POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

Development and Evaluation of Behavioral Models for the Detection of Malicious Web Accesses

Supervisors
Prof. Marco Mellia
Nikhil Jha, Ph.D.
Dr. Alberto Verna

Candidate
Francesco Gallo

Academic Year 2024-2025

Abstract

In recent decades, Internet services have become an essential component of modern society, providing a wide spectrum of applications. The diversity of users, some of whom have limited security knowledge, increases the likelihood of accessing malicious websites, specifically developed by attackers to steal sensitive information, money, or personal data.

Traditional defense countermeasures, such as blocklists, – manually curated lists of known malicious domains maintained by specialized companies – rely on comparing a website's identifier against those contained within the blocklist. Although this approach is widely adopted, it is purely reactive: it protects users only after the malicious site has been detected and included in the list.

To overcome this limitation, smart blocklists have been proposed. These rely on algorithms, heuristics, or machine learning models that evaluate features of a web resource to classify it as malicious or benign. However, because these characteristics are directly linked to the resource itself, attackers can deliberately manipulate them to avoid detection.

This thesis explores an alternative approach: instead of analyzing the intrinsic characteristics of a website, it examines the possibility of classifying a web resource by evaluating users' specific navigational patterns. In other words, the idea is to demonstrate that a malicious access is not isolated with respect to the previous ones, on the contrary, they are strongly correlated and can therefore be predicted.

To validate this idea, we developed a behavioral recognition model starting from a dataset of real users' browsing histories. The work starts with the collection and evaluation of open-source blocklists in order to understand their reliability and the extent of false positives. On this basis, we labeled the browsing data and extracted relevant features, which were then used to train and compare different machine learning models.

The results show that behavioral models can effectively distinguish malicious from benign accesses, providing preventive protection that is more robust than both traditional blocklists and smart blocklists. This work represents a step toward more resilient online security solutions, capable of anticipating attacks rather than merely reacting to them.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Overview

During the last decades, the Internet has established itself as a crucial component of modern society. Through the capability of providing instant access to information and facilitating global interaction and collaboration, the Internet has become an indispensable asset for a large user base, impacting on different aspects of daily life, such as communication, business, learning, and interaction. Consequently with the increasing of activity, Internet also become a valid target for malicious actors. The same connectivity that enables global interaction inherently introduces exploitable vulnerabilities to attack systems, data, and users. These attacks involve websites that mimic legitimate services with the purpose of stealing information, malicious websites that stealthily download spyware, malware, or ransomware exploiting victim's device, and fraudulent websites that intentionally deceive the user in order to get access to personal data, money, or other sensitive information.

The main target of these attacks is Personal data. Personal data is personally identifiable information about the victim, which can be exploited to perform more targeted attacks such as identity theft or fraud. For instance, personal data may include person's name, surname, or job position that may be used by an attacker to write a targeted email in order to convince the victim of being an acquaintance of his and asking for favors, like money or privileged access to the company where the victim works. Other than personal data, Internet communication produces a large quantity of data relating the user or his behavior during an online communication or session. The most common example is *cookies*, small pieces of text which can contain user identifiers, history of interactions and authentication tokens. If an attacker obtains this information, they can potentially impersonate the user for an entire session. To address this type of threat, it is necessary to adopt some countermeasures.

1.2 Blocklists

A *blocklist* contains a list of malicious identifiers that are compared with the identifier of the resource requested by the user. Depending on the desired level of detail, a blocklist

is based on

- *Domains*: identifier of a website.
- *URLs*: identifier of a web page.
- *IPs*: identifier of a machine in the network.

If the identifier is found within the blocklist the access is blocked. For their simplicity and efficiency in implementation, updating, and execution, blocklists are the most widely used tools against malicious access. Among the most popular blocklists is *Google Safe Browsing* (GSB) [14]. It operates at the URL level and is updated daily. At every access, GSB compares the URL with its own list and blocks the access if it finds a match.

A further and final detail, necessary for the purposes of this thesis, concerns the user access to a blocklists. In particular, an *open-source blocklist* is defined as one that makes its list of malicious resources public. GSB does not have this property. However, blocklists are subject to an inherent limitation. Indeed, we can define as a *window of exposure* the period during which the user can be exposed to a malicious website because the blocklist does not yet recognize the website as malicious. This period cannot be absent because:

1. The website must be recognized as malicious. Sometimes, this step coincides with a malicious access by a victim.
2. The administrator of the blocklist must be aware of the maliciousness of the website.
3. The user must properly update the blocklist.

Throughout this period, and excluding the final phase (which may depend on the user themselves) a malicious site is able to operate without restrictions. From this, it is clear that a solution must be found to reduce the window of exposure as much as possible. Academic literature explores a wide range of solutions involving algorithms, heuristics, and artificial intelligence models that operate to improve various aspects of blocklists. These studies aim to automatically detect and mitigate misclassified domains [15], analyze the characteristics of the web page, such as HTML structure, hyperlinks, and input forms [46], inspect URL syntax elements to identify malicious domains [55], or adopt hybrid approaches that combine different features and techniques [20, 45]. Finally, the literature has highlighted the possibility to mitigate the window of exposure problem, by supporting proactive detection of malicious domains [77, 79], but unlike blocklists they have neither a standardized structure nor a standardized operation. By definition, choosing the set of attributes used to analyze a domain plays a fundamental role in the effectiveness of classification. If this set is too selective, some malicious domains may evade them and bypass security measures. Consequently, if it is too loose, some benign sites may be blocked due to misclassifications. Finally, the exclusive use of site attributes could lead attackers to create ad-hoc websites to evade them. For instance, an index of a site's maliciousness could be a poorly maintained and low-quality structure, but the attacker, knowing that this feature is taken into account for site classification, could design a well-structured site so that it is mistakenly classified as benign.

1.3 Thesis Objectives

The main objective of this thesis is to propose and evaluate a behavioral model that analyzes user access patterns to accurately classify them for the detection of malicious websites, aiming to overcome some of the typical limitations of blocklists. We explore the idea of classifying a web resource by evaluating users' specific navigational patterns. In this context, we define a navigational pattern as a set of ordered accesses performed by a user. These accesses can be grouped according to similar characteristics with other ordered sets of accesses. In other words, the idea is to demonstrate that a malicious access is not isolated with respect to the previous ones and, on the contrary, they are strongly correlated. Consequently, we explore the possibility of deriving a certain number of features from a user's browser history that can be interpreted by a behavioral recognition model to predict, with a high grade of accuracy, whether the next access will be malicious. With regard to the window of exposure, the model, similarly to [46, 55], aims to classify a site through its characteristics, thus providing proactive protection. Unlike the latter, however, the idea is to take into account the user's user behavior. The rationale behind this choice is that a model capable of recognizing behavioral patterns before malicious access occurs is trained using features that belong to the user and cannot be modified by an attacker. In particular, the thesis aims to:

- analyze the limitations of traditional blocklists;
- design a behavioral model that exploits user navigation patterns as a source of useful information for identifying potential threats;
- implement and test the model, comparing its performance with approaches based on blocklists;
- evaluate the effectiveness of the model classification through standard classification metrics, such as accuracy, precision, recall, and F1-score.

