion to our work, summarizing our achievements and setting the stage for future endeavors.

6.1 Conclusion

Our project initiated with an extensive research of passive DNS, its pivotal role in the field of cybersecurity, and the methodological framework essential for its successful implementation. This comprehensive research phase not only offered us a glimpse into the upcoming stages but also facilitated the development of a profound comprehension and expertise in the project itself, as well as the broader subjects intertwined with it. This foundational groundwork became the cornerstone upon which we built our project, enabling us to navigate the intricate landscape of passive DNS and its significance in cybersecurity with confidence and precision.

In the course of this project, we introduced two distinct approaches for gathering passive DNS data. Prior to the application of machine learning algorithms, our initial crucial step was to amass this passive DNS data. Our project's development journey commenced with the implementation of the first approach, which involved utilizing the D4 project. This phase proved to be the most challenging aspect of our project, a challenge we detailed in the preceding chapter.

Confronted with the challenges we encountered during the D4 project phase, where we faced limitations in data collection due to time constraints and sensor limitations, we

made a strategic decision to pivot to our second approach for passive DNS data collection. In this revised approach, we sourced data from the CIRCL database to address the data shortage. Following the successful data collection from CIRCL, we proceeded with the essential tasks of feature extraction and dataset labeling, distinguishing domains as either malicious or benign. To conclude this phase, we recognized the importance of balancing the dataset, ensuring it was prepared and optimized for machine learning applications while addressing the data scarcity issue.

Upon applying a range of machine learning algorithms, our models have shown their effectiveness in detecting malicious domains. In particular, the LDA model achieved a detection rate of 61% on our test dataset, while the KNN, SVM, GNB, and Logistic Regression models exhibited accuracy rates ranging from 60% to 62%. The Random Forest model achieved 62% accuracy on the test dataset. Based on these results, we can confidently assert that the project's objectives have been met successfully.

6.2 Future Work

While we have successfully achieved all our project objectives, it's important to acknowledge that there exist opportunities for further improvement and future developments based on the foundation we have laid. One prominent area with potential for enhancement pertains to the applicability of the D4 project sensor. Despite having completed the implementation of the D4 project, we encountered a notable challenge during the process - the inability to effectively process the collected binary data into the database. This challenge was primarily attributed to the format of the DNS data we had gathered, which was in binary form.

To address this issue and facilitate smoother data handling in the future, one significant avenue for improvement lies in automating the storage of DNS data directly into the database. By implementing a mechanism that can seamlessly transform and store binary DNS data, we can enhance the efficiency and reliability of our data management processes, thereby contributing to the overall robustness of our project's infrastructure. This improvement will not only streamline our current operations but also pave the way for more seamless and scalable future developments in this domain.

Furthermore, there is potential for enhancing our model's feature extraction process by exploring additional features that could improve model performance. Additionally, further fine-tuning and experimentation with different models and hyperparameters could provide additional insights into improving model performance and robustness.

Bibliography

[1] Cricket Liu, Paul Albitz, "DNS and BIND, 5th Edition," *O'Reilly Media, Inc*, ISBN: 9780596100575, May 2006.

[2] Goossens, M., Mittelbach, F., Samarin, *A LaTeX Companion*, Addison-Wesley, Reading, MA, 1994.

[3] Kopka, H., Daly P.W., *A Guide to LaTeX*, Addison-Wesley, Reading, MA, 1999.

[4] PDNSTool Available at: `https://github.com/gamelinux/passivedns`.

[5] *D4-core*, D4 Project, 2019. `https://github.com/D4-project/d4-core/tree/master/server`

[6] *D4-goclient*, D4 Project, `https://github.com/D4-project/d4-goclient`

[7] *D4-analyzer*, D4 Project, `https://github.com/D4-project/analyzer-d4-passivedns`

[8] *PyPdns Library*, PyPdns Library, `https://github.com/CIRCL/PyPDNS`

[9] *Cof-Format*, QOF-format, `https://datatracker.ietf.org/doc/html/draft-dulaunoy-dnsop-passive-dns-cof`

[10] *Alexa and DGA Domains Dataset*, GitHub Repository, `https://github.com/chrmor/DGA_domains_dataset/blob/master/dga_domains_full.csv`

[11] *Blocklist Project*, GitHub Repository, `https://github.com/blocklistproject/Lists`

[12] *Tiago Toledo*,Website, Available at: `https://tiagobigode.com.br/o-que-e-o-zero-trust-priviled/`

[13] *Digital Business*,Website, Available at: `https://digitalbusinessblog.wordpress.com/2018/09/27/what-is-smart-data-how-can-we-make-it-actionable/`

[14] *GeeksforGeeks*,Website, Available at: https://www.geeksforgeeks.org/what-is-a-dns-amplification-attack/

[15] *GeeksforGeeks*,Website, Available at: https://www.geeksforgeeks.org/dns-spoofing-or-dns-cache-poisoning/

[16] *Security investigation*,Website, Available at: https://www.socinvestigation.com/how-dns-tunneling-works-detection-response/

[17] *Unit 42*,Website, Available at: https://unit42.paloaltonetworks.com/fast-flux-101/

[18] *Spancept*,Website, Available at: https://spanceptvideomarketing.com/what-is-dns-hijacking-and-how-to-prevent-it-from-happening-on-your-website/

[19] *Spamhaus*, Website, Available at: https://www.spamhaus.com/resource-center/what-is-passive-dns-a-beginners-guide/

[20] *D4 Project. Passive DNS Tutorial*, Webisite, Available at: https://d4-project.org/2019/05/28/passive-dns-tutorial.html.

[21] *Circl Passive DNS*, Website, Available at: https://www.circl.lu/services/passive-dns/.

[22] *IDC Global DNS Threat*, Website Available at: https://efficientip.com/resources/2021-idc-dns-threat-report-release/

[23] [Yury Zhauniarovich, Issa Khalil, Ting Yu, Marc Dacier], A Survey on Malicious Domains Detection through DNS Data Analysis, *Journal*,https://arxiv.org/pdf/1805.08426v1.pdf,22 May 2018

[24] [Issa Khalil, Ting Yu , Bei Guan], Discovering Malicious Domains through Passive DNS Data Graph Analysis, *Journal*,https://dl.acm.org/doi/pdf/10.1145/2897845.2897877, 30 May 2016

[25] [Zhenyan Liu, Yifei Zeng, Pengfei Zhang, Jingfeng Xue, Ji Zhang, Jiangtao Liu], An Imbalanced Malicious Domains Detection Method Based on Passive DNS Traffic Analysis , *Journal*,https://www.hindawi.com/journals/scn/2018/6510381/, 20 Jun 2018

[26] [Qing Wang, Linyu Li, Bo Jiang, Zhigang Lu, Junrong Liu, Shijie Jian], Malicious Domain Detection Based on K-means and SMOTE, *Article*,https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7302825/, 15 Jun 2020

[27] [Gopinath Palaniappan, Sangeetha S, Balaji Rajendran, Sanjay, Shubham Goyal, Bindhumadhava B S], Malicious Domain Detection Using Machine Learning On Domain Name Features, Host-Based Features and Web-Based Features, *Article*,https://www.sciencedirect.com/science/article/pii/S1877050920310383, 2020

Appendix A

Project Proposal

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Curso de Licenciatura em Engenharia Informática
Project 3th Year - School year of 2022/2023

Passive DNS Analytics

	Artificial intelligence		Multimedia/Augmented reality/…
	Networks and systems management		Mobile apps
	Web platforms	X	Cybersecurity

* - Other area

Supervisor: Tiago Pedrosa - pedrosa@ipb.pt

Co-supervisor: Rui Pedro Lopes – rlopes@ipb.pt

1 Goals

The main goals of this project can be divided as follows:

- Understand how passive DNS works and its applicability in cybersecurity intelligence
- Implement a solution to gather the information on resolving time in a testing network and how to create and manage the historical database
- Make exploratory data analysis.

2 Details

DNS is one of the central services of the Internet, almost every connection and use of services is preceded by a request to this protocol. Therefore, the capability to have historic information about the IP, FQDNs, registrars, and other information present on the DNS may enable to understand the normal updates and anormal update behaviour. It can be used to know if some actors also have a historic of being used or are related to known attacks and thereby assign a level of risk.

Students will work with Linux systems, to create a service architecture to capture, store and manage the passive DNS data. It is also expected that the students will make exploratory analysis to better understand the data available.

