UNIVERSITÀ DI TRENTO

Department of Information Engineering and Computer Science

UNIVERSITY OF TWENTE.

Department of Computer Science

Master's Degree in
Computer Science - ICT Innovation Cyber Security

Final Dissertation

Domain Name Analysis with Machine Learning: Enhancing Efficiency and Reducing Analyst Strain

Supervisor UniTrento

Roberto Doriguzzi Corin

Supervisor UTwente

Florian Hahn

Student

Irina Voloshina 247040

Academic year 2023/2024

Acknowledgements

I want to express my deepest gratitude...

...to my dearest friend Nistha Kumari, who supported my motivation and my sanity during this work;

...to my wonderful professor Roberto Doriguzzi Corin, who dedicated a lot of time to getting me through this work and ensured its quality;

...to all the amazing people who kept telling me "You got this" until they got me believing in that a little bit too: my friends Christian Romeo and Fabian Krüger, my father Alexandr, my grandmother Elena, and my sister Viktoria;

...to the great people who made this research possible in the first place: Massimo Giaimo, Simone Cagol, Mirko Ioris, Francesco Pavanello, Giacomo Giallombardo, Beatrice Dall'Omo, Federico Corona, Lorenzo Bevilacqua, Luca Zeni, Daniel Degasperi, Alessio Paternoster.

Contents

Abstract

In today's era of unprecedented internet connectivity, domain names have become the digital identities of individuals, businesses, and organizations. As digital identities, domain names are strongly associated with an individual or organization's brand and reputation. Therefore, protecting the domain name and its reputation from various kinds of abuse is crucial for a company to function on a digital platform. As domain names are an integral part of organizational infrastructure, they present an attractive target for attackers. Attackers have long utilized techniques such as domain squatting to commit various kinds of fraud and malicious activities. Such activities include misguiding the victims from a legitimate website to an attacker-owned one to gain traffic or present fraudulent content; alternatively, such spoofed domains might be used to disseminate phishing emails to harvest credentials for further attacks. Thus, proactive recognition of potentially malicious domains that might be utilized to perform such attacks on customers is an important activity of a threat intelligence service in a cybersecurity company.

This study aims to shed light on the current state of the domain analysis methodologies and proposes a practical implementation of a machine learning algorithm for the domain analysis to integrate within threat intelligence activities. The algorithm aims to reduce the workload on threat intelligence analysts by identifying false positive (i.e. domains that do not present risk for the customers) with high confidence, leaving only the suspicious domains for further manual investigation, achieving more efficient time and resource utilization by the cybersecurity analysts. The results obtained in the study aim to provide insights on the advantages and challenges of such implementation, as well as suggest further direction for improvement of automated methods in domain name analysis.

1 Introduction

In 2022, the revenues from cybercrime were estimated to be approximately 8.44 trillion U.S. dollars and are expected to reach 17.65 trillion in 2025 [42]. The increasing prevalence of cybercrime has led to a growing need for automated methods to combat the ever-increasing volume of cyberattacks. Current manual approaches are challenging to scale, motivating the development of automation methods capable of assisting analysts in handling large volumes of diverse security alerts. The effectiveness of the automated techniques has been noted by the researchers [8] [42] in various cybersecurity-related contexts of the organization.

One of the most prevalent threats the cybersecurity industry is trying to combat is phishing. Phishing is an attack where a threat actor employs various social engineering techniques to perform identity theft, i.e., spoofing a legitimate entity or individual, to obtain sensitive information [2]. Obtaining this information is not necessarily an end goal; a phishing attack can be the first stage of a more complex, multifaceted attack. While the consequences of phishing attacks may be significant, including loss of sensitive information, intellectual property, compromise of business continuity, or even national security [2], there is no single efficient method for stopping phishing attacks. A multitude of attack vectors and attack techniques make it challenging for companies to prevent phishing altogether.

Phishing often involves spoofed domains via electronic means, such as emails or websites. These domains may resemble a legitimate entity or individual and can be used to host fraudulent pages aimed at gathering information. However, spoofed domains can be used for other purposes that either are malicious or lie in a grey area. One of the measures to reduce risks associated with such domains is to preventively block them on a client network perimeter (e.g., with a firewall or IPS) or, if the domain is proven to be used for malicious activities, request a takedown from its hosting provider. To determine the legitimacy of the domain, security analysts must investigate the domain properties, often combining data from multiple sources. While crucial, this task is highly time- and effort-consuming. Considering the routine nature of the task, it has a great potential for automation.

This study addresses the growing challenge of phishing and other threats to legitimate organizations that involve the use of spoofed domains, highlighting gaps in current domain analysis methodologies. As cyber threats continue to evolve, it is increasingly difficult for threat intelligence analysts to efficiently manage the overwhelming volume of potentially malicious domains. To mitigate this problem, the study aims to develop a machine learning algorithm designed to enhance domain analysis by reducing false positives, allowing analysts to focus their efforts on truly suspicious cases. This approach aims to optimize resource allocation and provide practical insights into the feasibility of such algorithms in a practical context.

This study has been done in collaboration with a medium-sized consulting company offering threat intelligence services to its clients. Specifically, this study investigates the possibility of automating a manual domain name analysis process based on the real dataset of the company. Domain name analysis allows clients to take action against potentially fraudulent communications by blocking suspected domains at the perimeter, thus preventing communication between client employees and the suspicious domain. This approach is one of the many preventive measures that clients use against phishing attacks.

For automation, the proprietary dataset of the company's client data was used to build and train a machine-learning model that closely simulated the analyst's decision-making process. The dataset includes domain name features, such as string characteristics and registrant information, which are highly varied (e.g., numerical, categorical, ordinal). Additionally, the dataset was imbalanced, requiring preprocessing steps to enhance model performance. One of the unique characteristics of the given dataset is the presence of both the legitimate client domain names and the corresponding suspected domains, making it possible to extract features based on the comparison between these pairs of domain names. In contrast, previous studies in the field relied on the characteristics extracted from separate

benign and malicious datasets, which had no connection between them. The main shortcoming of this approach is its failure to account for the specific relationships or similarities that may exist between pairs of domains. In real-world scenarios, domain squatting typically involves creating domains that are deliberately similar to legitimate ones. By treating benign and malicious domains as entirely distinct entities without comparing them directly, this method loses valuable contextual information that could help in detecting subtle patterns of squatting for the lesser-known domains typically not present in such datasets. By contrast, extracting more nuanced features based on their direct comparison in a dataset where benign and suspicious domains are paired becomes possible.

The contributions of this work are summarized as follows:

- A novel approach was applied to domain classification using a dataset that captures the relationships between paired legitimate and suspicious domains, relying on the features extracted from their comparison. This comparison provides new insights into identifying squatted domains and automating this process to reduce the manual work for analysts.
- Several machine learning techniques, namely a random forest and a neural network, were compared to identify the most effective approach for this task. The models were tuned using a two-stage approach to optimize their performance on an imbalanced dataset.
- The project contributes practical insights into real-world applications of machine learning in cybersecurity by addressing the challenges posed by evolving domain squatting techniques and the varying practices across different businesses.
- The system is evaluated in real-world settings through temporal dataset splitting, simulating how the model would perform when deployed in a live environment on previously unseen data.
- The workflow developed in this research provides an example of integrating machine learning models into existing analyst workflows, reducing manual effort while maintaining high levels of accuracy and security.

To demonstrate the decision-making process of the machine learning model, the features of the dataset were ranked according to their importance, i.e., their contribution to the decision-making. This serves as an illustration of how closely the machine can mimic a human operator's thinking process. Additionally, the feature importance classification was used to identify noisy or irrelevant features that, while correlated to the result of the machine prediction, did not align with the decision-making process of a human operator.

The results indicate the potential of machine learning techniques in this domain, suggesting that higher accuracy and reliability could be attained with further data and model refinement. This study contributes to the ongoing development of automated cybersecurity threat detection and analysis tools as it showcases the applicability of neural networks and advanced hyperparameter optimization in identifying potentially malicious domains.

Moreover, establishing clear and systematic processes in domain classification, including preprocessing methods and feature engineering strategies, could further streamline the model training process. Such processes would not only aid in improving current models but also facilitate the development of future models in this domain, ultimately contributing to more effective and reliable cybersecurity solutions.

The remainder of this paper is structured as follows. Section 2 provides background information on the DNS and WHOIS protocols, the application of automation in cybersecurity, and the importance of human behavioral factors in applying these methods. Section 3 covers the current state-of-the-art research in phishing prevention and the application of machine learning methods for domain name analysis. Section 4 provides a company-based scenario and an overview of the initial state of the domain name analysis workflow in the given scenario, as well as the challenges in potentially improving this workflow. Section 5 describes the methodology of this research, with emphasis on the architectural structures of the created models. Section 6 details the experiment setup, including the dataset information, feature engineering, and evaluation metrics. In Section 7, the experiment results are described and interpreted. Finally, Section 8 provides suggestions for improvement and outlines the direction for future work in the field.

2 Background

2.1 Domain Name System

The core functionality of DNS is to translate IP addresses and domain names and navigate users to the requested resources efficiently. Without this navigation, clients of any type (human users and servers alike) could not reach their requested resources [18]. The term DNS refers to the Domain Name System and the protocol this system utilizes. Essentially, the DNS components form a distributed database containing information about domain names and other corresponding data, such as associated IP addresses or hosts [26]. A domain name is a string of alphanumeric characters used to identify a particular resource hosted on the Internet. The structure of the DNS name is depicted in the figure 2.1.

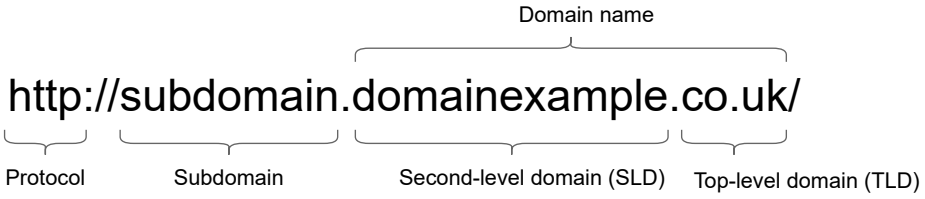

Figure 2.1: Domain name structure.

Disclaimer: All URLs and IP addresses used in this document have been defanged by replacing dots with square brackets (e.g., example[.]com instead of example.com). This practice prevents accidental access to potentially malicious domains or IPs. Defanging ensures that links remain inactive when the document is rendered by any software.

2.1.1 DNS security issues

The DNS (Domain Name System) is a complex infrastructure that initially had no built-in security measures and served the single purpose of translating the DNS names and IP addresses between each other [27] [18]. However, with the increased interconnectedness of the current digital landscape, the role of DNS has become crucial in ensuring the functioning of the Internet as a whole. At the same time, the DNS system has become an attractive target for attackers seeking to disrupt digital communications or perpetuate other forms of harm to organizations and individuals [31].

As a protocol, DNS suffers from a lack of security measures and thus is vulnerable to various threats. While some threats can compromise the security and privacy of the end users, specific threats can disrupt the performance of the Internet services that rely on DNS for their functioning [16][31]. The vulnerabilities of DNS include, among others, man-in-the-middle attacks, cache poisoning, DDoS attacks, registrar hacking, and domain squatting [27]. The following security issues of DNS are excluded from the scope of this study:

- Man-in-the-middle attacks are perpetuated by the absence of any mechanism for DNS server authentication, allowing the server reply packets to be modified by any third party able to sniff the unencrypted DNS traffic [27].
- DNS cache poisoning attacks occur because the current DNS protocol does not provide a fast and secure way to push new information to DNS caches. Therefore, an attacker can insert malicious domain names or IP addresses in the DNS cache, which may result in spreading malware and spam, among many other consequences [16] [27].
- Amplification and (Distributed) Denial-of-Service attacks impact the availability of Internet services relying on DNS for their operation [27]. The amplification attacks typically occur when

the response from the DNS server is significantly larger than the request, allowing the attacker to bombard their victim with a large quantity of DNS messages, causing a delay in performance or, in the worst-case scenario, a complete outage of the service. [17][31]

- Domain Generation Algorithms (DGA) allow attackers to generate and register a virtually unlimited number of domain names. The DGA domains are typically used in malicious infrastructure, such as for the CnC communication between the bots and the control center inside a botnet.

- Malicious domains. Multiple domains in the DNS infrastructure are associated with known malicious activity. These domains are associated with phishing campaigns, botnet management, spam, and malware distribution. Due to the fluid nature of the DNS system, new domains emerge fast while old domains are being taken down or abandoned. This fluidity makes static methods (such as blacklists) ineffective, and alternative methods are being developed in this field [2].

In the scope of this study, the following vulnerabilities related to DNS are considered:

- Phishing attacks. Phishing refers to the attack aiming to get sensitive information, such as personal, financial, or technical (such as organizational credentials) through impersonation as a form of deceit [2]. These attacks are not restricted purely to internet-based methods (such as emails); phishing attacks can also be conducted through phone calls or text messages, among other means. Phishing attacks are one of the rising trends not only by the volume of the attacks but also by the rate of the successful attacks [11]. As phishing attacks heavily rely on impersonation, domain names play a crucial role in attacks where threat actors mimic a legitimate organization or resource to misguide the victims. Despite an extensive research body in the area of phishing detection and prevention [17][16][29], there is no uniform solution for the organizations to combat phishing with 100% effectiveness [16].

- Domain squatting. Domain squatting is a technique attackers utilize to pose as a legitimate online entity of a known brand through domain names that closely resemble legitimate ones and, therefore, can be mistaken as belonging to a legitimate entity [16]. In phishing attacks, squatted domains can host websites resembling the ones belonging to a legitimate company. The purpose of these websites is to trick the victims into revealing their credentials as they believe they are entering them into a legitimate website. These credentials can be leveraged to gain unauthorized access to the company's information systems and perpetuate further harm [40]. In other scenarios, squatted domains can be monetized by exploiting user typos to capture some traffic for legitimate websites. To combat the problem of domain squatting, organizations often perform so-called defensive registration. In other words, companies register the domains that otherwise could have been used for squatting, and the traffic from these sites is redirected to the actual company-owned page. However, this approach is practical only for a limited subset of squatted domains, as it usually encompasses only users' most popular typos [40].

  The attacks exploiting domain squatting occur due to virtually unlimited possibilities to register domain names similar to known legitimate domains, as there is no technical mechanism to prevent these registrations. In some countries, typosquatting is considered illegal and is prosecuted. Under the 1999 Trademark Cyberpiracy Prevention Act, it is unlawful to misdirect consumers to fraudulent websites if the original domain name is associated with a trademarked business [24]. If such a website is discovered that meets the domain squatting criteria, it will be taken down by the registrar. However, domain squatting remains a severe issue in other situations. Not all squatted domains can be proven to have a fraudulent purpose, and not all have an associated website. Therefore, it remains the responsibility of the companies to monitor possible typosquatted domains and prevent potential exploitation of a brand name and reputation.

- Parked domains and their monetization. The term "parked domain" refers to a domain name registered but not actively used for website hosting. Instead, it is often "parked" on a simple placeholder page, typically displaying advertisements. The primary goal of parking a domain is monetization, where the domain owner earns revenue through ad clicks or affiliate links displayed

on the parked page. Parked domains can generate passive income if the domain name attracts significant traffic. This traffic can be generated by utilizing a typosquatted version of a legitimate domain so users can visit it through accidental typos in the domain name. Although parked domain monetization is technically a legitimate business practice, it frequently overlaps with other activities, such as click fraud, traffic spam, and hijacking. These activities can often distribute malicious content, including malware and spam [40][16].

2.1.2 Domain name analysis

Domain name analysis is one of the crucial tasks performed by cybersecurity analysts on the threat intelligence team. Identifying potentially malicious domains serves as a proactive approach for protecting clients from a wide range of threats, including:

- Incoming fraudulent communications. By blocking the suspicious domains at the organization's outer network perimeter, the clients can prevent incoming communications (such as emails) and protect their employees from phishing attacks. Phishing attacks are crucial to control as they often serve as a first step in a more sophisticated attack. Additionally, phishing attacks can lead to spam and malware distribution in the company network.

- Identity/brand spoofing. By identifying malicious domains that attempt to impersonate the organization, the clients can request the domain takedown and initiate legal actions in more severe cases [3].

- Other threats. A spoofed domain name reflects the potential for an attack, for instance, if several such domains are found in a short period with similar registration information. Monitoring for these domains can act as a proactive defense measure, flagging potential threats before they are launched. In threat intelligence terms, such domain name is considered an Indicator of Attack (IoA) [31].

Analysts can analyze the full DNS record associated with the domain from the domain name. The common types of the DNS records are [13]:

- **A records** are the most common DNS records, establishing a direct link between an IPv4 address and a domain name. An example of an IPv4 address is: 90.180.21[.]14.

- **AAAA records** connect domain names to IPv6 addresses with a longer format. IPv6 addresses are becoming more prevalent as IPv4 addresses are running out.

- **CNAME records** direct an alias domain to a canonical domain, typically linking subdomains to the domain's A or AAAA records. This allows changes to the root domain's IP address to update all linked CNAMEs automatically.

- **MX records** direct emails to the domain's mail server, enabling the setup of email accounts (e.g., user@example[.]com) linked to the domain.

- **NS records** indicate which DNS server is the authoritative nameserver for a domain, pointing to the domain's various records. They are essential for users to access the website.

- **SOA records** hold important administrative information about a domain, including details on updates and server refresh intervals.

- **PTR records** work in reverse of A records, mapping an IP address back to a domain name. They are used to validate legitimate email servers and detect spam.

- **TXT records** store textual information related to domains, including SPF, DKIM, and DMARC records for email verification.

Analysts can discover additional attributes associated with the given domain name using publicly available resources. These attributes help to understand the context of the domain usage and, with

sufficient confidence, determine whether the domain is malicious or can potentially be used for malicious purposes. It is important to note that the DNS infrastructure is fluid and changes constantly. Therefore, the same domain name can be reused for different purposes after a specific time. Thus, the domain analysis is not a one-time activity; the same domain needs to be rechecked with sufficient frequency for any changes. Discovering information from public sources is called open-source intelligence (OSINT).

2.2 WHOIS and RDAP protocols

The domain name space is managed by registries and registrars in a hierarchical structure. The Internet Corporation for Assigned Names and Numbers (ICANN) is responsible for creating Top-Level Domains (TLDs), such as *.com*, and delegates their operation to registries that manage the TLD zones. These registries, in turn, delegate customer service responsibilities to registrars, such as GoDaddy, who sell domain names to registrants or domain holders. All registrars and registries that manage generic TLDs (gTLDs) are contracted with ICANN through the Registrar Accreditation Agreement (RAA) and the Registry Agreement (RA) [21]. The figure 2.2 reflects the workings of the WHOIS ecosystem.

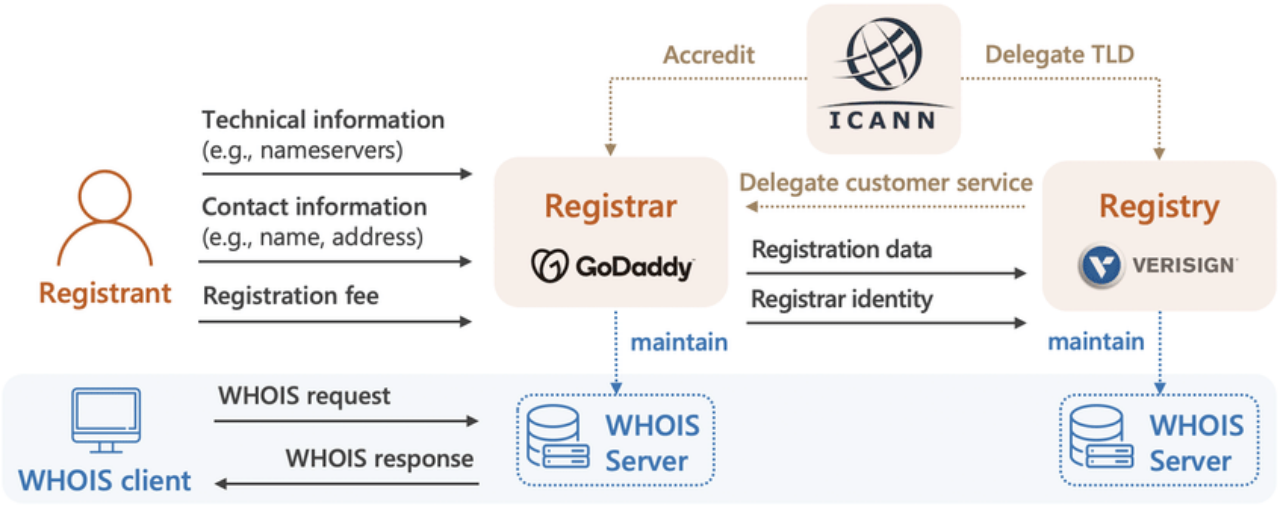

Figure 2.2: WHOIS ecosystem and its processes [21].

Having the domain name, it is possible to infer the following information through an additional OSINT investigation:

- Registration information: registrar, hosting, registrant (if not anonymized).
- IP history: any previous and current IP addresses associated with the domain.
- Certificate information: any certificates associated with the domain.
- Detected malicious activity: the domain name and associated IP addresses can be checked against known threats.
- Domain name reputation.

The registration information is typically retrieved from the WHOIS database. WHOIS is a protocol used to query the information about the registrant of a domain name. It operates on TCP port 43 and is officially documented in RFC 3912 [7]. The information provided includes the name and company of the individual or entity that registered the domain, as well as details regarding the DNS servers associated with that domain and the operators of those servers.

Although WHOIS offers valuable information, it is a poorly designed protocol. There are numerous WHOIS servers globally, each containing a portion of the total Internet domain names [30]. From a domain name, it is impossible to determine which particular WHOIS server holds the registrant

information for that domain name. Additionally, the format of WHOIS responses is not correctly standardized, making it challenging to parse the replies effectively [21][34].

Most countries maintain their own WHOIS server for their respective top-level domains (e.g., *.co.uk*, *.ie*), while international top-level domains like *.com*, *.net*, and *.org* are divided among large WHOIS servers or allocated by central servers across different regions. Some well-known WHOIS servers include *whois.networksolutions.com*, *whois.crsnic.net*, and *whois.ripe.net* [34]. When a domain is registered, data such as the registrant's contact information, domain creation and expiration dates, nameservers, and the registrar are stored in a publicly accessible WHOIS database. This information can be freely queried using the WHOIS protocol to identify the entity or individual behind a domain name. While WHOIS is a valuable tool for transparency, privacy concerns and the introduction of GDPR have led to data protection regulations and privacy services that anonymize some of the information in WHOIS records [21].

RDAP (Registration Data Access Protocol)[30] is a protocol designed as a successor to WHOIS with the goal of addressing multiple shortcomings of the WHOIS protocol. It utilizes a RESTful interface over HTTP to query and retrieve information. The data from an RDAP query is provided in a standardized JSON format (as opposed to non-standardized WHOIS), which can easily be parsed by scripts and tools while remaining readable for human operators. The HTTP interface can also be encrypted using TLS, ensuring data integrity. This is especially crucial in ongoing investigations where sensitive or confidential information should not be transmitted openly [30].

RDAP also supports authenticated and controlled access to its data, depending on the web server in use. The amount of information returned from an RDAP query can vary based on the user's authentication status. For instance, authenticated users may access more WHOIS data fields than unauthenticated users. This feature introduces the possibility of granting special investigative access to additional registry data without requiring a separate legal framework [30]. This approach should address the privacy issues around WHOIS without hindering any possible cybercrime investigations reported after introducing privacy regulations around WHOIS [21]. However, despite the increasing popularity of RDAP, WHOIS still is the most popular protocol for registration data retrieval, and replacing WHOIS with RDAP is a lengthy process [21].

While RDAP successfully addresses some of the challenges of WHOIS protocol, both WHOIS and RDAP share the common issue of inaccurate data. The registrant submits the registration information, and the quality of this data depends on the validation checks done by each registry provider [30][21]. The information retrieved via either of these protocols can serve as a source for domain name classification and ranking, leading to an analyst's ultimate verdict on the level of risk the domain presents for a client organization. Inaccurate information may hinder the investigation or lead to an incorrect verdict altogether.

2.3 Automation methods in cybersecurity

Due to the constantly increasing rates of cybercrime worldwide [8][11], the need for automation methods is felt firmly in the cybersecurity industry [42]. Automation methods are already used in areas such as automated threat detection, security management, and threat response [42]. By leveraging machine learning algorithms, cybersecurity systems can reliably detect potential threats in an organization's ecosystem and manage the organization's security perimeter. Additionally, through automating repetitive and mundane tasks, cybersecurity professionals are released to focus on more challenging and sophisticated tasks, thus increasing efficiency and reducing overall costs. [42].