1.4 Thesis outline

This thesis is structured as follows.

- 2 **Chapter 2** : presents definitions, examples, and academic works for the understanding of both the technical and theoretical contexts of the thesis.
- 3 **Chapter 3**: describes *Pimcity*, the dataset composed of browsing histories used in this thesis. The chapter also includes collection methods and data exploration.
- 4 **Chapter 4**: explores the method for the collection of several open-source blocklists up to realize a reliable dataset. Furthermore, we explores the classification and labeling process of the Pimcity entries
- 5 **Chapter 5**: explores the motivation, and the generation of relevant features from the datasets.

- 6 **Chapter 6**: presents the machine learning models used to classify malicious accesses, and the metrics used to evaluate them.
- 7 **Chapter 7**: reports and discusses experimental results.
- 8 **Chapter 8**: summarizes the main contributions of the study, discusses limitations, and outlines potential directions for future research.

Chapter 2

Background and related works

This chapter aims to explain the fundamental concepts for understanding the purpose, the context, and the problems that this thesis wants to address. To improve readability and simplify the structure, we divide the chapter into two sections. The Fundamental concepts section aims to illustrate a more detailed overview of Internet communication with a particular focus on the definitions of direct/indirect access, URL, initiator, and malicious access. In the second section, we review academic papers related to this thesis, in order to understand how the literature has addressed the problem, what solutions and results have been proposed, and which limitations still remain to be overcome.

2.1 Fundamental concepts

This section recalls the fundamentals of web communication by providing an high level explanation of how client-server communication works. Within this framework, the client is typically represented by the user's device, which starts a request for a resource available on the Internet, such as a webpage. On the other hand, the server is a remote machine that hosts the resource and provides access to it.

2.1.1 Example of communication

Any Internet communication follows these elementary steps:

- **Step 1**: Client request: The client sends a request for a specific resource. Any request is composed of: protocol (i.e. HTTP/HTTPS), method (GET, POST, PUT, DELETE), resource's URL, header (i.e. authorization tokens), and additional data (i.e. parameters in the body of the request if the method is PUT or POST).
- **Step 2**: Server processing: The server receives the request and verifies if all constraints are satisfied to provide the response. Constraints to satisfy include: authentication, authorization, verification of the parameters received, querying to database or external services, and error handling in case of invalid input.
- **Step 3**: Server response: The server constructs the response along with a status code that anticipates the outcome of the response (200 OK, 400 Bad Request,

500 Internal Server Error). In addition, the header is assembled with the necessary parameters that describe the response, and the body contains the resource requested by the user.

- **Step 4**: Client receives and renders the resource: Finally, the client receives the resource and its browser processes it as an HTML file. Firstly, the DOM (Document Object Model) is constructed, which is fundamental as it describes the structure of the file. Consequently, the browser applies all the additional elements necessary to display the resource to the user. For instance, additional elements include: CSS files for styling, or JavaScript scripts for functionality.

2.1.2 URL

URL (Uniform Resource Locator) plays a pivotal role during communication. Every resource on the web is associated with a unique *URL*. In other words, the URL is the resource's identifier, but, in addition, it even provides information about the communication between the client and the server. For example, URL specifies details about the protocol used to contact the host, which port is used, how to reach the resource, etc. In summary, if the client wants to access a resource, he must know the correlated URL.

The general structure of the URL is the following.

$$\underbrace{\text{protocol}}_{\text{http, https, ftp}} \text{ :// } \underbrace{\text{user:password@}}_{\text{john:123}} \underbrace{\text{domain}}_{\text{www.example.com}} \underbrace{\text{: port}}_{\text{443}} \underbrace{\text{/path}}_{\text{/path/to/file}} \underbrace{\text{?query}}_{\text{lang=it}} \underbrace{\text{\#fragment}}_{\text{33}}$$

- *Protocol*: It is the protocol used for the communication (i.e. HTTP, HTTPS, FTP). It establishes a set of mandatory operations performed by both the client and the server to communicate.
- *User:password* (optional): User's credential. Nowadays, they are rarely used because an attacker could intercept them during a communication (*URL sniffing*).
- *Domain*: Identify the server that host the resource.
- *Port* (optional): It is the port where the host received the request. If it is not specified, the standard port declared by the protocol is used, for instance, 80 HTTP, 443 HTTPS.
- *Path*: hierarchical path to reach the resource.
- *Query* (optional): key-values parameters interpreted by the server to specify conditions to obtain the resource.

  For instance, if the URL is `https://domain.com/resource?id=123&lang=it`, the server will use `id=123` to select the resource with ID equals to 123 and `lang=it` to return it in italian.
- *fragment* (optional): reference to a specific section of the resource.

2.1.3 Indirect accesses and initiator

Rarely do web pages consist of a single HTML file. They usually have a complex structure that involves images, CSS files, scripts, and other multimedia. When a user requests a web page, the server is responsible for providing all additional resources required to correctly respond to the user. Therefore, even in the absence of a direct request from the user, numerous resources are still provided, since they are implicitly encompassed within the initial request. The URL that initiates the communication is designated as the *initiator*. In summary, accessing a resource means initiating a process of requests that starts with the main request (*initiator*) and expands to all the necessary secondary resources (indirect accesses). In this context, we distinguish first-party accesses, which involve all the resources served by the same domain as the initiator, and third-party accesses,which include resources required by the initiator but served from different domains.

Example of communication with indirect accesses

- **Step 1**: Client request (direct access): The client sends a request to the server for a resource.
- **Step 2**: Server sends HTML: The server responds with the resource, but this one requires additional elements to be rendered. For example, a CSS file.
- **Step 3**: Client implicitly requests additional resources (indirect access): The client receives the document and automatically requests the additional resources referenced in it.
- **Step 4**: Server sends additional resources: The server provides the requested CSS file.
- **Step 5**: Client renders page: Finally, the client processes the initial resource using all the additional elements.

2.1.4 Malicious access

An access is defined as *malicious* when the initial request sent by the client or any other additional resources used to fulfill the first one contains malicious resources with the purpose of compromising data, privacy, or the user's control.