3 Project development strategy

The following working methodology is proposed:

1. Research about passive DNS and its applicability in cybersecurity intelligence
2. Research methods for passive DNS analytics
3. Propose an architecture to support the deploy of the solution for gather the information on resolving time on the network and the storing, management, and analytics components
4. Propose the methodology for data analysis

5. Implementation
6. Tests and improvements
7. Report Writing

Team size:	2
Required resources:	Student laptops and 3 VMs for deploy the solution components

Appendix B

Code for splitting One million domains in four halves respectively

```
import pandas as pd
# Load the original CSV file
original_csv_path = 'top-1m.csv'
df = pd.read_csv(original_csv_path)

# Split the DataFrame into two parts
df_part1 = df[:500000]
df_part2 = df[500000:]

# Save the two parts as separate CSV files
part1_csv_path = 'part1_domains.csv'
part2_csv_path = 'part2_domains.csv'
df_part1.to_csv(part1_csv_path, index=False)
df_part2.to_csv(part2_csv_path, index=False)

print("CSV␣files␣split␣successfully.")


#newcode line
#this one for splitting in four 250000 domains on each dataset
import pandas as pd

# Load the previously split CSV files
part1_csv_path = 'part1_domains.csv'
part2_csv_path = 'part2_domains.csv'
```

```
df_part1 = pd.read_csv(part1_csv_path)
df_part2 = pd.read_csv(part2_csv_path)

# Split the previously split DataFrames into two new parts
df_part1_1 = df_part1[:250000]
df_part1_2 = df_part1[250000:]
df_part2_1 = df_part2[:250000]
df_part2_2 = df_part2[250000:]

# Save the new parts as separate CSV files
part1_1_csv_path = 'part1_1_domains.csv'
part1_2_csv_path = 'part1_2_domains.csv'
part2_1_csv_path = 'part2_1_domains.csv'
part2_2_csv_path = 'part2_2_domains.csv'
df_part1_1.to_csv(part1_1_csv_path, index=False)
df_part1_2.to_csv(part1_2_csv_path, index=False)
df_part2_1.to_csv(part2_1_csv_path, index=False)
df_part2_2.to_csv(part2_2_csv_path, index=False)

print("CSV␣files␣split␣again␣successfully.")
```

Appendix C

Code for getting the Passive Dns Records from Circl using API and pypdns Library

```
import pypdns
import csv
import time

# Read the list of domains from the CSV file
alexa_domains = []
with open('part1_1_domains.csv', 'r', newline='', encoding='utf-8
   ') as csv_file:
    reader = csv.reader(csv_file)
    for row in reader:
        domain = row[1].strip()  # Assuming the domain is in the
           second column
        alexa_domains.append(domain)

if __name__ == '__main__':
    # Initialize the PyPDNS object
    pdns = pypdns.PyPDNS(basic_auth=('ipb.pt', '
       B5fzHNadddCF4j4sXYKzNOTXYXlDZ8OXACzO2wPHsk8='))

    # Specifying the CSV file path
    csv_file = 'passive_dns_data_part1_1.csv'
```

```
# Writing the Passive DNS data for each domain to the CSV
    file
with open(csv_file, 'w', newline='', encoding='utf-8') as
    file:
     writer = csv.writer(file)
     writer.writerow(['Domain', 'Passive␣DNS␣Data'])

     for domain in alexa_domains:
         try:
             results = pdns.rfc_query(domain)
             if results:
                 writer.writerow([domain, str(results)])
             time.sleep(3) # Sleep for 3 seconds to avoid
                potential restrictions
         except Exception as e:
             print(f"Error␣querying␣{domain}:␣{e}")
             continue

print(f"Passive␣DNS␣data␣for␣{len(alexa_domains)}␣domains␣
    saved␣to␣{csv_file}.")
```

Appendix D

Code for extracting fields from a string PDNSRecord object by converting it into a list of dictionaries and separating each feature into columns

```
import pandas as pd
import re

data = pd.read_csv('passive_dns_data_alexa_1m_part1.csv')
# Create empty lists to store the extracted fields
time_first = []
time_last = []
rdata = []
rrtype = []
rrname = []

# Iterating through each rows in the DataFrame
for index, row in data.iterrows():
    DNS_string = row["Passive␣DNS␣Data"]
    data_string = DNS_string.replace("[PDNSRecord(", "").replace(
       ")]", "")
    record_strings = re.split(r'\),\s*PDNSRecord\(', data_string)
```

```
    record_dicts = []
    for record_str in record_strings:
        record_dict = {}
        key_value_pairs = re.findall(r'(\w+)\s*=\s*("[^"]+"|\w+)'
           , record_str)
        for key, value in key_value_pairs:
            record_dict[key] = value.strip('"')
        record_dicts.append(record_dict)

    for record_dict in record_dicts:
        time_first.append(record_dict.get("time_first"))
        time_last.append(record_dict.get("time_last"))
        rdata.append(record_dict.get("rdata"))
        rrtype.append(record_dict.get("rrtype"))
        rrname.append(record_dict.get("rrname"))

# Create a new DataFrame with extracted fields
extracted_data = pd.DataFrame({
    "time_first": time_first,
    "time_last": time_last,
    "rdata": rdata,
    "rrtype": rrtype,
    "rrname": rrname
})

# Save the extracted data to a new CSV file
extracted_data.to_csv('extracted_data.csv', index=False)
```

Appendix E

Code for filtering data on the basis of RRtype either "A" or "NS" record type

```
import pandas as pd

df = pd.read_csv('extracted_data_comp.csv')

# Filter rows where "RRType" is either 'A' or 'NS'
filtered_df = df[df['RRType'].isin(['A', 'NS'])]

# Save the filtered DataFrame to a new CSV file
filtered_df.to_csv('filtered_data_comp.csv', index=False)
```

