Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas (IIMAS)

A Solitary Strategy to Protect Users From DNS Fingerprinting Attacks

T E S I S

que opta por el grado de

Maestro en Ciencia e Ingeniería de la Computación

PRESENTA:
Carlo Geovani Benita Maldonado

Tutor Principal:
Dr. Javier Gómez Castellanos. IIMAS, UNAM
Co-tutor:
Dr. Oscar Arana Hernández. IIMAS, UNAM

México,CDMX. (Septiembre) 2023

National Autonomous University of Mexico

Postgraduate in Computer Science and Engineering Programme

Applied Mathematics and Systems Research Institute
(IIMAS)

A Solitary Strategy to Protect Users From DNS Fingerprinting Attacks

T H E S I S

to obtain the degree of

Master in Computer Science and Engineering

Student:
Carlo Geovani Benita Maldonado

Adivisor:
Javier Gomez Castellanos PhD. IIMAS, UNAM
Co-advisor:
Oscar Arana Hernandez PhD. IIMAS, UNAM

Mexico City, Mexico. (September) 2023

Jury Assignment

CHAIRMAN: DR. LUIS FRANCISCO GARCÍA JIMÉNEZ

MEMBER: DR. JAVIER GÓMEZ CASTELLANOS

SECRETARY: DR. JOSÉ JAIME CAMACHO ESCOTO

ALTERNATE: DR. OSCAR ARANA HERNÁNDEZ

ALTERNATE: DR. MICHAEL PASCOE CHALKE

Thesis Advisor:	**Thesis Co-Advisor:**
DR. JAVIER GÓMEZ CASTELLANOS	DR. OSCAR ARANA HERNÁNDEZ

Advisor's Signature:	**Co-Advisor's Signature:**

Dedication

To my parents, who have been my unwavering pillar of support. I love you Mom and Dad.
To my brother Francisco, who has been my mentor at all times. Also to my sister, Sara.

Acknowledgements

To the National Autonomous University of Mexico and the Computer Science and Engineering Postgraduate program.

To my advisors, Dr. Javier Gómez and Dr. Oscar Arana, for their support through these two years.

Also, to my jury members for their corrections and comments.

To the project 320403 of CONAHCYT.

Resumen

En los últimos años, nuestras actividades en línea realizadas a través de Internet han sido cada vez más susceptibles de interceptación, recopilación y manipulación. Esta susceptibilidad se debe principalmente a los modelos de negocio emergentes y a las estrategias de marketing empleadas por terceras empresas. En consecuencia, el análisis de las búsquedas en Internet, en particular de las búsquedas en la web, ha permitido extraer valiosos datos sobre las preferencias individuales.

Para abordar este problema, nuestra investigación introduce un enfoque innovador. Proponemos, diseñamos y aplicamos una serie de algoritmos que, cuando se despliegan en un único nodo dentro de una red, disminuyen eficazmente la precisión de los algoritmos de aprendizaje automatizado comúnmente atribuidos como entidades maliciosas. Este proceso introduce posteriormente ofuscación en la huella DNS de los usuarios finales, mejorando así las medidas de privacidad y seguridad asociadas a nuestras actividades en línea.

Abstract

In recent years, our online activities conducted over the Internet have been increasingly susceptible to interception, collection, and manipulation. This susceptibility primarily arises from emerging business models and marketing strategies employed by third-party companies. As a result, the analysis of Internet searches, particularly web searches, has enabled the extraction of valuable insights into individual preferences.

To address this potential security concern from the user's perspective, our research propose, design, and implement a series of algorithms that effectively diminish the accuracy of machine learning algorithms commonly attributed as malicious entities when deployed on a single node within a network. The proposed algorithms subsequently introduce obfuscation to the re-identification of user´s traffic (e.g., DNS fingerprint), thereby enhancing the privacy and security associated with our online activities.

Contents

Jury Assignment **i**

Dedication **ii**

Acknowledgements **iii**

Resumen **iv**

Abstract **v**

Glossary **ix**

1 Problem definition **1**
§1.1 Online-activity and privacy 1
§1.2 User re-identification and DNS fingerprinting 3
§1.3 Concerns about online activity 3
§1.4 General goals 4
§1.5 Methodology 5
§1.6 Contribution 5
§1.7 Thesis Organization 6

2 Domain Name System (DNS) background **7**
§2.1 DNS architecture 7
§2.2 Elements of the DNS protocol 9
§2.2.1 Type of DNS Servers 9
§2.2.2 Types of DNS Records 10
§2.3 Caching 11

3 State of the art **13**
§3.1 Identification techniques 13
§3.2 Countermeasures 14
§3.2.1 Single-node-based strategies 14
§3.2.2 Intermediary/relay-based strategies 15

4 Threat Models **18**
§4.1 Multinomial Naïve Bayes 18
§4.2 Support Vector Machines 19

§4.3 Random Forest 20

5 Proposed algorithms 23

§5.1 Blocking-based algorithms 23

§5.1.1 P-Block 24

§5.1.2 Top-Block 25

§5.2 Swapping-based algorithms 27

§5.2.1 P-Mix-All 28

§5.2.2 P-Mix-Bottom 29

§5.2.3 Top-Mix-All 29

§5.2.4 Top-Mix-Bottom 29

6 Simulations & Results 31

§6.1 DNS traffic dataset definitions & statistical analysis 31

§6.1.1 Attributes of the dataset 31

§6.1.2 Statistical analysis and threat models evaluation 34

§6.2 Blocking-based algorithms results 37

§6.2.1 P-Block 38

§6.2.2 Top-Block 39

§6.3 Swapping-based algorithms results 41

§6.3.1 P-Mix-All 41

§6.3.2 P-Mix-Bottom 42

§6.3.3 Top-Mix-All 44

§6.3.4 Top-Mix-Bottom 45

§6.4 Blocking-based algorithms vs. Swapping-based algorithms 45

7 Impact of proposed algorithms 49

§7.1 Real-life experiment 49

8 Discussion and further work 54

9 Conclusion 56

A Code 61

§A.1 Data preprocessing 61

§A.2 Threat Models 62

§A.3 P-Block 63

§A.4 Top-Block 65

§A.5 Top-Mix-All 66

§A.6 Top-Mix-Bottom 67

§A.7 P-Mix-All 68

§A.8 P-Mix-Bottom 69

List of Figures

1.1 Number of Internet users by year. *from: Digital 2023 Global Overview Report, Kepios [1]* 2

2.1 DNS hierarchy example 8
2.2 Fully qualified domain name *from:[2]* 9
2.3 DNS architecture 10

3.1 Accuracy of DNS fingerprinting by Arana et. al.[3] 14

5.1 Block diagram of how our proposed algorithms work 24
5.2 Theoretical P-Block workflow 25
5.3 Theoretical Top-Block workflow 27

6.1 Proposed benchmark 33
6.2 accuracy for different data ratios 35
6.3 Confusion matrix of threat models in 70-30 ratio 36
6.4 Running total 37
6.5 Threat models performance implementing P-Block 38
6.6 Threat models performance implementing Top-Block 40
6.7 Threat models performance implementing P-Mix-All 42
6.8 Threat models performance implementing P-Mix-Bottom 43
6.9 Threat models performance implementing Top-Mix-All 44
6.10 Threat models performance implementing Top-Mix-Bottom 46
6.11 P & Top-Algorithms Overlap 47

7.1 New user's data 50
7.2 Implementation workflow 50
7.3 Bar plot of Top-Block performance 51

Glossary

DNS: Domain Name System.
TLD: Top Level Domain.
SLD: Second Level Domains.
FQDN: Fully Qualified Domain Name.
HTTPS: Hypertext transfer protocol secure.
TCP: Transmission Control Protocol.
UDP: User Datagram Protocol.
TLS: Transpor Layer Security.
DoH: DNS over HTTPS.
DoT: Dns over TLS.
DoQ: DNS over QUIC.
ODNS: Oblivious DNS.
ODoH: Oblivious DNS over HTTPS.
TTL: Time To Live.
IETF: Internet Engineering Task Force.
RFC: Request for Comments.
ISP: Internet Service Provider.
TF-IDF: Term Frequency - Inverse Document Frequency.
MNB: Multinomial Naive Bayes.
SVM: Support Vector Machines.
RF: Ranfom Forest.
HTML: HyperText Markup Language.

Chapter 1

Problem definition

1.1 Online-activity and privacy

In recent years, the world has witnessed an extraordinary surge in the number of Internet users, transforming it into a crucial part of everyday life for millions worldwide. Several factors, such as the widespread availability of mobile devices and the rise of social media and online platforms, have driven the rapid growth in the density of Internet users. The development of faster and more efficient Internet infrastructure and the increasing affordability of devices and data plans have further fueled this expansion. As more and more individuals utilize the Internet, there has been a fundamental shift in how people access information, communicate with one another, and conduct business. A statistical report from Kepios [1] has revealed a significant and stabilizing increase in the number of Internet users worldwide, as illustrated in Figure 1.1.

Over the past few decades, the exponential growth in Internet usage has emphasized the increasing importance of online privacy protection. With more and more people joining the online community each year, safeguarding personal information and ensuring privacy becomes even more crucial. As the user base expands, so does the potential risk of privacy breaches and unauthorized access to sensitive data.

As our online activity has become an integral part of our lives, it offers a window into our preferences and behaviors. Through various online interactions such as web searches, social media engagements, and online shopping habits, every click and scroll leaves a digital footprint, creating patterns unique to each online user. These online activity patterns have become increasingly valuable in today's digital landscape. The vast amount of data generated by our online interactions serves as a goldmine for businesses, marketers, and researchers, offering insights into our preferences, behaviors, and interests. Companies can tailor advertisements and content by analyzing these patterns to cater to our needs and desires. Additionally, researchers can utilize these patterns to gain a deeper understanding of societal trends and preferences, providing valuable information for various fields, such as psychology, sociology, and economics.

This large amount of information is a critical resource for machine learning algorithms to uncover these patterns and gain insights into our individual preferences [4, 5, 6]. By

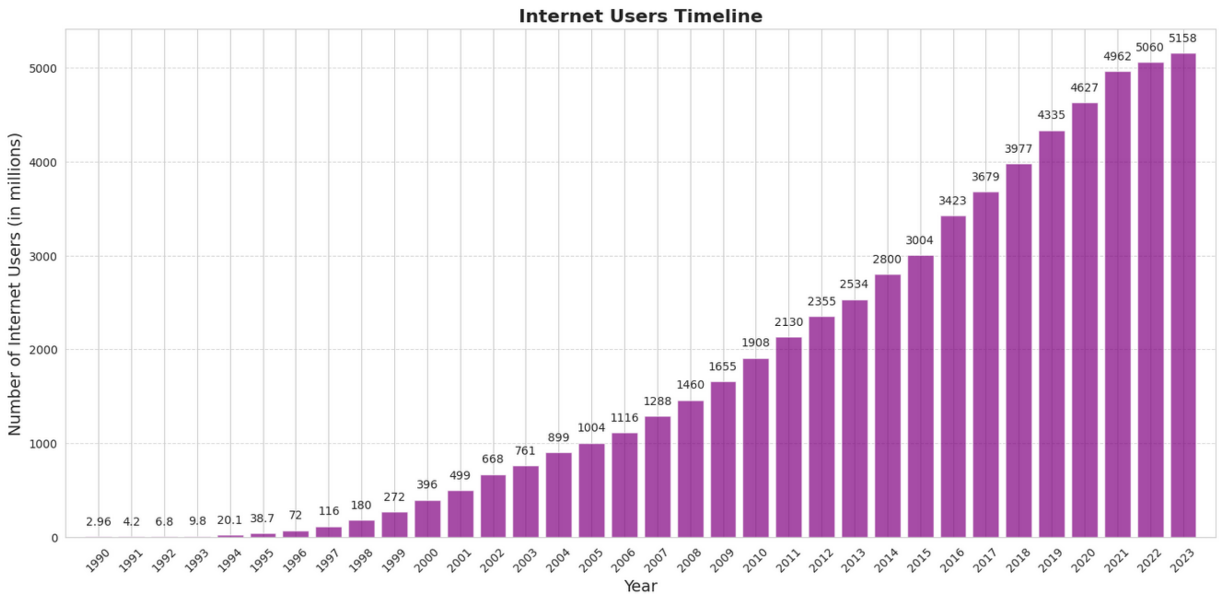


Figure 1.1: Number of Internet users by year.
from: Digital 2023 Global Overview Report, Kepios [1]

thoroughly analyzing our online activities, these algorithms can identify recurring trends, recognize correlations, and generate predictive models that aid in understanding user behavior and preferences. A well-known business model strategy: "behavioral-targeting," involves monitoring an individual's online activity [7, 4], and using this information to direct targeted advertising towards specific users.

The development of behavioral targeting [7] has led to more attention being focused on the privacy of people's online activities and the potential risks of being constantly tracked and identified by this type of artificial intelligence. Moreover, individuals should be aware of the various online tracking methods, such as cookies, device fingerprinting, and browser fingerprinting, and take actions to limit their exposure to these methods. While this practice can be advantageous for advertisers, it also raises concerns about the privacy of users' online activities.

According to a survey conducted by the independent research agency Toluna and highlighted by Kaspersky [8], many people need to gain the knowledge and skills to protect their online privacy effectively. The survey showed that 34% show interest in learn how to fully safeguard their privacy online. Furthermore, the survey revealed a troubling trend that over one in ten people (10%) have lost interest in learning how to improve their privacy further. For instance, when end-users accept cookies without reading the policies, terms, and conditions, they unknowingly grant permission to big-tech companies such as Facebook, Google, etc., to track their browsing habits, interests, and preferences. This vast amount of data enables these third-party companies to build comprehensive user profiles, which they then utilize to deliver tailored advertisements that align with the individual's preferences and behavior. Consequently, this practice raises legitimate concerns about the privacy and security of individuals' data, exposing them to potential misuse or unauthorized access.

Thus, it is evident that our online activity holds valuable information that machine learning algorithms can harness to better understand our preferences and enhance various aspects of our digital lives, but on the other hand it also represents a security breach that can potentially have harmful consequences for the the end user.

1.2 User re-identification and DNS fingerprinting

The Domain Name System (DNS) is a protocol used on the Internet to translate human-readable domain names (such as www.example.com) into IP addresses that computers use to identify each other on the network. It acts like a "phonebook" for the Internet, allowing users to access websites and services using user-friendly names.

In the context of online activities, the "user re-identification" paradigm [5, 3] plays an essential role in the behavioral-targeting strategy. Online interests are a subset of a person's overall interests. To clarify, if someone is interested in basketball, that interest can be represented online when visiting sports websites, online stores such as Amazon or eBay that sell basketball-related items; or even when searches for basketball players' names are made. The user re-identification paradigm aims to accumulate as many sessions as possible, enabling more precise targeting of advertising campaigns. A session refers to the time a user spends online, similarly searching for items or loading websites, Etc. This study defines a session as a day (24 hours) of Domain Name System (DNS) traffic.

Furthermore, behavioral targeting is a widespread technique that can utilize the DNS protocol, and leveraging DNS traffic can yield valuable insights about the devices and software users utilize. DNS is a critical element in this context because each query made by a user is typically associated with one or more DNS requests, making DNS traffic effective to identify users based on their manifested online interests. A DNS fingerprint can be generated by analyzing this DNS traffic, translating it into a user's online preferences and activities.

1.3 Concerns about online activity

As previously mentioned, our online activities generate a digital footprint that can be collected and analyzed, revealing patterns. One such pattern is the Zipf-like popularity distribution [9], which has implications for DNS fingerprinting. Zipf's law observed in DNS suggests that a small subset of domain names will be more popular than others, indicating that specific domains receive a significant number of requests. In contrast, most domain names receive relatively fewer requests. By analyzing this distribution pattern in our online activity, we can better understand DNS fingerprinting and its potential impact on privacy. This analysis helps identify online users' preferences and also raises implications for privacy concerns. By recognizing the logic behind the relationship between DNS fingerprinting and user preferences, we can collectively work towards creating a safer and more privacy-conscious online ecosystem.

Over the last years, several works have been done to avoid this re-identification. Importantly, it should be noted that among the various strategies proposed, certain approaches focus on either anonymization or obfuscation to improve the privacy. Anonymization involves the privacy technique of dissociating the data from personal identity. In other words, the goal is to ensure that the available information does not reveal our specific identity. In contrast, obfuscation allows for a partial unveiling of someone's identity. For instance, it might enable the prediction of a user's identity with a certain level of probability. We can categorize the proposed strategies in the literature as "Single-Node-Based" and "Intermediary"/"Relay-Based" techniques.

Single-node-based strategies might include solutions that implement cryptographic algorithms, such as DNS over HTTPS (DoH) [10], DNS over TLS (DoT) [11], and DoQ[12]. However, since the data is encrypted and relies on tunneling, the DNS recursive resolver can still gather the information since it is being sent through single ports. In addition, there are solitary strategies such as range query [13], which involve sending *dummy* or *fake* queries that can be easily ignored by machine learning algorithms, thus reducing the effectiveness of these countermeasures.

On the other hand, techniques that employ intermediaries or relays as actors of anonymization for each DNS query have also been implemented. Proposed solutions such as NQA [3], ODNS [14], μODNS [15], and Onion Routing (Tor) [16] aim to prevent this problem collaboratively. However, it should be noted that these implementations require either a partial or full level of trust in intermediaries within the network. Furthermore, in the case of Tor, there is a noticeable latency and longer response time for retrieving answers to requests, typically around 4.1 seconds [17]. This extended response time poses a potential drawback for end-users, significantly reducing the overall user experience. More detailed strategies are mentioned in Section 3.

Hypothesis

The accuracy of attackers using DNS fingerprinting employing machine learning algorithms can be significantly reduced through the selective modification of a minor fraction of a user's DNS traffic. By strategically introducing alterations to specific domain name frequencies, the attackers' capacity to re-identify an online user and discern its online activities is anticipated to experience a significant disruption.

1.4 General goals

The overall goal of this thesis is to bring anonymity and privacy to end-users based on their online activity, with a focus on achieving the below-mentioned objectives:

- Design and implement algorithms to obfuscate the DNS fingerprint.
- Quantify the efficacy of our proposed algorithms against re-identification attacks, such as machine learning classifiers.
- Measure and quantify the impact of the proposed algorithms on the user experience (UX).

1.5 Methodology

The research was conducted in two stages. In the initial phase, we performed comprehensive data preprocessing on a DNS dataset of eight real users. This preprocessing laid the groundwork for applying various machine learning classifiers. We designed, developed, and tested our algorithms in the second stage. These algorithms were subsequently subjected to the same threat models used previously, enabling us to measure and evaluate the effectiveness of our strategies. Our primary goal was to minimize the accuracy of threat models. Moreover, we conducted an in-depth analysis to assess these strategies' real-world implications and impact in various scenarios.

1.6 Contribution

Specifically, we diverged from the distributed strategies and put forth an alternative approach in our research. Our approach differs from conventional distributed architectures, as we opted for an single-node-based approach that does not rely upon any intermediary and/or third party to function effectively. The details are elaborated in Chapter 5, where we explore fresh opportunities and innovative solitary strategies to enhance user confidentiality on the Internet.

We conducted experiments that leveraged relevance metrics and stochastic-based algorithms to investigate these possibilities. These experiments were designed to evaluate and validate the effectiveness of our proposed strategies. In general terms, our research contributed in the following ways:

- A comprehensive collection of real DNS traffic traces was conducted to gather diverse parameters and relevant data, (e.g., domain names, IP adress of the source and the destination,etc.)

- The prediction accuracy of traditional machine learning algorithms was thoroughly evaluated, and the findings confirmed an accuracy rate of over 95% when identifying users based on DNS traffic.

- We have employed two approaches to propose multiple techniques for obfuscating the DNS fingerprint. To be more precise, these techniques involve utilizing stochastic processes and relevance metrics.

- The proposed algorithms were successfully implemented and achieved accuracy reductions greater than 50%. However, this achievement presents a trade-off between privacy and Internet browsing (user experience), demonstrating the substantial value of the results obtained.

- An exhaustive study was carried out to evaluate the impact of our strategies on real users.

1.7 Thesis Organization

The thesis is structured as follows: Chapter 2 provides an overview of the DNS protocol. This is followed by Chapter 3, which includes a literature review and an examination of the state-of-the-art methodologies in identification techniques, focusing on machine learning algorithms for DNS fingerprinting and their countermeasures. Chapter 4 presents our main threat models, consisting of three machine learning classifiers. Chapter 5 presents the proposed strategies to minimize these threat models' accuracy. Next, Chapter 6 covers the simulations and results of our proposed strategies. Chapter 7 explores the real-life implications of our algorithms, analyzing their impact in practical scenarios. Then, Chapter 8 offers potential directions for future research. Finally, in Chapter 9, our conclusions of this thesis.

Chapter 2

Domain Name System (DNS) background

DNS is a hierarchical, and distributed system, forming the foundation for almost all communication on the Internet, starting with a DNS lookup. The DNS infrastructure comprises distributed databases accessible throughout the network, providing information related to requests made by hosts or users within the network. In most simple terms, it translates a domain name to an IP address; For instance, when users such as Internet browsers or mobile devices issue DNS queries, they are essentially seeking the IP address of the server containing the information they requested initially. This ensures that the content requested by the end user can be displayed correctly on web browsers or mobile devices.

The stability, consistency, and efficient management of DNS are crucial for the smooth operation of Internet. Properly functioning DNS ensures that applications can be accessed and used reliably, making it a vital aspect of the configuration of any Internet node. DNS can be managed locally, allowing for adding, removing, or modifying information for specific hosts. DNS ensures the Internet works smoothly, loading content we ask through HTTP/HTPPS requests quickly and efficiently. It is an essential component of how the Internet works. With DNS, it would be easier for users to remember and access the vast amount of content available on the Internet.