Malicious accesses classification

Similarly, to the aforementioned distinction between direct and indirect accesses, we may classify two categories of malicious accesses.

1. *direct malicious access*: The initial request directly points to a malicious resource.
   (a) The user receives a link to a malicious web site and he clicks on it requiring the a malicious resource. (i.e. `https://malicious.com/`).

(b) The server responds with a page designed to steal credentials or sensitive information. To illustrate this point, imagine a fraudulent banking interface that mimics an official banking site with an input form where the user is prompted to enter their banking data.

(c) By visiting the page, the user inserts his information which are forwarded to the attacker.

In this case, the *malicious access is direct* because the user initiated the request.

2. *indirect malicious access*: The initial request is legitimate, but the calling chain caused from additional resources contains a URL that points to a malicious resource. For example, an attacker is able to inject a malicious script within a web site with the purpose of executing it on the user's browser when the web page is requested.

   (a) The user requests a legitimate page: `https://legit.domain.com/resource`.

   (b) The page includes a malicious script `https://legit.domain/malicious-script.js`. For instance, the script could have been injected by an attacker in the comment section of the legitimate web page. When the user's browser renders the HTML, it automatically executes the script (*Stored XSS attack*).

   (c) The injected or compromised script automatically sends the user's sensitive information to the attacker.

   In this case, malicious access is indirect, because it has been triggered by a malicious dependency of a legitimate page.

Targets

Common targets of attackers exist.

- *Session cookies*: temporary text of files automatically generated by the server to remember user details such as preferences or passed action on the site. Most websites use cookies to identify users' sessions; attackers can impersonate users' requests by stealing victims' cookies.
- *Personal data*: For personal data we include all the data able to recognize unambiguously a person. This cluster comprises name, surname, job position, phone number, email address, but even passed Internet researchers and interests. They are used by the attacker to gain specific information about an user to repeat a more accurate attack or impersonate him.
- *Credentials*:Usually involve username and password and are used by the user to prove his identity to the server. However, a deceived user may insert them into the form of a malicious website. If the attacker obtains those information, he can virtually impersonate the user.
- *localStorage* or *sessionStorage* contain user's sensitive data or metadata stored as key-values pairs used to improve navigation on Web. If an attacker was able to recover these values, they can lead to privacy breaches, or identity theft.

Kinds of attacks

An attacker has many possibilities to force a user to access a malicious resource. However, it is possible to group these attacks into two main categories.

- *Exploiting software vulnerabilities:* This type of attack usually requires a deep understanding of the systems used by the user for online communication. The attacker exploits weaknesses in the software or protocols to trigger undesired behaviors, in the meantime, the user may be unaware that he is under attack. Some examples of these attacks involve Shadow server, DNS cache poisoning, Man-in-the-Middle, and Cross-Site Request Forgery (CSRF).

  In our context, for instance, an attacker could force the injection of malicious scripts into legitimate websites making them malevolent.

- *Exploiting human vulnerabilities:* By human vulnerabilities is referred to *psychological or social pressure*. Attacks that take advantage of people's vulnerabilities are called *social engineering attacks*. In this case, the attacker forces the victim to perform an unwanted action through deception. The most popular example of this type of attack is *phishing*. Phishing occurs when an attacker sends an email containing a link to a malicious web page and convinces the victim to click it and provide sensitive information.

In summary, whether by exploiting software weaknesses or human behavior, attackers aim to gain unauthorized access or trick users into compromising security.

2.2 Literature Background

This section presents a series of articles and related works that address the subsequent steps of the thesis, starting with understanding user behavior on the web, as fundamental prerequisite for developing predictive models capable of estimating the risk that a user will access malicious content. Literature focuses on two aspects: the analysis of browsing history and domain classification, supported by *machine learning techniques*. This review highlights how various studies contribute knowledge and methodologies useful for constructing a robust behavioral model.

2.2.1 Risk Prediction Based on Browsing History

One of the first studies to explore the link between user behavior and the risk of exposure to malicious content is [16]. By analyzing users' browsing histories, researchers aim to identify those who, due to poor habits or limited cybersecurity knowledge, are more likely to visit malicious sites. The approach focuses on predicting general behavior rather than classifying individual URLs and introduces new features, such as the timestamp. This line of research is further explored by [62], which extends the prediction horizon to one month. Here, user risk is estimated by analyzing past behavior and cybersecurity knowledge, using a *random forest classifier* to determine the probability of exposure to malicious content. This approach highlights how non-sequential models are the first

choice if the goal is to predict general behavior. [44] addresses the classification in a more mathematical approach. Researchers discovered that by analyzing how predictable a website is, they could improve the model's performance by bringing together URLs from the same website. These results suggest that domain-level aggregation can improve the model's predictability performances.

2.2.2 Domain Classification as Support for Prediction

The prediction of the risk of occurring in a malicious access cannot be separated from the ability to classify visited domains. Several studies have addressed this problem by providing useful tools and methodologies. [43] demonstrates how, starting from a dataset containing username, timestamp, URL and other information such as mouse movement and keyboard activity , it is possible to recover the actual duration of visits to each URL. Using a machine learning model, domains are classified into categories such as Current, Next, or Past events, allowing the reconstruction of detailed browsing sequences. This approach provides valuable information about the context in which users navigate, improving the understanding of at-risk behaviors. Complementarily, [52] addresses the complexity of multi-tab browsing. Using LSTM and Word2Vec, the model predicts URL sequences considering the time spent on each and backtracking behavior. Identifying macro-behaviors such as *purposive*, *targeted*, and *explorative browsing* is crucial to distinguish the various users' behavior adopted during the navigation. Earlier studies, such as [19], highlights the importance of additional behavioral features, such as re-visitation rate and hierarchical site classification via DMOZ (now known as Curlie). This information helps to understand browsing routines and better define the behavioral context of each user. Finally, more recent classification approaches, such as [47, 80], introduce deep learning methods to categorize websites. Both articles demonstrate the ability to recognize the thematic area of a website by trending topics. Website categorization can be a crucial feature for predicting users' interests.

2.2.3 Choice of Machine Learning Algorithms

The selection of a predictive model plays a central role in building a reliable system. [63] show that *LSTM and stacked RNNs* can be used to predict future events based on past sequences. Despite the high accuracy achieved, models can be subject to deliberate evasion, a factor to consider when modeling web risk. Other approaches, such as [39, 49] explore various strategies for classifying suspicious behaviors using LSTM, HMM, or XGBoost and introduce techniques such as SMOTE for handling imbalanced datasets. These studies provide guidance on selecting algorithms that balance accuracy and interpretability.