Appendix F

Code for Feature extraction

```
import pandas as pd
import re
import math
import itertools
import editdistance
from statistics import mean

dataframe = pd.read_csv('filtered_data_comp.csv')

# Function to extract DNS records for a given domain (its used in
    distinct_ns_records)
def get_records(domain):
    records = dataframe[dataframe['Domain'] == domain].to_dict('
       records')
    return records

#function to get
def number_distinct_A_records(domain):
    records = dataframe[(dataframe['Domain'] == domain) & (
       dataframe['RRType'] == 'A')]
    a_records = set(records["RData"])
    return len(a_records)

# Function to extract distinct A records for a domain for the
   ip_entropy_domain_name
def distinct_A_records(domain):
```

```
    records = dataframe[(dataframe['Domain'] == domain) & (
        dataframe['RRType'] == 'A')]
    A_records = records["RData"].tolist()
    return A_records

#function for the ip entropy for domain
def ip_entropy_domain_name(domain):
    A_records = distinct_A_records(domain)
    suffixes = []
    for record in A_records:
        regex = r"(?:\d{1,3}\.){1}\d{1,3}"
        matches = re.finditer(regex, record, re.MULTILINE)
        for match in matches:
            suffixes.append(match.group())
            break
    unique_suffixes = set(suffixes)
    entropy = 0
    for n in unique_suffixes:
        suffix_frequency = suffixes.count(n) / len(A_records)
        entropy -= suffix_frequency * math.log2(suffix_frequency)
    return round(entropy, 2)


# Function to calculate distinct NS records for a domain for the
    similarity_NS_domain_name
def distinct_NS_records(domain):
    records = get_records(domain)
    NS_records = [r['RData'] for r in records if r['RRType'] == '
        NS']
    return set(NS_records)

# Function to extract NS records for a given domain
def get_NS_records(domain):
    ns_records = dataframe[(dataframe['Domain'] == domain) & (
        dataframe['RRType'] == 'NS')]
    ns_records = ns_records['RData'].unique()  # Get unique NS
        records
    return ns_records
```

```
# Function to calculate the number of distinct NS records for a
    domain
def number_distinct_NS_records(domain):
    ns_records = get_NS_records(domain)
    return len(ns_records)

# Function to calculate similarity of NS domain names
def similarity_NS_domain_name(domain):
    NS_records = distinct_NS_records(domain)
    results = []
    for a, b in itertools.combinations(NS_records, 2):
        results.append(editdistance.eval(a, b))
    if not results:
        return 0
    return round(mean(results), 2)


# Function to determine the size of a given domain name (removing
    dots)
def len_domain(domain):
    domain_without_dots = domain.replace(".", "")
    return len(domain_without_dots)

# Function to calculate the maximum length of labels (subdomains)
    in a domain
def max_len_labels_subdomain(domain):
    labels = domain.split('.')
    max_label_length = max(len(label) for label in labels)
    return max_label_length

# Function to calculate character entropy for a domain
def character_entropy(domain):
    all_chars = [char for char in domain.replace(".", "")]
    distinct_chars = list(set(all_chars))
    entropy = 0
    for char in distinct_chars:
        char_frequency = all_chars.count(char) / len(all_chars)
        entropy -= char_frequency * math.log2(char_frequency)
    return round(entropy, 1)
```

```
# Function to count the number of numerical characters in a
    domain
def number_numerical_characters(domain):
    numerical_characters = re.findall(r'\d', domain)
    return len(numerical_characters)

# Function to calculate the ratio of numerical characters to
    domain size
def ratio_numerical_characters(domain):
    numerical_count = number_numerical_characters(domain)
    domain_length = len_domain(domain)
    if domain_length == 0:
        return 0  # Avoid division by zero
    return numerical_count / domain_length

# Function to calculate the maximum size of continuous numerical
    characters in a domain
def max_len_cont_num_chars(domain):
    labels = domain.split('.')
    max_cont_num_chars = 0
    for label in labels:
        num_chars = re.findall(r'\d+', label)
        if num_chars:
            max_num_chars = max(num_chars, key=len, default="")
            max_cont_num_chars = max(max_cont_num_chars, len(
                max_num_chars))
    return max_cont_num_chars

# Function to calculate the maximum size of continuous alphabetic
    characters in a domain
def max_len_cont_alpha_chars(domain):
    labels = domain.split('.')
    max_cont_alpha_chars = 0
    for label in labels:
        alpha_chars = re.findall(r'\D+', label)
        if alpha_chars:
            max_alpha_chars = max(alpha_chars, key=len, default="
                ")
            max_cont_alpha_chars = max(max_cont_alpha_chars, len(
                max_alpha_chars))
```

```
    return max_cont_alpha_chars

# Function to calculate the maximum size of the same continuous
   alphabetic character in a domain
def max_len_cont_same_alpha_chars(domain):
    domain = ''.join(filter(str.isalpha, domain))
    max_same_alpha_chars = 0
    current_count = 1
    for i in range(len(domain) - 1):
        if domain[i] == domain[i + 1]:
            current_count += 1
        else:
            max_same_alpha_chars = max(max_same_alpha_chars,
               current_count)
            current_count = 1
    max_same_alpha_chars = max(max_same_alpha_chars,
       current_count)
    return max_same_alpha_chars


# Function to calculate the ratio of vowels in a domain
def ratio_vowels(domain):
    vowels = "AEIOUaeiou"
    vowel_count = sum(domain.count(v) for v in vowels)
    domain_length = len_domain(domain)
    if domain_length == 0:
        return 0  # Avoid division by zero
    return vowel_count / domain_length


# Function to calculate the maximum size of continuous consonant
   characters in a domain
def max_length_continuous_consonants(domain):
    labels = domain.split('.')
    max_cont_consonants = 0
    for label in labels:
        consonant_chars = re.findall(r'[b-df-hj-np-tv-z]+', label
           , re.IGNORECASE)
        if consonant_chars:
```

```
            max_consonant_chars = max(consonant_chars, key=len,
                default="")
            max_cont_consonants = max(max_cont_consonants, len(
                max_consonant_chars))
    return max_cont_consonants


# Create a list of unique domains from the CSV file
unique_domains = dataframe['Domain'].unique()

# Create a dictionary to store the results
results = {
    "Domain": unique_domains,
    "len_domain": [],
    "max_len_labels_subdomain": [],
    "character_entropy": [],
    "number_numerical_characters": [],
    "ratio_numerical_characters": [],
    "max_len_cont_num_chars": [],
    "max_len_cont_alpha_chars": [],
    "max_len_cont_same_alpha_chars": [],
    "ratio_vowels": [],
    "max_length_continuous_consonants": [],
    "number_distinct_A_records": [],
    "ip_entropy_domain_name": [],
    "number_distinct_NS_records": [],
    "similarity_NS_domain_name": []
}

# Calculate and store the results for each unique domain
for domain in unique_domains:
    results["len_domain"].append(len_domain(domain))
    results["max_len_labels_subdomain"].append(
        max_len_labels_subdomain(domain))
    results["character_entropy"].append(character_entropy(domain)
        )
    results["number_numerical_characters"].append(
        number_numerical_characters(domain))
    results["ratio_numerical_characters"].append(
        ratio_numerical_characters(domain))
```

```
        results["max_len_cont_num_chars"].append(
            max_len_cont_num_chars(domain))
        results["max_len_cont_alpha_chars"].append(
            max_len_cont_alpha_chars(domain))
        results["max_len_cont_same_alpha_chars"].append(
            max_len_cont_same_alpha_chars(domain))
        results["ratio_vowels"].append(ratio_vowels(domain))
        results["max_length_continuous_consonants"].append(
            max_length_continuous_consonants(domain))
        results["number_distinct_A_records"].append(
            number_distinct_A_records(domain))
        results["ip_entropy_domain_name"].append(
            ip_entropy_domain_name(domain))
        results["number_distinct_NS_records"].append(
            number_distinct_NS_records(domain))
        results["similarity_NS_domain_name"].append(
            similarity_NS_domain_name(domain))

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame(results)

# Save the results to a new CSV file
results_df.to_csv("results.csv", index=False)
```

Appendix G

Code for creating malicious_matchings_comp.txt file

```
import pandas as pd

df = pd.read_csv("completed_1m.csv", header=None)

# Extract the first column as a list of strings
csv_strings = df[0].tolist()

# Read the text file into a list of strings
with open("mal_col.txt", "r") as f:
    text_strings = f.read().splitlines()

# Check if each string in the text file is present in the CSV
   file
found_strings = []
not_found_strings = []
for s in text_strings:
    if s in csv_strings:
        found_strings.append(s)
    else:
        not_found_strings.append(s)

# Save the matching strings to a separate text file
with open("malicious_matchings_comp.txt", "w") as f:
    for s in found_strings:
        f.write(s + "\n")
```

```
# Print the results
print(f"Found␣{len(found_strings)}␣matching␣strings.")
print(f"Could␣not␣find␣{len(not_found_strings)}␣strings.")
```

Appendix H

Machine Learning Code

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from sklearn.discriminant_analysis import
   LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,
   confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC

# new code line (xDomain means without domain)
dataset = pd.read_csv("balanced_data_xDomain.csv")

# new code line
# Encode the labels
label_encoder = LabelEncoder()
dataset["CLASS"] = label_encoder.fit_transform(dataset["CLASS"])
dataset["Domain"] = label_encoder.fit_transform(dataset["Domain"
   ])
```

```
# new code line
# this was used on the added extra features named short_lived and
    duration
#this was for more extra features#
#dataset["Short_Lived"]=label_encoder.fit_transform(dataset["
   Short_Lived"])

# new code line
array = dataset.values
X = array[:, 0:14]
Y = array[:, 14]
validation_size = 0.30
X_train, X_test, y_train, y_test = train_test_split(
X, Y, test_size=validation_size, random_state=7
)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# new code line
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
print("Accuracy of K-NN classifier on training set: {:.2f}".
   format(knn.score(X_train, y_train)))
print("Accuracy of K-NN classifier on test set: {:.2f}".format(
   knn.score(X_test, y_test)))

# new code line
gnb = GaussianNB()
gnb.fit(X_train, y_train)
print(
"Accuracy of GNB classifier on training set: {:.2f}".format(
gnb.score(X_train, y_train)
)
)
print(
"Accuracy of GNB classifier on test set: {:.2f}".format(
gnb.score(X_test, y_test)
)
)
```

```
# new code line
from sklearn.discriminant_analysis import
   LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
print("Accuracy␣of␣LDA␣classifier␣on␣training␣set:␣{:.2f}".format
   (lda.score(X_train, y_train)))
print("Accuracy␣of␣LDA␣classifier␣on␣test␣set:␣{:.2f}".format(lda
   .score(X_test, y_test)))

# new code line
logreg = LogisticRegression(solver="lbfgs")
logreg.fit(X_train, y_train)
print(
"Accuracy␣of␣Logistic␣regression␣classifier␣on␣training␣set:␣{:.2
   f}".format(
logreg.score(X_train, y_train)
)
)
print(
"Accuracy␣of␣Logistic␣regression␣classifier␣on␣test␣set:␣{:.2f}".
   format(
logreg.score(X_test, y_test)
)
)

# new code line
svm = SVC(gamma="auto")
svm.fit(X_train, y_train)
print(
"Accuracy␣of␣SVM␣classifier␣on␣training␣set:␣{:.2f}".format(
svm.score(X_train, y_train)))
print("Accuracy␣of␣SVM␣classifier␣on␣test␣set:␣{:.2f}".format(svm
   .score(X_test, y_test)))

# new code line
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=0)
# Fit the classifier to the training data
```

```
rf.fit(X_train, y_train)