Security Information and Event Management (SIEM) and Security Orchestration, Automation, and Response (SOAR) systems are the most popular use cases for automation in cybersecurity. SIEM systems are designed to collect, aggregate, and analyze data from various sources across a network, providing centralized monitoring for security events. SIEM systems offer a comprehensive view of an organization's security environment, helping detect threats that might go unnoticed if individual systems were monitored separately. However, the large volume of alerts produced, including false positives, can overwhelm security teams, leading to alert fatigue. SOAR systems are designed to enhance the efficiency of security operations by automating response workflows and coordinating security tools. SOAR systems help automate the investigation and response to threats, reducing

manual intervention and improving the speed of incident resolution.

However, challenges are associated with introducing the automated methods into the cybersecurity operations. First, automation can be complex and require a lot of investment, which can be off-putting to some organizations despite the evident benefits (including economical) [42]. Additionally, implementing automation requires specific skills that may be limited in the organization. Even when those skills are available, allocating a portion of the workforce to the development of an automation system might be considered wasteful by an organization despite long-term cost savings.

In particular, the complexity of the automation and lack of required data can impede the development and adoption process for machine learning models. Inaccurate machine learning models provide little to no value for an organization. Therefore, the quality of the implementation of any automation must be assessed thoroughly before introducing it as an integral part of a company's operations.

The increasing sophistication of the attacks makes implementing automated methods in certain areas challenging. The recent onset of Generative AI technology increased the volume and complexity of phishing attacks as they benefit heavily from automation [35]. AI-powered automation systems are finding their way into the cybersecurity industry; however, human operators are still vulnerable to phishing attacks, and their increased volume and success rate make it crucial to prevent phishing attacks from reaching human operators altogether. This study considers domain analysis as one field that could benefit from automation.

2.4 Human factor in domain squatting

When researching domain squatting, a distinction should be made between *active* and *passive* user roles when interacting with a domain. User role is defined as active when the user manually enters the domain name in the browser and, therefore, might misremember the domain name or make a typo. In contrast, the passive user role implies that the user does not manually enter the domain name; instead, the user interacts with a prepared domain name link crafted by another party. In other words, the user copies the existing domain link or clicks on an existing link. When considering phishing, the user typically assumes a passive, i.e., receiving role. As this research primarily focuses on the phishing case of squatted domains, the user typos lie outside this scope.

Phishing is a special case of the application of squatted domains. Threat actors utilizing phishing aim to use domains that are difficult to distinguish from legitimate ones. In other words, a user must not be able to spot the difference easily between a legitimate domain and a squatted one. This effect can be achieved by applying techniques such as:

- Insertion of an extra character inside a domain name, with several special cases:
  - Added hyphenation (e.g., compare example[.]com vs. ex-ample[.]com)
  - Repetition of a character inside a domain name (e.g., compare hello[.]com vs. helllo[.]com)
  - Subdomain - when the domain name is "broken" by a dot in the middle of a domain name, resulting in an added subdomain (e.g., compare hello[.]com and h[.]ello[.]com)
- Addition of a character at the end of a domain name, with the plural being treated as a special case (e.g., step[.]com versus steps[.]com)
- Transposition - when the two consecutive characters in a domain name are swapped (e.g., compare example[.]com vs. exmaple[.]com)
- Changed order - when a character of the domain name is misplaced into a different position (e.g., compare example[.]com vs. exalmpe[.]com)
- Wrong TLD - using the same hostname with a different top-level domain (e.g., compare example[.]com vs. example[.]net)
- Omitting a TLD or moving the TLD inside a domain name while using a different TLD (e.g., compare hello[.]it and hello-it[.]com)
- Added TLD - using an extra top-level domain (e.g., compare example[.]co and example[.]co[.]uk)

- Wrong second-level domain - in the case where the sensible part of the domain name is located on a subdomain, the second-level domain might be substituted with a different one (e.g., compare onetwo[.]at[.]it and onetwo[.]it[.]it)
- Omission of a character from a legitimate domain name, with dot omitting being treated as a special case of omission (e.g., compare www[.]example[.]com vs. wwwexample[.]com)
- Dictionary, or combosquatting - concatenating the original domain name or sensible part of it with another word (e.g., google[.]com vs. googlesupports[.]com)
- Replacement of a character with another within the domain name. There are special classes of replacement techniques:
  - Homoglyph - replacing a character with a visually similar one, thus representing a higher chance of a human eye missing a typo (e.g., compare go0gle[.]com vs. google[.]com, where "o" is replaced by "0")
  - Vowel swap - when one or more vowels in the domain name are swapped with different one(s) (e.g., compare example[.]com vs. exomple[.]com)
  - Interchanging dots and dashes, often with an addition of extra TLD (e.g., compare mydomain[.]com vs. my-domain[.]com[.]net)
  - Bit squatting - a modification that exploits possible bit-level errors in memory or network data. These errors can emerge due to hardware issues or data transmission errors. Attackers register domain names that contain characters that might appear due to such bit flips (e.g., compare google[.]com vs. gootle[.]com)

These techniques can be applied separately or in combination. However, the smaller the manipulation, the harder it is for a user to notice the change, arguably. Additionally, when considering single-character manipulations, the position of a changed character and the length of a domain name play an essential role in the success of a domain impersonation. Consider the examples in the table 2.1.

Legitimate domain	**Fake domain**
instagram[.]com	knstagram[.]com
neteye[.]guide	net-eye[.]guide
fivefingerdeathpunch[.]com	fivefinqerdeathpunch[.]com

Table 2.1: Single-character manipulation examples

Spotting the manipulations in the fake domains is not always trivial, especially if the user does not remember how the domain name is written. In the example of the neteye[.]guide, the user might be misled by the extra hyphen in the fake domain if they do not remember the exact spelling of the domain name since the sensible part of the domain name does not change by adding a hyphen. For a longer domain name with homoglyph (visually similar) character replacement, the user has an even smaller chance of spotting the manipulation.

From a phishing perspective, homographic manipulations in the middle of a long domain name are among the most dangerous. The reason lies in the particularities of human perception of a written text.

Multiple studies on reading comprehension demonstrate that the human brain does not read text word by word, let alone letter by letter. Researchers [5] found a striking difference in the reading speed depending on how predictable the word or sentence is. Higher predictability and commonality lead to higher reading speed and improved reading efficiency. Another study [41] argues that the word recognition process is performed by splitting the words into morphemes rather than individual letters when it comes to complex words. Authors of the framework for language comprehension [20] point out that, based on measured brain activity during reading, more attention is paid to the beginning of the phrase or a sentence as it helps to activate the predicting mechanism.

Another study [12] supports these findings by investigating so-called *transposed word effect*, when readers fail to notice grammatical errors in the sentences where two words are switched. Their findings show that the failure to notice these errors is likely caused by readers' reliance on prior knowledge and speedy inference of the intended meaning of the message.

While these studies differ in their approaches, a common conclusion is that the human brain invokes a prediction mechanism during reading based on a limited input of words as a whole or morphemes as parts of words rather than individual letters. This is why it can be argued that modifying a single character at the beginning of the domain name can be far less effective than changing a character in the middle. This is why the position of the character modification is to be considered as a feature in this research.

When it comes to phishing, it can be assumed that the threat actors would try to impersonate the domains that the users *expect* to see, such as belonging to their organization or a partner organization, service, etc., therefore the domain name can be considered as moderately or highly predictable for the user, increasing a chance of missing a modification in the domain name.

3 Related work

Numerous studies on DNS and DNS vulnerabilities have proposed solutions that address DNS vulnerabilities, including the ones perpetuating the phishing problem. Researchers [16] identify multiple areas in the DNS threat landscape that directly or indirectly link to the utilization of specific domain names or domain name characteristics. Mainly, the domain names are used for brand impersonation or traffic diversion with the goal of monetization [16]. Phishing is one of the broadest fields heavily relying on spoofed or squatted domain names. Overall, squatted domain names can potentially be involved in the following scenarios:

- Phishing attacks. When attackers register a domain that is similar to a legitimate domain and host a fraudulent website on this domain, they can attempt to trick users into revealing sensitive information such as credentials, PII, or financial data [40][2].

- Distribution of malware. Squatted domains can also be used as a delivery mechanism for malware. Once a user visits the malicious domain, the website may automatically attempt to exploit vulnerabilities in the user's browser or operating system to install malware or trick users into downloading malicious software [16].

- Business email compromise (BEC). Squatted domains can be utilized in BEC attacks, where attackers register domains that closely resemble a company's official domain. The attackers can then impersonate company executives or employees to trick others into making unauthorized financial transfers, providing sensitive information, or granting access to internal systems [1]. For example, a domain squatter might register a domain like "example-com[.]com" instead of "example[.]com" and send emails from this domain to deceive recipients.

- Traffic diversion and advertising fraud. Squatted domains may also be used to divert traffic from legitimate websites. It is commonly done to generate ad revenue through pay-per-click schemes, where users are redirected to websites that display ads, and the attackers earn money from each click. In other cases, the goal might be to redirect users to competitor websites or fraudulent e-commerce sites. [40].

- Domain resale or ransom. Attackers may register domain names to sell them back to the rightful owner at a higher price. This is especially common with domains similar to well-known brands or businesses. In some cases, attackers might attempt to extort companies by threatening to use the domain for malicious purposes unless a ransom is paid. There are several known cases of such use of a domain; however, the poor intent needs to be proven before making a decision [3].

- Brand damage or reputation attacks. Squatted domains can be used to damage the reputation of a company or brand. Attackers may host defamatory content or misleading information on the squatted domain to harm the company's public image [3]. Sometimes, the squatted domain may host counterfeit products or services, further damaging the brand's credibility.

Researchers have been proposing different solutions to simplify and enhance domain name analysis. The solutions can be clustered into proactive and reactive approaches.

The *reactive* approach suggests techniques for analyzing the traffic between the target domain and the client (benign) host, which assumes the presence of a communication attempt from either the client host or the domain. The works encompassing this approach primarily focus on identifying malicious domains used as command and control servers, spreading malware, or belonging to the threat actors identified as APTs (Advanced Persistent Threats). In such scenarios, when the communication between the client and the malicious domain has already occurred, a domain name serves as an Indicator of Compromise (IoC) in terms of threat intelligence. It can be used to pivot, i.e., discover

more information about the attacker and their infrastructure. This approach largely involves traffic analysis and features obtained from the communications between the client and the malicious domain.

Another group of researchers focuses on identifying malicious domains *before* any attempted communication between the benign client and a suspicious domain, representing the *proactive* approach. This is the same use case for domain analysis as for the company focused on in this research. For this study, the related works of this group are divided into two categories: works focusing on identifying and preventing domain squatting attacks and works focusing on identifying malicious domains as a general category. These categories differ in the applied identification and classification methods. Additionally, in the works applying machine learning-based approaches, the features contributing to the final decision-making differ based on the goal of a study.

3.1 Identification of malicious domains

Researchers have long been interested in finding effective measures to protect legitimate organizations from various attacks, including measures based on domain analysis and classification. While domain name analysis has traditionally been performed manually, the growing volume and rate of phishing attacks and the high fluidity of DNS infrastructure call for more efficient implementations, fully or partially based on automated methods. The high volume and rate of phishing attacks and spam communications require effective countermeasures as early as possible, meaning these communications would be stopped long before they reach a human operator. Traditional approaches to automated domain classification include domain blacklisting and IP blacklisting.

Blacklisting refers to the process of adding specific domains or IP addresses to a global list that marks them as untrustworthy or malicious. When a domain or IP is blacklisted, it means that it has been associated with suspicious activities, such as sending spam, hosting malware, or being part of phishing attacks. As a result, services like email providers, browsers, and security systems may block access to these blacklisted domains or IPs to protect users from potential harm. Reputable organizations and entities support several widely known domain blacklists. Examples of such organizations are *MXToolBox* [28] and *DNSChecker*[15].

While domain blacklisting protects customers from known malicious domains, it does not account for the domains that exploit the domain squatting techniques. Domain blacklisting does not account for variations in the domain names and, as a static countermeasure, does not keep up with the emergence of new domains. Blacklisting IP addresses instead of domain names is even less efficient as the domains can change the hosting server without significant effort.

Techniques like IP-Fast, Domain-Flux, and advanced Double-Flux evade these conventional detection approaches. Flux techniques refer to switching between multiple IP addresses or domain names to avoid blocking and ensure continuous operation of the attacker's infrastructure [37][38]. This technique is widely applied to domains that serve as Command-and-Control (C&C) servers in the malicious infrastructure. The IP addresses typically belong to the bots in the botnet infrastructure controlled by the malicious actor. Given the low effort needed to create a fast flux network, IP and domain blacklisting has become a severely outdated approach [44]. Figure 3.1 reflects the fast-flux technique's fundamental principle.

Another shortcoming of the blacklisting approach is related to DGA domains. DGA stands for Domain Generation Algorithm, meaning that these domain names are not composed manually but are generated in an automated fashion. These domains are mainly used for malicious purposes, such as C&C servers of an attacker's infrastructure. Researchers[19] found that only 1.2% of DGA domains are on blacklists. The automated nature of generating such domains allows to generate new domains in a very short period, whereas identifying and blacklisting such domains is not a trivial task. This suggests that traditional malicious domain detection methods, such as firewall blocking and using blacklists, are largely ineffective against more flexible domain attack techniques.

Finally, maintaining blacklists is a challenging task due to the need for constant updates to reflect the latest changes in domain registrations and ownership. The dynamic nature of DNS can lead to legitimate domains being blacklisted by mistake (false positives), or malicious domains escaping detection (false negatives), complicating security efforts [19]. New domain registrations happen daily, adding to the globally stored DNS records and necessitating constant updates. Domains often change

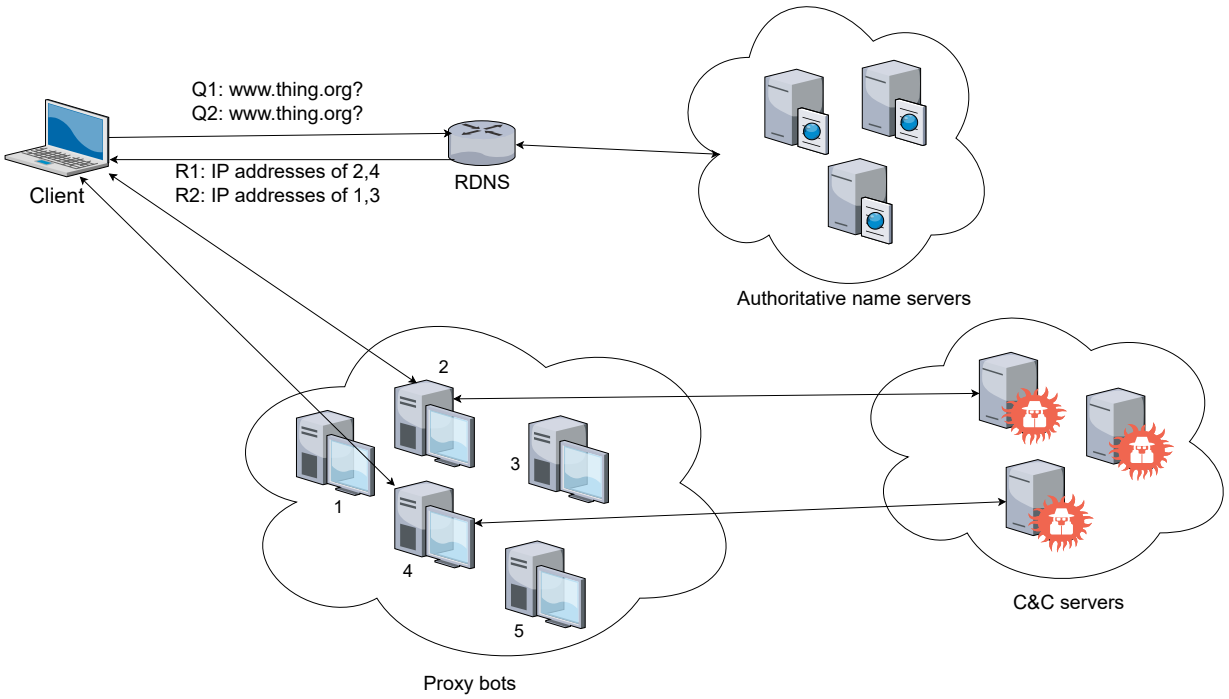

Figure 3.1: The basic principle of fast-flux technique.

ownership due to business transactions or expired registrations, each change requiring updates in DNS records and potentially in blacklists. The limited number of IPv4 addresses further complicates this scenario, as these addresses are frequently reassigned by ISPs (Internet Service Providers), causing frequent changes in DNS mappings. Once legitimate domains are hijacked or sold, they can be used for malicious activities like phishing, spreading malware, or hosting fraudulent websites.

These complexities in managing DNS highlight the need for more advanced and adaptive security measures to ensure the effectiveness of blacklists, as well as the need for measures beyond blacklists.

Another study [39] proposed graph-based identification that employs graph theory to discover associations between IP addresses, domain names, hosting providers, and registrars. In this approach, pieces of information (an IP address, a domain name, etc.) are represented by graph nodes, and their relationships are defined by graph edges. When combined with novel technologies such as machine learning techniques for domain classification [39], this approach can provide reasonable results on malicious domain identification. However, there are several drawbacks to such an approach. First, graph-based identification implies an association-based approach, meaning associations and relationships must exist with other domains. Newly registered domains with few relations with others might be identified by such a system with less certainty as the identification would be performed based on weaker features [39]. Second, the associations and relationships must be meaningful to provide valuable features for domain identification and classification. Depending on the sources of information, coincidental relationships (such as malicious and benign domains residing on the same hosting provider) could interfere with identification accuracy.

Researchers [32] also attempted to use lexical, DNS-based, and web-based domain features to train a logistic regression ML model for domain classification. Using web-based features related to web traffic, content category, and time spent by website visitors brought novelty to the field; previously, DNS-based data was the primary source of information. However, the study only used accuracy as a measurement metric, which does not provide a comprehensive insight into the model's performance. Additionally, the proportions of the benign and malicious domains in the dataset are unknown. The study reported 60% accuracy, which leaves potential for improvement.

3.2 Identification of squatted domains

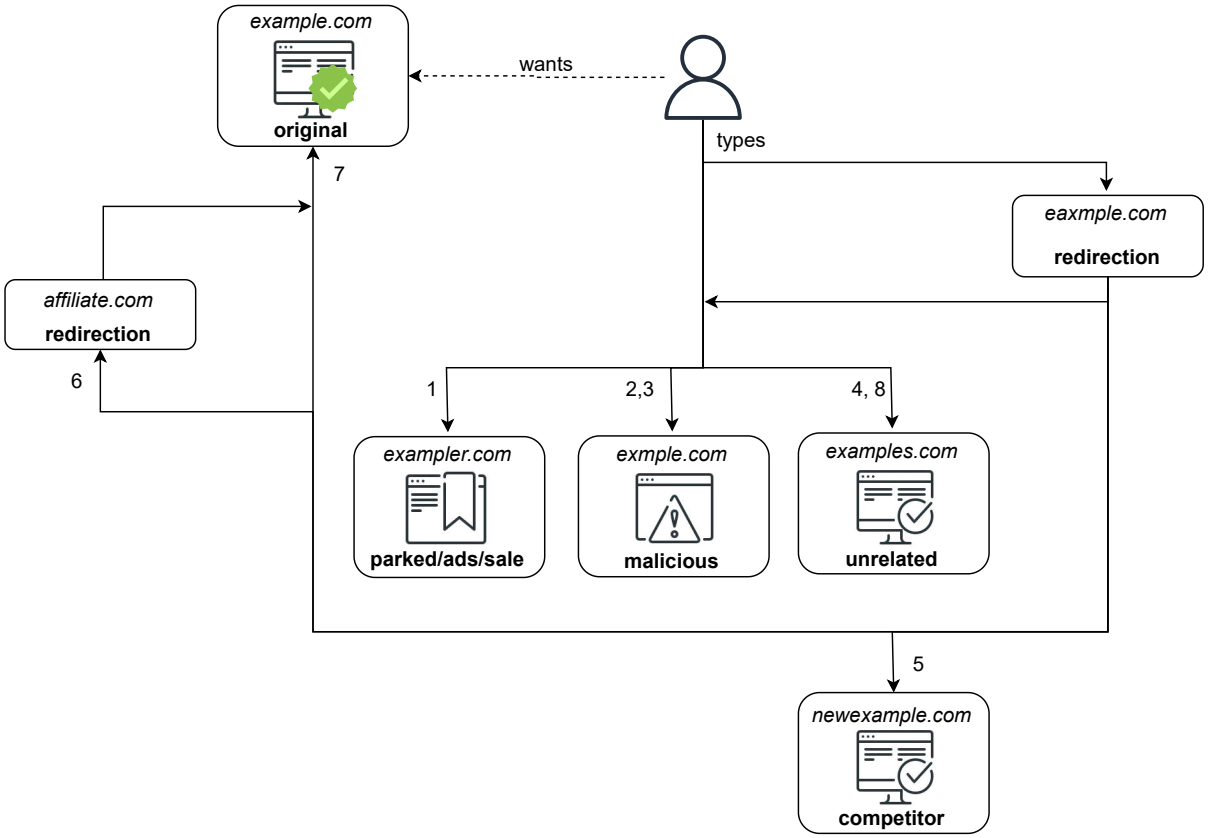

Figure 3.2: The typosquatting ecosystem.

To assess the scale of domain squatting issue, researchers [40] attempted to identify existing squatted domains, particularly the ones used for malicious purposes. Their methodology consists of several consecutive steps. First, the dataset of benign domains is obtained from Alexa's top 1 million sites. Then, this dataset is used to generate the possible variations of the benign domains, using common operations such as character addition, deletion, substitution, etc., to obtain a set of potential squatted domains. The existence of these domains is verified using additional information from DNS records, WHOIS records, and web crawlers. Finally, this information categorizes the domains according to their monetization strategy. The monetization strategy 3.2 can be divided into the following categories:

- 1) Serving third-party advertisements
- 2) Impersonating legitimate domain with the purpose of phishing
- 3) Serving malware
- 4) Redirection to another (landing) domain
- 5) Redirecting to competitor domains
- 6) Affiliate marketing
- 7) Defensive registrations by the domain owner, redirecting to the target domain. The study [40] indicates that less than 2% of typo-ed domains belong to this category.
- 8) Unrelated content, typo is coincidental

Additionally, these domains are checked against known blacklists to discover the domains that are referred to as "true malicious" domains.

The study [40] has discovered that most squatted domains do not appear to have a malicious purpose; instead, these domains are used for various monetization strategies. However, this type of monetization is not legal as it exploits the legitimate name of a company or a brand. Additionally, the study has identified typosquatting domains' use in malicious campaigns for quiz scams, spam survey sites, offering deceptive downloads or serving adult content, and other illegitimate purposes. The researchers also argue that the extent of typosquatting varies as it depends on the popularity of the domain, i.e., there would be more typosquatting attempts (typosquatted domain names) for a domain such as facebook[.]com compared to someunknowndomain[.]com.

While the researchers managed to develop a domain classification tool that produced reasonable results, their research encompassed only domains within the [.]com TLD (top-level domain), focusing on the domain names obtained through single-character modifications (addition, deletion, substitution, transposition). However, domain squatting can be performed using other methods, such as replacing a top-level domain or performing more than one single-character modification. Additionally, the authors acknowledge the difficulties in the definitive identification of squatted domains since the linguistic properties of the domain alone are not always enough to reliably identify the purpose of the domain name registration.

Another study [27] focused on implementing a machine-learning approach to the issue of domain squatting. The authors propose an ensemble learning classifier composed of five classification algorithms for detecting suspicious domains. One of the methodologies considered in this study outlines the features of the domain name that are not related to the features derived from traffic analysis. These features include, for instance, the percentage of numerical characters within the domain name, the number of unique characters in the domain name, the length of a domain name, and registrar information. The authors used a supervised machine learning-based classifier to detect malicious/suspicious domains. They selected decision trees (C4.5), K-nearest neighbors (K-NN), logistic regression (LR), Naive Bayesian (NB), and Support Vector Machines (SVM) to combine into a majority-voting ensemble learning classifier. This classifier was trained on a dataset consisting of legitimate domains as benign and DGA domains as malicious samples. The study identified the differences in linguistic features between legitimate and DGA domains, allowing the classifier to perform well on the unlabeled dataset. However, the typosquatted domains, by their lexical composition, are usually close to legitimate domains (for instance, via single-character modification). Therefore, typosquatted domains are highly likely to have features nearly identical to those of legitimate domains, unlike DGA domains.