2.3 Discussion

Although malicious access is a popular problem in the cybersecurity field, thus is not enough to provide sufficient countermeasures. On the contrary, even in the last years, malicious accesses have become a priority threat against the users' web activity. To contextualize this, referring to some reports, due to *generative AI*, online phishing attacks

increased by 1265% between 2024 and 2025 [61]. In addition, the number of phishing emails that use *infostealer* to steal user credentials is increasing by 84% weekly [78]. 72% of business leaders reported an increase in cyber-risks, with the malicious use of generative AI as their primary concern. Furthermore, over 40% of organizations have experienced social engineering attacks in the past year [27]. The report [48] highlights a shift towards proactive security, with a focus on understanding attacker behavior in order to implement effective countermeasures. As users rely more and more on the Internet, which offers applications that require the intensive use of personal data or sensitive metadata; this can only increase the number of reasons that lead an attacker to carry out illegal actions. In addition, the spread of generative AI allows to produce complex websites on large scale that are more convincing and more tailored to the victim. [59]

To mitigate the risk is necessary to:

- understands the hierarchy between initiator and secondary resources,
- monitor and log relationships between requests,
- apply different appropriate countermeasures,

Chapter 3

Pimcity Dataset

This chapter provides a detailed analysis of the datasets used for the study carried out in this thesis. The analysis begins with data collection; subsequently, every dataset field will be explored separately. Finally, all the limitations will be highlighted.

3.1 Initial Presentation

Pimcity is composed of browser histories, collected by voluntary participants who installed a browser extension that collects all their accesses. This opportunity was proposed by Politecnico di Torino [21] for a period of more than six months. At the end of this period, it has been recorded the contribution of 345 *users* and a total of 392,902,891 *accesses*. However, to focus on essential information, only a subset of access characteristics was recorded. In particular, each entry in the dataset contains five specific characteristics of the access.

- *URL*: Although in the dataset it was defined as a URL, only the domain of the web page was stored. Any other information, such as protocol, path, port, etc., was discarded during the acquisition.

- *username*: This is the identifier of the user who made the request. To preserve privacy, all users were anonymized using specific suites of k-anonymity algorithms [54].

- *time*: Timestamp of access in the format `YYYY-MM-DD hh:mm:ss`.

- *initiator*: If this field is NULL, the stored domain is considered the initiator. Otherwise, the column specifies the initiator that requested the resource.

- *type*: The type of resource: image, script, CSP report, etc. Any type will be explained in the appropriate subsection 3.3.2.

username	time	url	type	initiator
ae4dced3-d9b3-4db6-a242-d6fcad26ebaf	2022-08-21 16:16:46	static.xx.fbcdn.net	image	www.facebook.com
ae4dced3-d9b3-4db6-a242-d6fcad26ebaf	2022-08-21 16:16:46	scontent-maa2-2.xx.fbcdn.net	xmlhttprequest	www.facebook.com
ae4dced3-d9b3-4db6-a242-d6fcad26ebaf	2022-08-21 16:16:46	scontent-maa2-1.xx.fbcdn.net	xmlhttprequest	www.facebook.com
ae4dced3-d9b3-4db6-a242-d6fcad26ebaf	2022-08-21 16:16:46	scontent-maa2-1.xx.fbcdn.net	xmlhttprequest	www.facebook.com
ae4dced3-d9b3-4db6-a242-d6fcad26ebaf	2022-08-21 16:16:46	scontent-maa2-1.xx.fbcdn.net	xmlhttprequest	www.facebook.com

Table 3.1. Example: 5 rows in the dataset

3.2 Data cleaning

The distinction between direct and indirect accesses is imperative in the development of a behavioral recognition model, since direct accesses more accurately reflect users' intent, while indirect ones represent only additional accesses made by the host to correctly send to the user the directly requested resource. For this reason, we introduce a subset containing only the direct accesses of *Pimcity*: *Pimcity-mainframe*. The *Pimcity-mainframe* dataset plays an essential role in achieving a full understanding of the concepts of *Pimcity*, strictly bonded to the users' habits and behavior. This is due to the fact that *Pimcity-mainframe* excludes a significant amount of automatic generated requests for additional resources.

3.2.1 Missing values

Before proceeding with the dataset analysis, it is essential to perform data cleaning, which involves identifying and handling errors, suspicious values, or missing values. This process ensures that the data will be reliable and ready for meaningful analysis and next steps.

As a primary rule, all data fields, except for the initiator, must be non-nullable. In non-technical vocabulary, the values contained in any entry, but the initiator, are not optional, and its absence indicates an error. Therefore, the entry is considered incomplete and is excluded from the dataset. All columns respect this condition, but the URL column contains 93,417 null values (almost 0.02% of *Pimcity*). This presence of missing values in the URL column is probably caused by a bug that occurred during data acquisition, then the rows result unusable for our purposes, and was consequently deleted from the dataset.

3.2.2 Inappropriate values

The last access in *Pimcity* appears with a suspected value of 11 August 5790. Focusing on the *Pimcity-mainframe* accesses, we obtain a reasonable range of dates – from 06 May 2022 to 13 November 2022 – which we consider to be the correct interval, and all accesses in *Pimcity* outside this range are discarded. The results of the corrections applied to missing and inappropriate values are as follows: the original entries of 392,902,891 were adjusted to 392,807,209 (99.99% of Pimcity).

3.3 Data exploration

At this point, we can conduct a more detailed examination of the characteristics of *Pimcity*'s fields.

3.3.1 URL

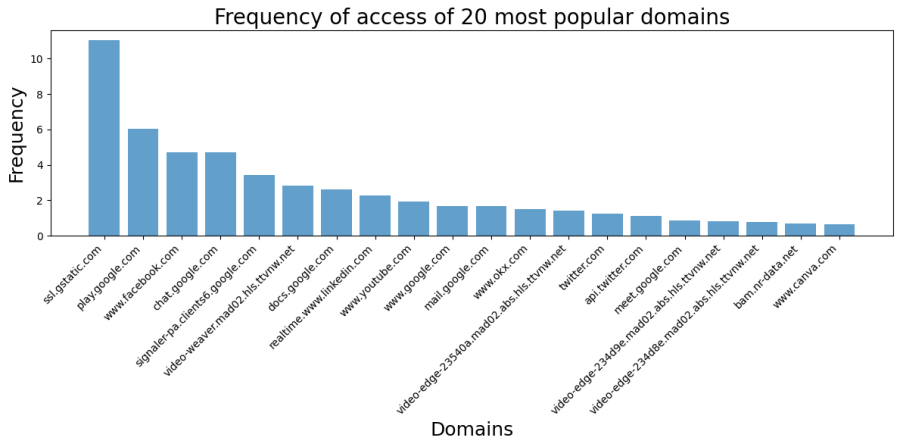

Figure 3.1. Most popular URLs in Pimcity, them total accesses.