2.1 DNS architecture

DNS is divided into three main parts: the namespace, the resolver, and the servers. The namespace is the system section containing the domain names and their corresponding IP addresses. It is organized like a tree structure, starting from the root domain and branching out to specific subdomains. The resolver, also known as the recursive resolver, functions as an intermediary responsible for processing DNS queries initiated by the client (end-user) and for receiving DNS responses. By Utilizing the DNS namespace, the resolver navigates the hierarchical structure and accurately locates the essential information. Consequently, the resolver plays a pivotal role as a crucial link in the process of translating domain names into IP addresses.

The DNS servers are machines distributed all over the Internet and provide DNS information. They are responsible for answering the resolvers' DNS queries and providing the necessary information to translate domain names to IP addresses and vice versa. These three components work together to create a functionally delegated system that resolves domain names to IP addresses and vice versa, allowing the end-user to access the desired website or service by simply typing its domain name in the web browser.

DNS hierarchy

The entire collection of DNS administrative domains worldwide is organized in a hierarchy known as the DNS namespace. This section explains how the organization of the namespace impacts local domains and the Internet. The DNS hierarchy, the domain name space, can be visualized as an inverted tree structure. At the top of this structure is a single domain called the root domain, denoted by the symbol ".". Like the UNIX file system, DNS domains are organized as descending branches, resembling the roots of a tree. As shown in Figure 2.1, each branch represents a domain, and each subbranch represents a subdomain.

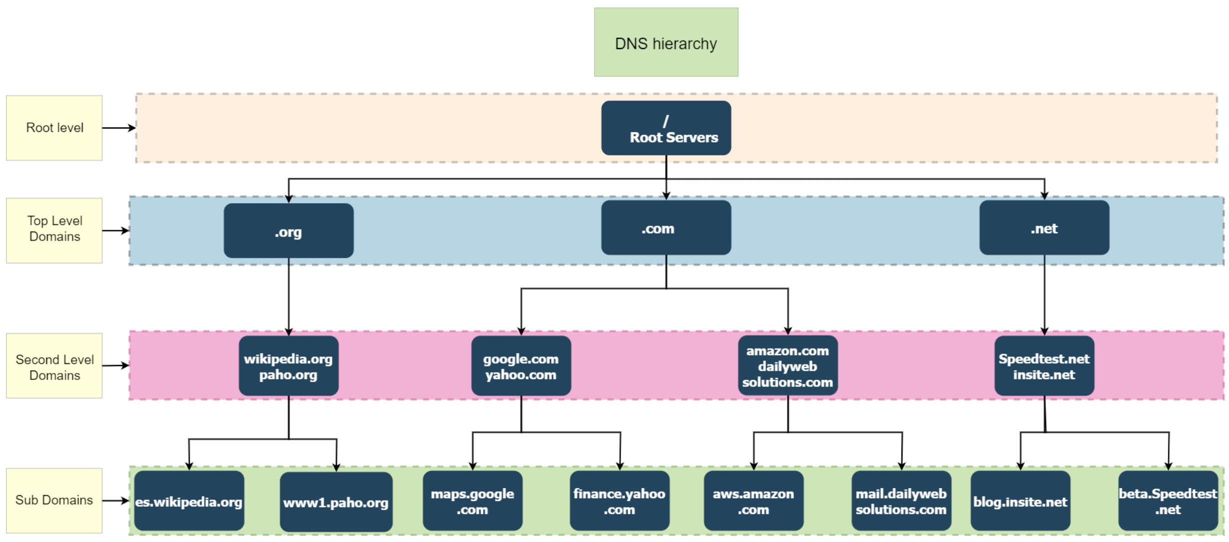


Figure 2.1: DNS hierarchy example

It is important to note that the terms "domain" and "subdomain" are relative. A specific domain is considered a subdomain to the domains above it in the hierarchy, while it acts as a parent domain to the subdomains beneath it.

Under the root domain are the top-level domains that divide the DNS hierarchy into segments containing second-level domains, sub-domains, and hosts. Hence, the DNS hierarchy is comprised of the following five levels:

- Root Level Domain
- Top Level Domains (TLD)
- Second Level Domains (SLD)

- Subdomains
- Hosts

While FQDN stands for Fully Qualified Domain Name, it is essential to note that FQDNs are used in data processing for this work. It represents a complete and unambiguous domain name that specifies the precise location of a specific host within the DNS hierarchy. An FQDN consists of multiple parts, separated by dots, with the most specific information appearing on the left and the root domain on the right. It comprises the hostname (or subdomain), the domain name, and the top-level domain (TLD). FQDNs play a crucial role in uniquely identifying and locating specific resources, such as websites or network devices, on the Internet.

The example of the elements of an FQDN can be best described in Figure 2.2

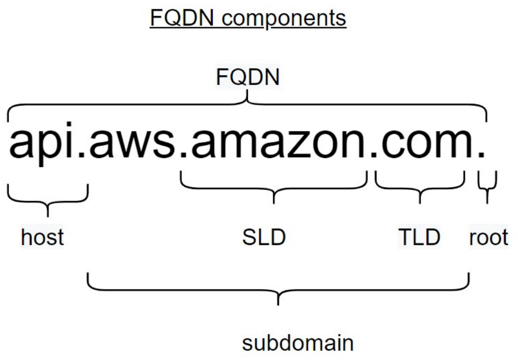


Figure 2.2: Fully qualified domain name
from:[2]

2.2 Elements of the DNS protocol

The DNS protocol encompasses various elements that work together to ensure accurate name resolution for end users. These elements are integral to translating domain names into IP addresses, enabling seamless communication over the Internet.

2.2.1 Type of DNS Servers

The most common and essential DNS server types are used to resolve hostnames into IP addresses.

DNS Resolver, a recursive resolver, or DNS recursor, acts as an intermediary between a client and DNS nameservers. Figure 2.3 depicts the process in steps 1 and 2. Upon receiving a DNS request from the client, the resolver checks its cache for the requested domain information. If the necessary information is already cached, the resolver sends the response back to the client. However, if the required information is not found within the cache, the resolver proceeds to step 3. During this stage, the DNS resolver initiates the corresponding sequence of request(s), starting with the root nameserver. The DNS Root

Server serves as the initial step in the process of mapping a hostname to its corresponding IP address. The DNS Root Server extracts the Top Level Domain (TLD) from the query, such as "www.example.com." It then forwards the request to the TLD Name Server responsible for that specific TLD (step 5), for instance, the .com TLD Name Server. This TLD Name Server is responsible for providing information related to domains within the .com, like "example.com.". Which in turn guide the resolver to the authoritative nameservers responsible for the specific subdomain. Upon reaching the authoritative nameservers, the resolver obtains the corresponding IP address. Subsequently, the resolver provides the resolved IP address to the client (steps 8 and 9).

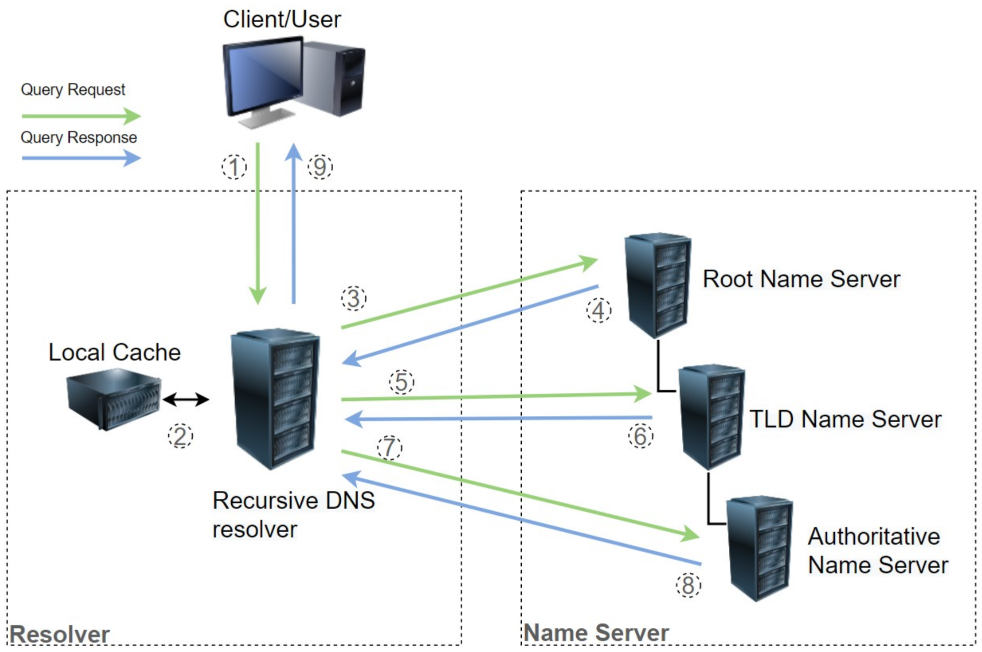


Figure 2.3: DNS architecture

2.2.2 Types of DNS Records

The DNS system uses a set of resource records, each with a specific purpose. These records are used to store information about a domain name and its associated IP address. Some of the most common resource records are:

- **A**: It holds the IPv4 address for a domain name. It maps a domain name to its corresponding IP address.
- **AAAA**: It holds the IPv6 address for a domain name; it is similar to the A record but is used for IPv6 addresses.
- **CNAME**: It is used to forward one domain or subdomain to another domain. It does not provide an IP address but allows a domain to have multiple names.

In the context of this work, our focus is directed towards A records. Although CNAME records provide an alternative method for domain resolution, their function of forwarding domains is not well-suited for the precise IP address association required by classifiers. On the contrary, A records assume a significant role in shaping the outcomes of our machine learning classifiers. They play a crucial part as they signify the final step in the resolution process, returning the IP addresses associated with particular domain names.

2.3 Caching

Caching [18] is also essential to DNS resolution. Once a DNS resolver has looked up the IP address for a domain name, it will cache that information for a certain amount of time, called the Time to Live (TTL), so that it can quickly return the IP address in the event of a subsequent request for the same domain name. This reduces the load on the DNS hierarchy and speeds up the resolution process.

The TTL is a critical component in the DNS protocol. This value sets the duration for which a particular DNS record will remain in the DNS resolver's cache [19]. The TTL value significantly impacts DNS performance and can directly affect a website's availability and user experience. Understanding the importance of TTL values and their role in DNS caching is crucial for website operators and network administrators. *"This value is controlled by the original (authoritative) DNS server and represents the length of time that other DNS servers and applications are allowed to store the given DNS Record before they must discard it and, if needed, request its new copy. In practical terms, the TTL value determines the validity period of the provided 'symbolic-name to IP-address' mapping"* [20, p. 467].

According to RFC 6195, [21] when a requested resource record R is not found in the cache, the client initiates a request to a bottom-level DNS server, usually belonging to the Internet Service Provider (ISP). If the record R is not found in the local cache, the request is forwarded to a higher-level server in the hierarchy. This process continues until the requested record R is obtained from either a cache or the disk of an authoritative server. Once the server providing R is located, it is sent back to the client via the reverse path between the answerer and the client, leaving a copy of R at each cache on this path. The time-to-live (TTL) value assigned to each copy of R by the answerer determines the duration for which the copy may be cached. This TTL value ensures that all copies of the record along the path are cached for a similar duration. This is called *TTL rule* in the literature.

Caches that adhere to this rule are referred to as traditional DNS caches, while those that ignore the TTL value and choose a locally defined value instead are known as modern DNS caches [22].

Traditional DNS caches were designed to minimize the time it takes to look up a domain name by storing the results of recent queries. These caches are typically implemented as part of the DNS server software, and they work by storing the results of previous DNS queries in memory. This approach is efficient, but it has some limitations. For instance, traditional DNS caches can quickly become overwhelmed by the sheer number of DNS queries that are processed by modern Internet networks. Furthermore, traditional DNS caches can be prone to certain types of attacks, such as DNS cache poisoning, which can compromise the integrity of the cache and cause it to return incorrect results.

Modern DNS caches, on the other hand, are designed to address these limitations by utilizing sophisticated algorithms and data structures that can handle large volumes of DNS queries while maintaining high levels of performance and security. For example, modern DNS caches are often implemented as standalone applications that run on specialized hardware or virtual machines. These caches are optimized to handle a wide variety of query types and are designed to adapt to changing network conditions and DNS traffic patterns. Modern DNS caches often employ advanced security features, such as secure communication protocols and Domain Name System Security Extensions (DNSSEC) validation, which help protect against DNS-based attacks. Overall, modern DNS caches are more efficient, secure, and scalable than traditional DNS caches, making them an essential part of today's Internet infrastructure.

Figure 2.3 shows the immediate interaction between the resolver and the local cache in step number 2. This means that if a requested domain by the client is in the cache, the response will be immediate since the query is already stored in the cache, and the query does not have to be forwarded to a higher-level DNS server. The process of query forwarding continues until a result is returned to the user program. Therefore, caching allows the best user experience in the website/application and decreases the DNS traffic network's flow.

Chapter 3

State of the art

3.1 Identification techniques

Machine learning algorithms have been extensively studied and proven to be a powerful tool in various fields. These mathematical models demonstrate their efficacy in diverse applications, including DNS fingerprinting, where they can predict classes with a certain probability. These models are often employed to identify users, represented by unique identifiers like IP addresses (classes). For instance, Herrmann et al. [5] implemented Multinomial Naive Bayes (MNB) for re-identification attacks using DNS traffic. Although MNB is a simple model that naively assumes conditional independence of each feature, it is essential to acknowledge that this assumption is complex to model in real-life scenarios. However, despite its simplifications, the efficacy of MNB has been widely proven in the literature, making it a valuable and robust tool for various classification tasks with an accuracy of around 90% when identifying users. Additionally, Arana et al. in [3] introduced linear classification models such as Support Vector Machine, achieving accuracies above 96%. This demonstrates the potential of machine learning methods in accurately predicting user identities based on DNS data.

In 2023, a group of researchers delved into the domain of bot detection and DNS fingerprinting, as described in [23]. Their investigation highlighted the effectiveness of ensemble learning models, such as decision trees, showcasing an accuracy of 99.5% in making predictions. This emphasizes the significance of employing advanced algorithms for tasks related to DNS fingerprinting and its applications.

The successes observed in these studies further reinforce the notion that machine learning algorithms play a pivotal role in DNS fingerprinting attacks, enabling precise predictions of user identifiers and facilitating efficient predictions. Moreover, Vekshin et al. [24] demonstrated that DNS over encrypted channels can also be accurately identified, achieving an impressive accuracy rate of 99%. To achieve such results, they utilized additional algorithms, including K-Nearest Neighbors, AdaBoosted Decision Tree, and Random Forest. This further exemplifies the versatility and effectiveness of machine-learning approaches in DNS-related tasks.

The utilization of machine learning algorithms in DNS fingerprinting continues to showcase its potential to predict user identifiers and identify DNS over encryption channels

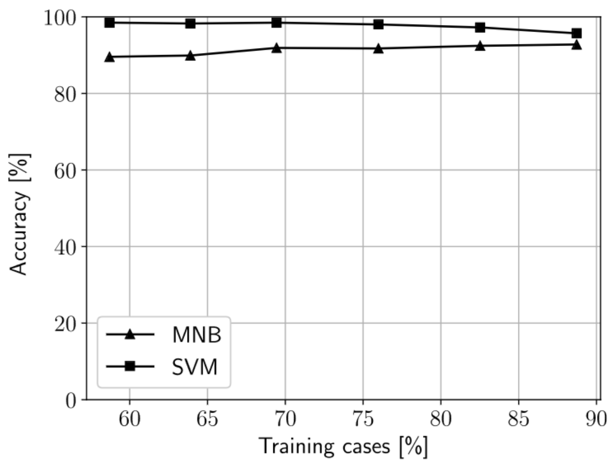


Figure 3.1: Accuracy of DNS fingerprinting by Arana et. al.[3]

accurately. This underscores the significance of these techniques in network analysis and intrusion detection systems.

3.2 Countermeasures

As mentioned in Chapter 1, several works have been proposed to protect DNS and try to obfuscate its fingerprint. These protocols and implementations help to secure DNS by encrypting/anonymizing the communication between the client and the server, reducing the risk of tampering and eavesdropping. However, it is worth noting that adopting these security strategies often entails high computational costs, complex implementation processes, and, in a few cases, high latency, which can pose challenges for widespread implementation and deployment, thus harming the end-user experience. In addition, most of the proposed strategies embrace intermediates or relays. Meanwhile, the approaches that solve this problem in a solitary approach might leverage certain information to the recursive resolver that can still targeting us. Moreover, we can categorize the existing strategies as "Single-node-based strategies" or "Intermediary/relay-based strategies" countermeasures.

3.2.1 Single-node-based strategies

Various implementations utilize encryption-based algorithms to enhance DNS security. For instance, DNS over HTTPS (DoH) [10] ensures secure DNS communication by encapsulating DNS queries and responses within HTTPS, utilizing encryption to protect the data in transit. Nevertheless, as mentioned earlier, in [24], this strategy can be easily detected. EncDNS [25] takes a similar approach, emphasizing encryption to safeguard DNS traffic from unauthorized access and tampering.

Furthermore, some solutions propose tunneling methods, such as DNS over TLS (DoT) [11]. In DoT, clients and local or recursive resolvers establish a Transport Layer Security (TLS) session to create a secure tunnel for sending DNS queries. Similarly, DNS over QUIC (DoQ) [12], which was introduced to the IETF in 2017, offers similar privacy properties to DoT. DoQ aims to simplify the traditional HTTPS stack (consisting of HTTP/2, TLS, and TCP) by multiplexing multiple HTTP connections over a QUIC tunnel operating on top of the UDP transport layer. Thus, makes the collection of information for the recursive resolver simpler. Although these features protect user DNS traffic from passive eavesdropping by intermediary nodes, they may still allow DNS resolvers to collect traffic information, which could be utilized to estimate user fingerprints. This limitation arises because all queries are transmitted over specific ports that can be detected, potentially compromising the privacy and security of the communication.

Furthermore, certain implementations incorporate intricate mathematical operations to enhance DNS security. DNSCurve [26] employs elliptic curves and uses public key cryptography, where the client and the resolver have a unique pair of public and private keys. When the client sends a query to the resolver, it encrypts it using its public key. The resolver can then decrypt the query using its private key, process it, and send the response back to the client. Similarly, F. Denis and Y. Fu proposed DNSCrypt [27], which employs cryptographic signatures to verify the authenticity of DNS responses and ensure the integrity of DNS transactions. It is important to acknowledge that these encryption and mathematical operations may introduce additional computational overhead while effectively improving overall security, resulting in increased response times for real-time applications. Striking a balance between security and performance remains crucial when implementing these protocols in real time.

Moreover, there have been proposals for strategies to send additional queries to obscure the fingerprints. Herrmann et al. introduced a technique called "Range Query" [13]. When a client needs to issue a DNS query to a resolver, it randomly selects (without replacement) $N-1$ *dummy* domain names from a database. It sends a total of N queries to the resolver. After receiving all the replies from the resolver, the responses for the *dummy* queries are discarded, and the desired reply is presented to the application that initiated the query. However, it has been observed that machine learning algorithms can learn to disregard these *dummy* queries, thereby affecting the effectiveness of these solutions.

3.2.2 Intermediary/relay-based strategies

Strategies that rely on collaborative nodes or intermediaries acting as middle-man in the network have also been proposed. Liu et al. proposed in 2020 [28] a new third-party DNS service called T^{2}DNS, which aims to address various privacy and trustworthiness concerns using a hybrid solution. T^{2}DNS consists of an agent installed on each client and a proxy between the clients and the Internet DNS service. The communication between the agent and the proxy uses a secure protocol, while the communication between the proxy and the Internet DNS servers uses obfuscation techniques.

The "Never Query Alone" approach, as outlined in [3], is also centered on an intermediary/collaborative strategy. The fundamental idea behind this technique is to divide users' queries among a trusted group of nodes rather than relying on a single node. Each node in the group is responsible for handling a subset of the user's queries, which are distributed within the network. Essentially, each node belongs to a multicast group in a local network, and they work together to minimize the risk of being tracked by DNS resolvers, sending its queries to "neighbors" in the network.

In addition, one of the most used strategies today is The Onion Router [16]. Regarding DNS, when a user requests to access a website, the request is first sent to a local DNS resolver to resolve the domain name to an IP address. However, when using Tor, the request is sent to the circuit's first onion router. This router acts as a local DNS resolver and resolves the domain name to an IP address. Then, the request is encrypted and sent to the next onion router in the circuit, and so on, until it reaches the final onion router, called the exit node. The exit node then sends the request to the destination IP address on behalf of the user. Nevertheless, the most noticeable drawback of this solution is the high latency of around 4.1 s [17] per query, making it hard to use as a day-to-day solution.

In 2021, Jun Kurihara and Takeshi Kubo proposed Oblivious DNS (ODNS) [14], a strategy that leverages an existing full-service resolver over Do53 as a relay. This relay acts as an intermediary, forwarding encrypted DNS messages between the ODNS resolver and the user, hiding the user's address from the ODNS resolver.

Additionally, various solutions have been proposed that combine encryption algorithms with intermediaries or relays to enhance DNS privacy. Approaches like Oblivious DNS over HTTPS (ODoH) [14, 29], which combines the privacy benefits of DNS over HTTPS with the obfuscation provided by ODNS. This is accomplished by encrypting the DNS query using the HTTPS protocol, which is commonly used for secure web browsing. The ODoH client encrypts the DNS query and sends it to a server known as a "proxy resolver." The proxy resolver then decrypts the query, performs the DNS lookup, and encrypts the response before sending it back to the client.

As a modification of ODNS and ODoH [14, 29, 30], authors in [15] aim to protect the privacy of DNS users by hiding their DNS queries from the DNS resolver and any intermediate parties. The main difference between Mutualized Oblivious DNS and Oblivious DNS is that in Mutualized Oblivious DNS, the encryption and decryption of the queries take place at the client and server ends, respectively. Also, the queries are grouped together with other users' queries before being encrypted. This way, the queries are "mixed," and it is difficult to tell which query belongs to whom. This "Mix-Net" technique is used to increase privacy and anonymity.