3.3 Machine learning in domain analysis

Machine learning has been applied in multiple use cases related to domain analysis, including attempts to detect typosquatting. Some researchers use machine learning to identify the DGA (algorithmically generated) domains that benign hosts might communicate with [44]. The reasoning behind this use case is the vast usage of DGA domains as C&C servers in the malicious infrastructure. Therefore, identifying these domains can help prevent bot infections and other attacks. In identifying DGA domains, linguistic features of the domain names are prioritized. DGA domain names have different properties than usual: they have little to no meaningful strings and are a product of a pseudo-random generator [27]. Automatic generation provides a quick way to register new domains. Through quick registration, the C&C servers in malicious infrastructure can be replaced with low effort in case of domain takedown or blocking. By identifying distinctions in features between legitimate and DGA domains, researchers [27] achieved reasonable results.

Another group of researchers focuses on identifying malicious domains based on the temporal and spatial features of the traffic between the benign and malicious hosts. This methodology is applied in the context of APTs (Advanced Persistent Threats), established criminal groups with known techniques, tactics, and procedures, referred to as TTP. These techniques and tactics have been linked to DNS exfiltration [25] or C&C communications. The authors argue that the behavior of DNS domains is persistent across the attacker's infrastructure and that the domains belonging to the same strain of malware will display identical or nearly identical behavior. Therefore, identifying

malicious domains based on their behavior can lead to detecting attacks from known APTs. However, domain name analysis is less significant in this use case as the approach relies heavily on network traffic analysis.

The most straightforward approach to malicious domain detection via domain name analysis is demonstrated by several studies [27][32][45] focusing on the features of the domain name itself, with extra information coming from the WHOIS' records and IP information. However, identifying contributing features is a tedious task, and the result largely varies depending on the purpose of the study. One study [27] evaluates the contribution of various features to the final decision of the domain classification and highlights the correlation between the features and the domain classes. However, the study only presents a limited set of features and their correlation to the final result.

Several researchers pinpoint the challenges in applying machine learning techniques to the domain name analysis procedure. One study [37] identified a challenge of obtaining the ground truth dataset that could be used for tuning the machine learning algorithms. This challenge arises mainly from the fluidity of the DNS infrastructure and the constant changes happening within. While blacklists and whitelists could serve as ground truth data, it is hard to verify their content and origin, as well as their relevance. Reputation lists can provide a certain degree of insight; however, reputation is irrelevant if the domain is blacklisted for reasons other than ones related to network security. For instance, a domain can host unethical or questionable content, yet from the network security perspective, it is not associated with malicious activity. Another study [40] found it challenging to obtain a labeled dataset even through manual inspection as the purpose of any given domain name registration cannot be established reliably in all cases, mainly due to missing or obscured information.

3.3.1 Feature engineering

This section summarizes and categorizes the features proposed by different studies. This summary further serves as a foundation for feature engineering for the solution used in this paper. As features contribute differently to the outcome, the usefulness of the features needs to be evaluated according to the algorithm's goal. The table 3.1 summarizes the proposed features across related studies. The features are divided into four categories according to their class:

- Lexical features. These features are retrieved from the domain name itself and encompass linguistic features of the domain used for textual analysis.
- Web-based features. When a domain has an associated website, it is possible to determine characteristics relevant to the associated webpage and its traffic.
- IP- and DNS-based features. These features are retrieved from the WHOIS database and include information about the registration of the domain and associated hosts and IP addresses.
- Other features.

To provide a better overview, similar features are grouped in the same row (e.g., "Number of distinct IP addresses" and "IP address associated with the domain" are grouped as they contain the same or very similar information).

Table 3.1: Proposed features

Feature	**Category**	**Studies**
Length of Domain Name	Lexical	[27] [22] [40]
Number of Unique Characters/Letters/(unique) numbers	Lexical	[27] [32]
Ratio of Letters/Numbers to Domain Length	Lexical	[27] [22] [6]
Ratio of Unique Letters/Unique Numbers to Unique Characters	Lexical	[27]
The ratio of consonant characters/special characters/vowel characters in the domain	Lexical	[22]

Feature	Category	Studies
Presence of strange characters (non-alphanumeric)	Lexical	[22] [32]
The maximum number of consecutive consonants/vowels/numerics/special chars in the domain	Lexical	[22]
Baseline DNS used to enrich data (derive features)	Lexical	[22]
DNS record type queried	DNS	[22]
The response from a DNS request for the record type MX/TXT	DNS	[22]
PTR record: reverse lookup, determine domain from IP	DNS	[32]
If the DNS response has Sender Policy Framework attribute	DNS	[22]
If the DNS response has Domain Keys Identified Email attribute	DNS	[22]
If the DNS response has Domain-Based Message Authentication	DNS	[22]
The IP for the domain / Distinct IP addresses	DNS	[22] [32] [6]
If the domain is registered in the Alexa DB / Alexa rank	DNS	[22] [32] [40]
If the domain is available for common ports*	DNS	[22]
The country code associated with the IP(s) of the domain/IP Geolocation	DNS	[22] [32] [6]
The country code defined in the domain registration process (WHOIS)	DNS	[22]
The creation/expiry date/last update of the domain (WHOIS)	DNS	[22] [32] [40]
The Autonomous System Number for the domain	IP	[22] [32]
The HTTP/HTTPS response code for the domain	Web	[22]
The organization name associated with the domain (WHOIS)	DNS	[22] [32]
Registrar: name and contact details	DNS	[32]
The number of sub-domains for the domain	DNS	[22]
The Top Level Domain for the domain	DNS	[22]
The result of the blocklisted search for the IP/domain	Web	[22]
Number of webpages	Web	[32]
Time spent by visitor	Web	[32]
Web referrals	Web	[32]
Web traffic	Web	[32]
Content category	Web	[32]
Parked / serving ads / Affiliate marketing	Web	[40]
Number of redirections	Web	[40]
Illegitimate contents: list of illegitimate words	Web	[32]
Short life of the domain	DNS	[6]
Daily similarity in request patterns	DNS	[6]
The class of the domain (malicious/non-malicious)	Other	[22]
The Shannon Entropy of the domain name	Other	[22]
The mean value of the entropy for the sub-domains	Other	[22]
End of Table		

*common ports: 80, 443, 21, 22, 23, 25, 53, 110, 143, 161, 445, 465, 587, 993, 995, 3306, 3389, 7547, 8080, 8888

3.4 Research novelty

While most studies focus on identifying malicious domains as a general category, the current research aims to answer the following questions:

- Can the machine learning approach reduce the workload for the analysts by excluding a high number of false positives, leaving only suspicious domains for investigation?
- Given pairs of legitimate and suspicious domains, is it possible to clearly distinguish between the domains with coincidental similarity and the domains that aim to exploit domain squatting techniques?
- Is the implementation of such an algorithm feasible in a real-world scenario where imperfect samples might lack essential information?

Having a dataset of real-world samples brings the research closer to the industry; however, it also means that the samples are not "perfect," as some features would be impossible to extract due to the missing information. The final model would need to maintain high performance in these conditions. Additionally, as new samples are discovered regularly, the guidelines should be placed on regular retraining and updating the model, as new samples might significantly differ from the ones found previously. This is especially important as the new legitimate domains are added to the dataset since different techniques might be employed to spoof these domains (e.g., long vs. short domain names or domains containing general language words vs. domains containing rare combinations of letters and words).

4 Research context

This chapter describes the operation of the target company's threat intelligence platform. The domain analysis is performed through this platform, and the current practices and policies provide the foundation for the automated solution in the domain analysis module. In the context of this research, domain analysis is one of the services offered to cybersecurity company customers. Analyzing domain names is one of the crucial tasks of cybersecurity analysts in the Security Operations Center (SOC), and it is part of their daily work routine.

4.1 Existing platform

The company has an automated threat intelligence platform that serves as an information hub for the operators. The data from different sources is automatically collected, aggregated, filtered, and visualized for further processing by analysts. The findings provided by the platform are referred to as evidence. Some evidence is gathered automatically to speed up the investigation process.

An alert is raised if a domain corresponding to the squatting criteria is discovered. In the given study, the requirements regarding typosquatting are set by software-based tools that automatically recognize domains that differ insignificantly from the original.

The alerts are produced in batches following the workflow described in figure 4.1. First, the scheduled research (information gathering) is triggered in the background. This research takes the client domain name as an input (e.g., *newclient[.]com* and the variations of this domain name produced by the automated tools such as *DNSTwist* (e.g., *newcliient[.]com*, *niewclient[.]com*, etc.). With this input, the DNS records are queried for the existence of those domains, i.e., if there are registration records associated with these domain names. If such a domain name is found, the following checks are performed automatically:

- The domain name already was handled by an analyst (or is in the queue to be handled). If no previous alerts are associated with the domain name in question, an alert is raised to the analyst.
- If there was an alert in the past, the outcome of the alert resolution is checked. If the alert was handled as other than a false positive (meaning an action was likely taken against this domain name by a client), the domain name is dismissed from further checks.
- If the alert was handled as a false positive, the new data of the domain is compared to the previously seen data. If there is a change in the information (e.g., a domain has changed ownership), an alert is raised. A new alert will also be issued to recheck the information if the domain name was handled over six months ago.

The process above is performed for each domain name variation the research handled. This approach eliminates a significant amount of manual work by providing some information associated with domain names and filtering out the domain names that do not have associated DNS records (i.e., are not registered and, therefore, do not present any risk at the moment of their discovery). The following information is gathered automatically for the alerts and stored in the database:

- Fully Qualified Domain Name - the domain name that needs to be investigated
- Type of modification required to transform the customer domain name into the discovered domain name
- Indication of whether the domain is marked as a phishing domain by reputable organizations
- SHA256 hash of the page content

- MX record related to the domain
- Indication of whether the IP address of an A record of the domain is found in known blacklists
- Indication of whether the IP address of an MX record of the domain is found in known blacklists
- Indication of whether the host is found in known blacklists
- The parking man rating - the rating given by the tool evaluating the likelihood of the domain being parked.
- The date of the last update of the DNS record
- Full WHOIS information in raw format if available

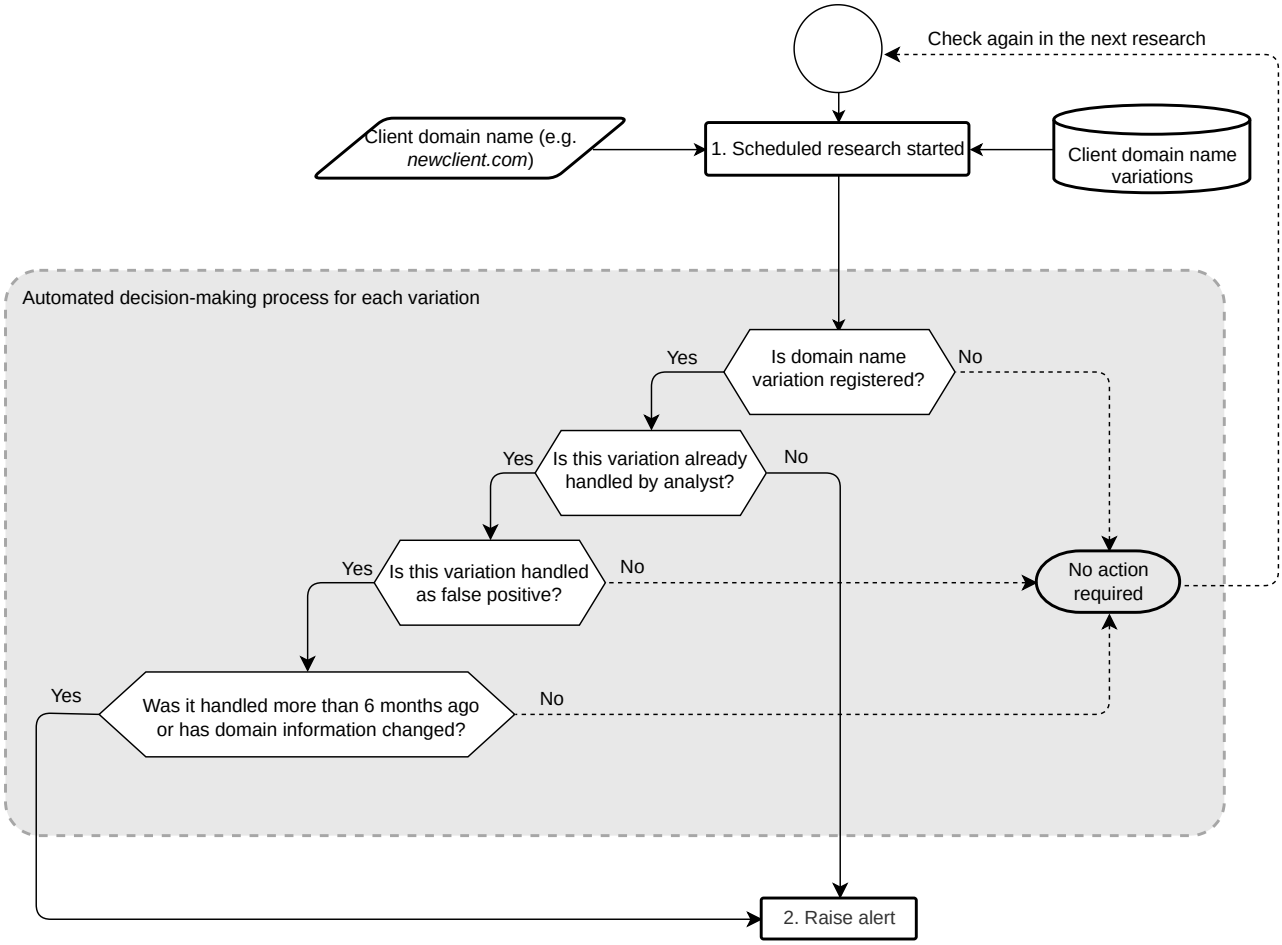

Figure 4.1: A high-level overview of the alert creation decision-making process.

Cyber security analysts must manually verify whether the discovered domain presents a (potential) risk for the customer. During this process, analysts aim to discover additional information about the domain by answering the following questions:

- Is the domain similarity coincidental, i.e., the flagged domain belongs to a legitimate organization/individual?
- Is the domain parked or for sale?
- Has the domain been associated with known malicious activity?
- Is the domain name associated with a known malicious IP address?

However, this task is often resource- and time-consuming as domain analysis requires manual information gathering and analysis.

Most alerts are classified as false positives (i.e., the domains in question are benign), and only a handful of flagged domains require further investigation and monitoring. With many false positives (around 80% in the given scenario, as the dataset analysis will show), analysts are prone to alert fatigue, which might result in missing suspicious or malicious activity. Additionally, the limited time and resources required to investigate every domain, including benign domains, leave less time for detailed investigation of suspicious domains. As the company grows continuously, and new customers are constantly being added to the platform, the number of alerts grows, and analysts face the challenge of allocating sufficient time and resources for the domain name investigation. This problem created the need for an automated or semi-automated solution to reduce the workload of the analysts in the context of domain name analysis.

4.2 Domain analysis process

The domain analysis process is a manual task that involves several checks (at the minimum) performed by the operator. Ultimately, the operator decides whether to escalate the request to the client's attention if the domain appears suspicious and might present a risk to a client. The client can take action against a domain if it is considered reasonable. An example of client action is submitting a takedown request to the hosting provider if the domain is deemed illegitimate and exploiting the client's brand name (such as impersonating the client's website). However, the client's decision-making process is outside the scope of this work.

The platform produces the findings as a result of a scheduled run of various tools for domain discovery. The tools used to discover the domains and subdomains are *DNSRecon*[9] and *DNSTwist* [10]. Figure 4.2 provides a high-level overview of the analysis process and its outcomes:

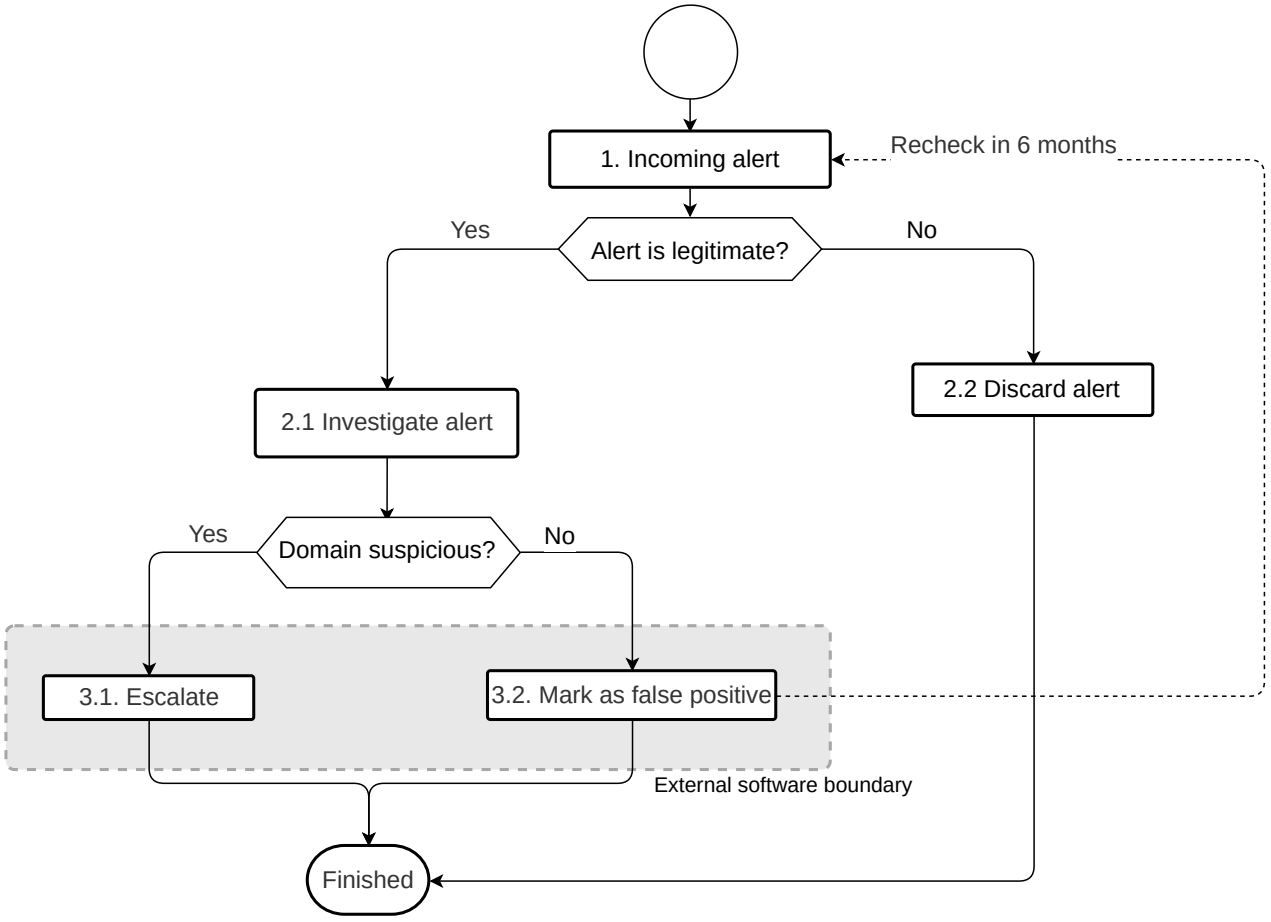

Figure 4.2: Domain link analysis process.

- The process begins when an alert is received and requires human evaluation to determine its legitimacy (1). The operator reviews the findings gathered after a platform run to discard irrelevant findings, referred to as "evidence" in the context of the platform operation. The basis for this decision at this stage is solely the domain name. The evidence is discarded if the domain name is considered irrelevant (2.2), meaning that the domain name does not represent a possible risk to a customer. If the domain name cannot be discarded with high confidence at this stage, the evidence is accepted to be processed at the next stage. The domain name is deemed irrelevant if it is, by the operator's judgment, too different from the customer's domain name and, therefore, has very little chance of impersonating it. The most common false alerts are generated by the client domain that uses a general word or a combination of general words. In such a situation, it is more likely to encounter domains that use the exact words or phrases. For instance, a fictitious domain such as *stop[.]com* can generate false positives such as *marvelou**stop**ic[.]com* where the domain name coincidentally can be found inside the string, and the domain is unlikely to try to impersonate the customer.

- The alert is passed to an analyst who conducts a thorough investigation (2.1). This involves checking various data points related to the alert, such as the domain registration details, web content, and other relevant indicators. The analyst determines whether the alert is legitimate or a false positive based on the information available at the time. The analysts evaluate the domains following practices developed inside the cybersecurity team. These practices consist of various manual verification steps using publicly accessible reputable tools, such as *VirusTotal*[43]. The following attributes of the domain are evaluated:

  - Domain lexical similarity. Some domains may appear similar due to coincidences, particularly if the domain name is generic or contains words from (typically) the English language. For instance, the similarity of the domain *datagroup[.]com* to the domain *dategroup[.]com* is less suspicious than the similarity of the domain *wvrth[.]com* to *wurth[.]com* as the first pair of domains contains generic words from the English language. In contrast, the second pair of domains includes a rare combination of vowels and consonants.
  - Domain configuration. An analyst can use tools such as *dig* to determine if the domain has associated DNS records, particularly A, MX, NS, and TXT records. These findings help define the domain's use and pivot to the subsequent investigation stage.
  - Hosting and web features. If the domain name has an associated IP address, the website's content can be checked to determine the use of the domain name.
  - WHOIS information. The registrant and registrar information contributes to the domain's evaluation and registration date.
  - Malicious indicators. Both the domain name and (if applicable) associated IP address are checked against spam blacklists evaluated in VirusTotal. However, in the case of shared hosting with many domains, the IP evaluation does not necessarily contribute to the overall domain evaluation.

  After these initial steps, an additional investigation might be performed if necessary. For instance, based on *WHOIS*[34] registrant data, a further check can be performed to verify the existence of the given organization/individual. Another common situation is the "parked" domains – when the domain name is registered, and a website is hosted but is not actively used or developed, typically directing the website visitor to a placeholder page or a page with advertisements. In most cases, parked domains are flagged as false positives by analysts.

- If the analyst deems the domain or alert suspicious, it is escalated to the client (3.1) by creating a ticket. This ticket is a formal notification that further action may be necessary on the client's part, such as blocking the domain or investigating the associated activities. An analyst can manually assign a level of urgency (lowest, low, medium, high, highest) to the ticket. The client's decision-making is excluded from the diagram.

- If the analyst concludes that the alert is not a risk, it is marked as a false positive. However, the process does not end here. In cases of false positives, the domain is flagged for rechecking in the future. This step is crucial to account for the possibility that the status or information related to the domain may change over time. During this recheck, the analyst revisits the domain to see if any new information has emerged that would alter the initial judgment. The goal is to ensure that previously dismissed alerts are safe or to escalate them if new evidence of suspicious activity arises.

Instead of a traditional classification of benign and malicious, this research uses the terms "suspicious" and "non-suspicious" for the domains. Non-suspicious domains are marked as false positives, and "suspicious" are escalated to the customer, according to Figure 4.2. First, the operators are not required to establish whether the domain is generally malicious. The business need dictates the requirement to gather sufficient basis to classify the domain as suspicious or non-suspicious concerning a particular customer. Therefore, even if the domain is classified as non-suspicious, some malicious activity can still be associated with such domain. Vice versa, a suspicious domain does not always represent malicious activity. Labeling a domain as suspicious merely indicates that additional checks must be performed to verify the risk status of the domain. Finally, the goal of the classification is to reduce the number of false positives on the platform. Thus, instead of focusing on identifying malicious domains with high confidence, the aim is to identify and discard non-suspicious domains so that analysts can only investigate the domains deemed suspicious.

However, it is important to consider that the reduction in false positives must not be achieved by introducing significant increase in the number of false negatives. In the given scenario, minimizing false negatives remains more crucial because missing a critical event could have severe consequences, while false positives, though undesirable, are more acceptable and manageable.

The following assumptions are made about suspicious and non-suspicious domains:

- **Non-suspicious domain**: non-suspicious domains can be classified based on specific factors, such as belonging to a legitimate individual/organization. However, the absence of specific malicious features is the most critical factor. The reasoning for this definition is that malicious domains often masquerade as legitimate domains and possess some of the same features as benign domains.