# Calculate and print the accuracy on the training set
train_accuracy = rf.score(X_train, y_train)
print("Accuracy of Random Forest classifier on training set: {:.2
    f}".format(train_accuracy))

# Calculate and print the accuracy on the test set
test_accuracy = rf.score(X_test, y_test)
print("Accuracy of Random Forest classifier on test set: {:.2f}".
    format(test_accuracy))

# new code line
print("KNN prediction")
pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

# new code line
print("GNB prediction")
pred = gnb.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("\nClassification Report:\n", classification_report(y_test,
    pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, pred))

# new code line
print("LDA prediction")
pred = lda.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("\nClassification Report:\n", classification_report(y_test,
    pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, pred))

# new code line
print("LOG prediction")
pred = logreg.predict(X_test)
print(confusion_matrix(y_test, pred))
```

```
print(classification_report(y_test, pred))
print("\nClassification Report:\n", classification_report(y_test,
    pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, pred))

# new code line
print("SVM prediction")
pred = svm.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("\nClassification Report:\n", classification_report(y_test,
    pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, pred))

# new code line
print("RandomForest prediction")

# Make predictions on the test set
pred = rf.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, pred))
print("\nClassification Report:\n", classification_report(y_test,
    pred))


# Here we use the codes to make Bargraph of Classifiers (Accuracy
    and F1 Score) for each classifiers
#newcode line
import matplotlib.pyplot as plt

# Define classifiers and their accuracy scores
classifier_names = ["K-NN", "GNB", "LDA", "Logistic Regression",
   "SVM", "Random Forest"]
accuracy_scores = [knn.score(X_test, y_test), gnb.score(X_test,
   y_test), lda.score(X_test, y_test),
                   logreg.score(X_test, y_test), svm.score(X_test
                       , y_test), rf.score(X_test, y_test)]

# Create a bar chart for accuracy
```

```
plt.figure(figsize=(10, 6))
plt.bar(classifier_names, accuracy_scores)
plt.xlabel("Classifier")
plt.ylabel("Accuracy")
plt.title("Accuracy of Different Classifiers")
plt.ylim(0, 1)  # Set the y-axis range (0 to 1 for accuracy)
plt.xticks(rotation=45)  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()

#newcode line
import matplotlib.pyplot as plt

# Define classifiers and their F1-Scores
classifier_names = ["K-NN", "GNB", "LDA", "Logistic Regression",
   "SVM", "Random Forest"]
f1_scores = [f1_score(y_test, knn.predict(X_test)), f1_score(
   y_test, gnb.predict(X_test)),
             f1_score(y_test, lda.predict(X_test)), f1_score(
                y_test, logreg.predict(X_test)),
             f1_score(y_test, svm.predict(X_test)), f1_score(
                y_test, rf.predict(X_test))]

# Create a bar chart for F1-Scores
plt.figure(figsize=(10, 6))
plt.bar(classifier_names, f1_scores,color = "lightcoral")
plt.xlabel("Classifier")
plt.ylabel("F1-Score")
plt.title("F1-Score of Different Classifiers")
plt.ylim(0, 1)  # Set the y-axis range (0 to 1 for F1-Score)
plt.xticks(rotation=45)  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()


#Here we used the code for creating heatmap of confusion matrix
   of each classifiers
#newcode line
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.metrics import confusion_matrix

# Define a list of classifiers and their names
classifiers = [knn, gnb, lda, logreg, svm, rf]
classifier_names = ["K-NN", "GNB", "LDA", "Logistic Regression",
   "SVM", "Random Forest"]

# Create confusion matrix heatmaps for each classifier
plt.figure(figsize=(12, 10))
for i, classifier in enumerate(classifiers):
    # Get predictions from the classifier
    y_pred = classifier.predict(X_test)

    # Calculate the confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Create a subplot for each classifier
    plt.subplot(2, 3, i + 1)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False
       )
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {classifier_names[i]}")

plt.tight_layout()
plt.show()

#Here we used the code to create the ROC Curves of each
   classifier
#newcode line
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Define a list of classifiers and their names
classifiers = [knn, gnb, lda, logreg, svm, rf]
classifier_names = ["K-NN", "GNB", "LDA", "Logistic Regression",
   "SVM", "Random Forest"]

# Create ROC curves for each classifier
plt.figure(figsize=(10, 8))
```

```
for i, classifier in enumerate(classifiers):
    # Get predicted probabilities for the malicious class (class
       1)
    y_prob = classifier.predict_proba(X_test)[:, 1]

    # Calculate ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)

    # Calculate the area under the ROC curve (AUC)
    roc_auc = roc_auc_score(y_test, y_prob)

    # Plot ROC curve
    plt.plot(
        fpr,
        tpr,
        lw=2,
        label=f'{classifier_names[i]} (AUC = {roc_auc:.2f})'
    )

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend(loc="lower right")
plt.show()


#Here we used the code to create the Precision-Recall Curve of
   each classifier
#newcode line
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve,
   average_precision_score

# Create Precision-Recall curves for each classifier
plt.figure(figsize=(10, 8))
for i, classifier in enumerate(classifiers):
    # Get predicted probabilities for class 1 (malicious class)
```

```
    y_prob = classifier.predict_proba(X_test)[:, 1]

    # Calculate Precision-Recall curve
    precision, recall, _ = precision_recall_curve(y_test, y_prob)

    # Calculate Average Precision (AP)
    ap = average_precision_score(y_test, y_prob)

    # Plot Precision-Recall curve
    plt.plot(
        recall,
        precision,
        lw=2,
        label=f'{classifier_names[i]} (AP = {ap:.2f})'
    )

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curves')
plt.legend(loc="lower left")
plt.show()
```

Appendix I

Code For Dataset Features Analysis in Histogram

```
#new code line
import seaborn as sns

#new code line
dataset = pd.read_csv("balanced_data.csv")

#new code line
names = [
    "len_domain",
    "max_len_labels_subdomain",
    "character_entropy",
    "number_numerical_characters",
    "ratio_numerical_characters",
    "max_len_cont_num_chars",
    "max_len_cont_alpha_chars",
    "max_len_cont_same_alpha_chars",
    "ratio_vowels",
    "max_length_continuous_consonants",
    "number_distinct_A_records",
    "ip_entropy_domain_name",
    "number_distinct_NS_records",
    "similarity_NS_domain_name",
]
```

```
#new code line

for it in names:
    plt.subplots(figsize=(7, 6), dpi=80)
    sns.distplot(
        dataset.loc[dataset["Class"] == "benign", it],
        color="green",
        label="benign",
    )
    sns.distplot(
        dataset.loc[dataset["Class"] == "malicious", it],
        color="red",
        label="malicious",
    )
    plt.title(f"{it} Histogram")
    plt.legend()
    plt.savefig(f"{it}.png")
```

Appendix J

Code For Machine Learning Model Evaluation

```
#newcodeline
#After executing the ml codes for saving the trained model to a
   pickle file
# Load the saved K-NN model
loaded_knn = joblib.load('knn_model.pkl') #knn/rf/lda/logreg/svm/
   gnb

# Define the input data as a list of lists
input_data_list = [
    [7, 4, 2.5, 0, 0.0, 0, 4, 2, 0.5714285714285714, 1, 7, 1.15,
       6, 6.27],
    [7, 4, 1.7, 0, 0.0, 0, 4, 2, 0.5714285714285714, 1, 1, 0.0,
       4, 1.0],
    [10, 7, 3.1, 0, 0.0, 0, 7, 1, 0.3, 3, 1, 0.0, 4, 8.83],
    [12, 9, 2.9, 0, 0.0, 0, 9, 1, 0.3333333333333333, 2, 2, 1.0,
       7, 2.62],
    [7, 5, 2.5, 0, 0.0, 0, 5, 1, 0.1428571428571428, 2, 2, 0.0,
       2, 1.0],
    [13, 10, 3.4, 0, 0.0, 0, 10, 1, 0.3076923076923077, 3, 0,
       0.0, 2, 1.0],
    [13, 10, 3.2, 0, 0.0, 0, 10, 1, 0.3076923076923077, 3, 0,
       0.0, 4, 9.0],
    [6, 3, 2.6, 0, 0.0, 0, 3, 1, 0.1666666666666666, 3, 1, 0.0,
       2, 3.0],
```