Furthermore, an anonymized version of DNSCrypt [31, 32] offers a different approach to enhancing DNS privacy. DNSCrypt anonymizes DNS queries by encrypting and routing them through a network of relays, making it difficult for eavesdroppers to trace the origin of the queries.

Summary

As mentioned, most strategies [28, 3, 16, 14, 15] depend on intermediaries or relays to hide the DNS fingerprint of users. While such approaches might employ a combination of cryptographic methods and possess a robust structure, any potential failure in these intermediary nodes can significantly undermine the overall effectiveness of these methodologies. We propose shifting towards a single-node-based strategy to address this vulnerability, which aims to obfuscate our DNS fingerprint without relying on intermediaries. By adopting this approach, our solution demonstrates its efficacy through a streamlined DNS configuration that can be implemented natively in any Unix-like system. This enhances the system's overall reliability and simplifies the implementation process, making it more accessible to users with varying technical expertise. In the upcoming chapters, we will explore the details of this strategy based on a single node. We will also highlight its benefits in effectively achieving a secure obfuscation of the DNS fingerprint.

Chapter 4

Threat Models

As previously discussed, using machine learning classifiers in the context of DNS traffic analysis is a growing concern for organizations and individuals. With the ability to identify individual users with increasing accuracy, attackers can pose a significant threat to the privacy and security of users and organizations alike. Therefore, proactive measures must be taken to mitigate these threats and protect against these sophisticated cyber-attack forms.

In this thesis, we evaluated three of the most commonly used machine learning classifiers in the literature that can classify users based on real DNS traffic data described as follows.

4.1 Multinomial Naïve Bayes

The Multinomial Naive Bayes model is proposed based on the characteristics of the data, as our problem involves multi-class classification (involving one class per user modeled). Although Naive Bayes and related probabilistic classifiers naively assume independence in the occurrence of its attributes (which is often not the case for real-world problems), it has been employed due to its simplicity and overall performance; in addition, is it the most commonly used model in the literature. [3, 5, 24].

For instance, each session representing a day of DNS traffic can be seen as an individual instance denoted by c_i belonging to a specific class C. These classes correspond to different users in the classification task. Within each session, there is a multiset of destinations denoted as $(x_1^{f_{x_1}}, x_2^{f_{x_2}}, x_3^{f_{x_3}}, ..., x_m^{f_{x_m}})$, where $x_i, i = 1, ..., m$ represents a particular destination (such as a domain name) requested by a user, and f_{x_i} represents the frequency or number of times that destination was visited during the session. These frequencies are non-negative integers and provide information about the importance or popularity of each destination within the session.

From this collection of destinations, an attribute vector $x = f = (f_{x_1}, f_{x_2}, f_{x_3}, ..., f_{x_m})$ can be derived, representing the visited websites for that session. This attribute vector captures the frequency of visits to each destination, quantitatively representing the user's activity during that session. To determine the probability that an instance or session represented by the attribute vector f belongs to a particular class c_i. By calculating the conditional probability $P(f \mid c_i)$ for each attribute vector f and each class c_i in the training data, we can utilize these probabilities to classify new instances or sessions based on their observed attribute vectors. Thus, given m unique destinations the classifier evaluates the probability given instance f belongs to some class c_i as follows:

$$P(f|c_i) \sim \prod_{j=1}^{m} P(X = x_j|c_i)^{f_{x_j}} \tag{4.1}$$

Where $P(X = x_j|c_i)$ describes the probability that a destination x_j comes from the aggregate multiset among all visited websites of the training class c_i, the value of f_{x_j} describes the number of times visited the destination x_j in the test instance.

4.2 Support Vector Machines

We also introduced a more complex attacker model that involves support vector machines (SVM). It is essential to note that in its most basic form, SVM does not inherently support multi-class classification. Instead, it excels in binary classification, where data points are separated into two classes. To handle multi-class classification, we employ a One-to-One approach [33], which involves breaking down the multi-class problem into multiple binary classification problems. The idea is to map data points to a higher-dimensional space to achieve linear separation between each pair of classes. This involves creating a binary classifier for each pair of classes.

SVM manipulates simple mathematical models of the form $y = w \cdot x^T + \gamma$, where $x \in X$ represents the data domain and $y \in Y$ is the response variable. The central concept behind SVMs is to enable the creation of linear boundaries within a domain by manipulating this straightforward mathematical model using regression plane fitting, which serves as the separation boundary for the given data domain. The weight vector w and the bias term γ are obtained by solving a quadratic programming problem in its dual form.

As mentioned, the input vectors are transformed into a high-dimensional feature space using a nonlinear mapping function of $\varphi(x)$. This transformation allows for constructing a bounding plane function using the transformed data. The representation of the bounding plane function is as follows:

$$y_i \left[w \cdot \varphi(X_i^T) + \gamma \right] \geq 1 - \xi_i, \quad \forall i, i = 1, ..., m \tag{4.2}$$

Equation 4.2 implies that for every input data point X_i, the transformed feature space is computed using $\varphi(X_i^T)$. The distance between the feature point and the hyperplane is then calculated as $w \cdot \varphi(X_i^T) + \gamma$, which should be greater than or equal to 1 to meet the criteria for correct classification. The slack variable ξ_i is introduced to relax the constraint when the data is not linearly separable.

The SVM model aims to find the optimal weight vector w and bias term γ that minimize the objective function $J(w, \xi)$ while satisfying the constraints. The optimization problem can be formulated as follows:

$$
\begin{aligned}
&min_{x,\gamma\ ,\xi} J(w,\xi) = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i \\
&s.t. \begin{cases} y_i[w\varphi(X_i^T)+\gamma] \geq 1-\xi_i \\ \\ \xi_i \geq 0, i = 1,...,N \end{cases}
\end{aligned}
\tag{4.3}
$$

In Equation 4.3, the objective function $J(w,\xi)$ consists of two terms: the first term $\frac{1}{2}w^T w$ controls the complexity of the model. The second term $C\sum_{i=1}^{N}\xi_i$ penalizes the misclassification of training data. The parameter C is a regularization constant determining the trade-off between achieving a low training error and maximizing the margin. The constraints enforce that the transformed data points lie on the correct side of the separation boundary.

Solving the optimization problem in Equation 4.3 typically involves using algorithms such as the dual form of the problem with a linear cost function. By formulating the problem in this manner and solving it using appropriate algorithms, SVMs can effectively find the optimal separation boundary for the given data set, taking into account both the complexity of the model (controlled by the first term $\frac{1}{2}w^T w$) and the misclassification of training data (penalized by the second term $C\sum_{i=1}^{N}\xi_i$). This allows SVMs to balance achieving a low training error and maximizing the margin between the data points and the separation boundary.

The regularization constant C plays a crucial role in SVMs. When C is large, the optimization algorithm prioritizes minimizing the training error, potentially leading to a narrower margin and a higher risk of overfitting the data. On the other hand, when C is small, the algorithm focuses more on maximizing the margin, which can result in a larger margin but potentially at the cost of increased training error.

The *LinearSVC* implementation in sci-kit-learn provides an efficient way to solve the optimization problem in Equation 4.3 for linear SVMs. It utilizes the *liblinear* library [34], which is based on the linear programming solver for large-scale machine learning tasks. This implementation enables SVMs to handle large datasets and efficiently find the optimal weight vector w and bias term γ that define the linear separation boundary in the high-dimensional feature space.

4.3 Random Forest

As a third and more robustly proposed attacker model compared to the previous ones, we have defined an ensemble approach. Decision trees are non-parametric supervised learning models aimed at developing a predictive model for the target variable, in this case, an IP address. These decision trees learn straightforward rules from the domain names within the data, enhancing the accuracy of our predictions. A tree can be considered a piecewise constant approximation; regression and classification could be done with them. For our attacker modeling, the classification method was employed.

Since we are predicting the correct labeling of users with a certain probability value, we employ the ensemble learning algorithm Random Forest from the Scikit-Learn library [33]. Specifically, the class *DecisionTreeClassifier* is utilized, as it is well-suited for our multi-class classification problem.

The Random Forest algorithm builds a collection of decision trees that partition the feature space based on the training vectors $x_i \in \mathbb{R}^n$ and the corresponding label vector $y \in \mathbb{R}^l$. The goal is to group samples with the same labels or similar target values. The feature space is recursively partitioned to form a tree structure. At each node m of the tree, the data is represented by Q_m, which consists of n_m samples. To split the data at node m into left and right subsets, a candidate split $\theta = (j, t_m)$ is considered, where j represents a feature and t_m is a threshold. The data is then divided into subsets $Q_m^{\text{left}}(\theta)$ and $Q_m^{\text{right}}(\theta)$ based on this split.

$$\begin{gathered} Q_m^{left}(\theta) = \{(x, y) | x_j \le t_m\} \\ Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta) \end{gathered} \tag{4.4}$$

When using a decision tree for classification, the quality of a potential split at a node is determined by an impurity function denoted as $H()$. This function measures the level of disorder or randomness in the samples at the node under consideration. For classification, we utilize the impurity function:

$$G(Q_m, \theta) = \frac{n_m^{\text{left}}}{n_m} H(Q_m^{\text{left}}(\theta)) + \frac{n_m^{\text{right}}}{n_m} H(Q_m^{\text{right}}(\theta)) \tag{4.5}$$

The decision tree selects the parameters that result in the lowest impurity level, minimizing the impurity as follows:

$$\theta^* = \text{argmin}_\theta G(Q_m, \theta) \tag{4.6}$$

The process recurses for subsets $Q_m^{\text{left}}(\theta)$ and $Q_m^{\text{right}}(\theta)$ until the maximum allowable depth is reached ($n_m < \text{min}_{\text{samples}}$) or $n_m = 1$. In this context of classification, the Gini impurity formulation [35] is employed. Gini impurity is a metric used to determine how the features of a dataset should be used to split nodes and create the tree. It is a value between $0 \le gini \le 1$ that represents the probability of misclassifying new and random data if they were assigned a random class label based on the class distribution in the dataset.

For a target that is a classification outcome taking values $0, 1, ..., k-1$ for node m, let

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k) \tag{4.7}$$

be the proportion of class k observations in node m. The Gini impurity measure is then defined as:

$$H(Q_m) = \sum_{k=0}^{k-1} p_{mk}(1 - p_{mk}) \tag{4.8}$$

Additionally, the logarithmic loss or entropy can be calculated as follows:

$$H(Q_m) = -\sum_{k=0}^{k-1} p_{mk} \log_2(p_{mk}) \tag{4.9}$$

The decision tree effectively determines the splits that best separate the classes or target values by selecting the parameters that minimize the impurity function. The Random Forest algorithm further improves the predictive performance by aggregating multiple decision trees. Each tree is trained on a random subset of the training data using a technique called bagging, and the final prediction is obtained through majority voting or averaging the predictions of individual trees.

Summary

This Chapter outlines the characteristics and functionalities of each attacker model, highlighting their mathematical capacity to process and analyze DNS fingerprints based on real data. By employing machine learning algorithms, these attacker models demonstrate a sophisticated understanding of DNS traffic patterns, making them formidable adversaries in DNS fingerprinting.

The subsequent Chapter describes the core focus of the research: introducing bias into the aforementioned machine learning classifier models to make them prone to inaccurate predictions.

The results of the experiments reveal the effectiveness of the proposed bias-inducing algorithms. By introducing carefully crafted obfuscation techniques and perturbations to DNS fingerprints, the machine-learning classifiers exhibit notable deviations in their predictive accuracy. This analysis highlights the importance of understanding and mitigating biases in machine learning-based security systems.

Chapter 5

Proposed algorithms

As previously discussed in Chapter 1, our online activity generates a digital footprint that can be analyzed using a Zipf-like popularity distribution. In addition, these patterns are well identified by machine learning algorithms, as also previously discussed. Therefore, it is important to create countermeasures. This Chapter focuses on two methods for obfuscating the DNS fingerprint that implement selective domain swapping or blocking. The first approach involves the random alteration of incoming DNS queries directed toward the DNS recursive resolver. By employing this method, we can introduce stochastic bias into the machine learning classifiers utilized by potential adversaries. This, in turn, enables us to modify the patterns of DNS queries.

In addition, we adopted a more sophisticated approach to enhance our DNS obfuscation efforts further. We implemented established relevance metrics commonly used in text classification. This allowed us to evaluate the importance of each domain name (query) within our dataset and gain insights into their relative significance. By leveraging this metric, we could effectively analyze and prioritize the relevance of each domain name, making informed decisions regarding our DNS obfuscation strategies. We could identify each user's most relevant domain names by applying it to DNS, enabling us to implement effective DNS fingerprint obfuscation strategies. Thus, we can create a virtual table or ranking of the most important domain names by each user in the dataset.

Figure 5.1 outlines the general workflow of our methodology, which primarily centers around the defensive aspect of developing an obfuscation technique. We aim to safeguard users from potential attackers depicted in the red-dotted box. These attackers may include entities such as the DNS server (DNS resolver) or third-party agents seeking to exploit DNS fingerprint data for various purposes. It is pertinent to highlight that our implementation relies on the client-side, thus, avoiding the need of any intermediary/relay in the network.

5.1 Blocking-based algorithms

We conceptualized and developed a preliminary algorithm using stochastic process as the core mechanism named P-Block. This implementation integrates a blocking function to effectively handle the resolution of requested queries on a local level. By incorporating these functions, the algorithm can efficiently process, and a local DNS resolution responds

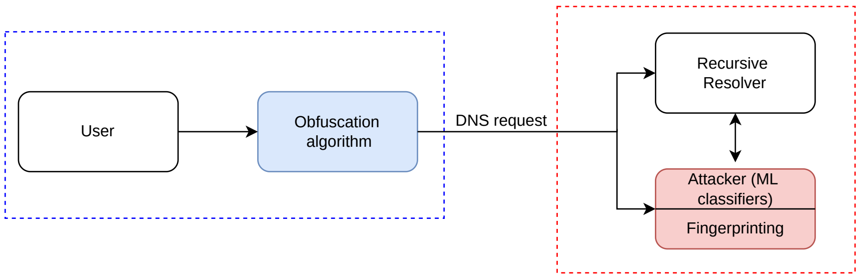


Figure 5.1: Block diagram of how our proposed algorithms work

to the queries.

It is worth noting that our *"blocking"* is defined as taking this domain from our initial local cache with a fixed TTL. As a result, the resolution will be made locally from the OS by a predefined and adjustable time instead of being sent to the resolver. This local resolution mechanism allows us to efficiently retrieve the domain from the local cache, which already contains the necessary information and thus avoids sending the query externally to a DNS resolver. By leveraging the local cache and a predetermined TTL, we can enhance our DNS resolution process's overall efficiency and responsiveness for frequently accessed domains. However, it is worthwhile to mention that a high TTL values may lead to inconsistencies in the data.

5.1.1 P-Block

As an initial experiment, we proposed the P-Block method, considered the most straightforward approach. This method uses a *"toss a coin"* with a uniformly distributed probability for each query. A control parameter p is defined and dynamically changed for different test cases based on the probability result in each requested domain. The decision to block a domain name depends on the outcome of the probability, and the comparison of the probability result with a value previously set of the control parameter. Although this method may be considered simplistic, it serves as a fundamental starting point for our research, by exploring into the potential of probabilistic-based obfuscation methods. In the subsequent sections, we will explore more advanced strategies that build upon the groundwork laid by the P-Block approach, aimed at further enhancing the efficacy of DNS fingerprint obfuscation.

Figure 5.2 shows a general workflow of the logic behind P-Block algorithm. Where we split the data to train the classifiers, and further, proceed with the stochastic process in the test data, to then, make predictions. To clarify the behavior mentioned above, during the actual algorithm testing, a random variable r is generated for each requested domain name, following a continuous uniform distribution. This random variable r captures the element of chance in the process. Let r_i denote the random variable generated for the i-th requested domain name during the testing phase. The probability distribution r_i can be denoted as $P(r_i)$. The static control parameter p, defined previously and adjusted for different test cases, acts as a threshold. Therefore, the corresponding query is blocked if the generated value falls below the established p value. On the other hand, if the generated value is higher than the p value, the query is allowed to proceed. This approach introduces an

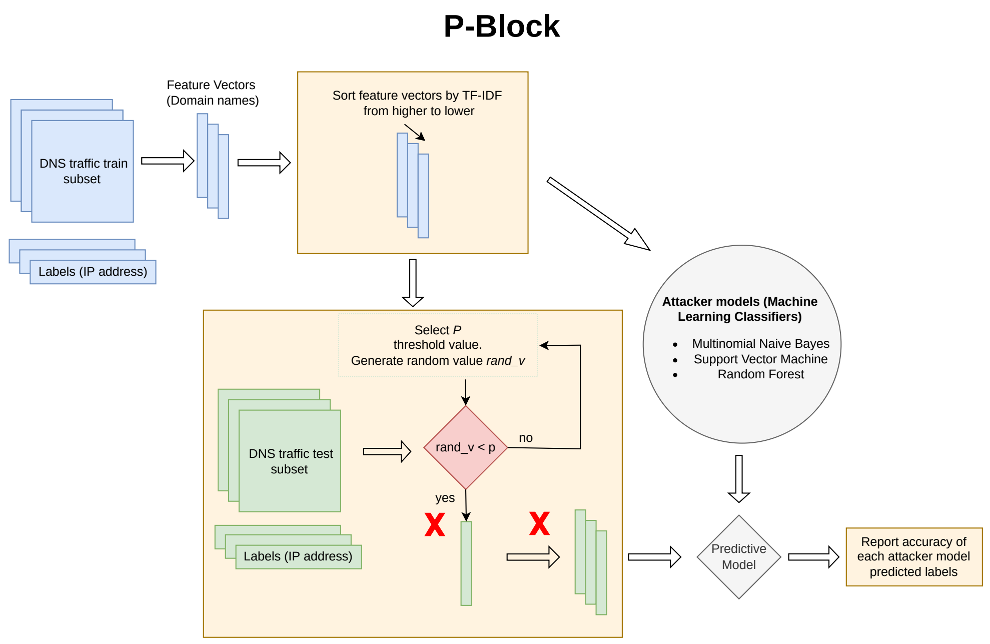

P-Block
DNS traffic train subset
Labels (IP address)
Feature Vectors (Domain names)
Sort feature vectors by TF-IDF from higher to lower
Attacker models (Machine Learning Classifiers)
Multinomial Naive Bayes
Support Vector Machine
Random Forest
Select P threshold value. Generate random value rand_v
DNS traffic test subset
Labels (IP address)
rand_v < p
no
yes
X
X
Predictive Model
Report accuracy of each attacker model predicted labels

Figure 5.2: Theoretical P-Block workflow

element of randomness into the decision-making process, aiming to make it more difficult to discern the user's identity. The control parameter p is defined as the following subset: $p \in (0.1, 0.99)$
The random variable r has a uniform distribution in the interval $(0, 1) = \{r \in \mathbb{R} : 0 \leq x \leq 1\}$, if its density function is constant in that interval, or, equivalently:

$$f(r) = \begin{cases} \frac{1}{b-a}, & a \leq r \leq b \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

5.1.2 Top-Block

Like our earlier approach with P-Block, we also incorporate a blocking mechanism in Top-Block. However, there is a significant distinction in this methodology. In Top-Block, we adopt a more targeted strategy, selecting only the top 10% of a fixed rank generated through the utilization of Term Frequency Inverse Document Frequency (TF-IDF). This entails creating a domain name ranking specific to each user, allowing us to discern the most pertinent and crucial domains associated with each node in the network. Creating personalized domain name ranks empowers the Top-Block system to make informed decisions about blocking or permitting queries. Users can still access essential information without compromising their privacy and identity, making Top-Block a sophisticated and user-centric approach to network security.

TF-IDF

Since our primary goal is to obfuscate the DNS fingerprint and minimize the accuracy of our threat models, a well-reviewed mathematical representation of text documents was implemented. TF-IDF [36] represents each session as a document of words without an order. It is a statistical measure of how important a word is in a document within a corpus, and this document represents a day of traffic for each user.

IDF is well defined as a scalar over all the datasets, but the TF definition is in terms of the documents. To better understand, the Term Frequency can best describe how many times a domain name appears in the document, and the Inverse Document Frequency measures how many sessions (days) the domain name has appeared.

Formally, the algorithm aims to represent each document d as a vector $\vec{d} = d^1, ..., d^{(|F|)}$ in a vector space, such that documents with similar content have similar vectors. Here, $|F|$ corresponds to a set of words, and each dimension in the vector space represents a word selected during the feature selection process. The values of the vector elements d^i for a document d are computed as a combination of the statistics of the term frequency $TF(w, d)$ and $DF(w)$.

The *term frequency* $TF(w, d)$ measures how often word w appears in document d, and the *document frequency* $DF(w)$ counts how many documents contain at least one occurrence of word w. The inverse document frequency $IDF(w)$ is derived from the document frequency. These vectors are utilized to capture the semantic similarity between documents, facilitating tasks like clustering or classification.

$$IDF = \log\left(\frac{|D|}{DF(w)}\right) \tag{5.2}$$

Here, $|D|$ represents the total number of documents. The inverse document frequency of a word is lower when it appears in many documents and higher when it appears in only one. We then calculate the value d^i for feature w_i in document d as their product.

$$d^i = TF(w_i, d) \cdot IDF(w_i) \tag{5.3}$$

To this end, a *rank* of each domain name can be created. Thus, we created a virtual rank for each user corresponding to the most important domain names. More precisely, the domain names were sorted by the sum of the TF-IDF values. This means we only worked with the relevant queries based on this statistical metric.