- **Suspicious domain**: suspicious domains are classified based on detecting certain malicious features. The malicious features are assigned different levels of importance. For instance, while a very recent (few days) domain registration date increases the level of suspiciousness, it does not alone indicate that the domain is malicious. However, if the domain is present in a known blacklist or if the website residing on the domain appears to be masquerading as a customer website while not being owned by the customer, an analyst will report this domain is malicious with high confidence.

  A particular case of suspicious domains is domains belonging to the customer but not monitored on the platform. Since these domains typically have features very similar to monitored domains, the system should flag these domains as suspicious for the possibility of impersonation. The most reliable way to establish the ownership of such domains is to escalate the ticket to the customer and inquire about the verification.

4.3 Solution challenges

Several challenges are associated with automating the domain analysis process in the given scenario. Some of these challenges can be tackled on a technical level, while some can be mitigated by setting clear guidelines for the domain analysis process to ensure consistent results. In the current research, these challenges are accepted as limitations that could be overcome in the future.

Definition of risk. One of the main challenges associated with the solution for the given use case is the definition of risk, which is tied closely to the knowledge of the client-owned domains. In other words, the domain is not classified as benign or malicious; it is classified as non-suspicious and suspicious, where suspicious does not always equal malicious. In fact, in most cases, the operators do

not possess sufficient time and resources to research all possible details about the domain to classify it as malicious, and there is no business need for this activity either. The risk for the customer is present if the domain:

- could be used to impersonate the webpage of the customer-owned domain
- could be used to attempt a phishing campaign on the customer
- could be used to attempt a phishing campaign on the customer organization's clients
- could be used to tarnish or exploit the brand reputation
- could be used to divert traffic from the original domain

Missing data. Some features are missing on a larger portion of the domain dataset. Additionally, while performing assessments, analysts do not solely rely on the information collected by the automated tools. Instead, they perform extra open-source intelligence gathering, and this process can be too sophisticated to mimic in a fully automated manner. This reliance on manual intelligence gathering introduces an additional layer of complexity, making it challenging to ensure the completeness of data in an automated system. Therefore, compensating for missing data or finding alternative data sources becomes critical to developing an effective model.

Data imprecision. Some assessments made by automated tools and scripts are not 100% accurate. When a human operator analyzes the domain name, this imprecision can have little effect; however, imprecise features might significantly degrade the model's performance when training a machine learning algorithm. Such features need to be either manually calibrated or excluded from the dataset. This requirement for manual calibration increases the preprocessing effort and raises the need for ongoing maintenance to adjust for changes in the data quality over time. Consequently, data imprecision remains a significant barrier to achieving consistent and reliable model outcomes.

Limited data. The original dataset used for the experiment is highly imbalanced, with the prevalence of false positive entries (approximately 80%). While having real, authentic data is advantageous, the under-representation of the suspicious samples limits the effectiveness of the model training. This data imbalance can lead to a bias in the model, making it less sensitive to the minority class. Considering that in the given scenario, suspicious samples are the minority class, this imbalance can have critical security implications. Techniques like oversampling, undersampling, or synthetic data generation should be applied to mitigate this imbalance, but these methods have limitations and the potential to introduce new biases.

Subjective verdict. The verdict given by an analyst may differ based on the experience level of an analyst and the amount of information available. While junior analysts are encouraged to seek support from more experienced peers, the accuracy of the verdict can differ, and the reasoning behind the verdict may also differ between analysts.

Flexibility and wide application. Manual tuning might ensure the model can be applied to different datasets. In the current experiment, the model is tweaked to mimic the analysis processes in a given company. This implies that the model must also be updated if new changes are introduced in the process. Additionally, the model performance is only as good as the process mentioned above is; in other words, the quality of the model performance is tightly coupled with the performance of human operators performing the analysis. This dependency on human expertise limits the model's scalability to other contexts. It creates a maintenance challenge where the model must continuously evolve alongside changes in human analysis practices and emerging threats.

5 Methodology

In this work, a new method of domain classification is explored based on the dataset that contains verified benign domain names and suspicious domain names with the corresponding labels. Most researchers in the field of domain analysis attempt to identify the malicious domains based on the features of benign domains and features of malicious domains, which come from two separate unrelated datasets - a benign dataset and a malicious dataset. However, in this research, the dataset consists of labeled rows, each containing a verified benign domain treated as ground truth and a domain identified as potentially suspicious by the automated tools employed by the platform. Therefore, the dataset provides a unique opportunity to investigate features based on differences between the verified legitimate domains and the potential threats instead of relying on separate features of benign and malicious domains.

Additionally, the dataset in this study is composed of real samples, meaning that the domain names (both legitimate and suspicious) are actual domains (or have been registered before) registered by individuals or organizations. This contrasts with synthesized lab samples, which may fail to accurately reflect the complexity and unpredictability of real-world scenarios. Real samples provide valuable insights into actual adversarial behaviors, leading to models that are more representative and better adapted to practical applications, thereby enhancing the reliability of the findings.

Feature identification and preprocessing. The dataset contains multiple features collected by the automated scripts. Potentially relevant features were grouped into two categories: linguistic and other. Linguistic features were engineered with the intent to mimic the analysis performed by the human operator during the alert handling. The identified features rely heavily on the observed differences between the pairs of original and suspicious domains. Additional (other) features were extracted from the available data. The full list of features is described in sections 6.3.1 and 6.3.2. A scaler is used for standardization to compensate for the different scales and ranges used with different features. The `StandardScaler`[36] scales the features of a dataset X to have a mean of 0 and a standard deviation of 1. Given a dataset with features $x_1, x_2, \ldots, x_n$, the scaling process for each feature x_i can be expressed as:

$$x_i' = \frac{x_i - \mu_i}{\sigma_i}$$

where:
x_i' is the scaled value of the feature x_i,
μ_i is the mean of the feature x_i,
σ_i is the standard deviation of the feature x_i.

The transformation ensures that the features are centered around zero with a unit variance. In mathematical terms, the resulting dataset has:

$$\text{mean}(x_i') = 0 \quad \text{and} \quad \text{std}(x_i') = 1$$

The primary motivation for using the `StandardScaler` in the given algorithm is to ensure that each feature contributes equally to the model's performance. Features with different scales can disproportionately influence the model. For instance, features such as normalized standard deviation of character modification position have a range of [-1, 1], while features like a number of character modifications have a theoretical range of [0, ∞]. If not accounted for this difference, features with more extensive numerical ranges might dominate those with smaller ranges, leading to biased results [4].

Dataset preprocessing. The experiment is repeated over three variations of the dataset:

- **Stratified splitting (unchanged)**. In this setting, no modification is applied to the dataset. The dataset is proportionally split between the training, validation, and test set. Let X represent the dataset, and let y represent the labels. The split is performed in a way that maintains the proportion of each class in each subset:

$$X_{\text{train}}, X_{\text{val}}, X_{\text{test}}, y_{\text{train}}, y_{\text{val}}, y_{\text{test}} = \text{StratifiedSplit}(X, y, \text{ratio})$$

  Here, StratifiedSplit ensures that the ratio of class labels in the original dataset y is preserved in the training, validation, and test sets.

- **Undersampling**. The dataset is split into the training, test, and validation sets in this setting. Additionally, the training dataset is undersampled according to the minority class. In the given scenario, the minority class is the true positive label. Therefore, the size of the original dataset is reduced to balance the difference between the classes. Mathematically, let N_{minority} and N_{majority} be the number of samples in the minority and majority classes, respectively, where $N_{\text{minority}} < N_{\text{majority}}$. Undersampling reduces the number of majority class samples to match the number of minority class samples:

$$N'_{\text{majority}} = N_{\text{minority}}$$

  The new training dataset size is $2 \times N_{\text{minority}}$. In this work, the `RandomUnderSampler` from the `imblearn` library is used to perform this operation:

$$X_{\text{train\_resampled}}, y_{\text{train\_resampled}} = \text{RandomUnderSampler}(X_{\text{train}}, y_{\text{train}})$$

  The undersampling is performed by randomly selecting samples from the majority class without replacement. The sampling is performed randomly, with the controlled random state parameter to ensure reproducibility of results. The inner mechanism relies on a uniform random sampling process, selecting samples from the majority class until the desired balance is achieved.

- **Oversampling**. The dataset is split into the training, test, and validation sets in this setting. Additionally, the training dataset is oversampled to artificially increase the number of instances in the minority class. In the given scenario, the minority class is the true positive label. Therefore, the size of the original dataset is increased to match the difference between the classes. Let N_{minority} and N_{majority} be the number of samples in the minority and majority classes, respectively. Oversampling increases the number of minority class samples to match the number of majority class samples:

$$N'_{\text{minority}} = N_{\text{majority}}$$

  The new training dataset size is $2 \times N_{\text{majority}}$. In this work, the `RandomOverSampler` from the `imblearn` library is used to perform this operation:

$$X_{\text{train\_resampled}}, y_{\text{train\_resampled}} = \text{RandomOverSampler}(X_{\text{train}}, y_{\text{train}})$$

  The `RandomOverSampler` duplicates existing samples from the minority class randomly to reach the desired number, which is a simple and naive strategy for increasing the number of samples of the minority class. The replication process ensures that the new samples retain the characteristics of the minority class, with randomness managed through the particular parameter for reproducibility.

Model building and hyperparameter tuning. The Neural Network (NN) model is built using Keras Classifier. This phase involves creating the initial structure of the model and further modification of this structure to discover the structure that yields the best performance. The modification of the model structure is performed by tuning hyperparameters. Hyperparameters are the parameters in the model that control the model's behavior and are determined before the training process. In this scenario, the following hyperparameters are being adjusted in the neural network:

- Learning rate. The learning rate indicates the step size during optimization; a higher rate speeds up learning but might miss the optimal solution, while a lower rate provides precision but slows the NN convergence.

- Batch size. Batch size influences training stability and speed—larger batches offer smoother gradients but increase memory usage.

- Dropout rate. The dropout rate affects the generalization ability of the model and its proneness to overfitting by randomly dismissing a predefined rate of neurons from further calculations.

- The number of layers and the number of neurons impact the model's ability to capture data and pattern complexity; networks with more layers and neurons per layer can model intricate relationships but risk overfitting.

The model training is performed in two stages. First, the initial hyperparameters are discovered through the randomized search. Then, the grid search is applied to the values near the ones discovered through the randomized search.

Randomized search explores a wide range of hyperparameter values by randomly sampling combinations from a predefined search space. Unlike grid search, which evaluates every possible combination, randomized search samples a fixed number of hyperparameter combinations, allowing it to cover a more extensive search space more efficiently. The parameter $n_{\text{iterations}}$ controls the number of iterations, which specifies how many random combinations should be tried. Randomized search can identify promising regions without exhaustive computation by sampling a subset of the hyperparameter space. Additionally, randomized search can explore combinations that grid search might miss, especially in cases where the hyperparameter space is vast and non-uniform.

Mathematically, if $\theta_1, \theta_2, \ldots, \theta_k$ represent the hyperparameters to be tuned, randomized search samples a combination of hyperparameters $\Theta = (\theta_1, \theta_2, \ldots, \theta_k)$ from the search space S randomly:

$$\Theta_i \sim \text{Uniform}(S), \quad i = 1, \ldots, n_{\text{iterations}}$$

The model is trained and evaluated for each sampled combination, and the combination with the best performance is selected for further tuning.

Grid search is employed for fine-tuning after the initial exploration with randomized search. Grid search exhaustively evaluates all possible combinations of a predefined hyperparameter grid. Given a set of hyperparameters $\Theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$, where each θ_i can take a range of values, grid search constructs a grid that spans all possible combinations:

$$\Theta_{\text{grid}} = \{(\theta_1, \theta_2, \ldots, \theta_k) \mid \theta_i \in V_i\}$$

where V_i is the set of values for hyperparameter θ_i. The total number of combinations is given by the product of the number of values for each hyperparameter:

$$|\Theta_{\text{grid}}| = \prod_{i=1}^{k} |V_i|$$

The model is trained and validated for each Θ_{grid} combination. Grid search fine-tunes the model by exploring a narrow hyperparameter space identified by randomized search. It ensures that the entire given space is covered, increasing the likelihood of finding the optimal hyperparameter values. Overall, the exhaustive nature of grid search complements the broader exploration of randomized search, ensuring that the model is finely adjusted.

When used independently, the two-stage approach helps overcome each method's limitations, leading to a more robust and well-tuned model. The approach is described in diagram 5.1.

The validation set (10% of the original dataset) is used to evaluate the model's performance iteratively during training, enabling the adjustment of hyperparameters to optimize performance. The model is trained on the training set and evaluated on the validation set using the evaluation metrics described in section 6.1.

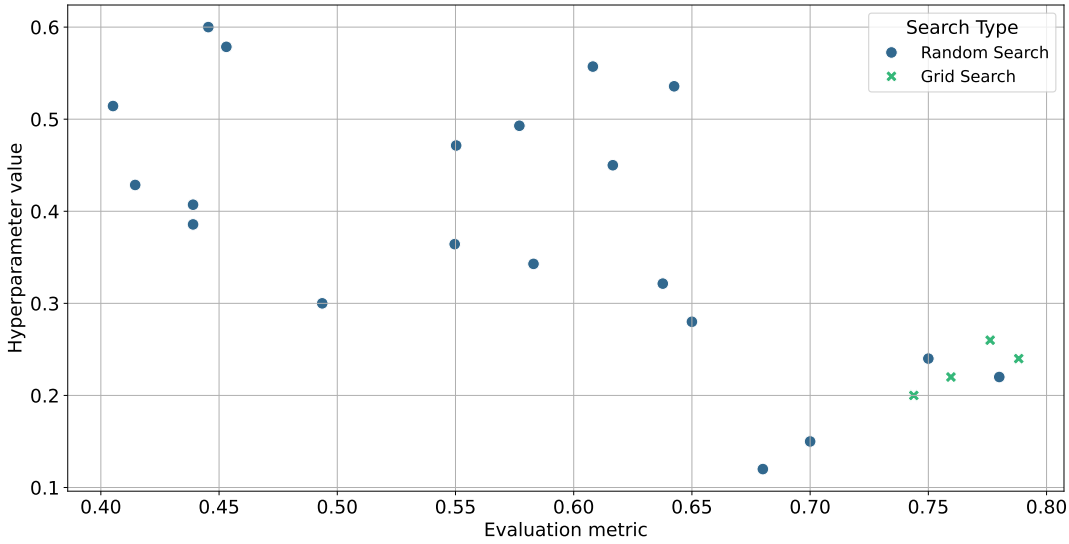

Figure 5.1: Two-Stage hyperparameter tuning process: randomized search followed by grid search.

Model testing and evaluation. The models derived from the previous stage are tested on two different test sets. A test set is a separate subset of the data not used during the training or validation phases. It assesses the model's ability to generalize on previously unseen data. The first test set contains 10% of the original dataset's records sampled randomly. The second test set consists of the newest data, i.e., the latest records from the dataset, also amounting to 10% of all records. Three models (trained on undersampled, oversampled, and stratified datasets) are tested with their respective best parameters on each test set. The evaluation metrics (see 6.1) are collected on all the experiments, and the performance of the models in all experiments is compared. This evaluation and the comparison of the results present an opportunity to see the impact of the dataset modification choice and the training set on the quality of the results, as well as hypothesize upon the further improvement of the model and the dataset.

5.1 Neural network architecture

The neural network architecture comprises multiple fully connected (dense) layers that perform binary classification. The input layer is structured according to the shape of the dataset features. Between the hidden layers, dropout regularization is employed to mitigate overfitting by randomly setting a fraction of input units to zero during training. The network comprises several fully connected (dense) hidden layers. Each hidden layer consists of multiple neurons and uses the ReLU activation function. The final output layer consists of a single neuron with a sigmoid activation function. This layer outputs a probability score indicating the likelihood of a domain being suspicious. The probability score p lies in the interval $[0, 1]$. This type of architecture allows the network to capture complex representations of the input data. To prevent overfitting, *dropout layers* are interspersed between the dense layers. Dropout is a regularization technique that randomly drops a subset of neurons during training, effectively reducing the co-adaptation of neurons and improving the model's generalization. Combining fully connected layers, dropout for regularization, and combined search techniques for hyperparameter tuning proved an effective strategy for optimizing the neural network. Randomized search offered a computationally efficient way to calculate the initial hyperparameters, while grid search allowed for further fine-tuning. Figure 5.2 illustrates the high-level architecture of the neural network.

Input layer. For a given dataset, the input layer contains neurons corresponding to the dataset's number of features. Each neuron in the input layer represents one feature of the data. If the dataset consists of n features, the input layer will have n neurons. The input to the neural network can be

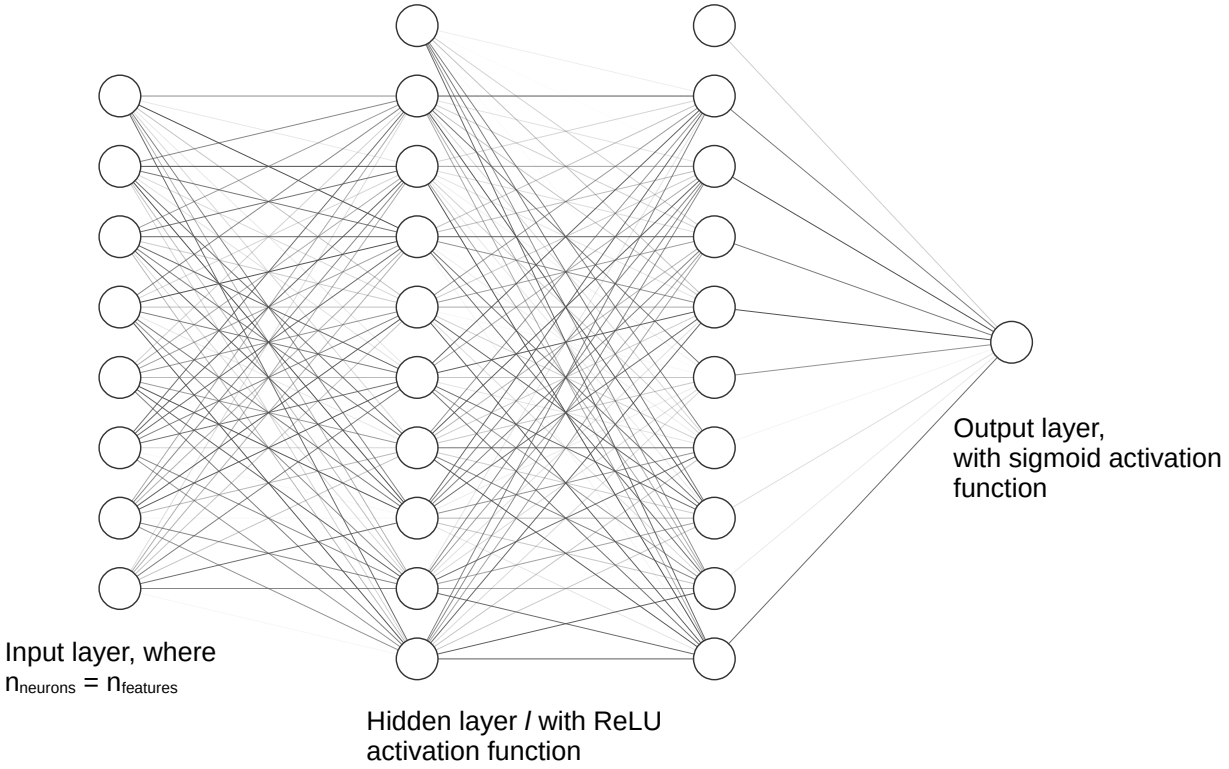

Figure 5.2: Neural network architecture: fully connected layers with multiple neurons, ReLU activation, dropout layers, and a sigmoid output layer for binary classification.

represented as a vector $\mathbf{x}$ of length n:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]$$

where x_i is the i-th feature of the input data.

Hidden layers. Each hidden layer consists of multiple neurons, where each neuron applies a weighted sum followed by a non-linear activation function to the inputs it receives from the previous layer.

For a given hidden layer l with m neurons, the output $\mathbf{h}^{(l)}$ of the l-th layer can be calculated as:

$$\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$$

where:
$\mathbf{W}^{(l)}$ is the weight matrix of the l-th layer,
$\mathbf{h}^{(l-1)}$ is the output from the previous layer (or the input vector for the first hidden layer),
$\mathbf{b}^{(l)}$ is the bias vector for the l-th layer,
σ is the activation function applied element-wise.
Each layer uses the ReLU activation function, which is defined as:

$$\sigma(z) = \max(0, z)$$

where z is the input to the activation function.

Output layer. The output layer is the final layer of the neural network and produces the final predictions. It is represented by a single neuron with a sigmoid activation function to produce the label probability measured between 0 and 1. The general operation of the sigmoid activation function is shown in figure 5.3.

For binary classification, the output $\hat{y}$ is calculated as:

$$\hat{y} = \sigma(\mathbf{W}^{(o)}\mathbf{h}^{(L)} + \mathbf{b}^{(o)})$$

$$\hat{y} \in [0, 1]$$

where:
$\mathbf{W}^{(o)}$ is the weight matrix of the output layer,
$\mathbf{h}^{(L)}$ is the output from the last hidden layer,
$\mathbf{b}^{(o)}$ is the bias vector of the output layer,
σ is the sigmoid activation function.

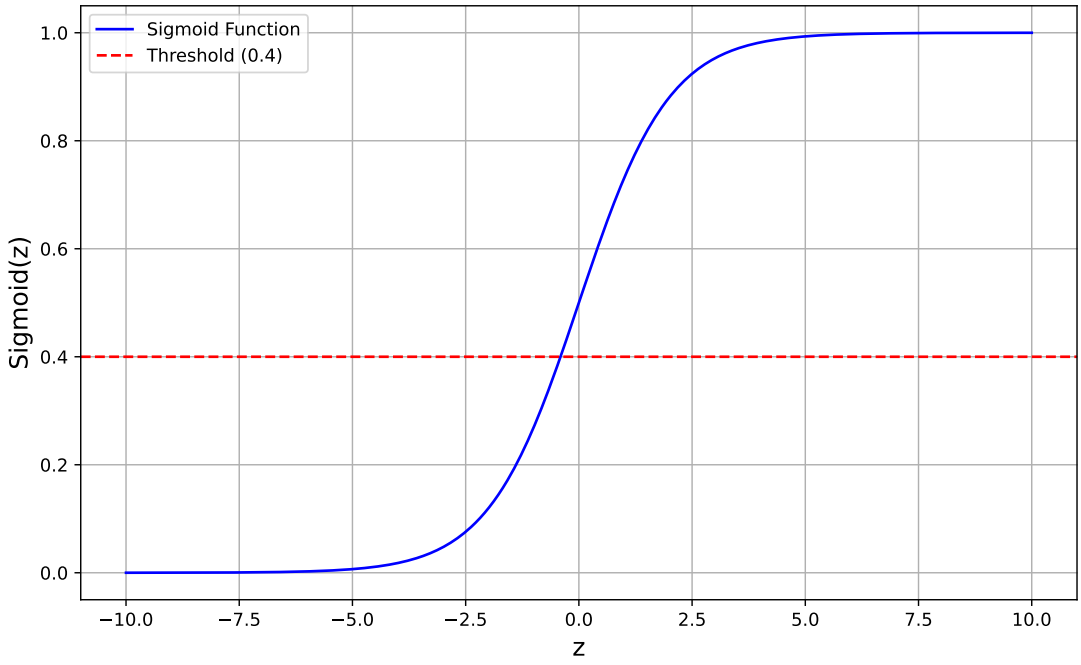

Figure 5.3: Sigmoid activation function output with a threshold.

The **threshold** plays a critical role in converting predicted probabilities into binary class labels. This threshold determines the decision boundary that separates the two classes based on the probability output of the model.
Given a probability p predicted by the model, the final class prediction $\hat{y}$ is determined using the threshold τ as follows:

$$\hat{y} = \begin{cases} 1 & \text{if } p \geq \tau \\ 0 & \text{if } p < \tau \end{cases}$$

where:
$\mathbf{p}$ is the predicted probability of the positive class,
τ is the threshold value.

Adjusting the threshold affects the model's sensitivity (true positive rate) and specificity (true negative rate). Lowering the threshold increases sensitivity but decreases specificity, and vice versa. This trade-off is essential for optimizing the model based on the application's requirements. While the goal of the system is to decrease the number of false positives, it should not be achieved by neglecting the false negatives. In security-specific settings, false negatives generally are deemed more dangerous than false positives, as missing a critical alert might cause serious consequences. Therefore, the model should achieve optimal performance in terms of false negatives and false positives.