As illustrated in the Figure 3.1, the most visited URLs in the dataset are auxiliary domains, such as `ssl.gstatic.com`, which serve static content, including images, CSS, and JavaScript. However, these requests reflect technical necessities rather than users' direct browsing choices.

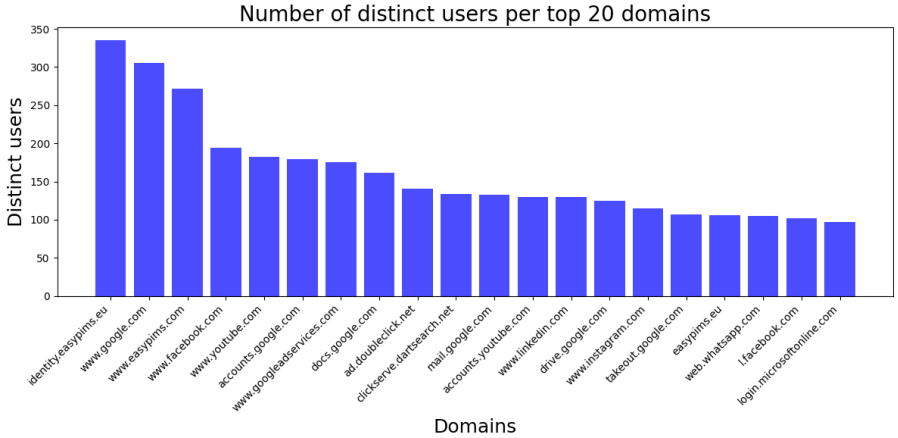

Figure 3.2. Most popular URLs in Pimcity-mainframe, them total accesses.

Sorting the *Pimcity-mainframe*'s domain for distinct users who made access to them (Figure 3.2), we observe that *easypims.eu* is the most popular domain, followed by

google.com, indicating an elevate use of research-related platforms and general web browsing activities. Other popular domains involve different *google.com* subdomains such as *accounts.google.com*, and *docs.google.com*, showing a significant use of Google services. Social media sites like *youtube.com*, *facebook.com*, *whatsapp.com* and *instagram.com* express that social interaction remains an important aspect of user activity, although it is secondary to work and learning platforms. Finally, the presence of *linkedin.com*, *microsoftonline.com*, suggests that professional networking are also frequently used.

Overall, this analysis indicates that the population primarily engages with a mix of search engines, social media, academic, and communication tools, with a strong inclination towards Google-related services.

3.3.2 Username

The dataset consists of 345 different users. Each user is properly anonymized to ensure privacy and security, while is maintained an identifier to allow recognition.

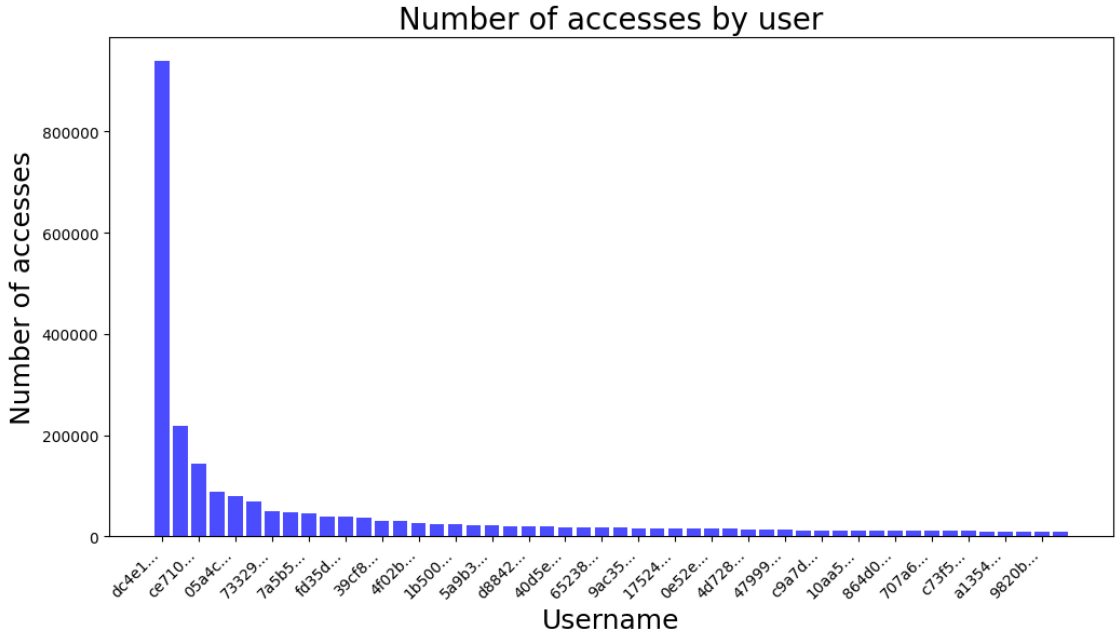

Figure 3.3. Users ranked per number of direct accesses.

In order to gain a more comprehensive insight into user behaviour, please refer to the Figure 3.3. This figure illustrates the user who has made the most direct accesses. However, the behavior of every user in terms of number accesses is highly variable. This variability allow us to conduct a comprehensive study encompassing different browsing patterns.

3.3.3 Time

The data collection period, which started on 6 May 2022 and ended on 13 November 2022, ensures a continuous collection for slightly over six months.

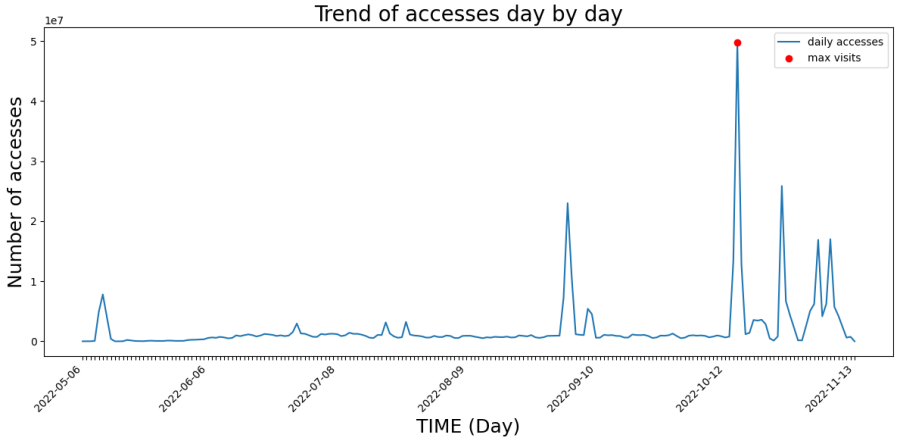

Figure 3.4. Trend of accesses (day by day)

The Figure 3.4 shows the number of accesses per day and, except for some peaks, the number of accesses per day is stable. To investigate these outliers, it is necessary to analyze the user's behavior in the Pimcity mainframe.