```
[15, 12, 3.4, 0, 0.0, 0, 12, 1, 0.3333333333333333, 5, 0,
   0.0, 4, 8.67],
[13, 8, 3.4, 0, 0.0, 0, 8, 1, 0.3846153846153846, 3, 0, 0.0,
   5, 8.8],
[8, 5, 2.5, 0, 0.0, 0, 5, 1, 0.375, 3, 0, 0.0, 6, 10.13],
[13, 11, 3.5, 0, 0.0, 0, 11, 1, 0.4615384615384615, 3, 1,
   0.0, 3, 1.0],
[6, 3, 2.3, 0, 0.0, 0, 3, 1, 0.1666666666666666, 3, 1, 0.0,
   5, 1.2],
[16, 13, 3.8, 0, 0.0, 0, 13, 1, 0.1875, 5, 20, 3.38, 8,
   12.11],
[9, 6, 2.5, 0, 0.0, 0, 6, 1, 0.4444444444444444, 1, 42, 4.8,
   4, 8.83],
[12, 9, 2.9, 0, 0.0, 0, 9, 1, 0.3333333333333333, 2, 1, 0.0,
   2, 1.0],
[14, 11, 3.2, 0, 0.0, 0, 11, 2, 0.4285714285714285, 3, 2,
   0.0, 10, 12.47],
[12, 9, 3.3, 0, 0.0, 0, 9, 1, 0.4166666666666667, 2, 2, 0.0,
   2, 3.0],
[7, 4, 2.5, 0, 0.0, 0, 4, 1, 0.4285714285714285, 3, 1, 0.0,
   11, 10.85],
[15, 12, 3.4, 0, 0.0, 0, 12, 1, 0.4, 2, 4, 2.0, 12, 9.65],
[13, 10, 3.4, 0, 0.0, 0, 10, 2, 0.3076923076923077, 3, 1,
   0.0, 13, 9.29],
[16, 13, 3.2, 0, 0.0, 0, 13, 1, 0.375, 3, 0, 0.0, 4, 3.33],
[12, 9, 2.9, 0, 0.0, 0, 9, 1, 0.25, 3, 2, 0.0, 2, 4.0],
[13, 10, 2.9, 0, 0.0, 0, 10, 1, 0.3846153846153846, 2, 1,
   0.0, 4, 8.5],
[11, 8, 2.9, 0, 0.0, 0, 8, 2, 0.4545454545454545, 2, 2, 0.0,
   2, 4.0],
[18, 15, 3.5, 0, 0.0, 0, 15, 1, 0.3888888888888889, 2, 5,
   0.72, 7, 11.29],
[10, 6, 2.8, 0, 0.0, 0, 6, 2, 0.5, 2, 40, 4.23, 7, 11.29],
[9, 6, 2.6, 0, 0.0, 0, 6, 1, 0.2222222222222222, 5, 1, 0.0,
   14, 9.85],
[8, 5, 3.0, 1, 0.125, 1, 4, 1, 0.25, 2, 5, 0.72, 4, 8.17],
[16, 13, 3.2, 0, 0.0, 0, 13, 1, 0.3125, 2, 3, 1.58, 14,
   4.51],
[18, 15, 3.4, 0, 0.0, 0, 15, 1, 0.3888888888888889, 2, 1,
   0.0, 6, 8.87],
```

```
    [13, 10, 3.5, 0, 0.0, 0, 10, 1, 0.3846153846153846, 3, 72,
       1.29, 12, 10.09],
    [13, 10, 3.1, 0, 0.0, 0, 10, 1, 0.3076923076923077, 3, 2,
       1.0, 10, 8.82],
    [11, 8, 3.5, 3, 0.2727272727272727, 3, 5, 1,
       0.2727272727272727, 2, 1, 0.0, 7, 14.05],
    [14, 11, 3.3, 0, 0.0, 0, 11, 1, 0.3571428571428571, 3, 6,
       0.65, 4, 16.0],
    [11, 8, 2.7, 2, 0.1818181818181818, 2, 6, 2,
       0.2727272727272727, 2, 3, 0.92, 6, 11.13],
    [7, 4, 2.8, 0, 0.0, 0, 4, 1, 0.2857142857142857, 2, 5, 0.72,
       8, 4.07],
    [12, 9, 2.9, 0, 0.0, 0, 9, 1, 0.4166666666666667, 2, 8, 0.0,
       7, 10.76],
    [10, 7, 2.9, 0, 0.0, 0, 7, 1, 0.4, 3, 0, 0.0, 8, 10.54]
]

# Initialize an empty list to store predictions
predictions = []

# Loop through the input data and make predictions
for data in input_data_list:
    input_data = pd.DataFrame({
        "len_domain": [data[0]],
        "max_len_labels_subdomain": [data[1]],
        "character_entropy": [data[2]],
        "number_numerical_characters": [data[3]],
        "ratio_numerical_characters": [data[4]],
        "max_len_cont_num_chars": [data[5]],
        "max_len_cont_alpha_chars": [data[6]],
        "max_len_cont_same_alpha_chars": [data[7]],
        "ratio_vowels": [data[8]],
        "max_length_continuous_consonants": [data[9]],
        "number_distinct_A_records": [data[10]],
        "ip_entropy_domain_name": [data[11]],
        "number_distinct_NS_records": [data[12]],
        "similarity_NS_domain_name": [data[13]]
    })

    X_input = input_data.values
```

```
    prediction = loaded_knn.predict(X_input)
    #if we want predictions on 0.0 as benign or 1.0 as malicious
    # predictions.append(prediction[0])

    if prediction[0] == 0.0:
        predictions.append("benign")
    else:
        predictions.append("malicious")