5.2 Random forest classifier

In this study, the random forest classifier ranks feature importance and performs predictions in the domain analysis process as an alternative to the deep learning model. The feature importance ranking provides a clear overview of the model decision-making process, allowing for easy elimination or modification of features that overly dominate in the decision-making or are irrelevant or confusing to the model. The manual analysis process, currently performed by analysts, is taken as a baseline for the feature importance assessment. For instance, if a model uses a domain registration timestamp as a dominating feature while analysts don't consider this feature as important, this feature might need to be eliminated from the dataset.

In addition to feature ranking, random forest is used as a baseline for evaluating NN model predictions since random forest, being an ensemble learning classifier, shows low sensitivity to outliers, which makes it less prone to overfitting.

The basic structure of the random forest classifier is shown in figure 5.4.

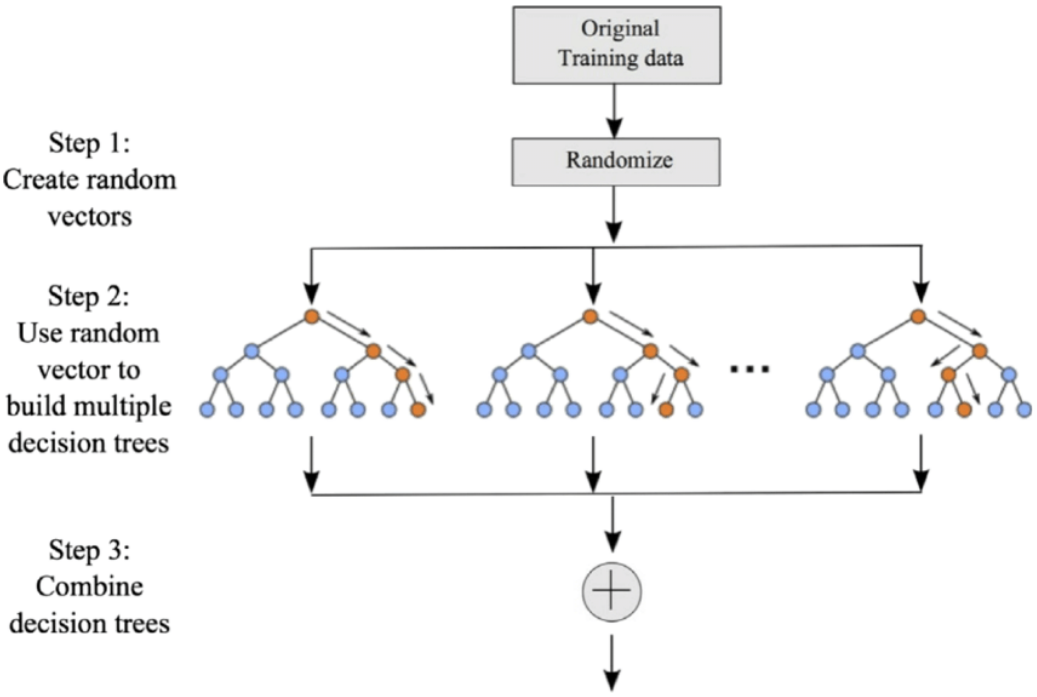

Figure 5.4: Random forest classifier algorithm [46].

The random forest classifier is constructed by generating many decision trees (5.5), where each tree is trained on a different subset of the data using a random sample of features. This randomization helps reduce overfitting and improves generalization [33]. Each decision tree makes a prediction, and the random forest aggregates the predictions through majority voting for classification. The trees are independently constructed, allowing the model to capture various data patterns. The classifier's randomness in selecting data samples and features ensures that the trees are diverse, mitigating the risk of any single tree dominating the prediction process. The number of individual trees in the forest can be tuned prior to the model training to achieve a balance between results quality and computation time.

Let $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ be the training data and $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$ be the corresponding labels, where n represents the number of data points. For each decision tree T_i ($i \in \{1, 2, \ldots, N\}$):

- Draw a bootstrap sample $\mathbf{X}_i$ from $\mathbf{X}$.
- Grow the decision tree T_i on $\mathbf{X}_i$ by recursively splitting the data based on a random subset of features until a stopping criterion is met (e.g., maximum depth or minimum number of samples at a leaf node).

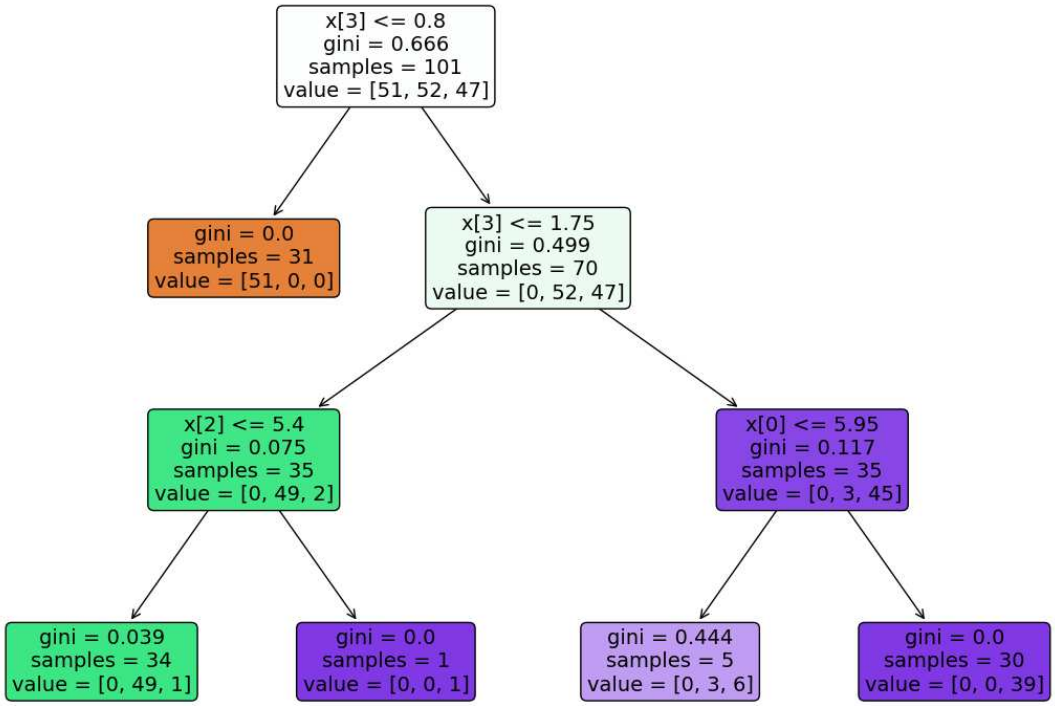

Figure 5.5: A sample decision tree from a random forest.

During the model tuning, three stopping criteria are considered: maximum depth of the tree, minimum samples per leaf, and minimum samples to split. All three criteria aim to limit the complexity of the tree by balancing the generalization ability with the sufficient learning ability of the model. If the values for minimum samples are set too low or maximum depth too high, the model might learn more complex patterns. Still, at the same time, it becomes prone to overfitting, while in the opposite case, the model might not be able to learn enough patterns from data.

The splitting criterion used for each tree is typically based on minimizing the impurity of the resulting subsets. For a node t, the impurity $I(t)$ can be calculated using measures like Gini impurity or entropy. For instance, the Gini impurity is given by:

$$I(t) = 1 - \sum_{k=1}^{K} p_k^2,$$

where p_k is the proportion of samples belonging to class k at node t, and K is the number of classes.

To predict a new input $\mathbf{x}$, each decision tree T_i provides a class prediction $h_i(\mathbf{x})$. The final prediction of the random forest is obtained by aggregating the predictions from all individual trees. For classification, majority voting is used:

$$\hat{y} = \text{mode}\left(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_N(\mathbf{x})\right).$$

One of the benefits of the random forest classifier is the ability to estimate feature importance. The importance of a feature f_j is calculated by measuring the total decrease in impurity due to splits on f_j, averaged over all trees in the forest. Let $\Delta I(f_j, T_i)$ represent the decrease in impurity for feature f_j in tree T_i; then the feature importance $FI(f_j)$ is given by:

$$FI(f_j) = \frac{1}{N} \sum_{i=1}^{N} \Delta I(f_j, T_i).$$

To ensure consistent results in the feature importance estimation, the feature importance is calculated on an unmodified dataset (i.e., not under- or oversampled).

Similar to the neural network, the random forest model in this project undergoes two tuning stages. First, randomized search explores a wide range of initial hyperparameters without extreme computational overhead. Once a promising set of hyperparameters is identified, grid search refines the model by exhaustively testing a narrower range. This approach allows the discovery of suitable hyperparameters more efficiently than a plain grid search.

The following hyperparameters for the random forest are tuned:

- Number of trees. The number of trees controls how many individual decision trees are built in the random forest. A larger number of trees generally leads to more stable and accurate results but also increases computation time.
- Maximum depth. The maximum depth limits the depth of each decision tree in the forest. A larger depth allows the model to learn more complex patterns, but it may lead to overfitting if the trees are too deep. A shallower tree might prevent overfitting but could result in underfitting if important patterns are missed.
- Minimum samples per leaf. This parameter specifies the minimum number of samples required to be at a leaf node. Higher values will force the model to have more generalized leaves, reducing overfitting, whereas smaller values might allow the model to overfit by capturing noise in the data.
- Minimum samples to split. This parameter controls the minimum number of samples required to split an internal node. If set too low, the trees may become very deep and complex, resulting in overfitting. Higher values make the model more conservative, reducing its ability to learn from nuances in the data.

The random forest classifier serves as a benchmark for performance comparison with the NN model in each experiment. Due to its simplicity and robustness to overfitting, random forest provides a reliable baseline. It offers straightforward, interpretable performance that can highlight the improvements achieved by a more complex neural network (or the absence of such). Since random forest does not require extensive hyperparameter tuning, comparing it to a neural network allows for assessing the value added by advanced architectures and learning capabilities while ensuring that the benchmark remains transparent and easy to understand.

6 Experiment setup

This chapter outlines the procedures used in conducting the experiments for this study. The experiment setup encompasses the configuration of the data preprocessing, model training, hyperparameter tuning, and evaluation processes. The aim is to ensure that the experiments are reproducible and that the results are reliable and valid.

6.1 Evaluation methodology

This study evaluates the model's performance using the following key metrics: accuracy, precision, recall, F1 score, and AUC (Area-Under-Curve). These metrics collectively enable a robust evaluation of the model's classification performance, addressing various aspects of prediction accuracy and reliability and comparing the models and models with varying parameters. When comparing the overall model performance, higher importance is given to the F1 score as it reflects the model performance most accurately, particularly in the case of an imbalanced dataset. Additionally, average inference time is measured to assess the speed of model prediction. The higher complexity of the model is reflected in the higher inference time, affecting the practical limitations of the model implementation.

Accuracy. Accuracy is the ratio of correctly predicted instances to the total instances in the dataset. It is defined as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

Precision. Precision, also known as positive predictive value, measures the accuracy of the positive predictions. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.2}$$

Recall. Recall, or sensitivity, measures the ability of the model to identify all relevant instances. It is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.3}$$

F1 Score. The F1 score is the harmonic mean of precision and recall, balancing the two metrics. It is beneficial when the dataset is imbalanced. The F1 score is defined as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.4}$$

Precision-Recall Curve and AUC. The Area Under the Precision-Recall Curve (PR-AUC) demonstrates the trade-off between precision and recall metrics over different thresholds. A higher PR-AUC indicates that the model maintains high precision even as recall increases, which is desirable in the given scenario as minimizing false negatives is critical. The PR-AUC is calculated as:

$$\text{PR AUC} = \int_0^1 \text{Precision}(\text{Recall})\, d\text{Recall}$$

This metric is handy in this work, as the goal is to effectively reduce the number of false positives while also keeping a low number of false negatives. A high PR-AUC implies the model performs well at distinguishing between legitimate and suspicious domains, even under varying thresholds. However, the model's efficiency in decreasing analyst's manual work decreases if the threshold is too low, resulting in a higher number of false positives.

Average inference time. Inference time measures the time the model requires to predict the test set. In the current study, the inference time is measured as an average of the ten runs of the best model.

$$T_{\text{inference}} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{end},i} - t_{\text{start},i}) \tag{6.5}$$

, where:
$T_{\text{inference}}$: Average inference time for the model.
N: Total number of samples (=10).
$t_{\text{start},i}$:Start time for the i-th inference.
$t_{\text{end},i}$: End time for the i-th inference.

6.2 Dataset preprocessing

The dataset contains 17.829 labeled rows. The labels have been previously assigned by the analysts who investigated the corresponding alerts. The labels are divided into two groups:

- False positives - the domains classified as not presenting a risk, further referred to as Class 0,
- Escalated - the domains classified as potential risk are referred to as Class 1.

The distribution between the class labels labels is shown in table 6.1.

Record type	**Number of records**	**% of the total records**
Class 0	13933	78,15%
Class 1	3896	21,85%
Total	**17829**	**100%**

Table 6.1: Dataset labels proportion.

Test set separation. The test set is extracted from the original dataset. In different experiments, a test set is extracted using one of the following approaches:

- Random. The test set is extracted randomly in a stratified fashion to ensure equal class proportions for the training and test sets.
- Newest data. The test set is extracted by selecting the most recent data in the dataset.

The label distribution between the random test set and the newest test set is shown in table 6.2. As shown in the table 6.2, the number of records in both test sets is the same.

	Random test set	**Newest data test set**
Records, total	1783	1783
Class 0, records	1393	1433
Class 1, records	390	350
Class 0, %	78,13%	80,37%
Class 1, %	21,87%	19,63%

Table 6.2: The proportions of 0 and 1 classes between the random test set and a test set of the newest data.

The dataset left after the separation of the *random* test set is split into the *training* and *validation* sets, where the validation set amounts to 10% of the total number of records in the original dataset, enabling for the efficient hyperparameter tuning. Three different sets of hyperparameters are discovered as a model is trained on undersampled, stratified, and oversampled datasets. Separating a random test set ensures there is sufficient data diversity in the remaining portion of the dataset, which should help the model discover more suitable parameters for generalization.

6.3 Feature identification and preprocessing

Feature extraction was conducted to emulate the human evaluation process for domain classification. Emphasis was placed on incorporating features indicative of domain legitimacy and potential malicious behavior, reflecting criteria commonly applied in human assessments. The feature set was chosen based on data availability and practical considerations observed in previous research. This approach aimed to capture the essential characteristics that human analysts would consider, thereby enhancing the model's ability to generalize and perform effectively in real-world scenarios. The features are divided into two subgroups:

- Linguistic features. Linguistic features encompass various attributes derived from the textual content of domain names. These features capture semantic and syntactic characteristics that indicate domain legitimacy or potential malicious intent. These features include the presence of suspicious keywords, the length of the domain name, and the similarity between the domain names. These features can reflect patterns and anomalies indicative of human-like evaluations of domain names. The section 6.3.1 presents a detailed list of linguistic features used in the experiments.

- Extra features. Extra features include a diverse range of attributes that do not fall under linguistic analysis but are equally crucial for domain classification. These features provide a comprehensive view of the domain's background and registration attributes, contributing to a more well-rounded classification process. The section 6.3.2 presents a detailed list of extra features used in the experiments.

6.3.1 Linguistic features

To mimic the evaluation process performed by the analysts, multiple features were extracted from the properties of the domain names and the perceived differences between the known, customer-owned domain names and suspicious domains. This step focuses primarily on the linguistic features of the domain names, i.e., the differences and particularities that human analysts would notice and apply during the evaluation process. Table 6.3 describes the features retrieved from the domain name in this step.

Table 6.3: Features extracted from the domain name.

Feature	**Acronym**	**Definition**
Punycode in the original domain*	puny_orig*	Binary value indicating whether the original domain uses Punycode in the domain link*
Punycode in the suspicious domain	puny_link	Binary value indicating whether the original domain uses Punycode in the domain link
Domain length	length_full	The length (in characters) of the (subdomain_{fake}.domain_{fake}.TLD_{fake})
Domain name length	length_dname	The length (in characters) of the (subdomain_{fake}.domain_{fake})
Levenshtein distance 1	lev_dist_dname	Levenshtein distance between (subdomain_{orig}.domain_{orig}) and (subdomain_{fake}.domain_{fake})
Levenshtein distance 2	lev_dist_full	Levenshtein distance between (subdomain_{orig}.domain_{orig}.TLD_{orig}) and (subdomain_{fake}.domain_{fake}.TLD_{fake})
Minimal Levenshtein distance	lev_min	Smallest Levenshtein distance among Levenshtein distance 1 and 2

Feature	Acronym	Definition
Jaro similarity 1	jaro_sim_domain	Jaro similarity between ($subdomain_{orig}$.$domain_{orig}$) and ($subdomain_{fake}$.$domain_{fake}$)
Jaro similarity 2	jaro_sim_full	Jaro similarity between ($subdomain_{orig}$.$domain_{orig}$.TLD_{orig}) and ($subdomain_{fake}$.$domain_{fake}$.TLD_{fake})
Different TLD	diff_tld	A binary value indicating whether the original and fake domain have different TLD
TLD modification type	tld_mod	Numerical value indicating the type of TLD difference (or its absence)
Subdomain modification	subdomain_mod	A binary value indicating whether a manipulation with subdomains was used to mimic the original domain (e.g., traffic[.]light[.]com to mimic trafficlight[.]com)
Character modifications in the domain name	char_num_mods	Number of the single-character modifications (insertion, deletion, replacement) to transform ($subdomain_{orig}$.$domain_{orig}$) into ($subdomain_{fake}$.$domain_{fake}$)
Mean modification position	char_mean_mod_pos	Mean character modification position to transform ($subdomain_{orig}$.$domain_{orig}$) into ($subdomain_{fake}$.$domain_{fake}$)
Standard deviation of modification position in the domain name	char_std_mod_pos	Standard deviation of character modification position to transform ($subdomain_{orig}$.$domain_{orig}$) into ($subdomain_{fake}$.$domain_{fake}$) in case of more than 1 modification
Character modifications in the domain link	char_num_mods_full	Number of the single-character modifications (insertion, deletion, replacement) to transform ($subdomain_{orig}$.$domain_{orig}$.TLD_{orig}) into ($subdomain_{fake}$.$domain_{fake}$.TLD_{fake})
Mean modification position	char_mean_mod_pos_full	Mean character modification position to transform ($subdomain_{orig}$.$domain_{orig}$.TLD_{orig}) into ($subdomain_{fake}$.$domain_{fake}$.TLD_{fake})
Standard deviation of modification position in the domain link	char_std_mod_pos_full	Standard deviation of character modification position to transform ($subdomain_{orig}$.$domain_{orig}$.TLD_{orig}) into ($subdomain_{fake}$.$domain_{fake}$.TLD_{fake}) in case of more than 1 modification
Added word length	combo_added_part_length	The value indicating the length of an added part to the domain name (more than two characters) as an attempt to detect combosquatting
Added word position	combo_norm_add_pos	The value indicating the position of an added part to the domain name (more than two characters) as an attempt to detect combosquatting

Feature	Acronym	Definition
Suspicious words	sus_words	Binary value indicating the presence of the words that increase the possibility of an intentional attempt to spoof the domain (e.g., "auth", "support", "identity" etc.)
* this feature was excluded from the algorithm since the given dataset had no client domains using Punycode. Included for informational purposes.		
End of Table		

While the domain name characteristics are important for the analysis, the analysis process typically includes information beyond these characteristics, such as registration data and DNS records. Consider the **hypothetical** pairs of the legitimate and corresponding suspicious domains as shown in table 6.4. The domains in pairs (1) and (2) have the same kind of linguistic difference - a single-character manipulation named *substitution* in the same position. However, this does not guarantee that the labels for these records are the same. For instance, an analyst could have checked that the suspicious domain in pair (2) has a website related to a different business or another activity. Therefore, the similarity is merely coincidental, leading to the analyst verdict of a benign domain (i.e., not a risk for the customer).
On the contrary, in the hypothetical pair (1), an unknown party could have registered the suspicious domain and had an MX record configured with a suspicious email provider. The MX record is typically used to evaluate whether the domain registrant might have potential phishing activity associated with them. The domain name previously associated with malicious activity could also be flagged on *VirusTotal*. Given these factors, an analyst would likely mark the suspicious domain in the pair (1) as suspicious and raise an alert.
In pair (3), the single-character manipulation in the middle of the word is used: a *homoglyph*. From the analyst's perspective, in this case, the linguistic features of the domain are the most prominent reason for raising the alert since the suspicious domain name appears to be purposefully mimicking the legitimate domain. Unless strong counterproof is found, an analyst would mark this domain as suspicious. This example serves as an illustration of the overbearing importance of linguistic features in specific scenarios.

#	Legitimate domain	Fake domain
1	basicword[.]com	basicworp[.]com
2	basicword[.]com	basicwork[.]com
3	normaldomain[.]com	normaidomain[.]com

Table 6.4: Domain names and linguistic characteristics.

6.3.2 Extra features

These features are not related to the linguistic properties of the domain name; instead, they are collected from different sources to provide more comprehensive information about the domain. The extra features are listed in the table 6.5.

Table 6.5: Extra features of the domains.

Feature	Acronym	Definition
Registrar	registrar_cluster	This feature describes a particular registrar used by the registrant for the domain.
Status	status_cluster	The status of the domain retrieved from the registry.
MX record	record_mx_cluster	This feature describes a particular MX record assigned to the domain or an absence of such record.

Feature	Acronym	Definition
Privacy service	priv_service	This feature evaluates whether the registrant of the domain uses anonymizing services to hide their personal information.
Blacklisted A record	blacklist_ip_a	This binary feature indicates whether the IP address associated with the domain is found in a known blacklist.
Blacklisted MX record	blacklist_ip_mx	This binary feature indicates whether the MX record associated with the domain is found in a known blacklist.
Blacklisted MX host	blacklist_host_mx	This binary feature indicates whether the host from the MX record associated with the domain is found in a known blacklist.
End of Table		

6.3.3 Feature importance ranking

After compiling the initial list of features, they are ranked by importance using the Random Forest classifier. This approach is crucial in identifying the most contributing features and removing the confusing or irrelevant features. Random Forest classifier quantifies the contribution of each feature in making predictions, helping to interpret and understand the model's behavior. Feature importance is computed by evaluating the contribution of each feature to the reduction in impurity across all trees in the forest. The importance of a feature is derived from its ability to decrease the impurity measure (e.g., Gini impurity or entropy) when used to split nodes in the decision trees.
For a given feature j, the importance can be computed as follows:

$$\text{Importance}_j = \frac{1}{T} \sum_{t=1}^{T} \text{Impurity Reduction}_{j,t}$$

where:
T is the total number of trees in the random forest.
$\text{Impurity Reduction}_{j,t}$ represents the total reduction in impurity attributed to feature j in tree t.
This study calculated the importance of features by running the random forest algorithm 100 times with different random seeds. The average importance score for each feature was determined by aggregating the importance scores from these 100 runs.
Let $\text{Importance}_{j,r}$ denote the feature importance for feature j in run r, where r ranges from 1 to R (the total number of runs). The average importance across R runs is computed as:

$$\text{Average Importance}_j = \frac{1}{R} \sum_{r=1}^{R} \text{Importance}_{j,r}$$

Averages across multiple runs help to ensure that the reported importance of features is not entirely influenced by any single run, therefore providing more stable and generalizable results. A more accurate and reliable measure of feature contribution is obtained by averaging importance scores across multiple runs, leading to better model interpretation. The feature contributions are reflected in figure 6.1.

The prevalence of the linguistic features in the top part of the graph indicates that the model generally follows a similar analysis pattern as a human operator. Since operators do not have a strict ruleset for the analysis process, operators might judge the importance of a certain feature differently. Nevertheless, all operators follow a similar procedure, and the linguistic features of the domain are usually the decisive factor for the domain evaluation.