Figure 5.3, shows the flow of how this approach works. Where we sort the feature vectors (domain names) by their importance based on the value of the results of TF-IDF. By this means, we can create the *rank* as previously discussed. In turn, block a small fraction (%) of this domain names.

To achieve this, we introduce a function $U_i(PR_i, p)$ for each user U_i, where PR_i represents the domain name visited by user U_i at rank R^i. This function acts as a gatekeeper, evaluating whether a domain should be blocked or allowed based on the page's rank visited by the respective user. To impart more flexibility and dynamism to our system, we incorporate a range of ranks controlled by the parameter p. This parameter modifies our previous function as follows: Let U_i represent the set of users where $U_i = \{1, 2, ..., N\}$. M

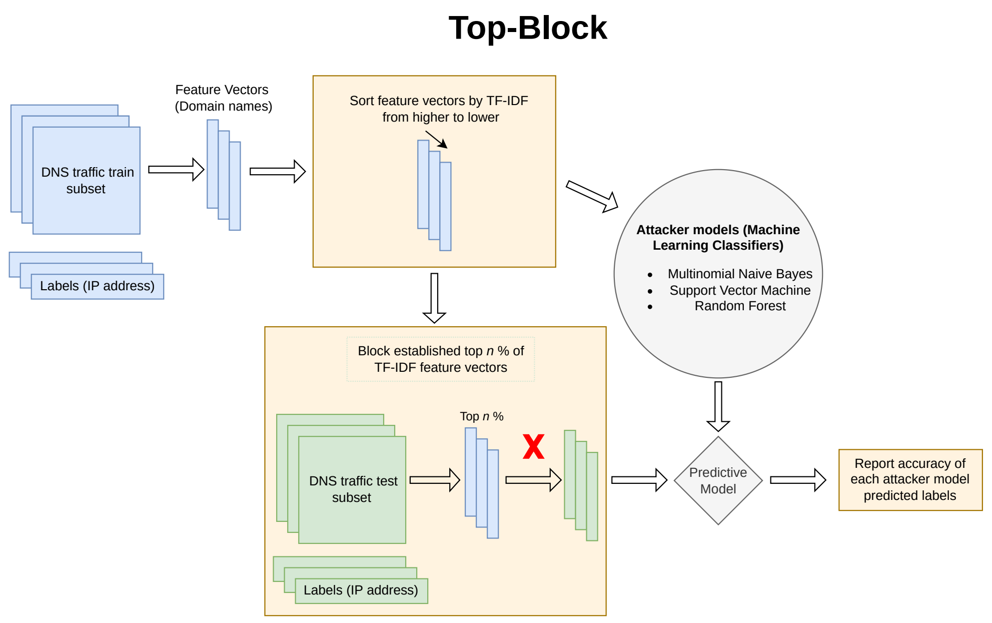

Top-Block
Feature Vectors (Domain names)
DNS traffic train subset
Labels (IP address)
Sort feature vectors by TF-IDF from higher to lower
Attacker models (Machine Learning Classifiers)
Multinomial Naive Bayes
Support Vector Machine
Random Forest
Block established top n % of TF-IDF feature vectors
Top n %
DNS traffic test subset
Labels (IP address)
Predictive Model
Report accuracy of each attacker model predicted labels

Figure 5.3: Theoretical Top-Block workflow

represent the total number of domains, where $P = \{1, 2, ..., M\}$. N_i represents the number of domain names in the ranked list for user U_i where $R_i = \{1, 2, ..., N_i\}$. Meanwhile, PR_i represents the domain name visited by user U_i at rank R^i where $PR_i \in M$. Thus, our function $U_i(PR_i)$ can be defined as follows:

$$U_i(DR^i, p) = \begin{cases} \text{Blocked}, & \text{if } R^i \leq \frac{p'}{100} \times N_i \\ \text{Allowed}, & \text{otherwise} \end{cases} \tag{5.4}$$

It is important to remark the range of p' within our experiments, where $p' \in [1, 10]$. Consequently, this shift results in a reduction of the blocking range when compared to our earlier method, P-Block, which blocking percentages ranging from 10 to 99%. This approach ensures a more refined and user-tailored blocking mechanism by introducing more bias to the classifiers. Top-Block is a sophisticated and single-node-based approach to network security regarding DNS obfuscation by allowing users to access essential information while preserving their privacy and identity.

5.2 Swapping-based algorithms

As variations of our two previously proposed algorithms, we have developed two distinct adaptations for each. In this approach, rather than implementing domain name blocking, we opt for domain name "swapping." In contrast to our blocking algorithms that retrieve the blocked domain from the local cache, these algorithms execute a "swap" operation. Similarly, domains that are not forwarded are resolved through the local cache, while a domain added or substituted within the local cache is directed to the recursive resolver.

5.2.1 P-Mix-All

In addition to designing the P-Block algorithm, we have also developed an innovative approach called P-Mix. While P-Block selectively blocks domain names based on a specified probability threshold, P-Mix takes a different approach. It introduces a "virtual table" concept, which serves as the basis for domain swapping, setting it apart from P-Block. This algorithm introduced the TF-IDF metric results, where a table of ranks of domain names per user is made. Thus, we can create this virtual table or ranks it according to its importance by each user.

More precisely, rather than blocking domains, P-Mix-All dynamically swaps the requested domain names with alternate ones from the table's remaining $1 - p$ data. By replacing blocked domains with suitable alternatives.

The P-Mix-All algorithm is designed to manage the selected domain names based on a predefined percentage, denoted as p. To facilitate this, the algorithm utilizes the same random variable r as P-Block, representing the probability of encountering a domain name to be blocked but swapped in this case. When a user initiates a request for a specific domain name d from their OS, the P-Mix-All algorithm enters its decision-making process. If the random variable r is found to be less than p ($r < p$), the algorithm triggers a query replacement mechanism. In this case, it randomly selects another domain name $d' \neq d$ from all available domain names except the original domain name d. The user's initial request for domain name d is then seamlessly replaced by the query for domain name d'. This allows for the dynamic substitution of specific domain names to enhance user privacy or control content access.

Conversely, the P-Mix-All algorithm follows a different path when the random variable r exceeds or equals the predefined percentage p ($r \geq p$). In such scenarios, the algorithm allows the user's query for domain name d to be sent as usual without any substitution or blocking. This means that when the likelihood of encountering a blocked domain name is less than than threshold p, the P-Mix-All algorithm introduces unpredictability in query responses, effectively obscuring the identity of requested domain names. However, when r has a greater likelihood of accessing a non-blocked domain, the algorithm allows the user's request to be resolved locally, ensuring seamless and unaltered access to the desired content.

By incorporating this probabilistic framework, the P-Mix-All algorithm effectively determines which domain names are blocked and which are allowed to proceed unhindered. Utilizing a random variable r in the decision-making process introduces an element of randomness, enhancing the obfuscation of the DNS fingerprint. Consequently, the P-Mix-All algorithm bolsters the overall security and privacy of the system by diversifying DNS query patterns and making it harder for potential adversaries to discern browsing behavior. Moreover, the parameter p can be adjusted to cater to specific requirements or desired levels of obfuscation, providing users with a flexible and customizable approach to DNS resolution.

5.2.2 P-Mix-Bottom

As an additional algorithm and keeping the stochastic-based approach, P-Mix-Bottom has been implemented, incorporating a similar underlying principle as the p-value. In contrast to the previously proposed algorithms, P-Mix-Bottom rather than selecting replacement values randomly from the entire test data subset $1-p$ as P-Mix-All. It generates a specific virtual table, as discussed before, enabling targeted domain exchanges. This algorithm is still implementing the generation of the random variable r as described in equation 5.1 and a similar technique of the virtual table of P-Mix-All.

With the result of the sum of TF-IDF, this metric assigns a scalar importance value to each unique domain for every user. Specifically, the lower segment of the 50th percentile is chosen, effectively creating a virtual division within the table for each user. Consequently, domain replacements are performed with domains with lesser importance for the user, as determined by the corresponding importance values. Thus, based on a continuous probability distribution, we generally swap the requested domain to the less important ones. If the "toss coin" falls below our parameter p, the query is swapped; otherwise, the domain goes through an ordinary DNS resolution. Therefore, with this approach we modify the data distribution (i.e., Zipf) that the attacker is receiving. Consecuently, we can manipulate with this stochastic process introducing bias to the threat models.

5.2.3 Top-Mix-All

As the next proposed strategy, we introduce an approach that diverges from the blocking mechanism utilized in Top-Block. In this new method, Top-Mix-All, we maintain the foundation of our earlier ranking algorithms, as discussed previously. However, instead of outright blocking a requested domain, we replace it with another domain from our importance set.

Similarly to the P-Mix-All algorithm, the fundamental principle of Top-Mix-All involves the utilization of the complement of p' in terms of our ranking tables by each user, specifically $1-p'$, to dictate the obfuscation. If the requested domain relies on $R^i \leq \frac{p'}{100} \times N_i$, we randomly select a domain from $1-p'$. Similarly to the P-Mix-All approach, a requested domain d would be swapped for a domain d', but instead of using the random variable r, it natively will choose a random domain of the remaining set. Thus, the domain will be swapped rather than blocked using Equation (5.4). By incorporating this approach, we introduce flexibility to our decision-making process. Rather than strictly blocking a domain, we can replace the original domain with an alternative, enhancing the diversity and randomness in the selection process.

The underlying mechanism remains anchored in using the domain ranks we meticulously generated in the past. This allows us to prioritize the most relevant and influential domains when choosing candidates for substitution.

5.2.4 Top-Mix-Bottom

Our last effort to minimize the accuracy of the machine learning classifiers relies on the Top-Mix-Bottom algorithm. Like its counterpart, P-Mix-Bottom, this approach leverages the lower segment of the 50th percentile of ranks per user. Furthermore, we utilize the identical Equation as in Top-Block (see Equation (5.4)). However, in contrast to blocking the domain, we implement the domain swap strategy.

We remain focused on minimizing the classifier's efficacy through strategic manipulations in the Top-Mix-Bottom algorithm. We target domains that exhibit specific patterns by harnessing the lower segment of the 50th percentile ranks assigned to each user. This means that instead of preventing access to specific domains, we theoretically exchange them with other less important domains according to our ranks, thus obscuring the original domain's identity and confounding the classifier's predictions.

By incorporating the Top-Mix-Bottom algorithm into our investigation, we extend our set of techniques to diminish the accuracy of the machine learning classifiers.

Chapter 6

Simulations & Results

This Chapter describes the simulations and results of our proposed strategies. The implementation phase involved bash scripting, Python 3.10, and Scikit-learn's built-in libraries [33]. These libraries were instrumental in the machine learning stage, as discussed in Chapter 4. Furthermore, we ensured that our obfuscation algorithms were implemented and tested on an Ubuntu 22.04.2 LTS (Jammy Jellyfish) system with an Intel(R) i7-7700 CPU @ 3.60GHz and 64GB of RAM. Even though our evaluation was in a unix-based system, our implementation can be migrated to any platform.

6.1 DNS traffic dataset definitions & statistical analysis

The theoretical implementation centers on analyzing and preprocessing a DNS traffic dataset from the UNAM Telecommunications Department. This extensive dataset originally covered DNS traffic spanning 108 days. However, for this thesis, we focused on 86 days after preprocessing, which involved removing the weekends and two weeks of public holidays. Below is a detailed description of the dataset and the statistical outcomes of the preprocessing and data analysis.

Within our research framework, two threat models sourced from the NQA strategy [3] serve as our baseline for comparison. In addition, we have incorporated a more robust attacker model, known as the Random Forest, as detailed in section 4.3. Including the Random Forest model significantly enhances the evaluation of our approach. Moreover, it accentuates our proposed strategies' impact on the accuracy of an ensemble such as a random forest.

6.1.1 Attributes of the dataset

During the data preprocessing phase, an analysis revealed that only 8 IP addresses demonstrated behavior resembling that of actual users. This observation suggests the existence of 8 genuine users within the dataset. In contrast, the rest of the users in the trace displayed sporadic behavior. As a result, we have decided to exclude this sporadic behavior data from further analysis. These eight users constitute an essential part of our analysis, contributing to evaluating our proposed strategies. The dataset comprises 14 parameters, while 4 played a significant role in the subsequent analyses. Below, we define each of these parameters.

1. **pkl_len:** It defines the packet length sent or received. The packet size attribute is an essential characteristic used to determine the amount of data in a single packet.

2. **s_ip:** The source IP address field is an important component of the query used to store the request's origin. It holds the IP address of the device or the computer that generated the query; it is essential in determining the origin of the request and can be used to trace the source of the query.

3. **s_port:** It represents the port number originating from the DNS traffic.

4. **dns_query_flag:** This field indicates whether the DNS traffic is a query (value = 0) or a response (value = 1) by setting the appropriate flag bit.

5. **dns_query_type:** This field specifies the type of DNS query being sent, such as A (address record), AAAA (IPv6 address record), MX (mail exchange record), etc.

6. **dns_ans:**This field contains the answer to the DNS query, if applicable, such as the IP address or mail server address.

7. **dns_query:** This field contains the actual DNS query being sent, such as the domain name being looked up.

8. **d_port:** Field that represents the port number to which the DNS traffic is being sent.

9. **dns_id:** This field is a unique identifier generated by the client for each DNS query, which matches queries with their corresponding responses.

10. **dns_rcode_flag:** This field indicates whether the DNS response was successful (0) or encountered an error (non-zero), such as NXDOMAIN (no such domain) or SERVFAIL (server failure).

11. **d_ip:** The destination IP represents the IP address of the DNS server to which the DNS traffic is being sent.

12. **dns_query_class:**This field specifies the class of DNS query being sent, such as IN (Internet) or CH (Chaosnet).

13. **timestamp:** This field represents the date and time when the DNS traffic was captured or observed, which can help analyze patterns or trends.

14. **dns_opcode_flag:** This field specifies the type of DNS operation being performed, such as QUERY (0), IQUERY (1), or STATUS (2), which can provide additional context for the DNS traffic.

After conducting a thorough analysis of the TF-IDF metric in our dataset, we enhanced the data's accuracy by modifying the relative frequency of the domain names of each user using six different methods. In the subsequent section, we present a detailed overview of our contributions and the benefits of each of the six previously discussed implementations. In order to establish a solid baseline, we constructed a benchmark by employing the training data without making any alterations to the traffic to set a level of accuracy as a starting point to minimize.

Accuracy is a metric that evaluates the overall performance of a classification model. It represents the fraction of predictions the model got correct from the total number of predictions. In other words, it measures how often the model makes correct predictions. Mathematically, accuracy is defined as:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total of number predictions}} \tag{6.1}$$

For binary classification, accuracy can be calculated in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.2}$$

In the case of multi-class classification, accuracy measures the number of times any class was predicted correctly, normalized by the total number of data points. It represents the overall predictive performance of the model across all classes.

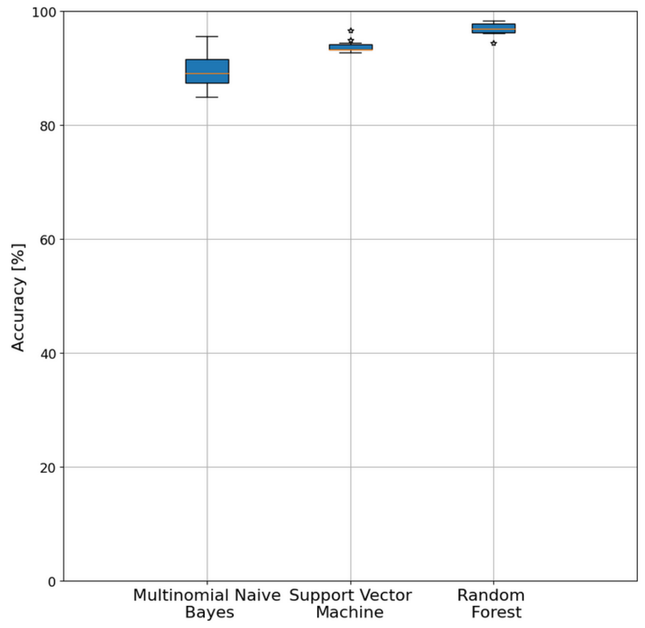


Figure 6.1: Proposed benchmark

To ensure our results were reliable, we executed each attacker model ten times using different seeds, as demonstrated in each boxplot in Figure 6.1. This enabled us to effectively compare the efficacy of our techniques against the baseline. It should be highlighted that we only kept the median value of each attacker model as a final value as our benchmark.

6.1.2 Statistical analysis and threat models evaluation

The analysis and theoretical implementation of the algorithms proposed in section 5 encompassed eight distinct users and their corresponding DNS primary server. The DNS traffic of each IP address was analyzed, and the primary DNS server was determined based on the number of responses received from the requests made by each IP. It was observed from the dataset that each s_ip might have more than one DNS server. To this end, we considered the primary DNS server the one that received more requests and provided more responses to each s_ip. The association of each user's IP address with their primary DNS server is presented in Table 6.1.

IP address (s_ip)	**Primary DNS server** (d_ip)
192.168.27.153	132.248.10.2
192.168.27.16	132.248.10.2
192.168.27.160	8.8.8.8
192.168.27.161	8.8.8.8
193.168.27.162	8.8.8.8
192.168.27.17	132.248.10.2
192.168.27.180	132.248.10.2
192.168.27.215	132.248.10.2

Table 6.1: Target users and primary DNS servers

Our primary focus was processing DNS queries; therefore, we primarily worked with the parameters dns_query_type, d_ip, s_ip, and dns_query_flag. We particularly emphasized responses ($dns\_query\_flag = 1$) that corresponded to the user's requests, as these responses contained valuable domain name information. To ensure precision in our analysis, we exclusively filtered out other query types, narrowing our attention solely to query type A ($dns\_query\_type = A$, which specifically refers to the Internet Protocol version 4 (Ipv4) address records.

When evaluating the accuracy of each attacker model proposed in section 4, we adopted a consistent proportion ratio of 70% data for training and 30% for testing, as shown in our benchmark 6.1. However, we tested with different ratios as depicted in Figure 6.2. This standardized ratio (70-30) has been extensively used in related literature and accepted within the machine learning community. Maintaining this well-established training-to-testing data ratio throughout the experiments ensured fair and reliable assessments of the attacker models' performance. We must note that we only modified the testing data in our experiments described in section 5. Thus, when training each model, it uses real, non-modified data. By doing so, when using the testing and modified data, we can ensure the performance and effectiveness of our algorithms.

We present a visualization depicting the frequency of accurate predictions made by the models, associating the predicted class (IP address/user) with its corresponding actual

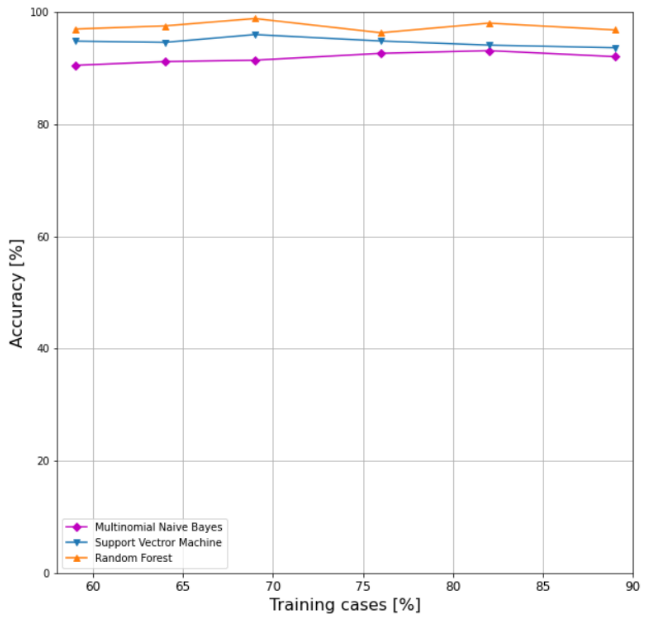


Figure 6.2: accuracy for different data ratios

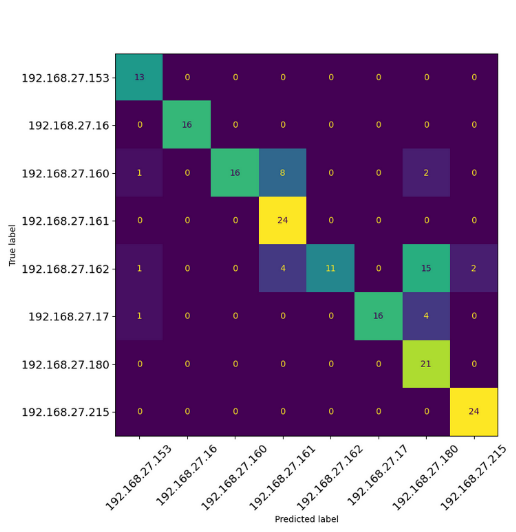
class by using confusion matrices. On the y-axis of the three Figures 6.3a, 6.3b, and 6.3c, we find the *True label* or the actual class, while on the x-axis, we observe the predicted class.

These confusion matrices provide a concise and insightful representation of the classifiers' performance, enabling us to assess their ability to predict and classify IP addresses (labels) based on domain names (features). Consequently, we obtained the rank of domains for each user by TF-IDF in our dataset, which we used to perform a detailed analysis of the running total. This analysis allowed us to estimate each domain's importance percentage and the total number of domains accessed by each user. This information proved invaluable in gaining insights into the browsing behavior of individual users and understanding their preferences when it comes to accessing different domains.