# Print the predictions
print(predictions)
```

Appendix K

Comprehensive Dataset Analysis Visualized Of Features Extraction

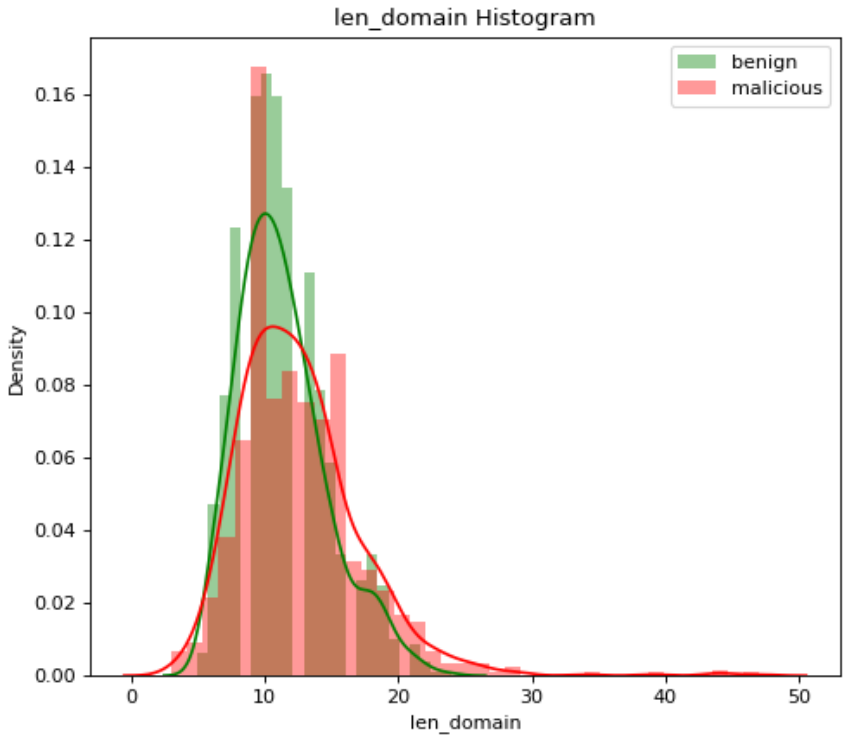

Figure K.1: len_domain feature density

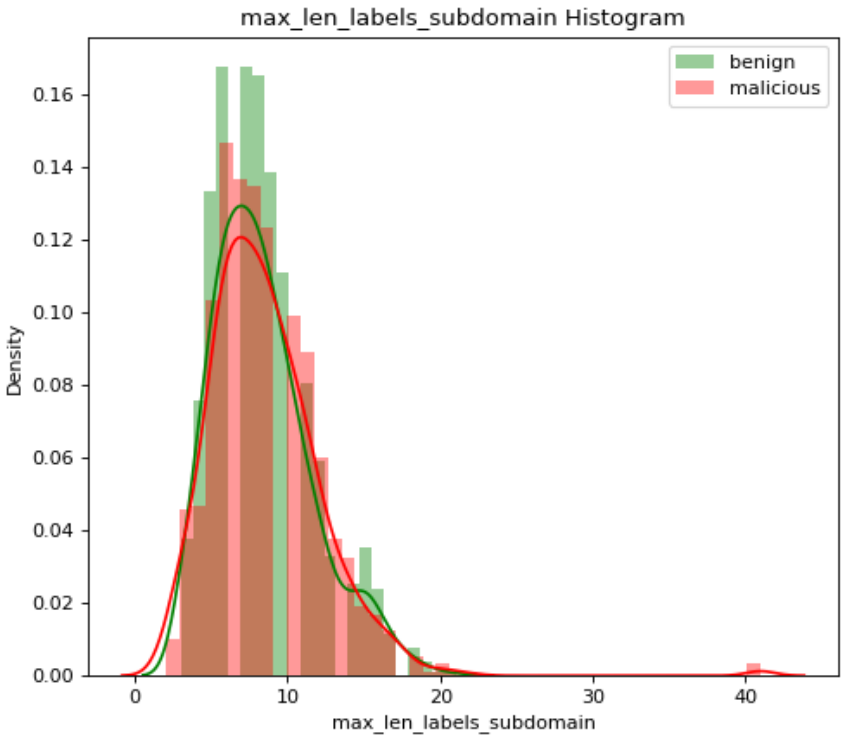

Figure K.2: max_len_labels_subdomain feature density

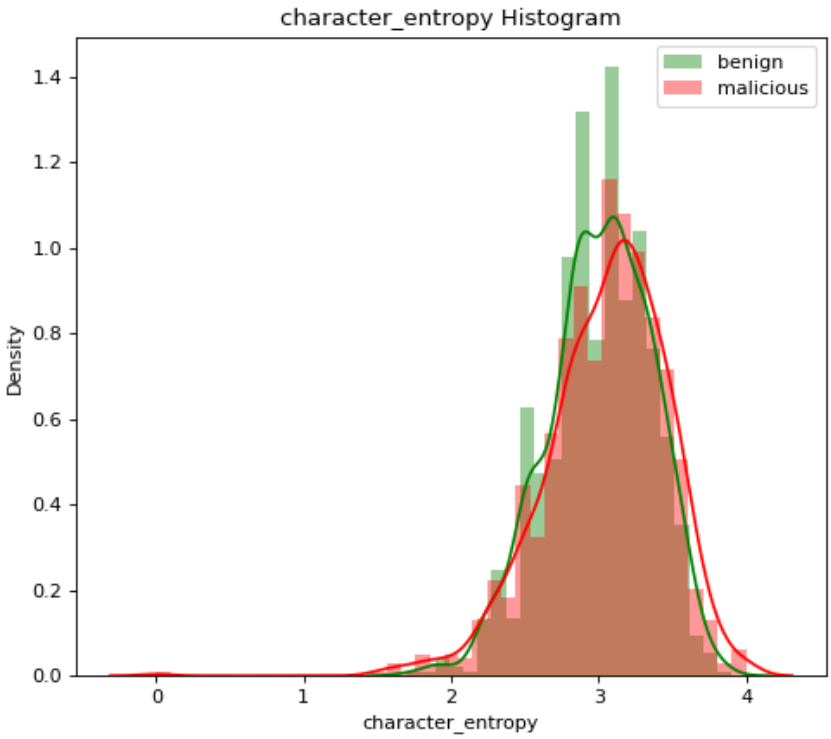

Figure K.3: character_entropy feature density

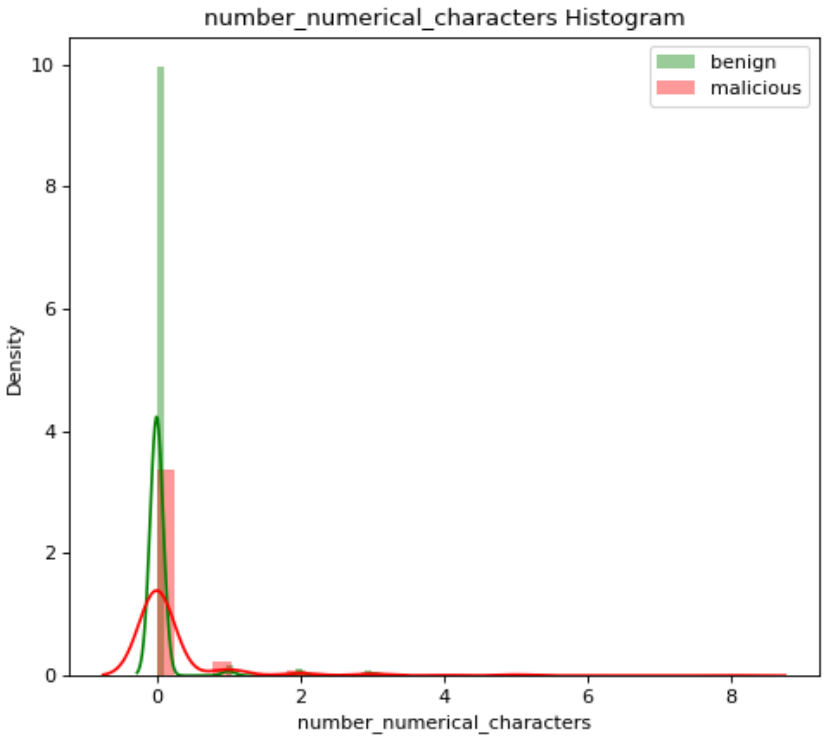

Figure K.4: number_numerical_characters feature density