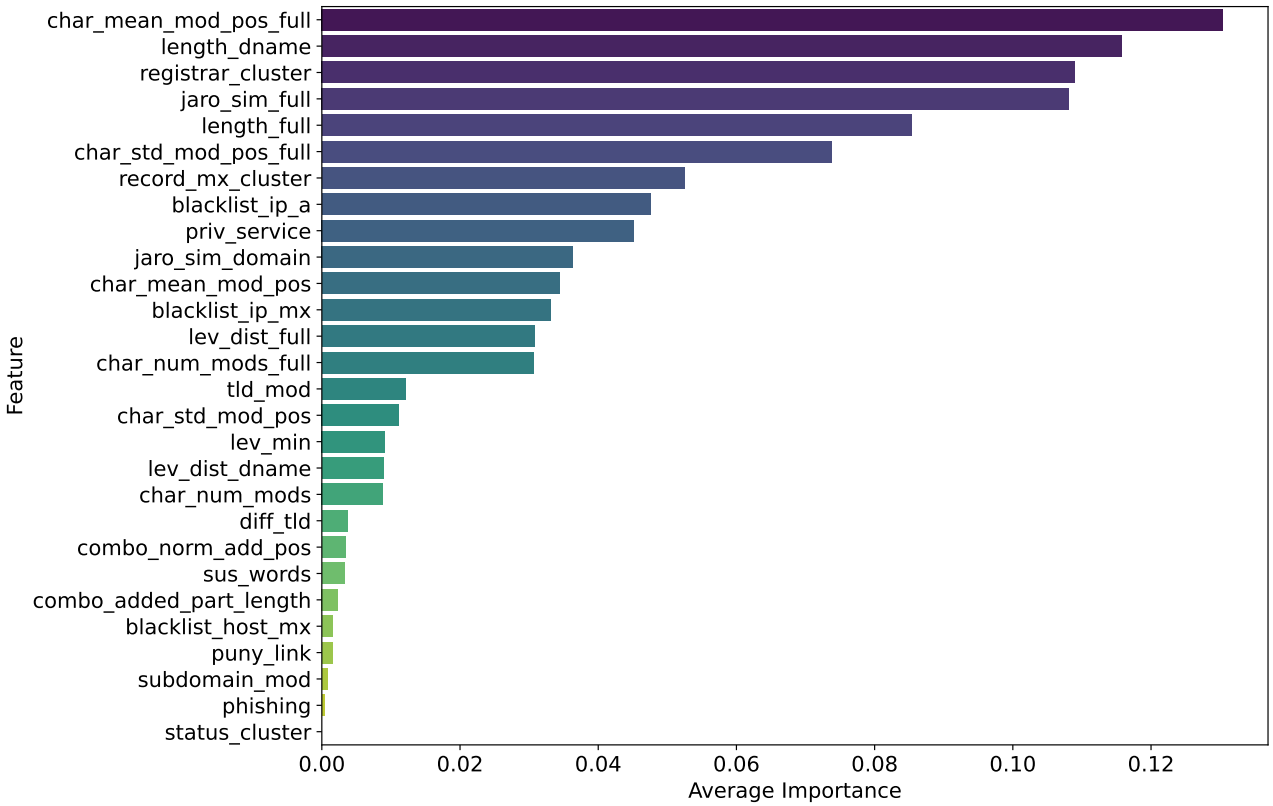

Figure 6.1: Average feature importance across 100 runs.

6.4 Hyperparameter tuning

This section describes the approach used for tuning the neural network model and random forest. In both cases, a similar two-stage tuning approach is applied to optimize computations and balance the performance of these models.

6.4.1 Neural network

The hyperparameters considered for tuning were the number of hidden layers, the number of neurons per layer, the learning rate, batch size, and dropout rate. Randomized search was configured to sample from these parameter distributions over a predefined number of iterations. A `KerasClassifier` was used as the model estimator, and the `RandomizedSearchCV` from `scikit-learn` library was employed to execute the first stage of the tuning process. Early stopping was incorporated during the model training to ensure that the training process would halt once validation performance stopped improving over a specified time interval. This approach helps avoid unnecessary epochs and minimizes overfitting risks by retaining models that demonstrate the best validation performance according to the F1 score. Table 6.6 describes the parameter distributions used for the hyperparameter tuning.

Following the randomized search, a more focused grid search was performed on the identified hyperparameter configurations. The grid search allowed for an exhaustive evaluation of specific combinations of hyperparameters, facilitating fine-tuning to enhance model performance. The hyperparameter tuning was conducted separately on undersampled, oversampled, and stratified (unmodified) datasets and evaluated with a validation set (10% of the original dataset).

The final parameters for each of the experiments are shown in table 6.7. For the model trained on the undersampled dataset, the hyperparameter set is denoted as HP_u, and this set is applied to all experiments involving an undersampled dataset. For the model trained on the oversampled dataset, the hyperparameter set is denoted as HP_o. Finally, for the model trained on the unmodified stratified dataset, the hyperparameter set is denoted as HP_s.

A single hyperparameter can affect the performance of the model as demonstrated in figure 6.2 where model performance on a validation set is plotted for different dropout rates. With the best parameters configured for each of the *Under*, *Over*, and *Strat* models, the effect of dropout rate value

Hyperparameter	Definition	Min value	Max value
Number of layers	The number of dense layers in the model. The number of the dropout layers corresponds to the number of dense layers	5	20
Number of neurons	The number of neurons per layer (except input layer and output layer)	50	300
Learning rate	The floating point value that controls how much to adjust the model in response to the error, affecting the speed and precision of convergence	0.001	0.01
Batch size	The number of samples processed before the model is updated during training	20	300
Dropout rate	The fraction of neurons randomly deactivated during training to prevent overfitting	0	0.6

Table 6.6: Variable hyperparameters of the NN model for randomized search.

	HP_u	HP_s	HP_o
Number of layers	8	8	8
Number of neurons	200	190	138
Learning rate	0.002	0.001	0.0015
Batch size	100	130	154
Dropout rate	0.1	0.15	0.15

Table 6.7: Final hyperparameters for each NN model.

varies depending on the sampling technique applied to the dataset.

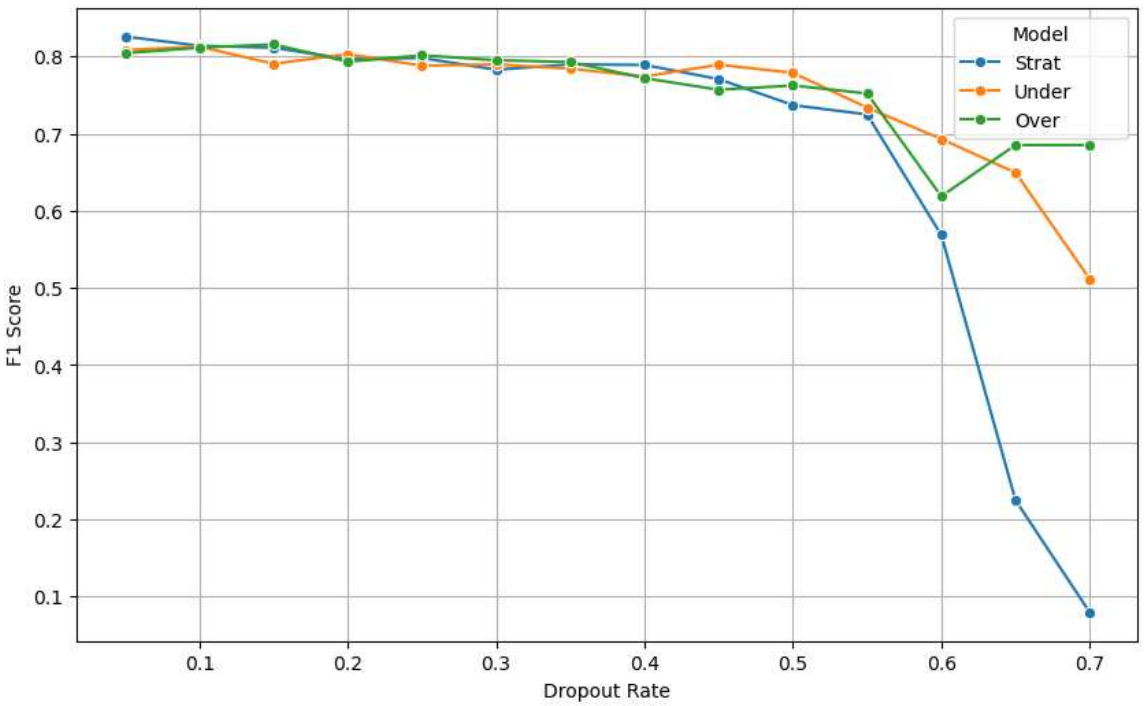

Figure 6.2: An effect of variable dropout rate on f1 score on a validation set.

The most severe impact is observed on a model trained on an unmodified imbalanced dataset, while on oversampled and undersampled datasets, the influence of the dropout rate is less dramatic. Due to the application of the balancing techniques, the neural network is less sensitive to the hyperparameter variations without favoring a majority class as opposed to the case of an imbalanced dataset. In other words, the training on a balanced dataset (under- or oversampled) is more stable and less prone to overfitting or underfitting.

6.4.2 Random forest

Similarly to the approach in neural network tuning, randomized search was used initially to efficiently explore a wide range of possible hyperparameter values.

After determining the initial hyperparameters through randomized search, a more fine-tuned optimization was performed using grid search. Grid search systematically tests every possible combination of the specified hyperparameter values within a predefined range. This approach is more exhaustive compared to randomized search and helps achieve more precise optimization of the hyperparameters. Table 6.8 describes the parameter distributions used for the hyperparameter tuning.

Hyperparameter	**Definition**	**Min value**	**Max value**
Number of trees	The number of individual decision trees in the model.	10	200
Maximum depth	The depth of each decision tree in the forest. A larger depth allows the model to learn more complex patterns but may lead to overfitting.	5	30
Minimum samples per leaf	The minimum number of samples required to be at a leaf node	2	10
Minimum samples to split	The minimum number of samples required to split an internal node. If set too low, the trees may become very deep and complex, resulting in overfitting.	5	10

Table 6.8: Variable hyperparameters of the random forest model for randomized search.

By first using randomized search to narrow down potential values and then grid search to precisely fine-tune the model, a more efficient and thorough approach was taken to achieve optimal performance.

The final parameters for each of the experiments are shown in table 6.9. For the model trained on the undersampled dataset, the hyperparameter set is denoted as HP_u, and this set is applied to all experiments involving an undersampled dataset. For the model trained on the oversampled dataset, the hyperparameter set is denoted as HP_o. Finally, for the model trained on the unmodified stratified dataset, the hyperparameter set is denoted as HP_s.

	HP_u	HP_s	HP_o
Number of trees	145	155	147
Maximum depth	20	20	20
Minimum samples per leaf	6	6	6
Minimum samples to split	5	5	5

Table 6.9: Final hyperparameters for each random forest model.

6.5 Model evaluation

The results presented in the table 6.10 and the table 6.11 are obtained using the validation set during the hyperparameter tuning stage from the neural network and random forest models.

After hyperparameter tuning and model training were completed, the models were tested on two

Metric/H-parameter set	HP_u	HP_s	HP_o
Accuracy	0.84	0.91	0.86
Precision	0.78	0.88	0.80
Recall	0.86	0.85	0.89
F1-Score	**0.80**	**0.86**	**0.82**
Average inference time	0.140	0.138	0.143
PR-AUC	0.75	0.82	0.78

Table 6.10: Tuning results from the NN models trained on differently resampled datasets.

Metric/H-parameter set	HP_u	HP_s	HP_o
Accuracy	0.82	0.91	0.88
Precision	0.77	0.91	0.82
Recall	0.86	0.91	0.91
F1-Score	**0.78**	**0.91**	**0.85**
Average inference time	0.056	0.067	0.044
PR-AUC	0.75	0.80	0.81

Table 6.11: Tuning results from the random forest models trained on differently resampled datasets.

test sets: a random test set (*UnderRand*, *OverRand*, and *StratRand* experiments) and the newest data test set (*UnderNew*, *OverNew*, and *StratNew* experiments).

A comparison of these experiments provides an opportunity to observe the differences likely stemming from the dataset's temporal characteristics. Additionally, differences between the models trained on differently sampled datasets provide insight into the importance of data quality.

While a random test set is likely to reflect the dataset diversity more accurately, in real-world scenarios, the model would be evaluating the data that might possess slightly different characteristics than previously seen. For instance, the model should be able to generalize on the pairs of domains it previously hasn't seen (e.g., if new client domains are added to the platform). By testing the model on the newest data test set, it is possible to get results that are very close to the ones that could occur in the scenario of a model implementation as part of the domain analysis process.

The simplified description of the evaluated experiments is presented in the table 6.12.

Experiment	Dataset modification	Test set	Hyperparameter set
UnderRand	Undersampled	Random	HP_u
OverRand	Oversampled	Random	HP_o
StratRand	Unchanged	Random	HP_s
UnderNew	Undersampled	Newest data	HP_u
OverNew	Oversampled	Newest data	HP_o
StratNew	Unchanged	Newest data	HP_s

Table 6.12: Description of the experiments.

The experiments are conducted using different sampling techniques to address the class imbalance:

- *UnderRand*. This experiment involves using previously discovered hyperparameters HP_u and a randomly extracted test set for the model performance assessment. The performance of the NN model is compared accordingly to that of a random forest model trained on the same sampling type of dataset (undersampled).

- *OverRand*. This experiment involves using previously discovered hyperparameters HP_o and a randomly extracted test set for the model performance evaluation.

- *StratRand*. This experiment uses previously discovered hyperparameters HP_s and a randomly extracted test set to evaluate model performance.

- *UnderNew*. This experiment involves retraining the model on an undersampled dataset using previously discovered hyperparameters HP_u and the test set consisting of the newest data. Since the test set is extracted differently in this experiment, the model needs to be retrained on the remaining training set. When the test set is extracted randomly, it is likely to contain samples from the newest data and the older data alike, meaning that the training set is likely to include some samples of the newest data. Since the data of the test set must not be previously seen by the model, the model needs to be retrained to exclude the possibility of having seen certain samples during the hyperparameter tuning stage as part of either the training or validation set. The same process applies to *OverNew* and *StratNew* experiments.

- *OverNew*. This experiment involves retraining the model on an oversampled dataset using previously discovered hyperparameters HP_o and the test set consisting of the newest data.

- *StratNew*. This experiment involves retraining the model on an unchanged stratified dataset using previously discovered hyperparameters HP_s and the test set consisting of the newest data.

Threshold tuning. To increase the model's sensitivity and reduce the number of false negatives, the default threshold was lowered to 0.4 instead of 0.5, and the same threshold was used across all the experiments. Additionally, the PR-AUC metric is employed to evaluate the effect of different thresholds on the F1 score of the model. The effect of threshold selection is reflected in figure 6.3. There is a dramatic difference in the threshold effect between the model trained on a stratified dataset and the other modules. While 0.4 appears to be the reasonable threshold for the models trained on undersampled and oversampled datasets, the only threshold with similar performance in terms of TPR and FPR for the stratified model is 0.1; however, to ensure consistency, the same threshold of 0.4 is applied to all three models.

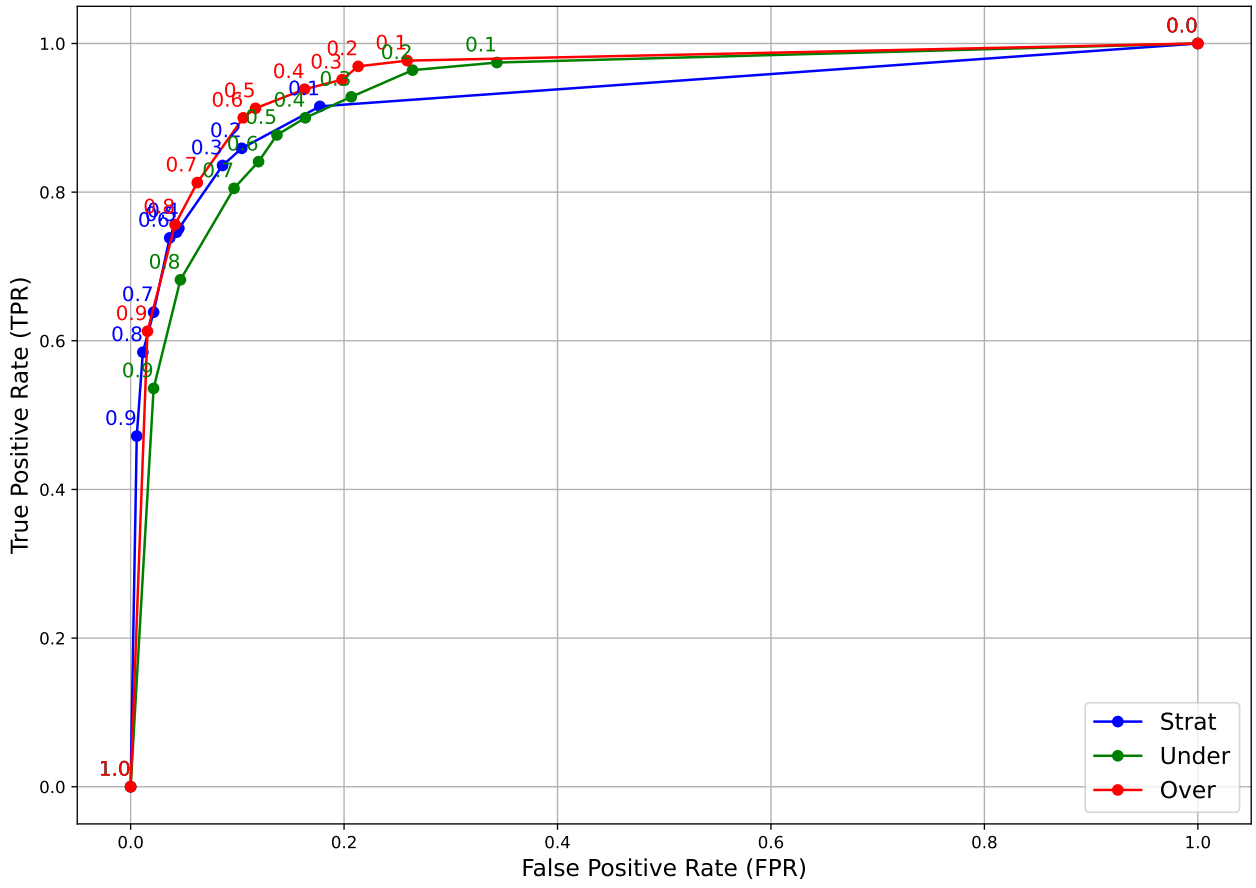

Figure 6.3: The effect of a threshold on a validation set for different NN models.

To ensure consistency across all experiments, the same threshold of 0.4 is applied to the random

forest classifier results. The effect of the threshold selection for the random forest models is reflected in figure 6.4.

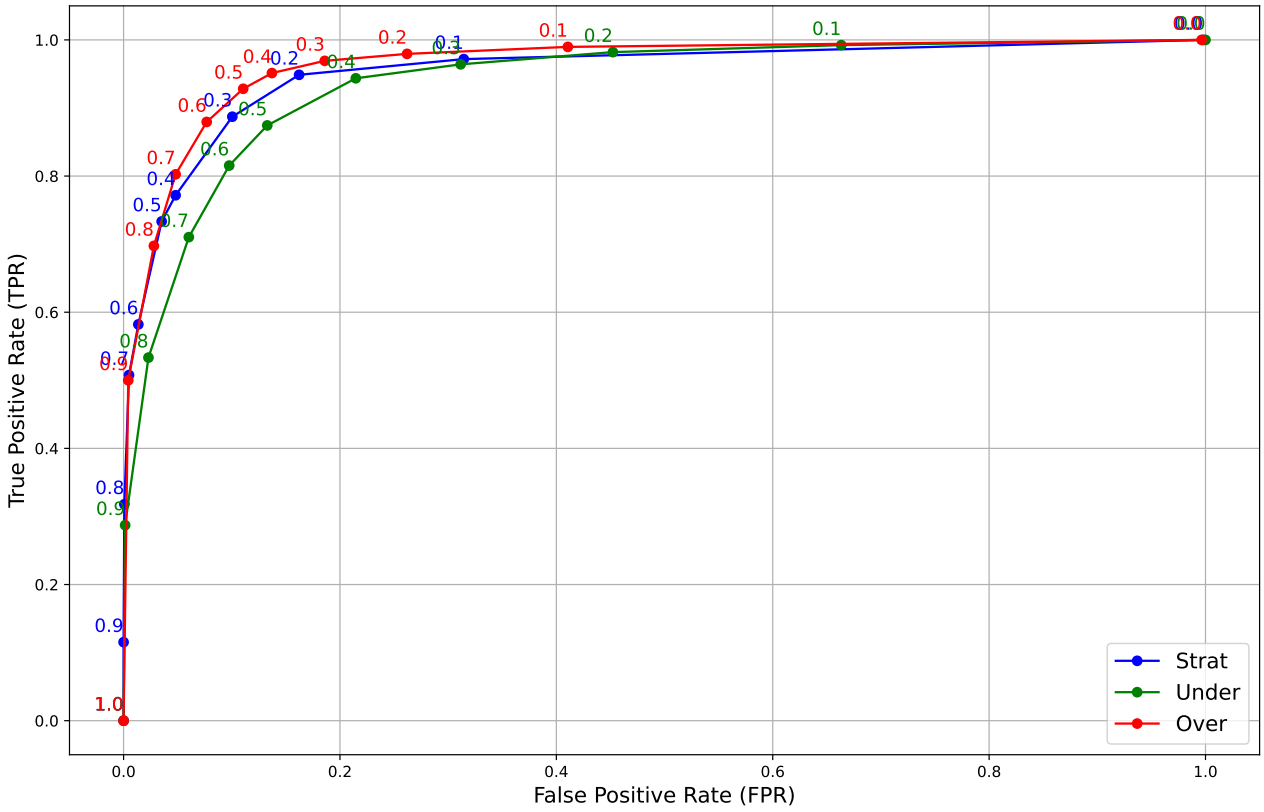

Figure 6.4: The effect of a threshold on a validation set for different random forest models.

7 Experiment results

This section outlines the results obtained from the models trained on undersampled, oversampled, and stratified datasets. The model is tested on two different test sets (random and newest data) with corresponding sets of hyperparameters to evaluate its applicability in real-world scenarios. Assessing the model's ability to correctly identify true negatives while maintaining a reasonable rate of false negatives is crucial in understanding the model's capability to reduce an analyst's workload by marking a limited number of alerts as positive.

7.1 Neural network

The neural network's performance results are collected across all experiments and compared by their performance within the same test set type (random vs. newest data) and according to the dataset sampling technique used for training (undersampled vs. oversampled vs. unmodified).

7.1.1 Performance on a random test set

The random test set consists of 1783 records, with 1393 negatives and 390 positives. This proportion reflects the label proportion of the dataset.

The results of the experiments were evaluated and compared with each other to investigate the effect of the dataset balancing techniques on the final result, as well as the effect of the test set selection. The comparison between these experiments can be summarized as follows:

- The model performed notably better when trained on the oversampled dataset regarding the lower number of false positives when compared to a model trained on an undersampled dataset. While both models had a similar number of false negatives, the model in the *OverRand* experiment was able to classify true negative labels more accurately. This difference led to the *OverRand* model's higher performance on the F1 score metric.

- The model in the *StratRand* experiment performs better across all metrics than the model in the *UnderRand* experiment. The *UnderRand* model training yields around 20% of false positives, whereas *StratRand* model has only 5% of false positives and *OverRand* model has around 14% of false positives. However, the *StratRand* model performs somewhat worse with class 1 (20% false negatives as opposed to 9% on *UnderRand* and *OverRand* experiment). While decreasing the number of false positives is important for the model's effectiveness, it should not be achieved by increasing the number of false negatives.

- As seen from the previous comparisons, the *OverRand* model has overall better performance than the *UnderRand* model, and the *UnderRand* model has more acceptable performance in terms of false negatives than *StratRand* model. However, the model from the *StratRand* experiment has the best overall performance across the experiments using a randomized test set due to its ability to distinguish best between benign and suspicious samples. This result is reflected in figure 7.1. Presumably, this model performance could be improved by adjusting the threshold to a lower value to increase its sensitivity to suspicious samples.