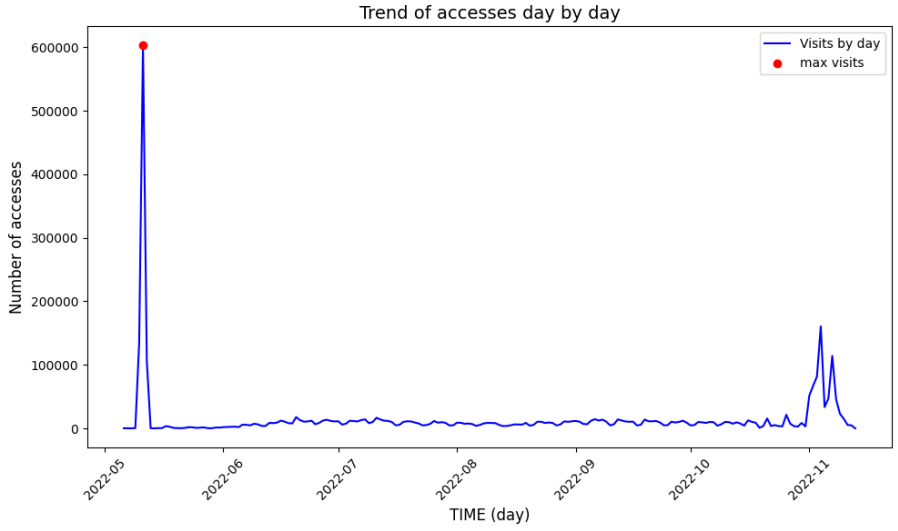

Figure 3.5. Trend of direct accesses (day by day).

The peak visible in the Figure 3.5 shows that the user `dc4e199b-4c5f-4986-b57a-2aaa01b6eeef` carried out 602,503 accesses out of a total of 602,747 (almost 99%). It is not possible for

a human being to execute such a number of accesses; therefore, this is suspected to be automated traffic or a bug in data collection. In both cases, the user is excluded from the database due to anomalous behavior. It is important to note that the suspicious behavior of the user `dc4e199b-4c5f-4986-b57a-2aaa01b6eeef` is also illustrated in Figure 3.3. This user, who is at the top of the list for number of accesses, performs a significantly high number of accesses. This number is strongly misaligned to the trend of the remaining users.

3.3.4 Type

Type represents the kind of resource that the user wants to access. There are 14 different types, defined as follows:

- *main_frame*: Request for the main document of the web page. It also refers to the first resource requested by the user.
- *sub_frame*: web page or HTML document loaded inside the mainframe. For example, an advertisement.
- *xmlhttprequest*: Used for APIs or dynamic data, it refers to HTTP request sent in the background by JavaScript.
- *image*: Request for an image.
- *media*: Request for multimedia files such as audio or video.
- *script*: Request for executable file.
- *ping*: Lightweight request used to send telemetry or tracking data.
- *stylesheet*: Request for a CSS file.
- *font*: Request for a font file. Used for text rendering.
- *history*: Navigation performed via the browser's history.
- *csp_report*: Report sent by the browser when a Content Security Policy (CSP) rule is violated.
- *object*: Request for content loaded via the `<object>` or `<embed>` tag.
- *webbundle*: Request for a WebBundle package, a format that allows grouping multiple web resources into a single file.
- *other*: Requests that are not in any other category.

The following Figure 3.6 shows the popularity of the various types in Pimcity: `xmlhttprequest` plays a central role.

With a total of 302,651,506 requests in the dataset `xmlhttprequest` represents almost 77% of Pimcity's accesses.

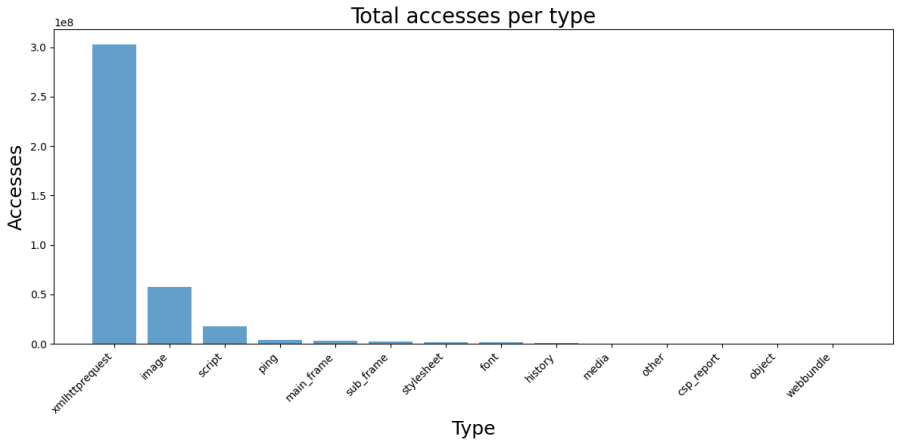

Figure 3.6. Type popularity per number of accesses.

3.3.5 Initiator

The initiator is crucial to understand who among the hosts required a malicious resource if it occurs. In Pimcity, there are only 90,072 different initiators, but they are sufficient to generate more than 98.96% of the traffic. The remaining 1.04% of the traffic is composed of direct requests without initiator. Therefore, most of the traffic was automatically requested by hosts rather than users. However, some initiators are more popular than others. The Figure 3.7 shows who the 20 most popular initiators in the dataset are.