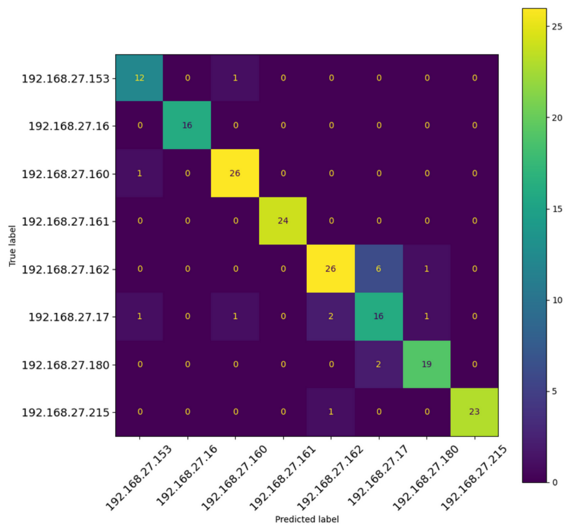
Figure 6.4 reveals a crucial finding regarding users' web browsing habits in the DNS dataset. Our analysis shows that for most users, a relatively small number of domain names (around 2000) account for almost 80% of the total importance of the domains visited. This implies that users frequent a select few websites more frequently than others. To clarify this, let tf_idf_table_i be a DataFrame for user i with a column named tf-idf_sum containing n_i floating-point values in descending order which is the TF-IDF sum calculation:

$$\text{tf\_cdf}_i = \frac{\sum_{j=1}^{k} \text{tf-idf\_sum}_i(j)}{\sum_{j=1}^{n_i} \text{tf-idf\_sum}_i(j)}$$

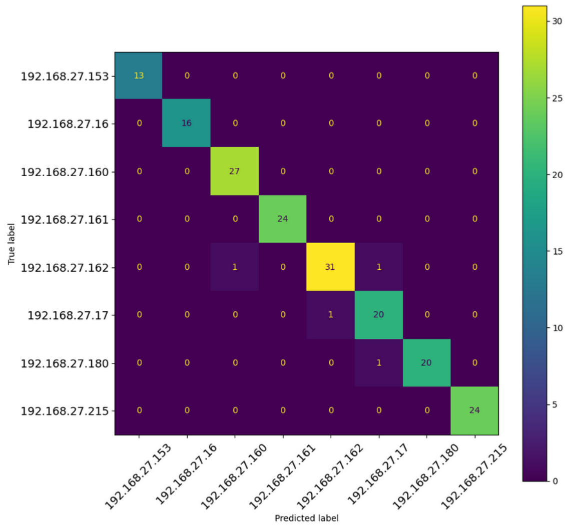
Where $\text{tf-idf\_sum}_i(j)$ represents the j-th value in the 'tf-idf_sum' column of user i in tf_idf_table_i and k is the current row being considered for the cumulative sum. The expression calculates the cumulative distribution function of the 'tf-idf_sum' column for user i by summing up all the values from the first row up to the k-th row, divided by the



(a) Multinomial Naive Bayes



(b) Support Vector Machine

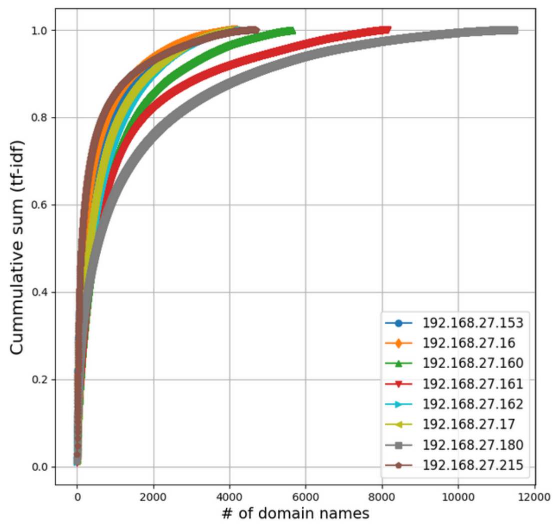


(c) Random Forest

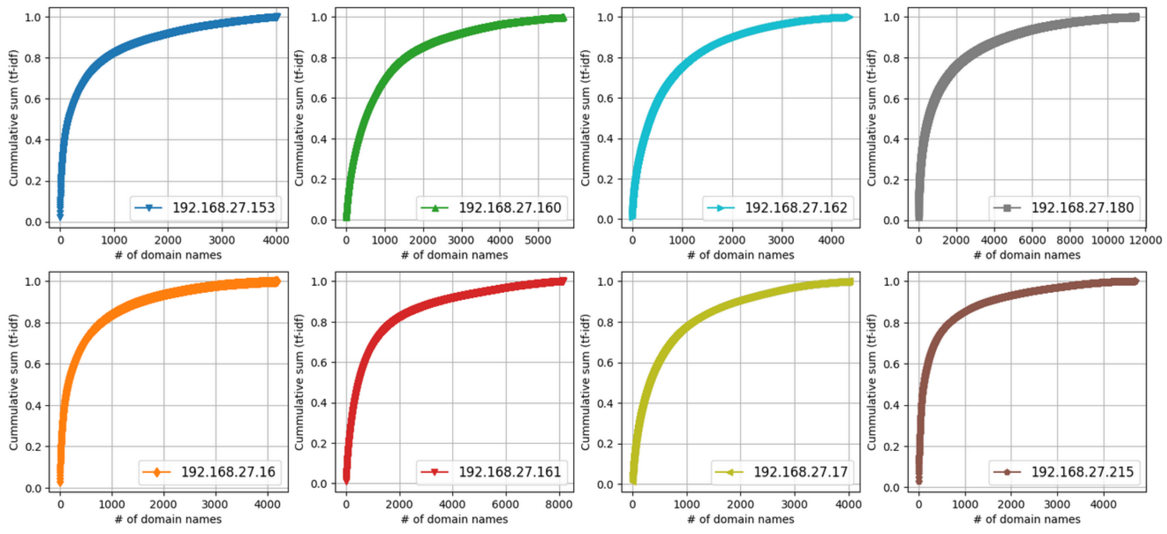
Figure 6.3: Confusion matrix of threat models in 70-30 ratio

sum of all the values in the entire 'tf-idf_sum' column for user i.

In the context of our research, as elaborated in Chapter 5, our primary focus revolved around the deliberate introduction of bias into the classifiers. Subsequently, we present the outcomes of our practical implementations of the theoretical Blocking-based and Swapping-based algorithms in the subsequent subsections, using genuine DNS data for our evaluations. By analyzing these results, we aim to gain insights into the efficacy of bias induction techniques and their impact on the classifiers' predictions in real-world DNS scenarios.



(a) 8 users



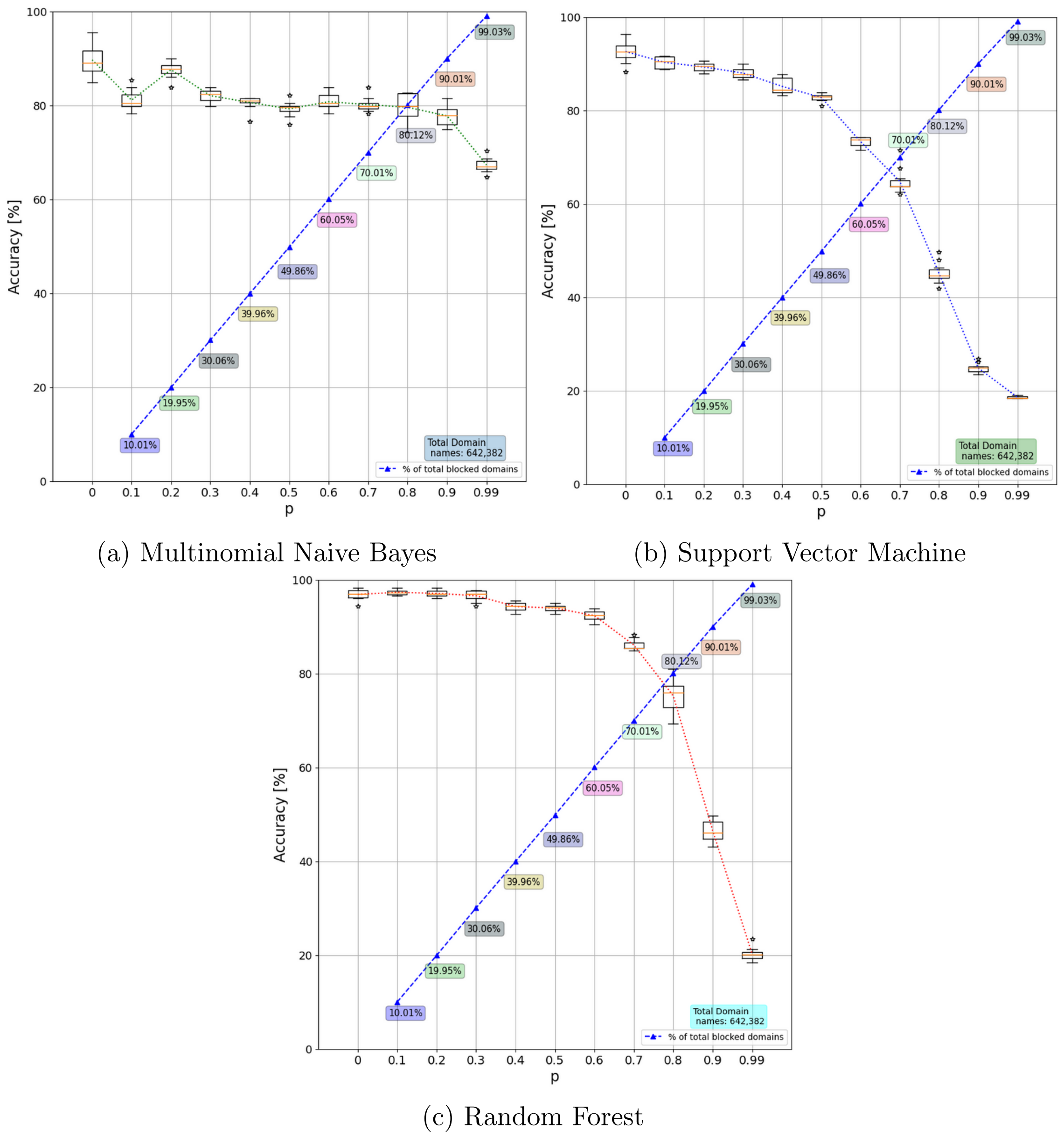
(b) Individual users

Figure 6.4: Running total

6.2 Blocking-based algorithms results

In this section, we present the results of our investigation into the Blocking-Algorithms, a set of techniques designed to address the challenge of DNS fingerprint obfuscation and mitigate the risks posed by machine learning classifiers. The Blocking-Algorithms encompass a series of the two approaches mentioned before, each contributing to the overarching goal of enhancing user privacy and minimization of the accuracy of our threat models. In addition, we highlighted the percentage of data manipulated in our dataset for the proposed results in every plot. This provides insight into the trade-off of the data that was either swapped or blocked, impacting the accuracy. More precisely, we can observe the linear behavior of the data blocking/swapping in these P-based algorithms. This is due to the proposed probability distribution when setting our control parameter p. It is also important to note that the accuracy of each classifier relies on the average accuracy of each of the eight users from the dataset.

6.2.1 P-Block



(a) Multinomial Naive Bayes (b) Support Vector Machine

(c) Random Forest

Figure 6.5: Threat models performance implementing P-Block

Among our stochastic-based algorithms, we focus on the P-Block method, which seeks to proactively block random domain names based on our *toss coin* to minimize the likelihood of identification and tracking.

The accuracy results for each attacker model for the whole range $(0.1 - 0.99)$ of our probability parameter of blocking p in the x-axis are best shown in Figure 6.5 as blocking methodology.

To obtain sufficient data, we conducted ten tests using three different threat models per each p value. Each test involved evaluating the impact of a specific p value on the domain names, where p represents the percentage of the domain names that are partially blocked.

The plots provide a comprehensive overview of the behavior exhibited by our attacker models while applying our theoretical algorithm, P-Block. In the plots, the initial box positioned at 0 on the x-axis denotes the benchmark scenario where no modifications are made to the data. Subsequently, the whisker boxes showcase the evaluations corresponding to the implementation of this algorithm. Compared to the other two models, a distinct contrast emerges in the accuracy minimization for Multinomial Naive Bayes, as illustrated in Figure 6.5a. Specifically, when our parameter p is set to 0.6 and 0.7, the incremental drop in accuracy becomes even more pronounced in the support vector machine and random forest.

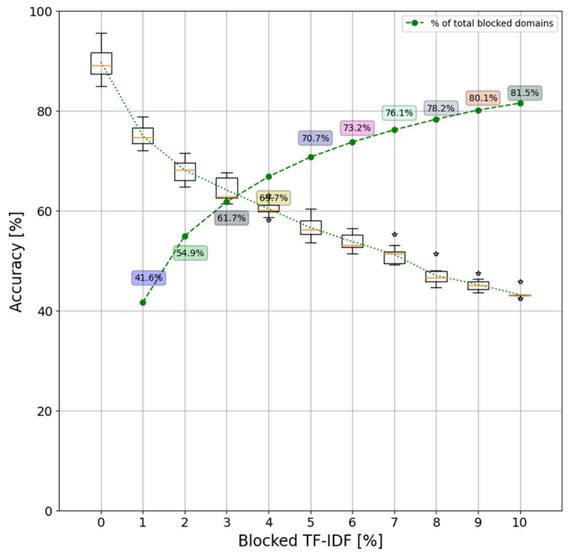
Interestingly, the accuracy minimization plot of the multinomial naive Bayes, as shown in the figure, exhibits a distinctive pattern compared to the other two models. Specifically, as we manipulate the parameter p to 0.6 and 0.7, the classifiers' accuracy experiences a less pronounced decrease compared to support vector machines and random forests. This observation can be attributed to multinomial naive Bayes's inherent assumption of independence between each domain. Despite this behavior, we retained this classifier in our further experiments to emphasize the contrasting behavior of each threat model. Doing so highlights each approach's unique characteristics and implications, providing valuable insights into their performance and effectiveness.

6.2.2 Top-Block

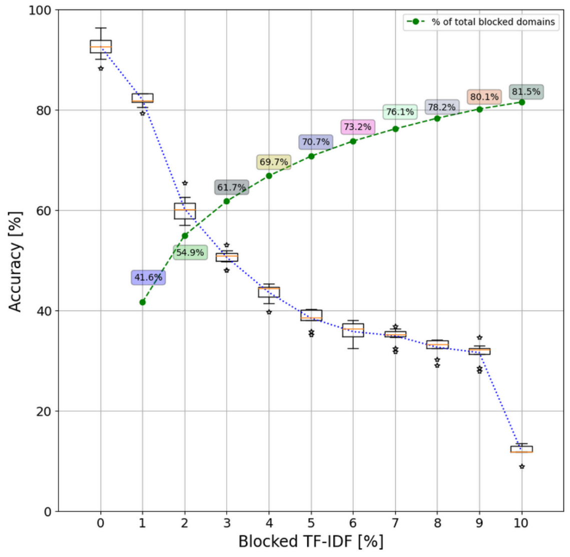
Our first attempt to modify the relative frequencies after P-Block algorithm centers with the Top-Block approach. With this implementation, our primary objective is to select a **specific range of top domains** $(1 - 10\% \equiv 0.01 - 0.1)$ for blocking, enabling us to introduce controlled disruptions to the data. In a more detailed perspective, we strategically block a certain percentage of domains within our dataset based on their perceived importance in the context of our threat models.

The outcomes of this implementation are shown in Figure 6.6, offering insights into the dynamic behavior of our threat models within the Top-Block approach. One aspect that captures our attention is the percentage of data due to its distinct curvature. This phenomenon can be largely attributed to the nature of DNS traffic and our relevance metric implied, which exhibits an exponential-like distribution. As we mentioned before, certain domains receive higher requests than others.

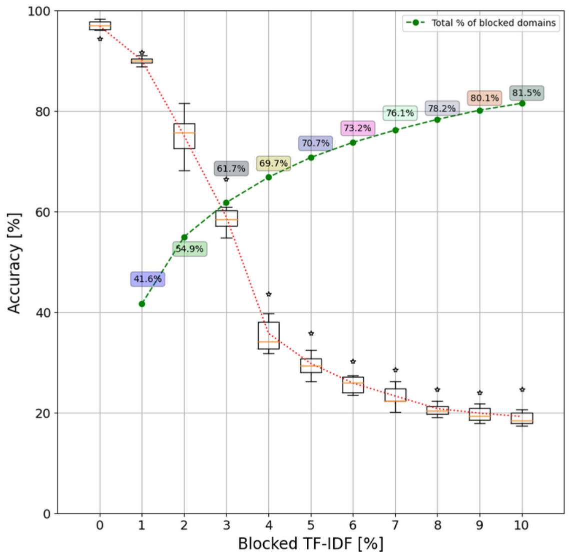
By purposefully blocking these domains, we initiate a process that yields interesting results, particularly within the lower 4% of the domains selected using TF-IDF. Remarkably, when employing Random Forest (as depicted in Figure 6.6c), we observe a substantial reduction in accuracy of approximately 60%. This finding underscores the efficacy of our Top-Block strategy in introducing bias in ensembles such as Random Forest classifiers.



(a) Multinomial Naive Bayes



(b) Support Vector Machine



(c) Random Forest

Figure 6.6: Threat models performance implementing Top-Block

Similarly, SVM 6.6b, tends to have the same behavior of RF but with a slower rate. On the other hand, MNB (6.6a) has linear-like minimization within our blocking ranges. Reaching from 91% to 57% when blocking 10% of the most important data for each user.

It is important to note that even with just a 10% blocking rate in the Top Block, the accuracy is significantly reduced. Herefore, exceeding this percentage yields only meaningful improvements. However, in terms of total traffic, a 10% block rate corresponds to an effective block rate of 81.5%, which is equivalent to setting a block rate of $p' = 0.8$ in our P-Block algorithm. Nevertheless, the most notable distinction lies in the observed accuracy decrease.

6.3 Swapping-based algorithms results

Continuing with our commitment to enhancing privacy and introducing bias into our threat models, we proceeded to deploy the second batch of functions designed to manipulate the relative frequencies of our dataset. This strategic approach minimizes accuracy by selectively blocking or swapping specific domains. Through the execution of these functions, we aimed to strike a balance between safeguarding user privacy and effectively mitigating the risks posed by DNS fingerprinting attacks. By introducing deliberate variations in domain frequencies, we aimed to disrupt the conventional patterns that machine learning classifiers typically rely on, thus rendering our threat models more resilient and less prone to accurate identification.

For visualization purposes within the data, the x-axis of each plot described below represents the top [%] of domain names either blocked or swapped for each user. A color-dotted line represents the percentage either blocked or swapped among our range in the whole test data.

6.3.1 P-Mix-All

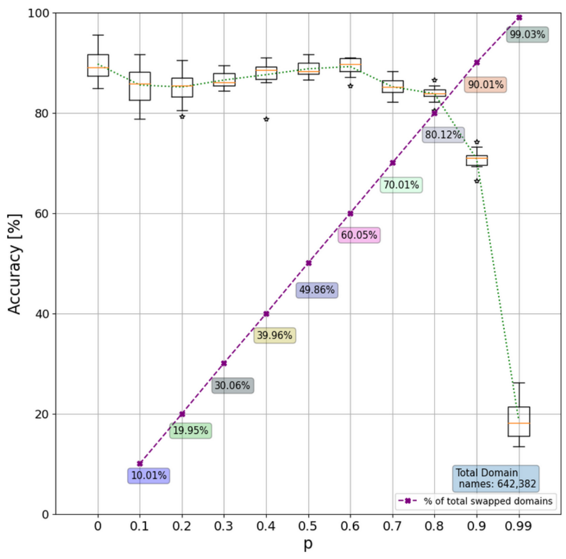
Compared to the P-Block algorithm, the results of our proposed strategy P-Mix-All, show a distinct behavior concerning accuracy across various percentages of swapped domains.

Notably, a pattern emerges when we attempt to swap domains based on the probability calculation for each requested domain.

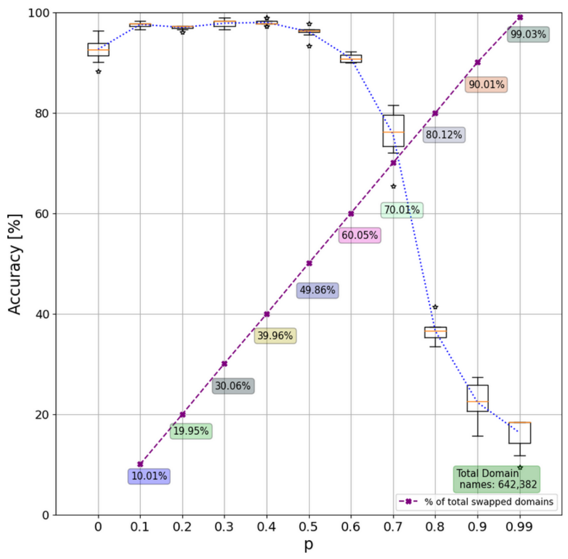
A notable increase in accuracy is observed in the initial half of the p range values for SVM, and while in RF, it almost keeps the same as our benchmark. Specifically, in the case of Multinomial Naive Bayes (MNB), a slight decrease in accuracy is noticeable within the first three values, followed by a subsequent slight increase, almost reaching the mean accuracy of our benchmark at $p = 0.6$. However, after this point, the accuracy begins to be minimized considerably. In the case of SVM (6.7b), this technique favors the first half of probability values of p, then decreases afterward.

This intriguing behavior exemplifies the efficacy of our P-Mix-All strategy, indicating that swapping domains based on probability calculations yields advantageous outcomes, particularly in the upper range of p values where the effect on the accuracy is notable. In this scenario, both SVM (6.7b) and RF (6.7c) exhibit comparable behavior, yet SVM demonstrates quicker minimization after reaching our parameter threshold of $p = 0.6$ when compared to RF. However, it is important to acknowledge that this is a theoretical approach, and its real-world performance may vary depending on various factors and datasets.