When the undersampling technique is applied, the number of benign samples is reduced, which might lead to the loss of information that the model could use to perform the classification of true negative samples more efficiently. Therefore, the model trained on an unmodified or oversampled dataset is able to reduce the number of false positives notably more than the model trained on the undersampled dataset. However, balancing the false positive rate (FPR) with the true positive rate (TPR) becomes more challenging on a stratified dataset. The model trained on a stratified dataset has somewhat worse TPR than other models. This difference presumably stems from the imbalance

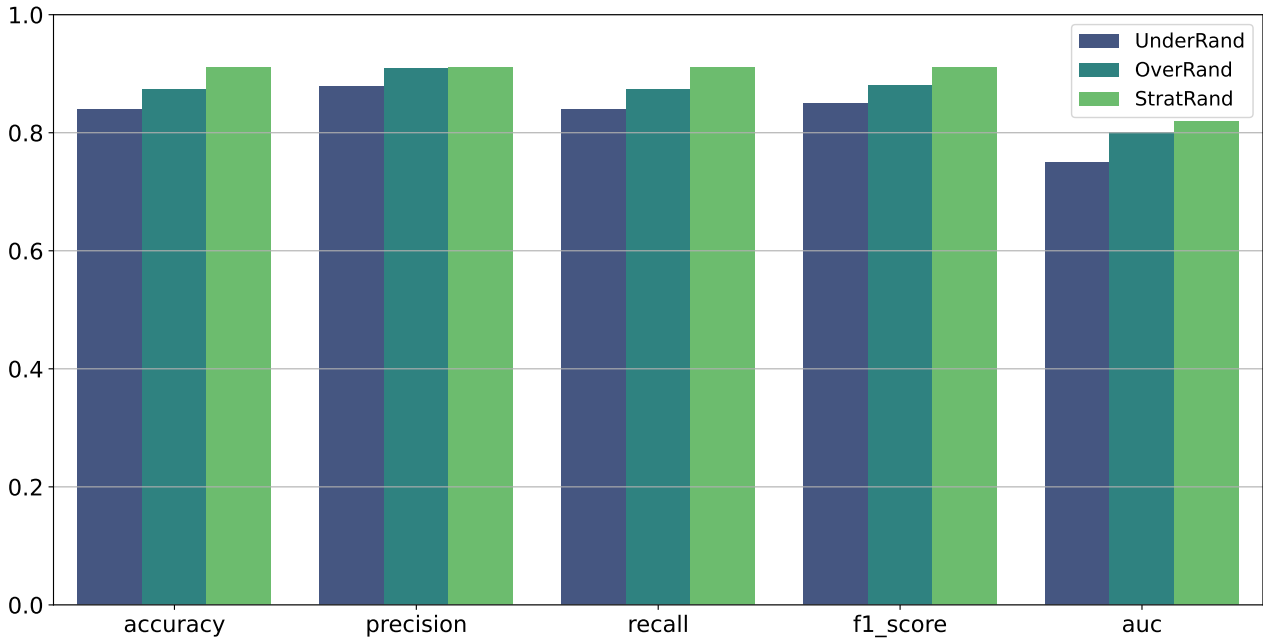

Figure 7.1: Model performances on a random test set.

of the information the model has on benign and malicious samples since no balancing technique is applied to the dataset. The relation between TPR and FPR for these experiments is reflected in figure 7.2.

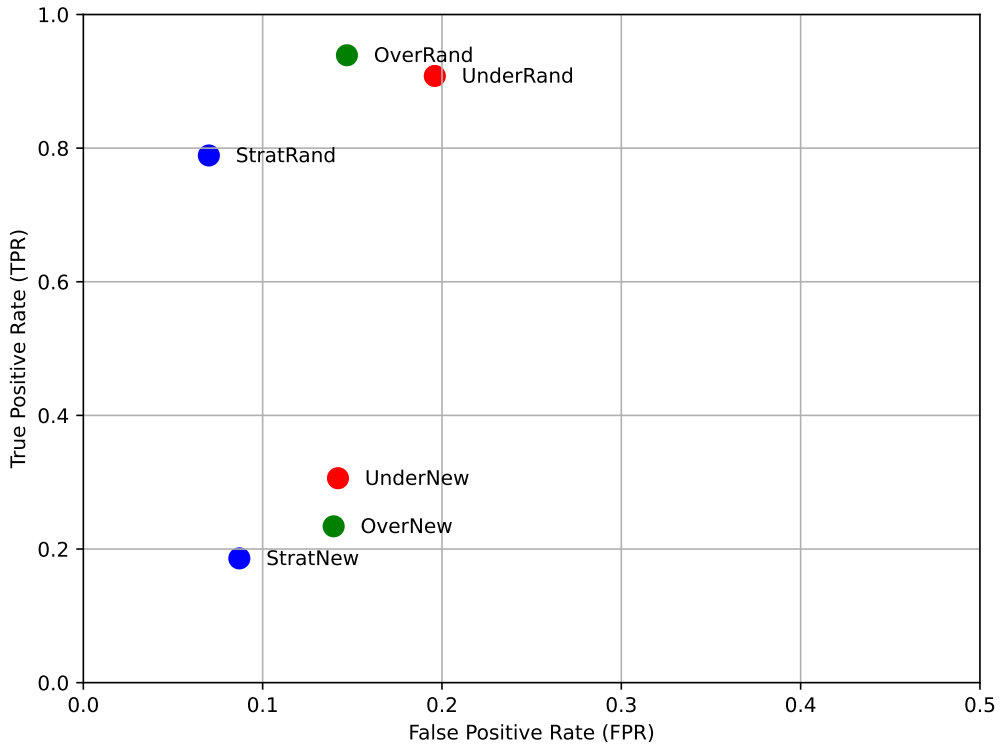

Figure 7.2: TPR vs FPR on the models assessed on different test sets with threshold 0.4.

In the real-world scenario, the analyst has to investigate each alert manually. Implementing a

model with any of the hyperparameter sets HP_u, HP_o, and HP_s provides a reduction in the false positives for around 80% on a random test set. Adjusting the threshold can significantly decrease this value; however, even a 50% reduction in false positives would be an acceptable result when compared with a situation when an analyst has to investigate 100% of alerts. In the case of the random test set, the model has the advantage of accessing the full diversity of data, which may not be the case with the newest data test set.

7.1.2 Performance on the newest data test set

In the second step of the experiment, the best models from the *UnderRand*, *OverRand*, and *StratRand* experiments are retrained on the training set left after the separation of the test set and a validation set. This step is required to exclude the overlap between the test set and training set since the test set is extracted in a different way. Next, the models are tested on a test set which consists of the newest data. Since the test set was extracted using a different method, the proportions of the classes are evaluated in the table 6.2. This evaluation demonstrates no significant difference between the proportions of the 0 and 1 classes.

The performance metrics for the experiments over the newest data test set are summarized in figure 7.3.

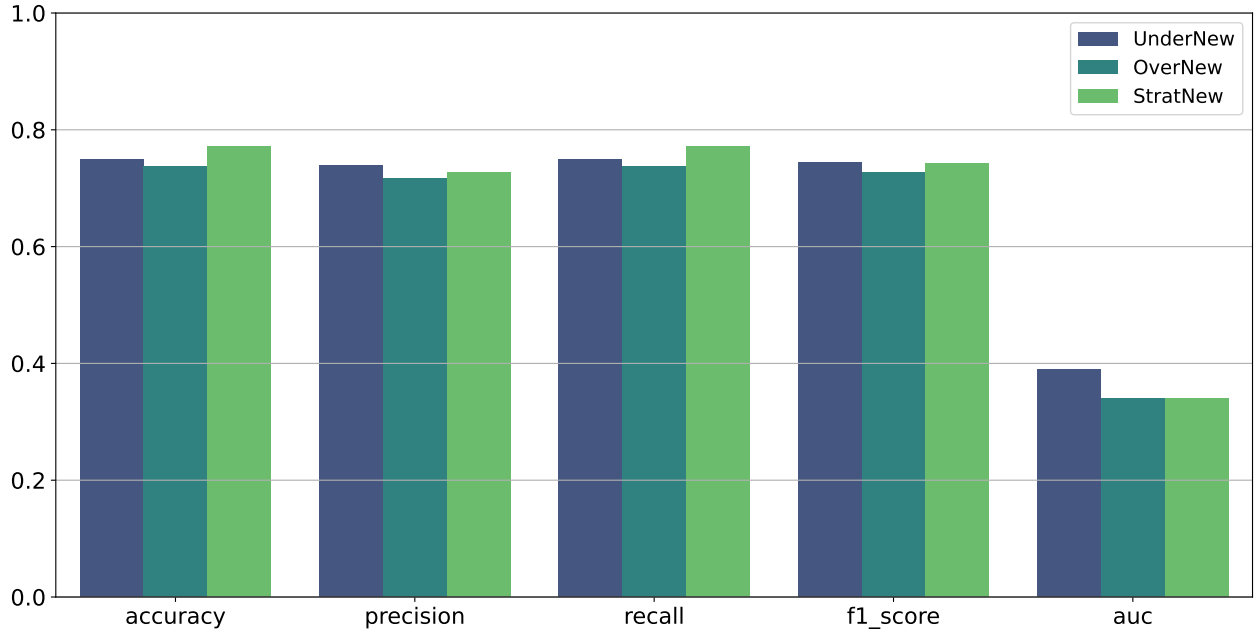

Figure 7.3: Model performances on the newest data test set.

The comparison between these experiments can be summarized as follows:

- The model trained on the undersampled dataset demonstrates the highest F1 score and the best sensitivity to the threshold tuning, and it demonstrates the best recall, i.e., the ability to identify the highest number of suspicious instances. However, the model is only able to identify around 30% of the total number of suspicious instances correctly. This hints at the model's limited ability to generalize on data, which can either stem from overfitting or significant differences between the training and test sets.

- All three models demonstrate similar results when it comes to the identification of the true negative instances. The model in the *UnderNew* and *OverNew* experiments identify 14% of non-suspicious instances as suspicious (i.e., false positive rate). In contrast, for the *StratNew*, the false positive rate is 8.7%.

- While *StratNew* model demonstrates the best false positive rate, its true positive rate remains the lowest (18.6%). In contrast, the model in *OverNew* experiment was able to identify 23.4%

of all suspicious instances, and the model in *UnderNew* experiment was able to identify 30.6% of total suspicious records.

The undersampling approach reduces the number of majority class instances, which could cause the model to learn more effectively from the minority class. This is why the undersampled model has the highest true positive rate but also suffers from a relatively high false positive rate, indicating that it is oversensitive to features associated with the minority class. Reducing the majority class instances might remove some of the essential information needed for the effective identification of true negative instances.

7.1.3 Performance comparison

The overall comparison of the metrics performed in all six experiments is reflected in figure 7.4.

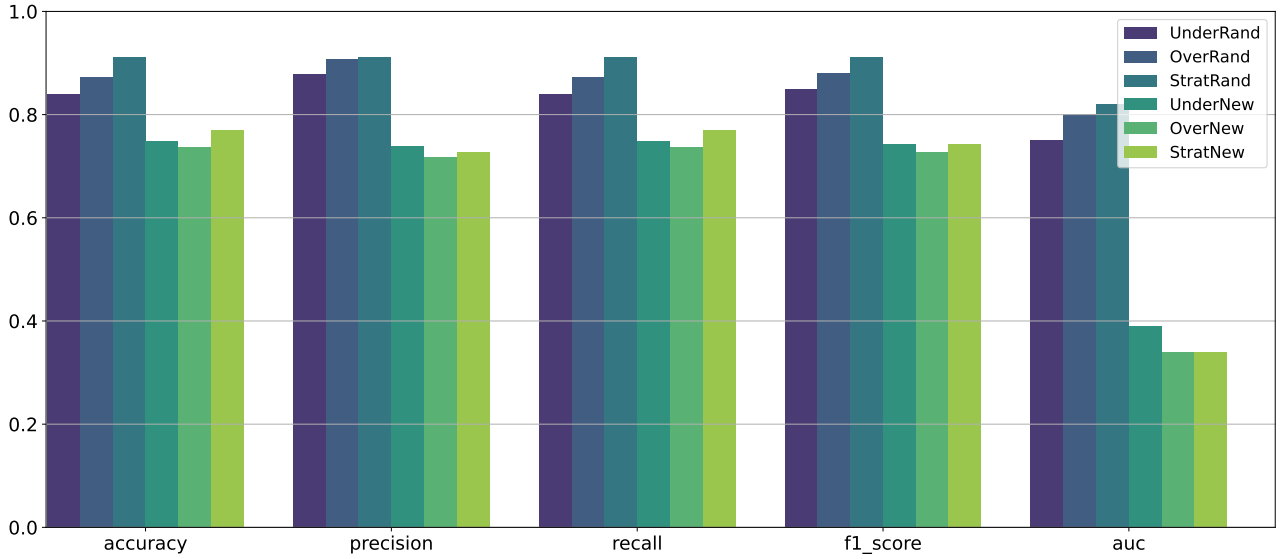

Figure 7.4: Experiment metrics.

All three models demonstrate notably worse performance across all metrics on the newest data test set when compared to the performance on the random test set. This discrepancy can be partially explained by the difference in the feature distributions across different test sets. In particular, the Jensen-Shannon Divergence (JSD) [23] was computed between the training set and both the most recent data set and the random test set, as shown in 7.5. The JSD value between the random test set and the training set is denoted as JSD_random, and the JSD between the newest data test set and the training set is denoted as JSD_newest.

The Jensen-Shannon divergence (JSD) measures the similarity between two probability distributions. It is symmetric and finite and provides a way to quantify their differences.

Given two discrete probability distributions P and Q, the JS divergence is calculated in two steps.

First, the average distribution is computed:

$$M = \frac{1}{2}(P + Q)$$

where M is the average of P and Q.

Next, the Jensen-Shannon divergence is calculated by the following formula:

$$\mathrm{JS}(P\|Q) = \frac{1}{2}\left(D(P\|M) + D(Q\|M)\right)$$

where $D(P\|M)$ and $D(Q\|M)$ are measures of how much P and Q differ from M, respectively.

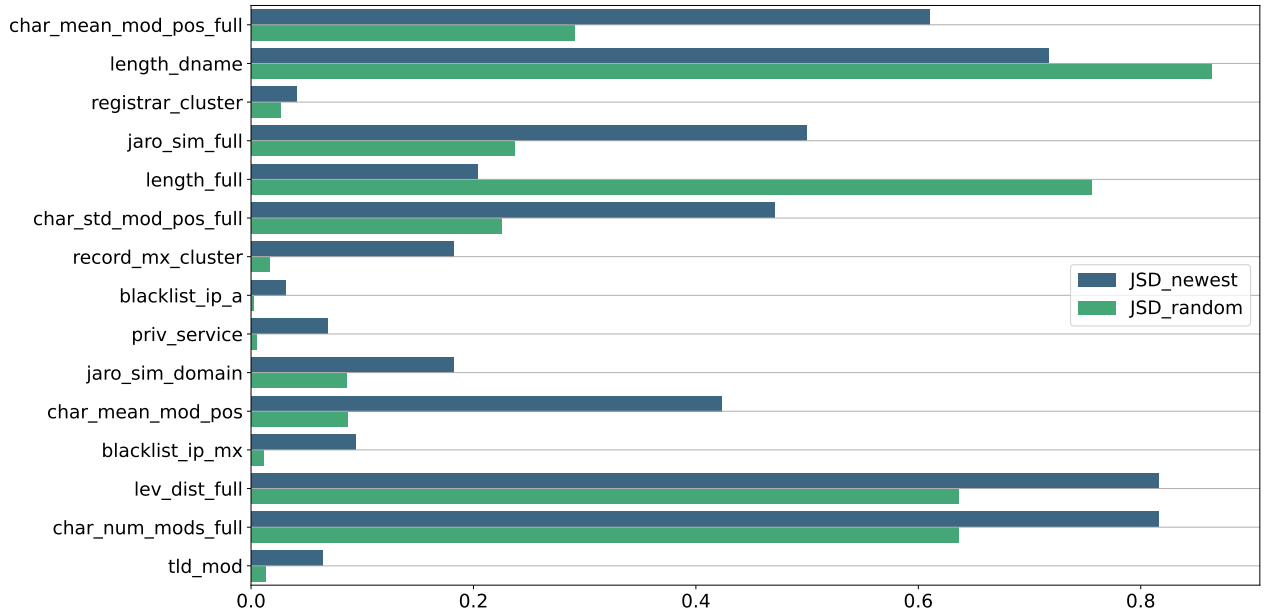

Figure 7.5: Jensen-Shannon divergence between the training and the most recent data sets.

The JSD value is symmetric, meaning that $\mathrm{JS}(P\|Q) = \mathrm{JS}(Q\|P)$, and it ranges from 0 to 1, where 0 indicates that the distributions are identical, and 1 indicates maximal divergence.

To assess the JSD between the training set and a test set, the JSD value for each feature was computed. As indicated in figure 7.5, the JSD values for multiple features are notably higher between the training set and the most recent data set than those observed between the training and random test sets. The JSD measures the dissimilarity between two probability distributions, and higher values indicate a more significant divergence between the feature distributions in the datasets. This divergence suggests that the statistical properties of the features in the most recent data set differ substantially from those of the training set, which likely contributes to the observed drop in model performance.

To compute the overall JSD value for each of the test sets, the weighted average of feature JSD values is computed where feature importance serves as a weight parameter:

$$\text{Weighted Average JSD} = \frac{\sum_{i=1}^{n} w_i \cdot \text{JSD}_i}{\sum_{i=1}^{n} w_i}$$

where
w_i is feature importance of the i-th feature,
JSD_i is Jensen-Shannon Divergence for the i-th feature,
n is the total number of features.

The overall JSD_newest value is 0.379, whereas the JSD_random value is 0.312, which shows a higher feature distribution difference between the training set and the newest data set.

The model was trained on the training set, assuming the feature distributions remain consistent. However, the significant divergence in feature distributions between the training and the newest data set, as reflected in the high JSD values, violates this assumption. As a result, the model's learned patterns from the training data may not be effective in generalizing the newest data set. In contrast, the relatively lower JSD between the training set and the random test set indicates a closer alignment in feature distributions, which explains the model's relatively better performance on this set.

This discrepancy highlights the importance of monitoring changes in data distributions over time, as distribution shifts can lead to degraded model performance. In this case, retraining the model with more recent data or incorporating techniques such as domain adaptation may be necessary to improve performance on the newest data set.

Additionally, when considering individual metrics, models trained on stratified datasets show

higher results on accuracy and precision metrics but lower recall metrics, which allows us to hypothesize that the models trained on this dataset raise fewer false positives but are more likely to have more false negatives. In other words, the model is able to identify true negatives with more certainty but is more likely to miss suspicious instances. This scenario reflects the imbalance of the dataset, where the number of suspicious samples is significantly lower than the number of benign samples.

The large disparity in the results obtained from the random test set and the newest data test set leads to the hypothesis that the temporal characteristics of the newest test set, in combination with the higher JSD value, cause significant deviation in the models' prediction and generalization ability.

7.2 Random Forest classifier performance

The random forest classifier showed overall similar performance in all cases of oversampled, undersampled, and stratified datasets when tested on both test sets. When it comes to the FPR and TPR rates, random forest demonstrates nearly identical tendencies across all experiments, with the performance difference fluctuating within 4%, and this difference could be attributed to the particular sample differences in the dataset. The TPR and FPR rates are reflected in table 7.1, where TPR and FPR for the neural network are denoted TPR_{nn} and FPR_{nn}, respectively, and TPR and FPR for the random forest are denoted TPR_{rf} and FPR_{rf}.

Experiment	TPR_{nn}	TPR_{rf}	FPR_{nn}	FPR_{rf}
UnderRand	91%	93%	20%	23%
OverRand	91%	95%	14%	14%
StratRand	79%	79%	5%	5%
UnderNew	30%	39%	14%	19%
OverNew	23%	26%	14%	13%
StratNew	19%	22%	9%	6%

Table 7.1: Description of the experiments.

Several conclusions can be drawn from the performance levels of the random forest and the neural network. First, the data patterns are likely to have low complexity since the neural network, being a more advanced structure, fails to show better results than the random forest model. This suggests that a more complex feature engineering approach could be explored that would show less obvious patterns in data, and these patterns could be discovered by the neural network more effectively. Next, similar performance suggests high consistency across feature importance ranking. While the feature importance ranking appears to be consistent with the manual analysis performed by the human operators, further detailed investigation is needed to fine-tune the feature importances. While both models seem to achieve decent performance on the random test set, which is likely to contain more diverse data than the newest data test set, further inspection is needed to understand the reasoning behind their failure to achieve similar results on the newest data test set.

8 Conclusion and future work

This chapter discusses the experiment's findings and proposes the direction of future work to improve the automated domain analysis capability. The findings demonstrate that a machine learning approach has the potential to reduce analysts' manual workload significantly, provided that the models can effectively capture complex data patterns. The results indicate that the performance of machine learning networks largely depends on the diversity and quality of the dataset used for training and consistency of the feature distributions, as well as the balancing techniques used for dataset preprocessing.

Specifically, the model showed acceptable performance for balancing false positives and false negatives when tested on a randomly selected dataset, suggesting its capability to generalize when exposed to data with sufficient diversity, which is consistent with training and testing sets. The neural network was able to minimize the number of false positive alerts by 80-95% while having a 91% true positive rate in two out of three experiments. However, its performance was notably poorer when evaluated on the newest, unseen data, highlighting limitations in adapting to recent changes or novel patterns. The best true positive rate obtained from the newest data test set did not exceed 31%, while false positives were reduced by the same 80-95%. This discrepancy emphasizes the importance of training models with adequately diverse and representative datasets to achieve consistent accuracy. This performance drop can be justified by the notable difference in feature distributions across the training and test sets; the feature distribution in the newest data test set differs from the training set more than the feature distribution in the random test set. In other words, the random test set feature distribution is more consistent with the training set, and the test set consisting of the newest data diverts more from the training set, and this difference is captured in the Jensen-Shannon Divergence (JSD) metric.

8.1 Further work

Various strategies can significantly enhance the effectiveness and performance of the machine learning model. This section outlines potential improvements to both the model and the dataset. These suggestions include dataset augmentation, feature engineering, a different approach to labeling, and accounting for temporal characteristics.

8.1.1 Dataset augmentation

It could be beneficial to augment the dataset to increase the number of instances representing the minority class through techniques other than the naive replication of minority entities. This can be achieved through generating synthetic data points using methods like the Synthetic Minority Oversampling Technique (SMOTE). Given a sample x_i from the minority class, SMOTE selects one of its k nearest neighbors x_j and generates a new synthetic instance by interpolation:

$$x_{\text{new}} = x_i + \lambda(x_j - x_i), \quad \lambda \in [0, 1]$$

, where λ is a random number between 0 and 1, and x_{new} is a synthetic sample that lies on the line segment joining x_i and x_j. This approach helps to create synthetic samples that introduce variability into the minority class, thereby improving the model's generalization capability.

SMOTE's primary advantage is that it generates new, diverse samples, reducing the risk of overfitting prevalent in random oversampling. This approach helps balance the dataset, allowing the model to learn better from underrepresented classes. In the given scenario, the simplest approaches—random oversampling and random undersampling—were used. The main drawback of random oversampling, however, is that duplicating the same instances may lead to overfitting in the model. While this technique helps balance the dataset, it may limit the model's ability to capture nuanced patterns within the minority class. Likewise, while random undersampling can mitigate overfitting by reducing the amount of redundant data in the majority class, it also risks discarding helpful information that

could contribute to model learning. In scenarios where the majority class contains diverse patterns, removing samples can lead to a loss of valuable data.

A given application scenario contains multiple interconnected features. Therefore, the dataset augmentation has to be performed carefully, considering the complex patterns of data in the dataset.

Another approach to avoid synthetic samples is to expand the dataset through the same process as already used by the existing intelligence platform. The dataset could be enriched through the use of popular legitimate domains (such as google[.]com or facebook[.]com) as the baseline for producing potential squatting domains through the same modifications as already performed on the platform. The reason for selecting popular domains is the high probability of the existence of lookalike domains since the attackers are more likely to attempt to spoof well-known domains. Therefore, there is a high potential for discovering a large number of domain pairs. This approach is similar to that used in existing research [40].

8.1.2 Feature engineering

Linguistic features were extracted from the domain names to capture the syntactic and semantic patterns indicating whether a domain is legitimate or malicious. These features included simple attributes such as the length of the domain name, the number of digits or special characters, and the use of common words or phrases. Such basic linguistic features are often correlated with the legitimacy of a domain. However, the potential for more complex linguistic features exists, and incorporating them could significantly enhance the model's predictive power, benefitting from the advanced potential of a neural network. For instance, domain names could be analyzed using natural language processing (NLP) techniques to infer deeper linguistic patterns. Some possible advanced linguistic features include:

- N-grams and word embeddings. Domain names can be tokenized into n-grams (sequences of n consecutive characters or words). Applying word embeddings makes it possible to capture semantic similarities between domain names that appear different but functionally related. This technique is used in numerous domain name analysis studies as a feature extraction method.

- Phonetic features. Phonetic algorithms could be applied to domain names to capture similarities in pronunciation. This could be particularly useful for detecting homophone-based attacks, where a malicious domain mimics the sound of a legitimate one. By identifying domains that sound similar to the client domains, the model could flag domains that might otherwise evade detection.

Furthermore, integrating additional data sources could enrich the dataset, providing more significant feature extraction and engineering potential. For instance, extracting website content analysis could provide additional signals about a domain's trustworthiness. Advanced NLP techniques, such as sentiment analysis, could assess the tone and language of the associated website content, offering further insights into the domain's legitimacy.