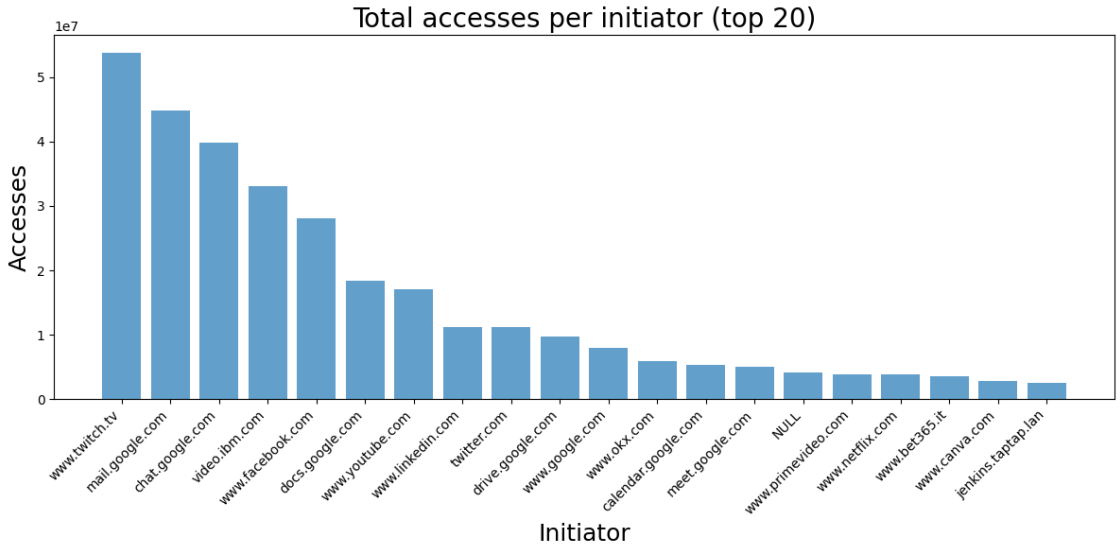

Figure 3.7. Top 20 initiators.

An interesting consideration concerns the relationship between initiators and URLs, and between initiators and types, as shown in Figure 3.8. The goal is to understand whether popular URLs gain their popularity because their initiator is popular as well, or if an unpopular initiator makes a URL popular because it requires a large number of resources from that URL.

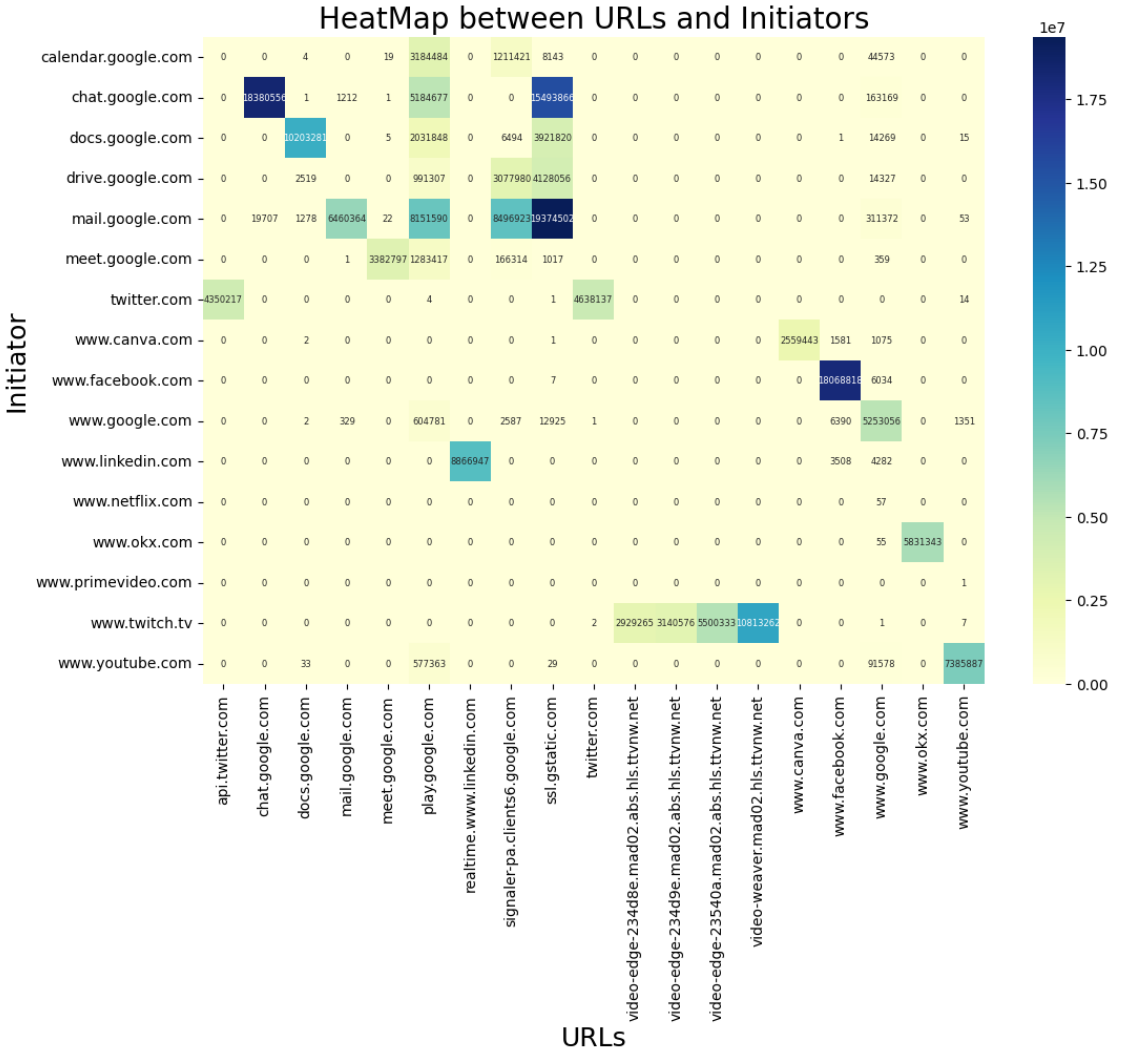

Figure 3.8. Correlation heatmap between initiator domains and accessed URLs.

The heatmap in the Figure 3.8 highlights the correlation between URL and initiator and it is a demonstration of the massive number of requests that any initiator executes. However, initiators have different behaviors. URLs like `google.com`, `ssl.gstatic.com`, or `play.google.com` have been called from different initiators such as `calendar.google.com`, `chat.google.com`, `docs.google.com`, `drive.google.com`, etc.

Instead, other initiators generate a localized traffic of resources. For example, `linkedin.com` only points to `realtime.www.linkedin.com`.