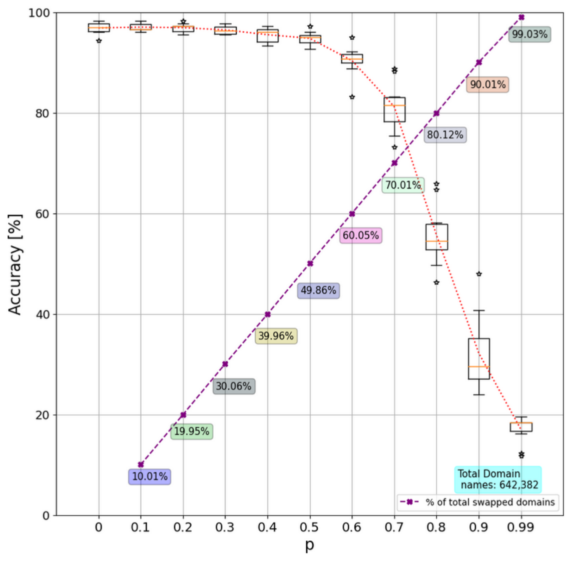
While the results are promising and highlight the potential benefits of the proposed algorithm, it is essential to consider the limitations and challenges that may arise when implementing it in practical scenarios. The algorithm's effectiveness could be influenced by factors such as the specific dataset used, the data's nature, and the underlying distributions' complexity.



(a) Multinomial Naive Bayes



(b) Support Vector Machine



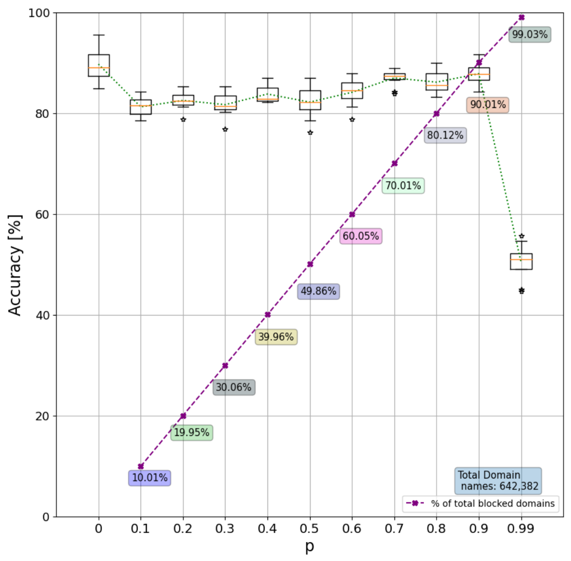
(c) Random Forest

Figure 6.7: Threat models performance implementing P-Mix-All

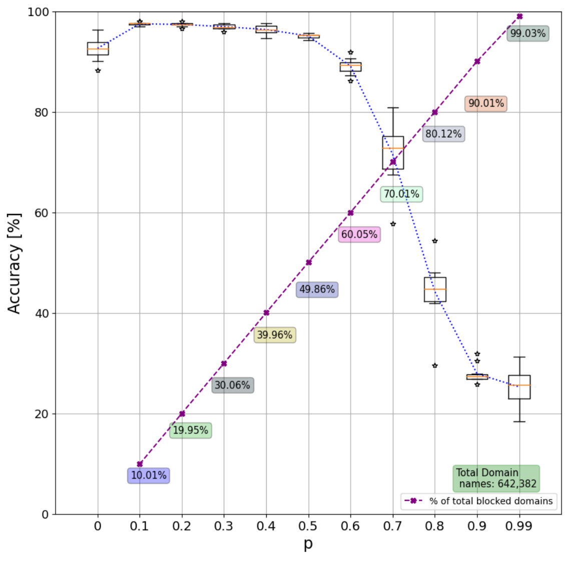
6.3.2 P-Mix-Bottom

Lastly, when considering our algorithms based on continuous probability distributions, our approach involves swapping domains with those of lesser significance, as depicted in Figure 6.8. In this context, we can observe a similar pattern in both SVM (6.7b) and RF (6.7c), as seen in our previous strategy, P-Mix-All.

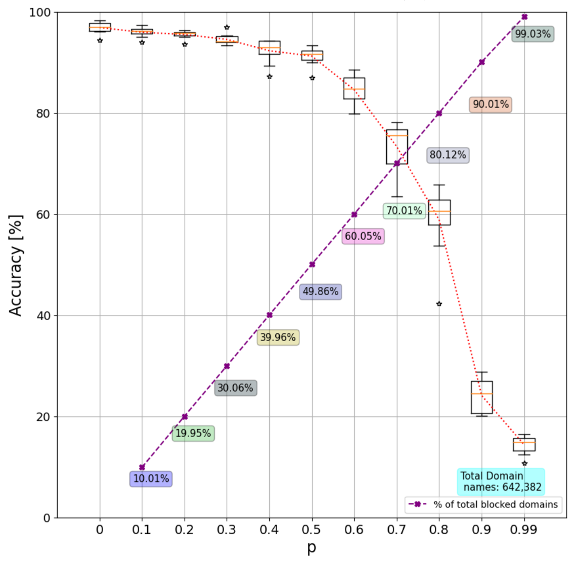
Notably, there is a noticeable increase in accuracy after the initial p value of 0.1. However, as we move towards the upper range of p, we observe a minimization of accuracy due to domain swapping. More specifically, the lower values of p exhibit a comparable trend in SVM and RF.



(a) Multinomial Naive Bayes



(b) Support Vector Machine



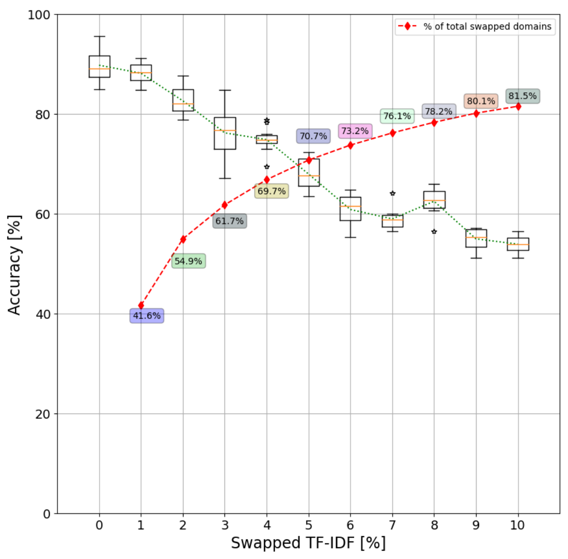
(c) Random Forest

Figure 6.8: Threat models performance implementing P-Mix-Bottom

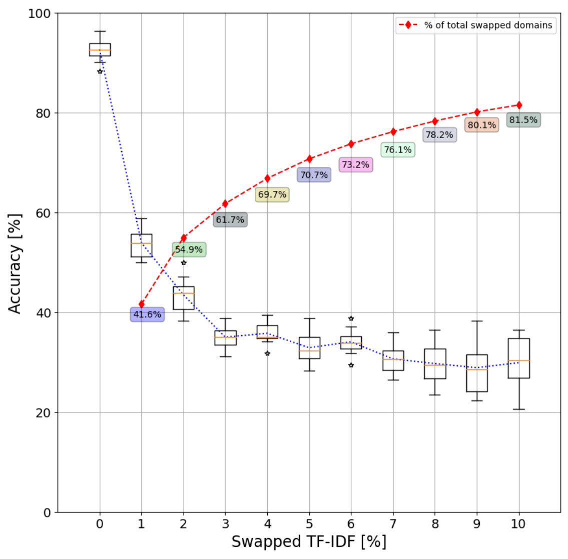
The results show different behavior in the case of MNB (6.7a). Initially, there is an approximately 10% decrease in accuracy when swapping domains that reaches 80%. However, based on the experimental data, we observe a promising trend where the accuracy increases as p approaches 0.9. However, it is essential to note that a significant drop in accuracy is observed beyond this point.

These observations generally highlight the nuanced behavior of our algorithms when utilizing continuous probability distributions for domain swapping. While SVM and RF demonstrate favorable accuracy improvements after an initial threshold, MNB presents a more challenging accuracy degradation, albeit with a potential for improvement in certain regions. This underscores the importance of considering the implications and trade-offs when selecting specific algorithms for domain-swapping tasks in different contexts and datasets.

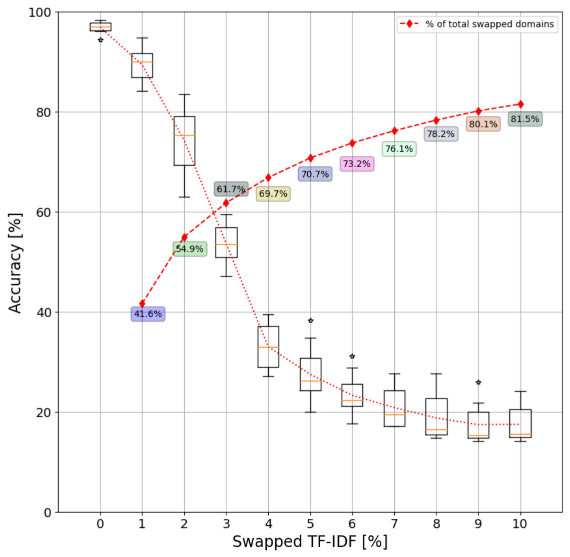
6.3.3 Top-Mix-All



(a) Multinomial Naive Bayes



(b) Support Vector Machine



(c) Random Forest

Figure 6.9: Threat models performance implementing Top-Mix-All

When we choose domain swapping as an alternative to blocking the relatively more "critical" domain names within the data, we encounter an intriguing scenario that serves as an excellent test case for assessing SVM's performance. Domain swapping involves interchanging domain labels or features, offering a unique opportunity to evaluate the model's robustness.

When we apply the swapping operation to a mere 1% of data from each user, a substantial drop in accuracy, approximately 39%, becomes apparent, notably impacting SVM's predictive ability. This significant decrease in accuracy emphasizes the sensitivity of SVM to domain swapping, making it a vital evaluation metric.

Conversely, the impact of domain swapping on RF becomes more pronounced within a specific range, particularly between 2 to 4%. It is within this range that RF exhibits noticeable variations in its predictive performance due to the domain-swapping technique. This observation highlights the differential responses of SVM and RF to this specific data manipulation, further illustrating the distinct behavior of both models.

6.3.4 Top-Mix-Bottom

Finally, we have made an interesting observation while implementing the swapping technique with the lower percentile of domains, explicitly targeting the less important ones, following the same generic idea as P-Mix-Bottom. The outcomes of this approach are illustrated in Figure 6.10, exhibiting behavior similar to our previous strategy, Top-Mix-All.

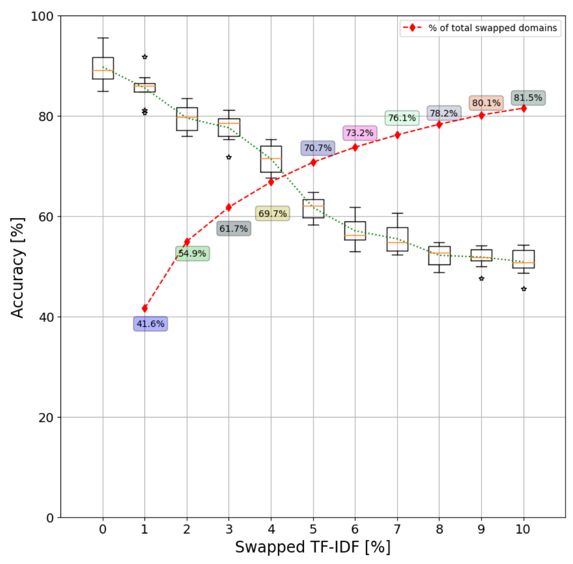
However, there is a slight difference in the drop of accuracy in SVM (as shown in Figure 6.10b) when compared to the swapping of only 1% in the Top-Mix-All strategy. We note a 6% difference in accuracy between the Top-Mix-Bottom approach and the swapping of 1% in the Top-Mix-All strategy. This difference emphasizes the impact of domain selection on SVM's performance. On the other hand, MNB 6.10a showed slightly better overall performance with the swapped domains when compared to its previous implementations. This observation indicates that the MNB algorithm is less sensitive to domain swapping and can maintain relatively better accuracy even with the less important domains being swapped.

6.4 Blocking-based algorithms vs. Swapping-based algorithms

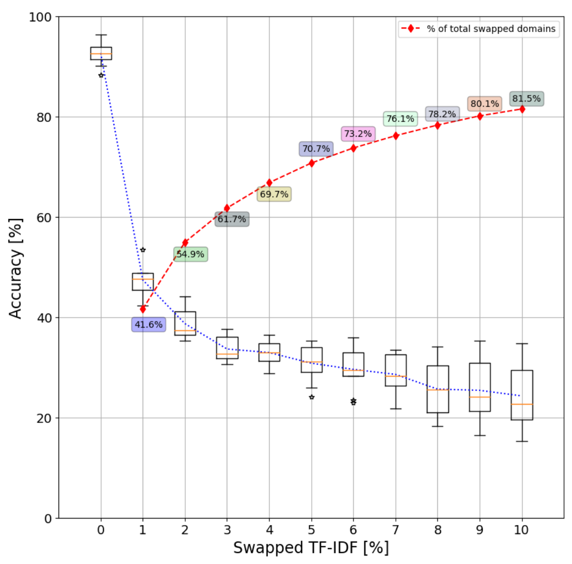
Figure 6.11 outlines the overlap of all our proposed algorithms. In the graph, multinomial naive Bayes is represented by the green lines, support vector machines (SVM) by the blue lines, and random forest by the red lines.

Using color-coded lines facilitates the easy identification and differentiation of each algorithm's performance trends. It should be noted that only the mean values of the accuracy have been plotted. By focusing on the mean accuracy values, we can comprehensively assess the overall performance of each algorithm. As depicted in both Figures, we can see the huge difference of data-trade-off between our two sets of implementations. In contrast, our algorithms based on a p value extend from 10 to 99% of the total data, while our algorithms based on the top TF-IDF block or swap only 1 to 10% of the unique domains per user. Nevertheless, in terms of total traffic, selecting 1% of the TF-IDF data from each user represents an overall 41.6% of data. It is worth mentioning that these results extend only to test cases of 8 users.

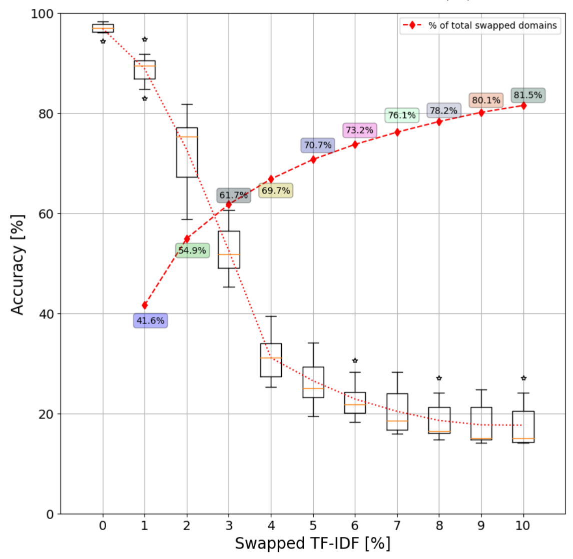
The performance summary of our Top-Block technique, specifically in terms of accuracy minimization, is documented in Table 6.2. This table provides a comprehensive and



(a) Multinomial Naive Bayes



(b) Support Vector Machine

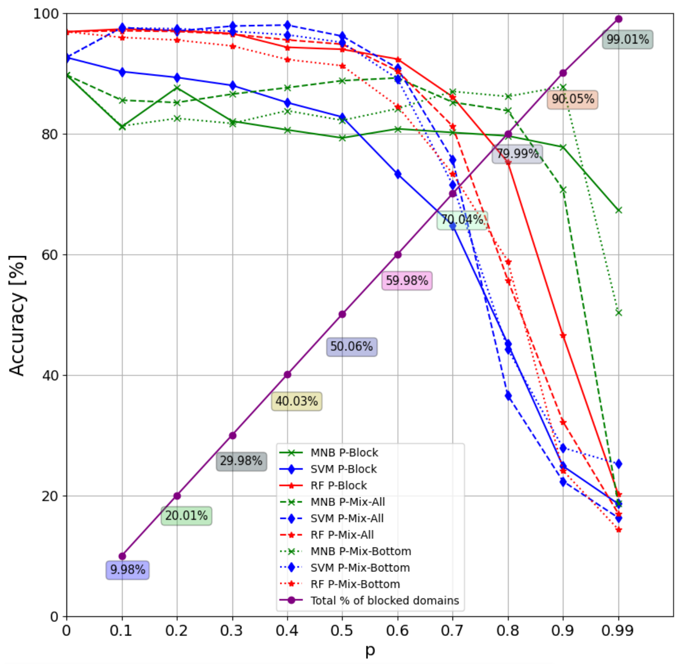


(c) Random Forest

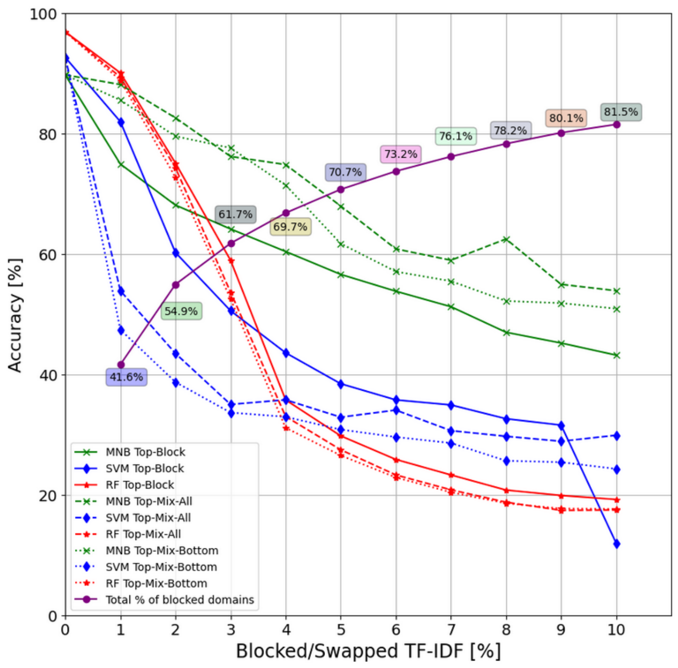
Figure 6.10: Threat models performance implementing Top-Mix-Bottom

Attacker	Accuracy [%] (avg)				
	Normal DNS traffic (benchmark)	Blocking Top 1% TF-IDF	Blocking Top 2% TF-IDF	Blocking Top 3% TF-IDF	Blocking Top 4% TF-IDF
Multinomial Naive Bayes	91.2	74.2	68.1	64.1	60.3
Support Vector Machines	95.2	81.9	60.22	50.5	43.4
Random Forest	98.8	90	75	58.9	35.75

Table 6.2: Summary of Top-Block performance



(a) Algorithms based on P-value



(b) Algorithmss based on Top TF-IDF

Figure 6.11: P & Top-Algorithms Overlap

detailed overview of the crucial trade-off-worthy results obtained from our in-depth data analysis, encompassing domain blocking ranging from the Top 1% to the Top 4%.

In the subsequent Chapter, we describe in detail the real implementation of this technique, providing a detailed account of its impact on end-user experience, mainly focusing on the implementation of the Top 1% blocking.

Chapter 7

Impact of proposed algorithms

7.1 Real-life experiment

In contrast to our previous approaches that relied solely on theoretical scenarios within the UNAM Telecommunications Department dataset, this chapter presents a real-life scenario along with the practical implementation of our algorithms to perform a qualitative analysis of user experience.

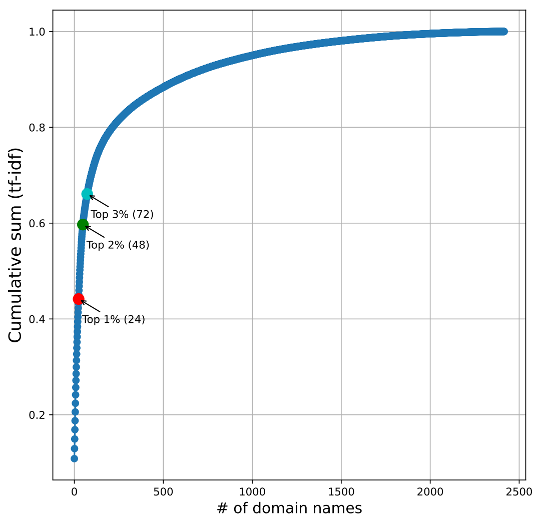
We collected actual DNS traffic from a mutually agreed-upon party over approximately three weeks to achieve this goal. This dataset provides us with authentic and fresh information to thoroughly process and evaluate the effectiveness of our techniques. By analyzing user behavior and observing the resulting impact using this data, we gain valuable insights into how our algorithms function in practical, real-world situations. Understanding how our solutions perform in actual, real-time scenarios is crucial for ensuring their effectiveness and applicability in practical settings.

The data gathering process involved sniffing the traffic on port 53 using *tcpdump*, allowing us to capture all the DNS activities during this time frame. Specifically, the data was collected from April 20, 2023, to May 12, 2023, and originated from the new IP address. Subsequently, we analyzed the DNS responses and created a cumulative sum of the most significant domains, employing TF-IDF. Figure 7.1a visually represents the total number of domains along with their respective Top 1%, 2%, and 3% occurrences. In addition, the three sniffed weeks of data gave us enough insights and/or patterns of user behavior; figure 7.1b shows the empirical data vs. the theoretical Zipf distribution of domain names.