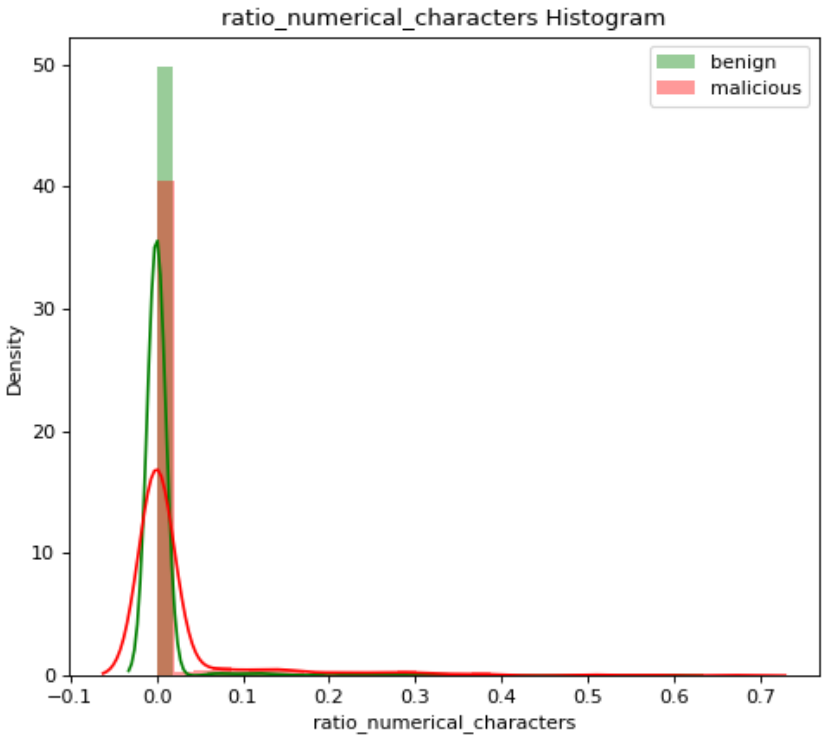

Figure K.5: ratio_numerical_characters feature density

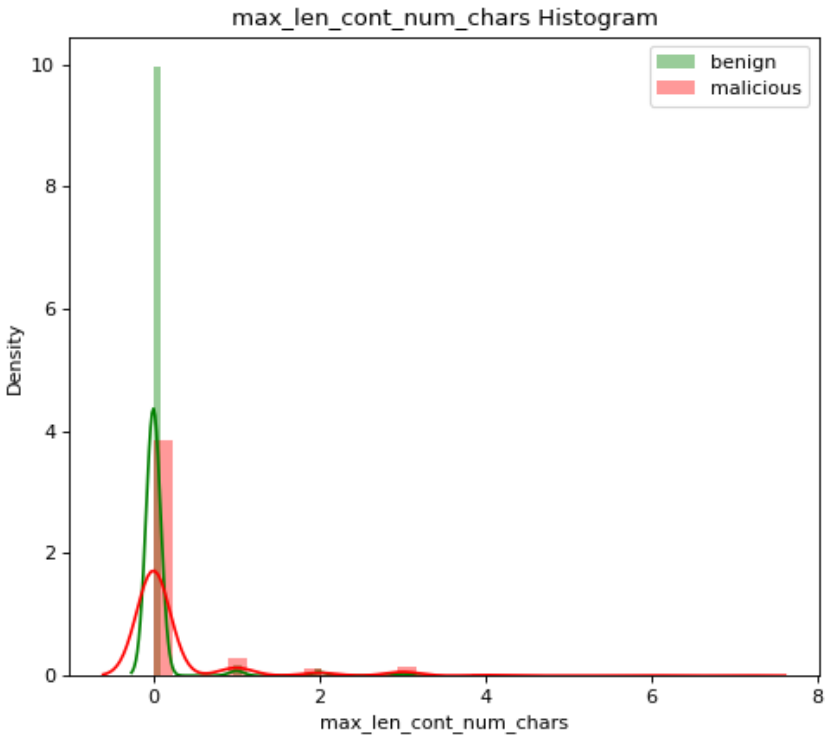

Figure K.6: max_len_cont_num_chars feature density

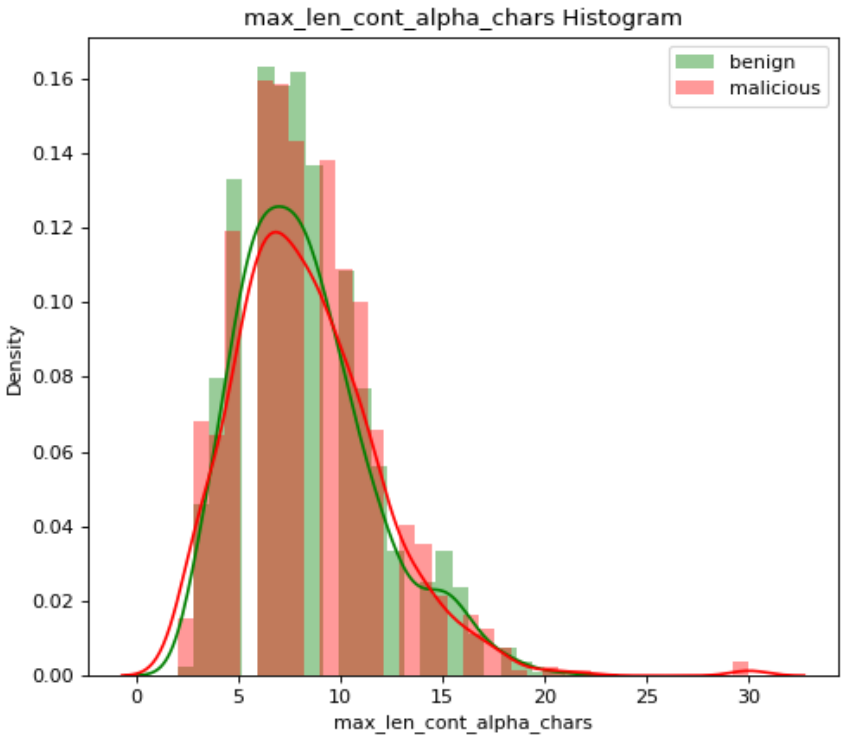

Figure K.7: max_len_cont_alpha_chars feature density

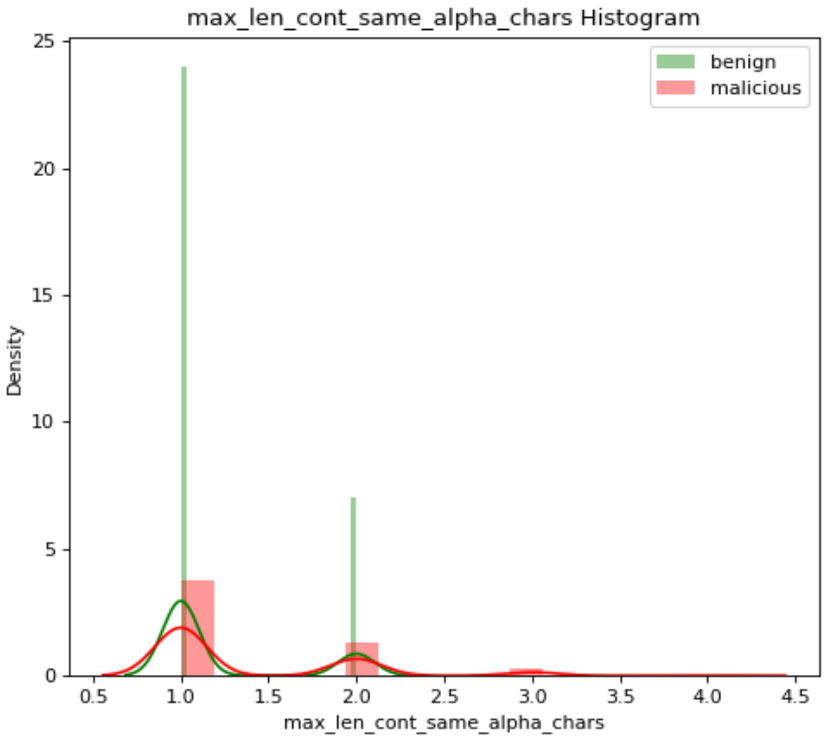

Figure K.8: max_len_cont_same_alpha_chars feature density

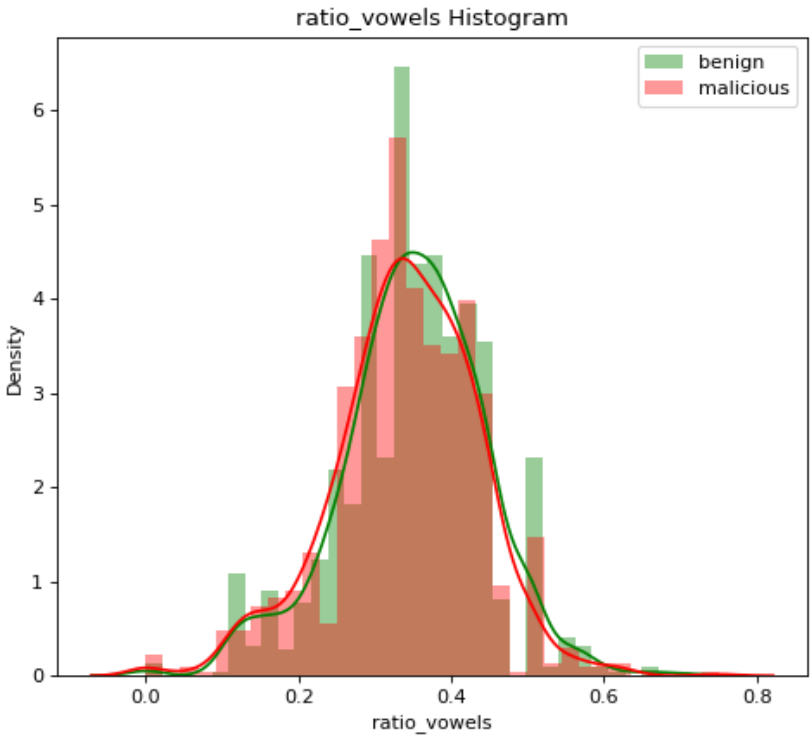

Figure K.9: ratio_vowels feature density