Recall that certain features were excluded from the dataset due to their unreliability. Improving the reliability of the sources of these features (e.g., automated scripts used for data collection) could significantly expand the model performance as those features could be used.

Regular updates and retraining of the model are also essential to maintain its effectiveness. As new data becomes available, the model should be periodically retrained to incorporate recent trends and patterns. This approach ensures that the model adapts to changes in the underlying data distribution, enhancing its predictive capabilities. Figure 8.1 shows an example of such a process. The steps of the workflow, as depicted in the diagram, are explained below:

- At stage (1), the alert is received as usual. After the initial screening by an analyst for legitimacy (as in the original analysis process), the domain is fed to the model for analysis (2).

- Regardless of the model's output, an analyst is tasked with manually investigating (3) the alert. This step ensures that human oversight remains a part of the process. The manual investigation serves two primary functions: verifying whether the alert was labeled correctly and identifying possible cases where the model might have misclassified the alert.

- The alert is added to the training set if the analyst confirms that the model's label is accurate (5). This reinforces the model's understanding of similar alerts as the training set grows with verified examples of both legitimate and non-legitimate cases.

- If the analyst finds the model's label is incorrect, the alert is added to the training set with a corrected label(4.2). This allows the model's performance on incorrectly labeled data to be tracked and analyzed. Misclassifications offer critical insights into the model's weaknesses, and adding these cases to the test set ensures that they will be specifically targeted during the next round of model evaluation. These misclassifications can be used during the model tuning (5) procedure to understand the reasons behind the incorrect labeling, allowing for a better adjustment of the model structure and hyperparameters.

This workflow integrates human oversight with machine learning, promoting continuous improvement and ensuring that correct and incorrect classifications are systematically used to refine the model.

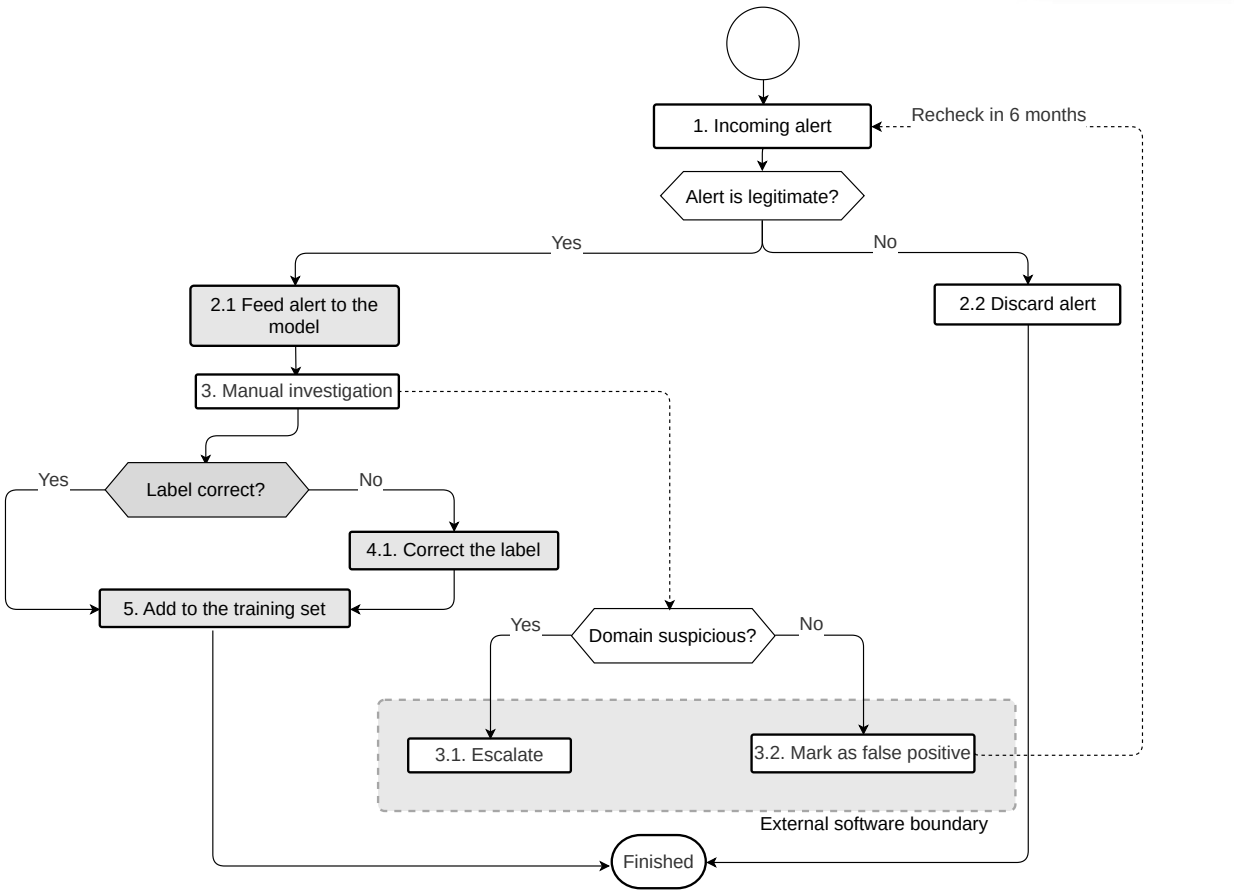

Figure 8.1: Possible workflow for model tuning and retraining.

8.1.3 Different approach to labeling

The research used binary classification for the machine learning training. Alternatively, a multiclass labeling technique could be applied, consisting of five classes that represent the level of urgency associated with the verdict provided by analysts during the evaluation process. The level of urgency ranges from lowest to highest, and it could potentially offer greater insight into the features contributing to the final decision. Several thresholds could be introduced, indicating a separation between the classes. This multiclass approach would enable a more nuanced analysis of the data, capturing variations in urgency and enhancing the model's interpretability and utility. The thresholds could be fine-tuned to reflect distinct decision boundaries, allowing the classification model to better differentiate between cases with different priorities. Moreover, training the model with multiple classes would provide analysts with a clearer understanding of how certain features impact various levels of urgency. This could

help in making more informed decisions and effectively allocating resources based on the predicted urgency level.

8.1.4 Temporal characteristics

In certain situations, multiple semi-suspicious domain names may be registered by the same individual or have other notable commonalities. Upon noticing numerous related records, an analyst might infer that these registrations indicate a coordinated effort, potentially aimed at a phishing attack or other malicious activities. However, the current model evaluates each domain individually, limiting its ability to detect such connections. Typically, these suspicious registrations may occur within a relatively short timeframe, making it feasible to detect such patterns through temporal analysis. A more advanced system should be capable of identifying such correlations autonomously, even in cases where data is randomized. By incorporating this capability, the detection framework could significantly improve its ability to recognize coordinated campaigns and provide timely alerts, thereby enhancing its effectiveness in preempting security threats.

8.2 Practical implementation

The analysis of the model's performance highlights a critical distinction between false positives and false negatives. While false positives may indicate reduced system effectiveness, leading to an increased workload for analysts, the implications of false negatives are more alarming from a security standpoint. False negatives can allow potentially malicious activities to go undetected, posing significant threats to the organization and its stakeholders.

The current model is tailored to align with the specific analysis processes employed by the company's analysts. However, it is essential to note that different organizations may adopt varying practices, necessitating additional tuning and re-evaluation of feature importance. Such adjustments would ensure that the model remains effective across different operational contexts.

Moreover, the system's performance could be enhanced by incorporating additional uncollected and critical features for comprehensive evaluations. Examples of these features include the content of web pages and the email addresses of registrants, when available. It is acknowledged that these features may not always be accessible for automatic collection. For instance, while a registrant's email address may provide little information in some cases, it could also reveal valuable insights in others. An analyst might discover that a registrant's email address is associated with multiple squatted domains, which raises suspicions about the registrant's intentions to gather illegitimate traffic or engage in spam campaigns. Conversely, a registrant's email address might be identified as a domain reseller on a dubious, country-specific forum, indicating potential malicious intent.

The complex nature of this information, derived from various sources, makes it inherently unpredictable and unsuitable for automatic collection. Ideally, the system should be able to recognize other potential issues within the records and prompt analysts for manual investigations. However, a single uncollectable feature raising suspicion could lead to a false negative if not adequately handled. One proposed solution would be to flag such records as "requiring more information." For example, if a registrant's email address is available (indicating that a privacy service is not utilized), the system could recommend that an analyst conduct a manual investigation into this feature.

While achieving full automation remains unlikely due to data limitations, the system can be refined to identify features requiring manual review and alert analysts accordingly. This adjustment would enhance the model's overall performance and empower analysts with actionable insights. Figure 8.2 reflects the potential implementation scenario.

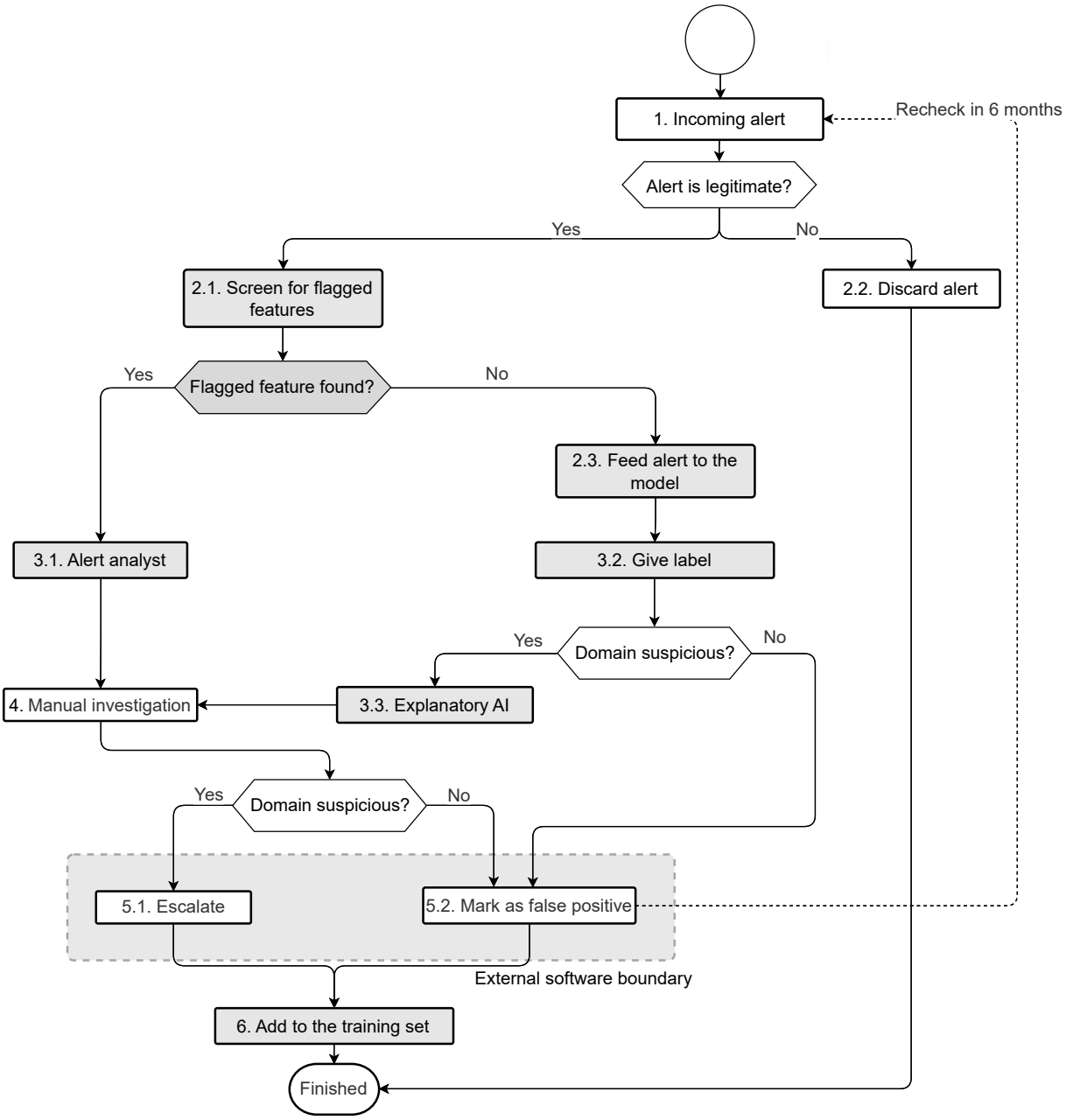

Figure 8.2: Possible workflow for model implementation in practice.

- At stage (1), the alert is received as usual. After the initial screening by an analyst for legitimacy (as in the original analysis process), the domain is fed to the algorithm for analysis (2.1) to find flagged (i.e., requiring manual review) features in the alert.

- If no flagged feature is found, the alert is fed to the model (2.3) for evaluation and labeling (3.2). Otherwise, a manual investigation is required by the analyst (3.1)

- If a flagged feature is found or a model has labeled the alert as positive (suspicious), an analyst is tasked with manually investigating (4) the alert. To simplify the manual investigation, a technology such as explanatory AI (XAI) could be implemented after the labeling stage. The term XAI applies to a set of processes and methods that allows human operators to understand the results produced by machine learning algorithms in a user-friendly way [14]. Since the decision-making process of neural networks is not transparent, XAI could provide human-readable insights into how the model predictions are made, adjusting for further adjustment of the model if necessary.

Therefore, using XAI, manual investigation could be significantly improved since only partial investigation might be needed.

- The alert is processed as usual in the external software, either escalated (5.1) or marked as false positive (5.2) according to the analyst verdict.
- The alert is added to the training set (6). If the analyst finds the model's label is incorrect, the alert is added to the training set with a corrected label. After a sufficient number of new samples is added to the dataset or a certain period has passed (for instance, 500 new samples added or six months passed), the model should be retrained to maintain the data diversity and enhance the robustness of the model.

8.3 Discussion

In conclusion, this research has demonstrated the feasibility of implementing machine learning for the automation of data analysis, significantly reducing manual workload by correctly identifying benign instances (thus reducing the number of false positive alerts) if certain adjustments and improvements are implemented. However, the evolving nature of the underlying data creates a need for a continuous learning process to maintain model relevance and accuracy. Establishing clear guidelines for domain-specific analysis is crucial to ensure consistent and effective model performance that would accurately reflect the decision-making process of analysts.

The results of the experiments demonstrate that the models are able to identify the true negatives (i.e., benign instances) but struggle to recognize true positives (i.e., suspicious instances) when given a dataset with different feature distribution diversity. This prompts the conclusion that the model needs more diverse samples of the suspicious domains to improve the learning. Currently, the models tend to dismiss the majority of suspicious domains as false negatives, leading to the conclusion that the model can't distinguish between suspicious and non-suspicious domains with sufficient precision when given samples that reflect real-world conditions.

The transparency of machine learning decisions is imperative to establish trust and enable analysts to interpret model outputs meaningfully. While the results are promising, further investigation is essential to refine the methodologies, improve model efficiency, and ensure the reliability of the findings. This work highlights the potential of machine learning in data-driven environments, emphasizing the importance of adaptability, transparency, and ongoing research for practical implementation. Such implementation is feasible if the needs for model update and retraining are taken into account during its lifecycle.

Bibliography

[1] Norah Saud Al-Musib, Faeiz Mohammad Al-Serhani, Mamoona Humayun, and N.Z. Jhanjhi. Business email compromise (BEC) attacks. *Materials Today Proceedings*, 81:497–503, 4 2021. doi:10.1016/j.matpr.2021.03.647.

[2] Ahmed Aleroud and Lina Zhou. Phishing environments, techniques, and countermeasures: A survey. *Computers & Security*, 68:160–196, 7 2017. doi:10.1016/j.cose.2017.04.006.

[3] Arbitration and Mediation Center. WIPO Domain Name Decision: D2000-0847, 10 2000. URL: https://www.wipo.int/amc/en/domains/decisions/html/2000/d2000-0847.html.

[4] Aniruddha Bhandari. Feature Scaling: Engineering, normalization, and Standardization (Updated 2024), 8 2024. URL: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

[5] Bruno Bianchi, Rodrigo Loredo, María Da Fonseca, Julia Carden, Virginia Jaichenco, Titus Von Der Malsburg, Diego E. Shalom, and Juan Kamienkowski. Neural bases of predictions during natural reading of known statements: An Electroencephalography and eye Movements co-registration study. *Neuroscience*, 519:131–146, 5 2023. doi:10.1016/j.neuroscience.2023.03.024.

[6] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. 01 2011.

[7] Leslie Daigle. RFC 3912: WHOIS Protocol Specification. URL: https://datatracker.ietf.org/doc/html/rfc3912.

[8] Paolo Dal Cin, Jacky Fox, Harpreet Sidhu, and James Nunn-Price. State of Cybersecurity Resilience 2023 — Accenture. Technical report, Accenture, 2023.

[9] dnsrecon — Kali Linux Tools. URL: https://www.kali.org/tools/dnsrecon/.

[10] Elceef. GitHub - elceef/dnstwist: Domain name permutation engine for detecting homograph phishing attacks, typo squatting, and brand impersonation. URL: https://github.com/elceef/dnstwist.

[11] Vishalkumar Ravindrakumar Gajjar and Hamed Taherdoost. Cybercrime on a Global Scale: Trends, Policies, and Cybersecurity Strategies. 1 2024. doi:10.1109/icmcsi61536.2024.00105.

[12] Jannat Hossain and Alex L. White. The transposed word effect is consistent with serial word recognition and varies with reading speed. *Cognition*, 238:105512, 9 2023. doi:10.1016/j.cognition.2023.105512.

[13] What are DNS records? — IBM. URL: https://www.ibm.com/topics/dns-records.

[14] Ibm. Explainable AI, 8 2024. URL: https://www.ibm.com/topics/explainable-ai.

[15] IP blacklist & Email blacklist check. URL: https://dnschecker.org/ip-blacklist-checker.php.

[16] Aminollah Khormali, Jeman Park, Hisham Alasmary, Afsah Anwar, Muhammad Saad, and David Mohaisen. Domain name system security and privacy: A contemporary survey. *Computer Networks*, 185:107699, 2 2021. doi:10.1016/j.comnet.2020.107699.

[17] Soyoung Kim, Sora Lee, Geumhwan Cho, Muhammad Ejaz Ahmed, Jaehoon Jeong, and Hyoungshick Kim. Preventing dns amplification attacks using the history of dns queries with sdn. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*, pages 135–152. Springer, 2017.

[18] Lars Kröhnke, Jelte Jansen, and Harald Vranken. Resilience of the Domain Name System: A case study of the .nl-domain. *Computer Networks*, 139:136–150, 7 2018. doi:10.1016/j.comnet.2018.04.015.

[19] Marc Kührer, Christian Rossow, and Thorsten Holz. *Paint It Black: Evaluating the Effectiveness of Malware Blacklists*. 1 2014. doi:10.1007/978-3-319-11379-1_1.

[20] Ashley G. Lewis and Marcel Bastiaansen. A predictive coding framework for rapid neural dynamics during sentence-level language comprehension. *Cortex*, 68:155–168, 7 2015. doi:10.1016/j.cortex.2015.02.014.

[21] Chaoyi Lu, Baojun Liu, Yiming Zhang, Zhou Li, Fenglu Zhang, Haixin Duan, Ying Liu, Joann Qiongna Chen, Jinjin Liang, Zaifeng Zhang, Shuang Hao, and Min Yang. From WHOIS to WHOWAS: A Large-Scale Measurement Study of Domain Registration Privacy under the GDPR. 2 2021. doi:10.14722/ndss.2021.23134.

[22] Cláudio Marques, Silvestre Malta, and João Paulo Magalhães. DNS dataset for malicious domains detection. *Data in brief*, 38:107342, 10 2021. doi:10.1016/j.dib.2021.107342.

[23] M.L. Menéndez, J.A. Pardo, L. Pardo, and M.C. Pardo. The Jensen-Shannon divergence. *Journal of the Franklin Institute*, 334(2):307–318, 3 1997. doi:10.1016/s0016-0032(96)00063-4.

[24] Faisal Misle. What is Typosquatting? Everything you need to know and how to prevent it, 3 2024. URL: https://blog.redsift.com/brand-protection/what-is-typosquatting-everything-you-need-to-know-and-how-to-prevent-it/.

[25] Application Layer Protocol: DNS, Sub-Technique T1071.004 - Enterprise — MITRE ATT&CK®. URL: https://attack.mitre.org/techniques/T1071/004/.

[26] P.V. Mockapetris. Domain names - concepts and facilities. Technical report, Internet Engineering Task Force, 11 1987. URL: https://www.rfc-editor.org/rfc/rfc1034, doi:10.17487/rfc1034.

[27] Abdallah Moubayed, Mohammadnoor Injadat, Abdallah Shami, and Hanan Lutfiyya. Dns typosquatting domain detection: A data analytics & machine learning based approach. 12 2018.

[28] MX Lookup Tool - Check your DNS MX Records online - MxToolbox. URL: https://mxtoolbox.com/.

[29] Bilal Naqvi, Kseniia Perova, Ali Farooq, Imran Makhdoom, Shola Oyedeji, and Jari Porras. Mitigation strategies against the phishing attacks: A systematic literature review. *Computers & Security*, 132:103387, 9 2023. doi:10.1016/j.cose.2023.103387.

[30] Bruce Nikkel. Registration Data Access Protocol (RDAP) for digital forensic investigators. *Digital Investigation*, 22:133–141, 8 2017. doi:10.1016/j.diin.2017.07.002.

[31] Erdal Ozkaya. *Practical Cyber Threat intelligence*. BPB Publications, 5 2022.

[32] Gopinath Palaniappan, Sangeetha S, Balaji Rajendran, None Sanjay, Shubham Goyal, and Bindhumadhava B S. Malicious domain detection using machine learning on domain name features, Host-Based features and Web-Based features. *Procedia computer science*, 171:654–661, 1 2020. URL: https://www.sciencedirect.com/science/article/pii/S1877050920310383, doi:10.1016/j.procs.2020.04.071.

[33] RandomForestClassifier. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[34] Fiach Reid. *Ping, DNS, and WHOIS*. 1 2004. doi:10.1016/b978-155558315-6/50013-x.

[35] Dan Shiebler. Generative AI enables bad actors to create more sophisticated attacks, 12 2023. URL: https://abnormalsecurity.com/blog/generative-ai-chatgpt-enables-threat-actors-more-attacks.

[36] StandardScaler. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[37] Matija Stevanovic, Jens Myrup Pedersen, Alessandro D'Alconzo, Stefan Ruehrup, and Andreas Berger. On the ground truth problem of malicious DNS traffic analysis. *Computers & security*, 55:142–158, 11 2015. doi:10.1016/j.cose.2015.09.004.

[38] Matija Stevanovic, Jens Myrup Pedersen, Alessandro D'Alconzo, and Stefan Ruehrup. A method for identifying compromised clients based on DNS traffic analysis. *International Journal of Information Security*, 16(2):115–132, 5 2016. doi:10.1007/s10207-016-0331-3.

[39] Xiaoqing Sun, Zhiliang Wang, Jiahai Yang, and Xinran Liu. Deepdom: Malicious domain detection with scalable and heterogeneous graph convolutional networks. *Computers & Security*, 99:102057, 9 2020. doi:10.1016/j.cose.2020.102057.

[40] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The long "Taile" of typosquatting domain names. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 191–206, San Diego, CA, August 2014. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/szurdi.

[41] Marcus Taft and Kenneth I. Forster. Lexical storage and retrieval of prefixed words. *Journal of verbal learning and verbal behavior*, 14(6):638–647, 12 1975. doi:10.1016/s0022-5371(75)80051-x.

[42] Michal Tonhauser and Jozef Ristvej. Cybersecurity automation in countering cyberattacks. *Transportation research procedia*, 74:1360–1365, 1 2023. doi:10.1016/j.trpro.2023.11.283.

[43] VirusTotal. URL: https://www.virustotal.com/gui/home/url.

[44] Han Wang, Zhangguo Tang, Huanzhou Li, Jian Zhang, and Cheng Cai. DDOFM: Dynamic malicious domain detection method based on feature mining. *Computers & security*, 130:103260, 7 2023. doi:10.1016/j.cose.2023.103260.

[45] Zheng Wang. Use of supervised machine learning to detect abuse of COVID-19 related domain names. *Computers & Electrical Engineering*, 100:107864, 3 2022. doi:10.1016/j.compeleceng.2022.107864.

[46] Ünal çavuşoğlu. A new hybrid approach for intrusion detection using machine learning methods. *Applied Intelligence*, 49, 07 2019. doi:10.1007/s10489-018-01408-x.