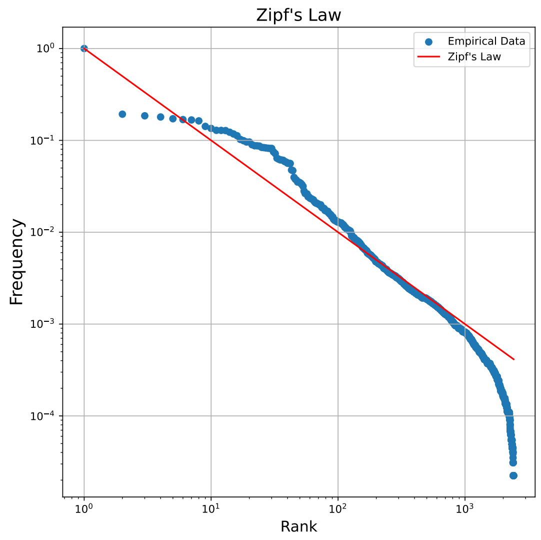
During the implementation, we focused on the top 24 domains, which constitute the Top 1% of the data. This selection process allows us to concentrate on the most relevant domains within the new dataset.

Testing phase

Initially, we performed a 12-hour evaluation of our Top-Block implementation. Additionally, we employed a well-known tool *dnsmasq* [37] to conduct local resolutions with fixed TTL values for the 24 domains representing the Top 1% of domains sorted by TF-



(a) New user TF-IDF



(b) Zipf's data distribution comparison

Figure 7.1: New user's data

IDF.

The testing process proceeded smoothly; nevertheless, to eliminate dependence on any third-party tool, we configured the */etc/hosts* file for this evaluation. Additionally, we included our local server, "localhost," with the IP address *127.0.0.1*, in the */etc/resolv.conf* file. This ensured that the resolutions were performed locally, as the */etc/resolv.conf* file natively manages the DNS resolvers on Linux-based operating systems.

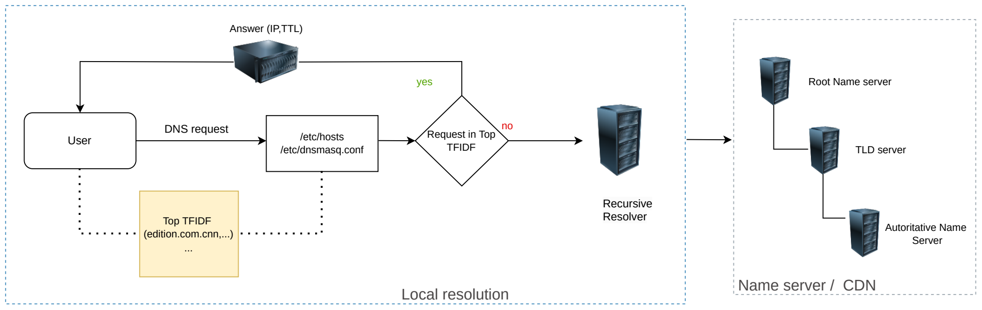
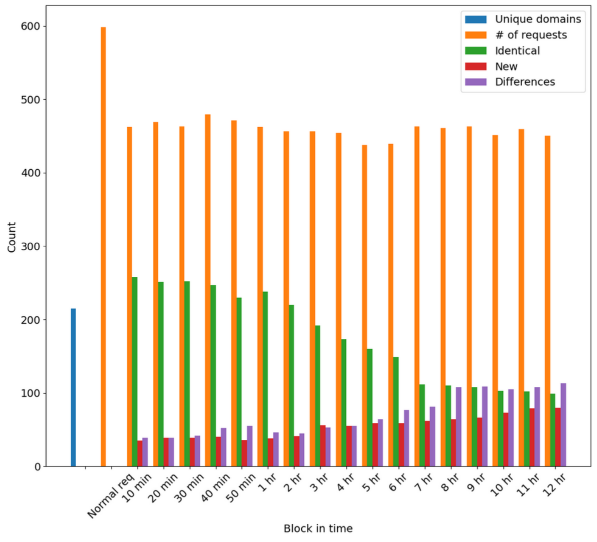


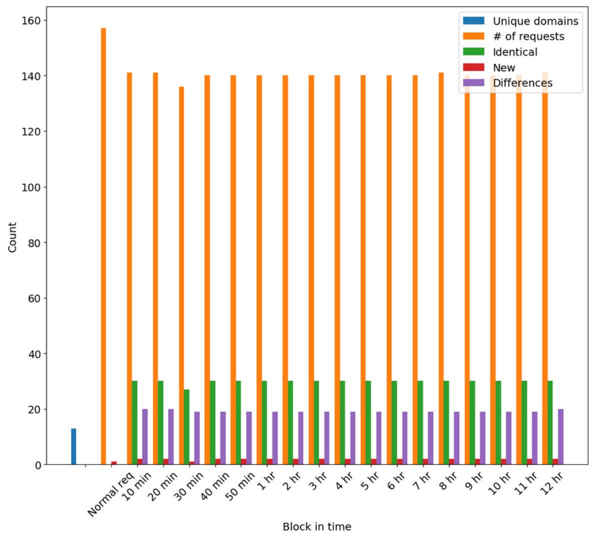
Figure 7.2: Implementation workflow

Any domain name present in this file (/etc/hosts) with a fixed TTL will resolve locally. We confirmed the functionality of the implementation by monitoring the outgoing traffic from our testbed using an open-source packet analysis framework (i.e., Wireshark). We accomplished this by applying a basic filter to capture only the DNS protocol traffic. Nonetheless, it's worth noting that the requests corresponding to the top 1% are not shown in the trace. This suggests that these particular requests might not be reaching the resolver as intended. Figure 7.2 illustrates the general outline of the implementation process.

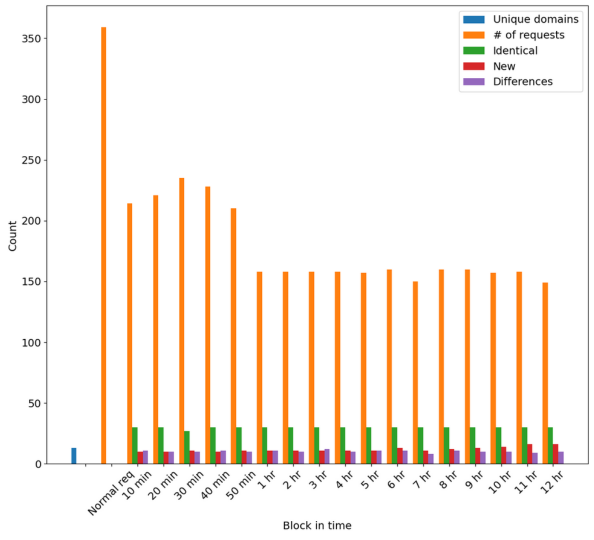
Our experiment began at 10:03 AM on Thursday, May 12, 2023, implementing Top-Block targeting the Top 1% domains. Over the initial 12-hour period, we meticulously observed and analyzed the behavior of three websites selected from this top subset. Figure 7.3 visually represents the critical statistical data acquired while monitoring these distinct websites within this Top 1%: CNN news, Spotify, and Netflix.



(a) CNN



(b) Netflix



(c) Spotify

Figure 7.3: Bar plot of Top-Block performance

An intriguing insight that emerged pertains to the distinct variations in behavior observed between the CNN news website and the other two platforms, namely Spotify and Netflix. This difference becomes particularly evident when examining the number of requests each site generates. Notably, more dynamic websites like news platforms tend to exhibit a significantly higher volume of requests than their entertainment-oriented counterparts like Spotify and Netflix. This contrast in request frequency underscores the distinctive nature of content and user engagement across these diverse online platforms.

Analyzing the bar plots in Figure 7.3, we notice that the green bars represent elements that remained the same compared to a previous request. Over time, these elements tend to decrease while new ones (marked in red bars) increase. This phenomenon can be attributed to the constant updates of content on news pages. The browser must request new elements as the news updates, resulting in the observed trend.

This discrepancy in request patterns highlights the distinct nature of various websites and emphasizes the importance of considering their dynamics while monitoring and optimizing web performance.

For instance, we meticulously observed the end-user experience (UX) during our web performance experiment and discerned various intriguing visual anomalies while navigating the website. Upon successfully deploying the Top-Block feature, we initiated an analysis and noted significant alterations in the website's behavior. Table 7.1 provides a snapshot of the website's behavior within a limited window of time monitored after subsequent implementation of the Top-Block algorithm.

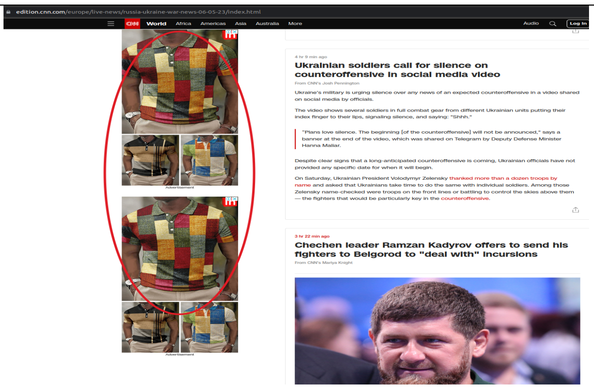
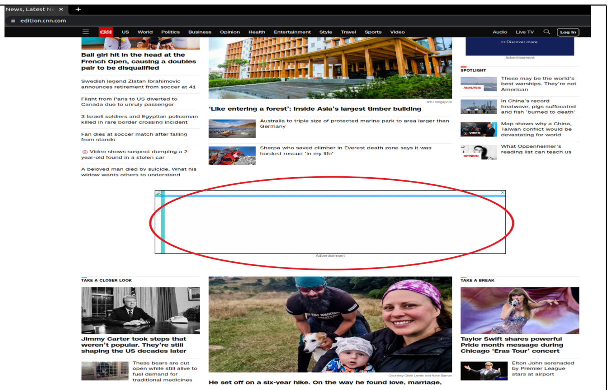
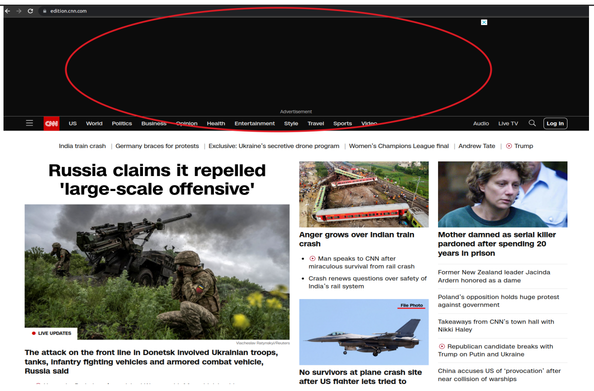
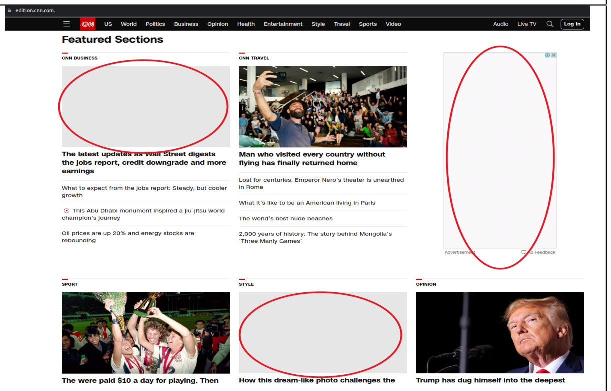
 (a) CNN anomaly 1	 (b) CNN anomaly 2
10:05 AM	10:06 AM
 (c) CNN anomaly 3	 (d) CNN anomaly 4
10:20 AM	4:23 PM

Table 7.1: Table of CNN anomalies implementing Top-Block w.r.t time

The initial anomalies detected are illustrated in subfigures 7.1c and 7.1b, where the advertisements on the CNN news site are blocked. However, we have uncovered another intriguing anomaly, which is better depicted in subfigure 7.1c. Despite the advertisement blocking, a duplicate of the same advertisement appeared on the page a few minutes after we initiated our implementation, as illustrated in figure 7.1c.

Furthermore, after 6 hours, substantial alterations in the website's data integrity became visible. Specifically, two images from the legitimate domain *edition.cnn.com* failed to display. These images are part of the news content, and their absence signifies a website's integrity disruption. This is more effectively depicted in Figure 7.1d. Additionally, even though the advertisements remained blocked, the empty HTML element of the image position*(div)* remained there. Usually, the developers add an alternative description with the alt word in the HTML code in case an image/video fails to load. Nevertheless, not even the alternative description seems to appear.

Given that the concept behind this implementation is to run indefinitely, certain anomalies might arise over an extended period, including missing or even outdated information. In the worst-case scenario, a broken link (Error 404) might occur, indicating that the server could not locate a webpage requested by the client.

Chapter 8

Discussion and further work

As presented in the previous two chapters, our implementations have demonstrated significant effectiveness in countering machine learning algorithms. However, it is essential to acknowledge that the limited size of our dataset, consisting of only eight users, resulted in blocking a substantial volume of traffic for each user. Precisely, blocking a minimal subset the TD-IDF vector that corresponds to $\approxeq 41.6\%$ of the total DNS traffic. This considerable impact on data volume raises concerns about the scalability of our approach in real-world scenarios. In response to this concern and to minimize any adverse effects on our proposed solutions, we present the following approaches:

- **Scaling dataset up for real-world scenarios:** In real DNS server environments, the user count can easily extend to the hundreds or even thousands. In scenarios with many users, it is expected that blocking only a smaller fraction of DNS traffic (much less than the 40% we require with the 8-user data set) will result in a similar reduction in the accuracy of attacker algorithms.
- **Reinforcement learning algorithm for anomaly detection:** The possibility of using reinforcement learning algorithms can also gain significance. This advanced algorithms would learn from the encountered anomalies found within the website, thereby enabling the identification of patterns that reveal aspects that may not have functioned as expected. This learning process empowers the algorithm to avoid repeating those aspects in future requests (e.g., solving locally a specific domain name for a shorter time). The reinforcement learning algorithm's adaptive nature and ability to refine decision-making based on past experiences make it well-suited for situations characterized by reduced variability. The algorithm can adjust its strategies to ensure optimal results in subsequent iterations by studying anomalies and their corresponding outcomes.
- **Overwriting TTL for longer cache retention:** In some scenarios, adjusting the Time-To-Live (TTL) values can effectively control the duration for which DNS resolutions are cached. Administrators can extend the cache retention time by overwriting the TTL values associated with DNS records, reducing query frequency to authoritative DNS servers. This can be particularly advantageous for frequently ac-

cessed websites, where a longer cache retention time can significantly decrease the query load on the DNS infrastructure and improve overall response times.

Extending TTL values does come with certain considerations. While longer cache retention reduces the number of DNS queries and enhances the user experience, it also introduces a potential delay in reflecting changes made to domain configurations. Administrators need to find a balance between cache duration and responsiveness to ensure that timely updates are propagated. Additionally, monitoring and analyzing the impact of extended TTL values on DNS traffic patterns and user experience is crucial to maintaining an optimal configuration.

- **Optimizing Resolutions with Machine Learning:** Another approach involves employing a machine learning algorithm that analyzes the relationship between domains and changes in the retrieved objects. Instead of extending or adjusting the TTL, consideration is given to generating a set of all possible resolutions for a site based on the pattern of changes over time. Take the example of Netflix: if we identify that the domain resolutions for Netflix remain unchanged for a certain period, we can gather a collection of all possible known resolutions and strategically employ them to achieve the desired resolution. Since this set of resolutions remains constant, we can leverage the changing dynamics of DNS queries to optimize and recycle resolutions in a process similar to DNS query recycling.

Chapter 9

Conclusion

This thesis presents a series of algorithms based on a single-node approach to obfuscate the DNS fingerprints through techniques such as stochastic processes and selective manipulation of domain names via blocking or swapping. Notably, these techniques demonstrate their effectiveness in minimizing the accuracy of the attacker models, thereby enhancing privacy, all without requiring "trust" or involving any other network nodes or intermediaries/relays, as suggested in prior works.

For instance, incorporating stochastic algorithms like P-Block and its various swapping variants, such as P-Mix-All and P-Mix-Bottom, has effectively reduced the identification accuracy of our threat models. However, it is worth noting that the domain name blocking or swapping rate needs to exceed 70-80% to achieve a substantial reduction in accuracy. Conversely, based on the TF-IDF metric, our second batch of algorithms strikes a more balanced compromise between privacy and the user experience on the internet, even blocking/swapping the top 1% of each user domain names. These algorithms demonstrate the potential for achieving significant minimization in identification accuracy while preserving a more acceptable level of service quality for users.

We validated the effectiveness of our approach by employing a limited dataset from our testbed, involving only eight users, which led to an accuracy reduction of approximately 40% with a blocking of 41.6% of DNS traffic. However, when considering a larger-scale DNS dataset with hundreds or even thousands of users, it is reasonable to anticipate a significant decrease in the necessary amount of DNS traffic to be blocked while achieving the same desired privacy level. The reduction required in large-scale experiments remains unknown but will be the focus of further studies.

The code of our implementation can be found at:
`https://github.com/Geobm/ObfuscaLone`

Bibliography

[1] S. Kemp, "Digital 2023: Global overview report - datareportal – global digital insights," Feb 2023. [Online]. Available: https://datareportal.com/reports/digital-2023-global-overview-report

[2] C. I. Services, "What is dns hierarchy architecture with examples (explained)," https://cloudinfrastructureservices.co.uk/what-is-dns-hierarchy/, 2021, accesed on 02 01, 2023.

[3] O. Arana, H. Benítez-Pérez, J. Gomez, and M. Lopez-Guerrero, "Never query alone: A distributed strategy to protect internet users from dns fingerprinting attacks," *Computer Networks*, vol. 199, p. 108445, 2021.

[4] R. Lambiotte and M. Kosinski, "Tracking the digital footprints of personality," *Proceedings of the IEEE*, vol. 102, no. 12, pp. 1934–1939, 2014.

[5] D. Herrmann, C. Gerber, C. Banse, and H. Federrath, "Analyzing characteristic host access patterns for re-identification of web user sessions," *Nordic Conference on Secure IT Systems*, pp. 136–154, 2010.

[6] D. Herrmann, C. Banse, and H. Federrath, "Behavior-based tracking: Exploiting characteristic patterns in dns traffic," *Computers & Security*, vol. 39, pp. 17–33, 2013.

[7] M. Kirchler, D. Herrmann, J. Lindemann, and M. Kloft, "Tracked without a trace: linking sessions of users by unsupervised learning of patterns in their dns traffic," in *AISec 1́6: Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. Association for Computing Machinery, 2016, pp. 23—-34.

[8] H. Diazgranados, H. Aver, L. Grustniy, and O. Svistunova, "Defending digital privacy: taking personal protection to the next level." [Online]. Available: https://www.kaspersky.com/blog/global-privacy-report-2020/

[9] R. P. Doyle, J. S. Chase, S. Gadde, and A. M. Vahdat, "The trickle-down effect: Web caching and server request distribution," *Computer Communications*, vol. 25, no. 4, pp. 345–356, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366401004066

[10] P. Hoffman and P. McManus, "Dns queries over https (doh)," Internet Engineering Task Force, RFC 8484, 2018.

[11] ——, “Specification for dns over transport layer security (tls),” Internet Engineering Task Force, RFC 7858, 2016.

[12] C. Huitema, S. Dickinson, and A. Mankin, “Dns over dedicated quic connections,” Internet Engineering Task Force, RFC 9250, 2018.

[13] D. Herrmann, M. Maaß, and H. Federrath, “Evaluating the security of a dns query obfuscation scheme for private web surfing,” in *ICT Systems Security and Privacy Protection*, N. Cuppens-Boulahia, F. Cuppens, S. Jajodia, A. Abou El Kalam, and T. Sans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 205–219.

[14] P. Schmitt, A. Edmundson, A. Mankin, and N. Feamster, “Oblivious dns: Practical privacy for dns queries: Published in popets 2019,” in *Proceedings of the Applied Networking Research Workshop*, ser. ANRW ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 17–19. [Online]. Available: https://doi.org/10.1145/3340301.3341128

[15] J. Kurihara and T. Kubo, “Mutualized oblivious DNS (μodns): Hiding a tree in the wild forest,” *CoRR*, vol. abs/2104.13785, 2021. [Online]. Available: https://arxiv.org/abs/2104.13785

[16] K. Loesing, S. J. Murdoch, and R. Dingledine, “A case study on measuring statistical data in the Tor anonymity network,” in *Proceedings of the Workshop on Ethics in Computer Security Research (WECSR 2010)*, ser. LNCS. Springer, January 2010.

[17] S. Müller, F. Brecht, B. Fabian, S. Kunz, and D. Kunze, “Distributed performance measurement and usability assessment of the tor anonymization network,” *Future Internet*, vol. 4, no. 2, pp. 488–513, 2012. [Online]. Available: https://www.mdpi.com/1999-5903/4/2/488

[18] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, “Dns performance and the effectiveness of caching,” *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, p. 74–74, 2002.

[19] P. Mockapetris, “Domain names - implementation and specification,” Internet Engineering Task Force, Tech. Rep., November 1987.

[20] N. Vlajic, M. Andrade, and U. Nguyen, “The role of dns ttl values in potential ddos attacks: What do the major banks know about it?” *Procedia Computer Science*, vol. 10, p. 466–473, 2012.

[21] D. Eastlake, “Domain name system (dns) iana considerations,” Internet Engineering Task Force, Tech. Rep., March 2011.

[22] N. C. Fofack and S. Alouf, “Modeling modern dns caches,” in *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, ser. ValueTools ’13. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, p. 184–193. [Online]. Available: https://doi.org/10.4108/icst.valuetools.2013.254416

[23] V. Quezada, F. Astudillo-Salinas, L. Tello-Oquendo, and P. Bernal, "Real-time bot infection detection system using dns fingerprinting and machine-learning," *Computer Networks*, vol. 228, p. 109725, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623001706

[24] D. Vekshin, K. Hynek, and T. Cejka, "Doh insight: Detecting dns over https by machine learning." New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3407023.3409192

[25] D. Herrmann, K.-P. Fuchs, J. Lindemann, and H. Federrath, "Encdns: A lightweight privacy-preserving name resolution service," in *Computer Security - ESORICS 2014*, M. Kutyłowski and J. Vaidya, Eds. Cham: Springer International Publishing, 2014, pp. 37–55.