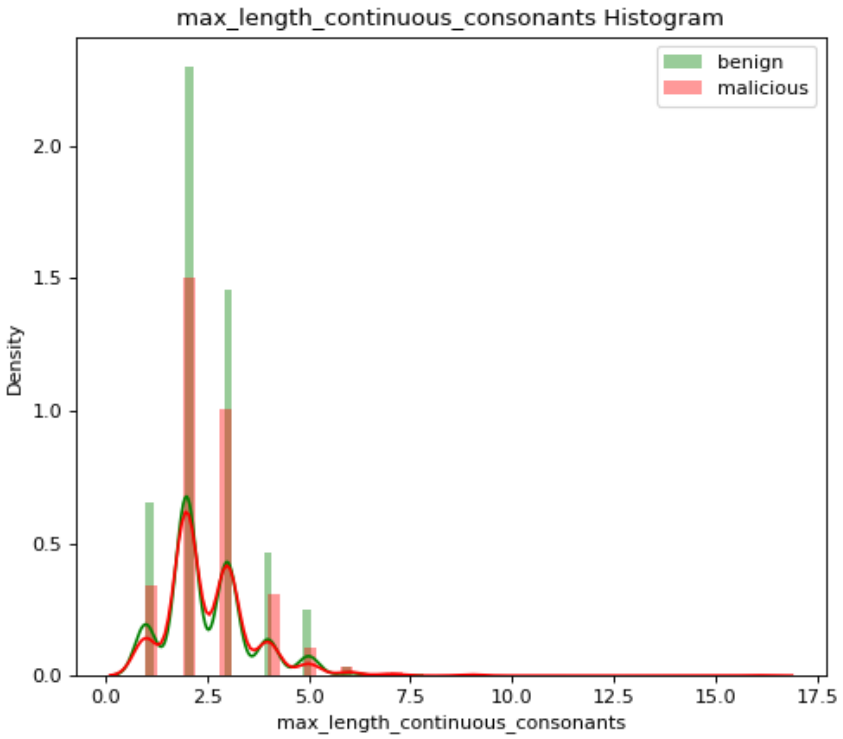

Figure K.10: max_length_continuous_consonants feature density

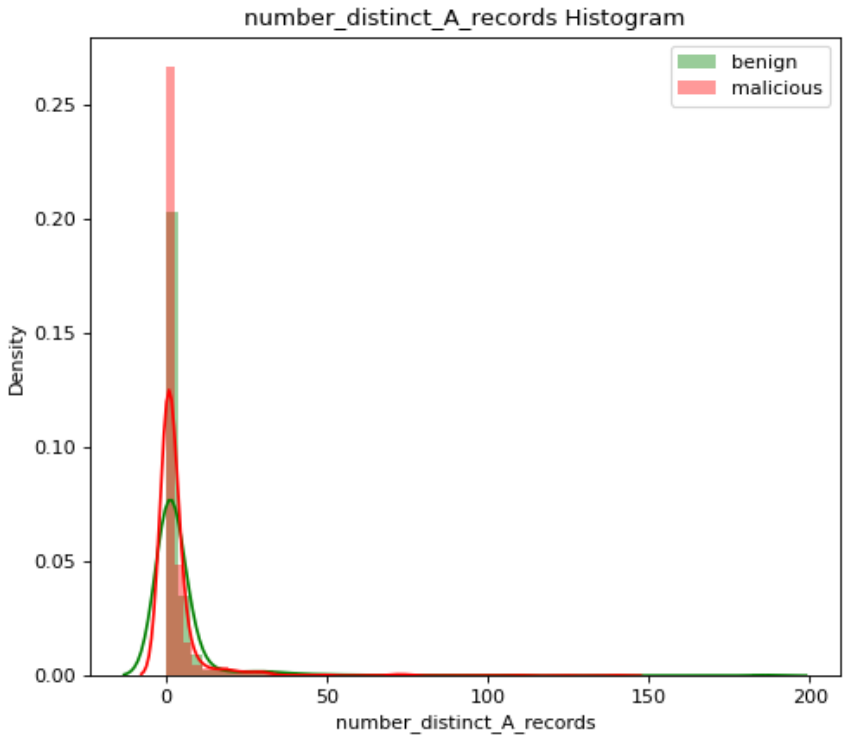

Figure K.11: number_distinct_A_records feature density

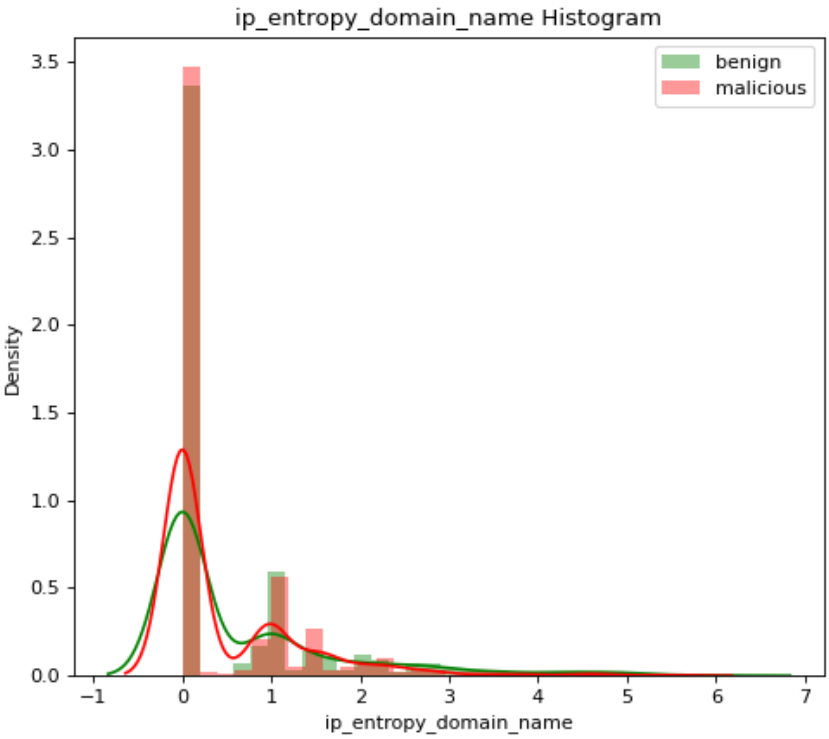

Figure K.12: ip_entropy_domain_name feature density

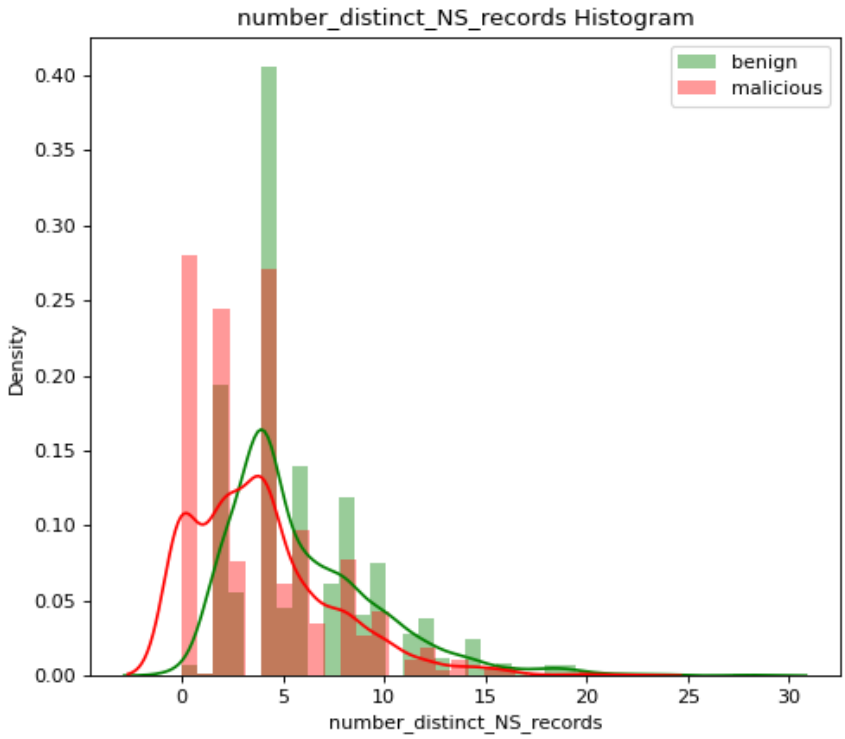

Figure K.13: number_distinct_NS_records feature density

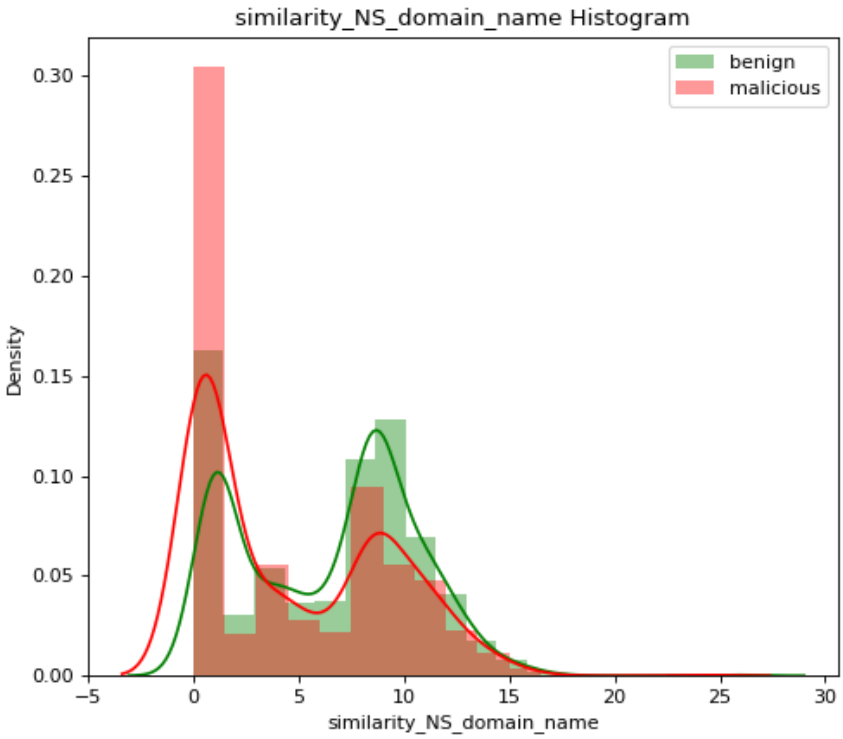

Figure K.14: similarity_NS_domain_name feature density