[26] T. Lange and D. J. Bernstein, "Dnscurve: Preventing dns forgery," https://tools.ietf.org/html/rfc4986, 2007.

[27] F. Denis and Y. Fu, "Dnscrypt version 2 protocol specification," https://www.dnscrypt.org/, December 2015.

[28] Q. Liu, W. Wu, Q. Liu, and Q. Huangy, "T2dns: A third-party dns service with privacy preservation and trustworthiness," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020, pp. 1–11.

[29] S. Singanamalla, S. Chunhapanya, M. Vavrusa, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan, and C. A. Wood, "Oblivious DNS over HTTPS (odoh): A practical privacy enhancement to DNS," *CoRR*, vol. abs/2011.10121, 2020. [Online]. Available: https://arxiv.org/abs/2011.10121

[30] H. Schulzrinne and R. Jesske, "Oblivious DNS over HTTPS (DOH)," Internet Engineering Task Force (IETF), Tech. Rep. 8484, 2018. [Online]. Available: https://tools.ietf.org/html/rfc8484

[31] F. Denis, "Anonymized dnscrypt specification," https://github.com/espressif/esp-idf/tree/v5.1, June 2020, v5.1.

[32] ——, "Anonymized dns," https://github.com/DNSCrypt/dnscrypt-proxy/wiki/Anonymized-DNS, January 2021, commit ID: 9e384ee.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, no. 61, pp. 1871–1874, 2008. [Online]. Available: http://jmlr.org/papers/v9/fan08a.html

[35] Y. Yuan, L. Wu, and X. Zhang, "Gini-impurity index analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3154–3169, 2021.

[36] L.-P. Jing, H.-K. Huang, and H.-B. Shi, "Improved feature selection approach tfidf in text mining," in *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2, 2002, pp. 944–946 vol.2.

[37] S. Kelley, "Dnsmasq," https://thekelleys.org.uk/gitweb/?p=dnsmasq.git, April 2001, version: dnsmasq-2.89.

Appendix A

Code

A.1 Data preprocessing

```
#importing libraries
import os
import pandas as pd
import numpy as np
import json
from joblib import parallel_backend
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from functools import reduce
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay,
    classification_report, confusion_matrix
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
from functools import reduce


def process_user_data(columns, s_ip_start, d_ip_target):
    user = columns[columns['s_ip'].str.startswith(s_ip_start) &
                   (columns['dns_query_type'] == 'A') &
                   (columns['d_ip'] == d_ip_target) &
                   (columns['dns_query_flag'] == 1)]

    if len(user) == 0:
        print("Empty doc for user in:", filename)
        return None
    else:
        user_servers = user.d_ip.value_counts()
        user = user.drop(columns=['dns_query_type', 'd_ip', 's_ip', '
    dns_query_flag'])
        user = user.reset_index(drop=True)
        return user, user_servers

def process_file(file_path, subset_list, subset_servers_list, y_list):
    data = pd.read_json(file_path)
    columns = data.loc[:, ['s_ip', 'dns_query', 'dns_query_type', 'd_ip'
    , 'dns_query_flag', 'dns_ans', 'timestamp']]

    users = [
        ("132.248.10.2", "192.168.27.153"),
        ("132.248.10.2", "192.168.27.16"),
        ("8.8.8.8", "192.168.27.160"),
        ("8.8.8.8", "192.168.27.161"),
        ("8.8.8.8", "192.168.27.162"),
        ("132.248.10.2", "192.168.27.17"),
        ("132.248.10.2", "192.168.27.180"),
        ("132.248.10.2", "192.168.27.215")
    ]

    for s_ip_start, d_ip_target in users:
        user_data, user_servers = process_user_data(columns, s_ip_start,
    d_ip_target)
        if user_data is not None:
            subset_list.append(user_data['dns_query'])
            subset_servers_list.append(user_servers)
            y_list.append(d_ip_target)

path = '/content/drive/MyDrive/anon_dns_data/'
subset_list = []
subset_servers_list = []
y_list = []

with parallel_backend('threading', n_jobs=5):
    try:
        for root, dirs, files in os.walk(path):
            for filename in sorted(files):
                file_path = os.path.join(root, filename)
                print("Processing:", filename)
                process_file(file_path, subset_list, subset_servers_list
    , y_list)
    except Exception as e:
        print(e)
```

A.2 Threat Models

```
def train_predict_multinomial_nb(X_train, y_train, X_test):
    mnb = MultinomialNB(alpha=5, fit_prior=False)
    mnb.fit(X_train, y_train)
    return mnb.predict(X_test)

```

```
def train_predict_linear_svm(X_train, y_train, X_test):
    linear_svm = svm.SVC(C=1e-2, kernel='linear',
   decision_function_shape='ovr')
    linear_svm.fit(X_train, y_train)
    return linear_svm.predict(X_test)

def train_predict_random_forest(X_train, y_train, X_test):
    rand_forest = RandomForestClassifier(n_estimators=100, n_jobs=1,
   random_state=42, verbose=0, class_weight='balanced', oob_score=True)
    rf = rand_forest.fit(X_train, y_train)
    return rf.predict(X_test)

# Tokenize the data
vect = CountVectorizer(token_pattern=r'\b(?:[A-Za-z0-9](?:[A-Za-z0
   -9\-]{0,61}[A-Za-z0-9])?\.)+[A-Za-z]{2,6}\b')
Xencoded = vect.fit_transform(matrix_df['traffic'])

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(Xencoded, y,
   train_size=0.7, test_size=0.3, random_state=42)

benchmark_mnb, benchmark_svm, benchmark_rf = [], [], []
random_states = [1, 10, 25, 38, 42, 55, 70, 80, 97, 100]

for state in random_states:
    X_train_split, _, y_train_split, _ = train_test_split(X_train,
   y_train, train_size=0.7, test_size=0.3, random_state=state)

    y_pred_mnb = train_predict_multinomial_nb(X_train_split,
   y_train_split, X_test)
    benchmark_mnb.append(accuracy_score(y_test, y_pred_mnb) * 100)

    y_pred_svm = train_predict_linear_svm(X_train_split, y_train_split,
   X_test)
    benchmark_svm.append(accuracy_score(y_test, y_pred_svm) * 100)

    y_pred_rf = train_predict_random_forest(X_train_split, y_train_split
   , X_test)
    benchmark_rf.append(accuracy_score(y_test, y_pred_rf) * 100)

# Print benchmark results
print("Multinomial Naive Bayes benchmark:", benchmark_mnb)
print("Linear SVM benchmark:", benchmark_svm)
print("Random Forest benchmark:", benchmark_rf)
```

A.3 P-Block

```
def p_block(train_data, test_data, p_values, random_states):
    mnb = MultinomialNB()
    linear = SVC(kernel='linear', decision_function_shape='ovr')
    rand_forest = RandomForestClassifier(n_estimators=100, n_jobs=1,
   random_state=42, verbose=0, class_weight='balanced', oob_score=True)
    vectorizer = CountVectorizer(token_pattern=r'\b(?:[A-Za-z0-9](?:[A-Za-z0-9\-]{0,61}[A-Za-z0-9])?\.)+[A-Za-z]{2,6}\b')

    all_mnb_accuracies, all_svm_accuracies, all_rf_accuracies = [], [], []

    for p in p_values:
        new_test_data = test_data.copy()
        mnb_accuracies, svm_accuracies, rf_accuracies = [], [], []

        for index, row in new_test_data.iterrows():
            traffic_list = row['traffic'].split(',')
            total_elements = len(traffic_list)
            removed_elements = 0

            #Compare rand variable here
            for element in row['traffic'].split(','):
                rand_var = int.from_bytes(os.urandom(8), byteorder="big") / ((1 << 64) - 1)
                if rand_var < p:
                    traffic_list.remove(element)
                    removed_elements += 1

            new_test_data.at[index, 'traffic'] = ','.join(traffic_list)

        X_train = vectorizer.fit_transform(train_data['traffic'])
        y_train = train_data['day_user'].tolist()

        X_test = vectorizer.transform(new_test_data['traffic'])
        y_test = test_data['day_user'].tolist()

        for state in random_states:
            X_train_split, _, y_train_split, _ = train_test_split(X_train, y_train, random_state=state)

            mnb.fit(X_train_split, y_train_split)
            y_pred_mnb = mnb.predict(X_test)

            linear.fit(X_train_split, y_train_split)
            y_pred_svm = linear.predict(X_test)

            rf = rand_forest.fit(X_train_split, y_train_split)
            y_pred_rf = rf.predict(X_test)

            mnb_accuracies.append(accuracy_score(y_test, y_pred_mnb) * 100)
            svm_accuracies.append(accuracy_score(y_test, y_pred_svm) * 100)
            rf_accuracies.append(accuracy_score(y_test, y_pred_rf) * 100)

        all_mnb_accuracies.append(mnb_accuracies)
        all_svm_accuracies.append(svm_accuracies)
```

```python
        all_rf_accuracies.append(rf_accuracies)

    return all_mnb_accuracies, all_svm_accuracies, all_rf_accuracies

# Load your train_data and test_data

p_values = [0.1, 0.2,0.3,0.,0.5,0.6,0.7,0.8,0.9,0.99]  # Adjust p values accordingly
random_states = [1, 10, 25, 38, 42, 55, 70, 80, 97, 100]

mnb_accuracies, svm_accuracies, rf_accuracies = p_block(train_data, test_data, p_values, random_states)

for i, p in enumerate(p_values):
    print(f"Results for p = {p}:")
    print("MNB:", mnb_accuracies[i])
    print("SVM:", svm_accuracies[i])
    print("Random Forest:", rf_accuracies[i])
```

A.4 Top-Block

```python
def top_block(user_ip, train_data, test_data, top_per_user):
    traffic_user_train = train_data[train_data['day_user'] == user_ip]
    traffic_user_test = test_data[test_data['day_user'] == user_ip]

    vectorizer = TfidfVectorizer(token_pattern=r'\b(?:[A-Za-z0-9](?:[A-Za-z0-9\-]{0,61}[A-Za-z0-9])?\.)+[A-Za-z]{2,6}\b')
    tfidf_vector = vectorizer.fit_transform(traffic_user_train['traffic'])
    tfidf_df = pd.DataFrame({
        f'queries user {user_ip}': vectorizer.get_feature_names_out(),
        'idf': vectorizer.idf_,
        'tf-idf_sum': np.asarray(tfidf_vector.sum(axis=0)).ravel(),
        'tf-idf_mean': np.asarray(tfidf_vector.mean(axis=0)).ravel()
    })

    tf_idf_table = tfidf_df.sort_values(by='tf-idf_sum', ascending=False)
    tf_idf_table['URL'] = tf_idf_table.index
    tf_idf_table = tf_idf_table.reset_index(drop=True)

    top_per = int(len(tf_idf_table) * top_per_user)
    domains_to_remove = tf_idf_table[f'queries user {user_ip}'][:top_per].tolist()
    traffic_user_test['traffic'] = traffic_user_test['traffic'].str.replace('|'.join(domains_to_remove), '', regex=True)
    traffic_user_test['traffic'] = traffic_user_test['traffic'].str.replace(',+', ',', regex=True)

    return traffic_user_test

```

```
user_ips=['192.168.27.153','192.168.27.16','192.168.27.160','192.168.27.161','192.168.27.162','192.168.27.17','192.168.27.180', '192.168.27.215', '172.18.41.231']

top_percent_values = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]
dataframes = []

for user_ip in user_ips:
    for top_per_user in top_percent_values:
        user_test_df = top_block(user_ip, train_data, test_data, top_per_user)
        dataframes.append(user_test_df)

merged_df = reduce(lambda left, right: pd.merge(left, right, on=['day_user', 'traffic'], how='outer'), dataframes)
merged_df = merged_df.sample(frac=1).reset_index(drop=True)
```

A.5 Top-Mix-All

```
def top_mix_all(train_data, test_data, users, vectorizer_tmix, top_percent_values):
    dataframes_list = []

    for top_percent in top_percent_values:
        dataframes = []

        for user in users:
            traffic_user_train = train_data.loc[train_data['day_user'] == user]
            traffic_user_test = test_data.loc[test_data['day_user'] == user]

            tfidf_vector = vectorizer_tmix.fit_transform(traffic_user_train['traffic'])
            tfidf_df = pd.DataFrame({'queries user': vectorizer_tmix.get_feature_names_out(),
                                     'idf': vectorizer_tmix.idf_,
                                     'tf-idf_sum': np.asarray(tfidf_vector.sum(axis=0)).ravel(),
                                     'tf-idf_mean': np.asarray(tfidf_vector.mean(axis=0)).ravel()})
            tf_idf_table = tfidf_df.sort_values(by='tf-idf_sum', ascending=False)
            tf_idf_table['URL'] = tf_idf_table.index
            tf_idf_table = tf_idf_table.reset_index(drop=True)
            top_per_user = int(len(tf_idf_table) * top_percent)
            domains_to_remove = tf_idf_table['queries user'][:top_per_user].tolist()
            remaining_domains = tf_idf_table['queries user'][top_per_user:].tolist()
```

```python
            domains_to_remove_mix = np.random.choice(remaining_domains, size=top_per_user, replace=False)

            traffic_user_test_copy = traffic_user_test.copy()
            for i, row in traffic_user_test_copy.iterrows():
                for j in range(len(domains_to_remove)):
                    row['traffic'] = row['traffic'].replace(domains_to_remove[j], domains_to_remove_mix[j])

            dataframes.append(traffic_user_test_copy)

        merged_df = reduce(lambda left, right: pd.merge(left, right, on=['day_user', 'traffic'], how='outer'), dataframes)
        merged_df = merged_df.sample(frac=1).reset_index(drop=True)
        dataframes_list.append(merged_df)

    return dataframes_list

# Define your vectorizer_tmix, users, and top_percent_values
vectorizer_tmix = CountVectorizer(token_pattern=r'\b(?:[A-Za-z0-9](?:[A-Za-z0-9\-]{0,61}[A-Za-z0-9])?\.)+[A-Za-z]{2,6}\b')
users = ['192.168.27.153', '192.168.27.16', '192.168.27.160', '192.168.27.161', '192.168.27.162', '192.168.27.17', '192.168.27.180', '192.168.27.215']
top_percent_values = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]

# Call the function
resulting_dataframes = top_mix_all(train_data, test_data, users, vectorizer_tmix, top_percent_values)
```

A.6 Top-Mix-Bottom

```python
def top_mix_bottom(train_data, test_data, vectorizer, top_percent_values):
    global user_ips
    dataframes = []

    for user in user_ips:
        traffic_user_train = train_data.loc[train_data['day_user'] == user]
        traffic_user_test = test_data.loc[test_data['day_user'] == user]
        tfidf_vector = vectorizer.fit_transform(traffic_user_train['traffic'])
        tfidf_df = pd.DataFrame({'queries': vectorizer.get_feature_names_out(),
                                 'idf': vectorizer.idf_,
                                 'tf-idf_sum': np.asarray(tfidf_vector.sum(axis=0)).ravel(),
                                 'tf-idf_mean': np.asarray(tfidf_vector.mean(axis=0)).ravel()})

        tf_idf_table = tfidf_df.sort_values(by='tf-idf_sum', ascending=False)
        tf_idf_table['URL'] = tf_idf_table.index
        tf_idf_table = tf_idf_table.reset_index(drop=True)
        for top_percent in top_percent_values:
            top_per_user = int(len(tf_idf_table) * top_percent)
            top_bottom_user = int(len(tf_idf_table) * 0.5)
            domains_to_remove = tf_idf_table['queries'][:top_per_user].tolist()
            remaining_domains = tf_idf_table['queries'][top_bottom_user:].tolist()
            domains_to_remove_mix = np.random.choice(remaining_domains, size=top_per_user, replace=False)
            traffic_user_test_copy = traffic_user_test.copy()
            for i, row in traffic_user_test_copy.iterrows():
                for j in range(len(domains_to_remove)):
                    row['traffic'] = row['traffic'].replace(domains_to_remove[j], domains_to_remove_mix[j])
            dataframes.append(traffic_user_test_copy)
    merged_df = reduce(lambda left, right: pd.merge(left, right, on=['day_user', 'traffic'], how='outer'), dataframes)
    merged_df = merged_df.sample(frac=1).reset_index(drop=True)

    return merged_df
```

A.7 P-Mix-All

```python
def p_mix_all(matrix_df, percent_to_remove=0.8):
    vectorizer = TfidfVectorizer(token_pattern=r'\b(?:[A-Za-z0-9](?:[A-Za-z0-9\-]{0,61}[A-Za-z0-9])?\.)+[A-Za-z]{2,6}\b')
    train_data, test_data = train_test_split(matrix_df, test_size=0.3, random_state=42)
    vectorizer.fit_transform(train_data['traffic'])
    domains_dict = {}
    new_test_data = test_data.copy()

    for user in test_data['day_user'].unique():
        traffic_user_train = train_data.loc[train_data['day_user'] == user]
        traffic_user_test = test_data.loc[test_data['day_user'] == user]

        tfidf_vector_train = vectorizer.fit_transform(traffic_user_train['traffic'])

        tfidf_df = pd.DataFrame({'queries': vectorizer.get_feature_names_out(),
                                 'idf': vectorizer.idf_,
                                 'tf-idf_sum': np.asarray(tfidf_vector_train.sum(axis=0)).ravel(),
                                 'tf-idf_mean': np.asarray(
   tfidf_vector_train.mean(axis=0)).ravel()})

        tf_idf_table = tfidf_df.sort_values(by='tf-idf_sum', ascending=
   False)
        tf_idf_table['URL'] = tf_idf_table.index
        tf_idf_table = tf_idf_table.reset_index(drop=True)

        top_percent = int(len(tf_idf_table) * percent_to_remove)
        domains_to_remove = tf_idf_table['queries'][:top_percent].tolist
   ()
        remaining_domains = tf_idf_table['queries'][top_percent:].tolist
   ()
        domains_to_remove_pmix = np.random.choice(remaining_domains,
   size=top_percent, replace=True)

        domains_dict[user] = (domains_to_remove, domains_to_remove_pmix)

        for index, row in traffic_user_test.iterrows():
            ip_address = row['day_user']
            domains_to_remove, domains_to_remove_pmix = domains_dict[
   ip_address]
            new_traffic = []

            for domain in row['traffic'].split(','):
                rand_var = int.from_bytes(os.urandom(8), byteorder="big"
   ) / ((1 << 64) - 1)
                if rand_var < percent_to_remove:
                    if domain in domains_to_remove:
                        idx = domains_to_remove.index(domain)
                        new_traffic.append(domains_to_remove_pmix[idx])
                    else:
                        new_traffic.append(domain)
                else:
                    new_traffic.append(domain)

            new_test_data.at[index, 'traffic'] = ','.join(new_traffic)

    return new_test_data
```

A.8 P-Mix-Bottom

```
def p_mix_bottom(train_data, test_data, p_values, random_states):
    vectorizer_pmix_bottom = TfidfVectorizer(token_pattern=r'\b(?:[A-Za-
   z0-9](?:[A-Za-z0-9\-]{0,61}[A-Za-z0-9])?\.)+[A-Za-z]{2,6}\b')

    new_test_data = test_data.copy()

    domains_dict = {}

    for user_ip, random_state, top01_per in zip(test_data['day_user'].
   unique(), random_states, p_values):
        train_user_data = train_data[train_data['day_user'] == user_ip]
        test_user_data = test_data[test_data['day_user'] == user_ip]

        tfidf_vectorizer = vectorizer_pmix_bottom.fit(train_user_data['traffic'])
        tfidf_table = pd.DataFrame({
            f'queries {user_ip}': vectorizer_pmix_bottom.get_feature_names_out(),
            'idf': vectorizer_pmix_bottom.idf_,
            'tf-idf_sum': np.asarray(tfidf_vectorizer.sum(axis=0)).ravel(),
            'tf-idf_mean': np.asarray(tfidf_vectorizer.mean(axis=0)).ravel()
        })

        tf_idf_table = tfidf_table.sort_values(by='tf-idf_sum', ascending=False)
        tf_idf_table['URL'] = tf_idf_table.index
        tf_idf_table = tf_idf_table.reset_index(drop=True)

        pmixbottom = int(len(tf_idf_table) * 0.5)

        domains_to_remove = tf_idf_table[f'queries {user_ip}'][:top01_per].tolist()
        remaining_domains = tf_idf_table[f'queries {user_ip}'][pmixbottom:].tolist()

        domains_to_remove_pmix_bottom = np.random.RandomState(random_state).choice(
            remaining_domains, size=top01_per, replace=True
        )

        domains_dict[user_ip] = (domains_to_remove, domains_to_remove_pmix_bottom)

    p_values_dict = dict(zip(test_data['day_user'].unique(), p_values))

    for index, row in new_test_data.iterrows():
        ip_address = row['day_user']
        p = p_values_dict[ip_address]
        domains_to_remove, domains_to_remove_pmix_bottom = domains_dict[ip_address]
        new_traffic = []
        for domain in row['traffic'].split(','):
            rand_var = int.from_bytes(os.urandom(8), byteorder="big") / ((1 << 64) - 1)
            if rand_var < p:
                if domain in domains_to_remove:
                    idx = domains_to_remove.index(domain)
                    new_traffic.append(domains_to_remove_pmix_bottom[idx])
                    print(Fore.GREEN + f"This domain:"+ Fore.RED+ f"{domain}"
                        + Fore.CYAN+ " is being swapped for:"+ Fore.
   MAGENTA
                               + str(domains_to_remove_pmix_bottom[idx])
                    )
                else:
                    new_traffic.append(domain)
            else:
                new_traffic.append(domain)
        new_test_data.at[index, 'traffic'] = ','.join(new_traffic)
    return new_test_data
```