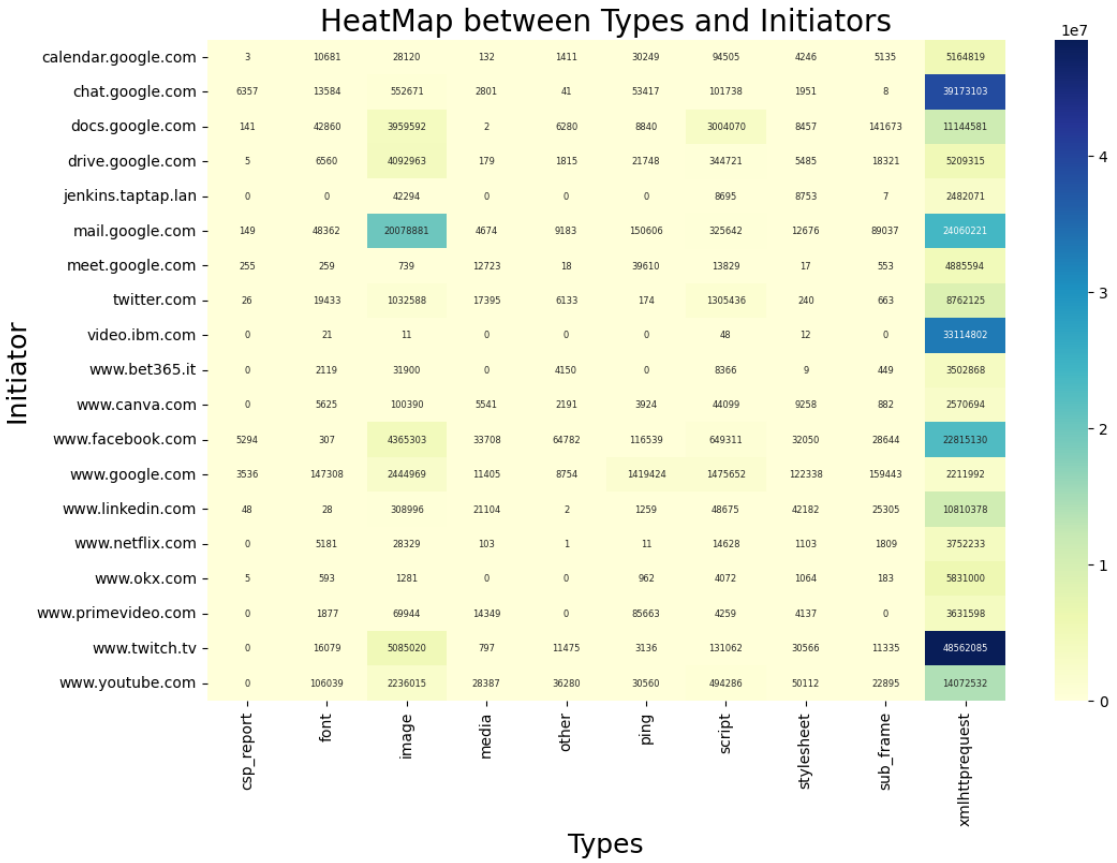

Figure 3.9. Correlation heatmap between initiator domains and accessed types.

The Figure 3.9 shown the correlation between initiator and types requested per initiator. Every popular initiator requires almost every type of resource. It means that the additional resources' traffic, is independent by the initiator that requires it. It is worth noting that the `xmlhttprequest` request type is the most requested among all initiators. The `xmlhttprequest` mechanism is a tool that allows clients and servers to change the content of a webpage without reloading the entire page, thereby improving navigation and communication performance between client and server. Consequently, many web-pages use `xmlhttprequest` to dynamically retrieve data, for example when loading new search results, updating a dashboard or fetching user-specific information.

3.4 Issues

Data exploration just performed is sufficient to list some limitations and problems that the thesis must address in order to develop a reliable behavioral recognition model.

1. *Academic navigational patterns*: The browsing behavior of an academic user, specialized in STEM subjects, is different from that of inexperienced users. Ideally, this problem will be addressed by collecting more heterogeneous data.

2. *Domain-only storage*: Without other information but domain, we lost the difference between web page and web resource. The presence of path is crucial to understand if the host is responsible of the presence of the malicious resource, because it is designed with malicious purpose in mind, or it is a legitimate host who rely on third parties malicious resources.

3. *Lack of other information*: Nowadays the behavior on web is really complex. Multi-tab, presence of security mechanism, history clicks, links, focus, incognito browsing are just some of the example of information that would help to better apprehend the behavior of an user.

Chapter 4

Unified Blocklist Dataset

This chapter presents in detail all the steps involved in constructing a dataset of blocklists from scratch. The steps carried out include an explanation of the limitations imposed by *Pimcity*, the selection and justification of the blocklists, and data collection. Finally, the new dataset, *Unified Blocklist Dataset*, is presented through a data exploration.

4.1 Introduction

The thesis' goal is to develop and evaluate a behavioral recognition model able to classify a user access as malicious or benign.

In order to achieve this objective, we use Pimcity, a dataset comprising sequential accesses executed by voluntary participants. At this juncture, it is necessary to label every access in the database as malicious or benign, to establish a solid foundation for the development of a reliable web access classifier. Considering that blocklists represent one of the most widely used tools for identifying malicious accesses, in this chapter, we describe the methods used to retrieve, download, and integrate different blocklists, up to the creation of the *Unified Blocklist Dataset*. Consequently, a preliminary analysis of the dataset is performed in order to provide an overview of its main characteristics. We also explore how to combine *Pimcity* and *Unified Blocklist Dataset* to detect every malicious access, and how we develop a methodology to ensure reliable labeling of domains through a study on possible false positive reports in blocklists.

4.2 Limitations

The first problem that we address is the limitation of the access of the blocklist. Security services such as Google Safe Browsing (GSB) [14] or Virus Total (VT) [69] allow API services to classify URLs or domains. However, the API services' limitation rate makes this type of classification unfeasible. Therefore, open-source blocklists are required, as they allow us to download the entire list of malicious domains and work on it locally.

Other concerns about data collection to generate reliable labeling are strongly correlated with data stored in *Pimcity*. The labeling of Pimcity accesses places in the conditions to address two fundamental limitations.

1. *Domain only*: As explained in the previous chapter, collecting only the domain of any access and not further information, we lost the difference between malicious host and malicious hosted resource. To address this problem, it is necessary to focus on the blocklists that contain domains.

2. *Historic data*: This work was done in March 2025, but the stored accesses in Pimcity date back to March 2022 - November 2022. The use of updated blocklists would cause unreliable labeled accesses, since they do not reflect the Internet context of 2022. In addition, malicious websites have a volatile nature due to their activity. Today many malicious web page has been blocked, reported, deleted, renamed, or other.

The most challenging of the imposed limitations is that which forces us to derive the historical data of the blocklists. In fact, most popular blocklists do not have historical information dating back to the period of Pimcity data collection.

After a thorough search of the available blocklists, our choice fell on the numerous blocklists projects available on GitHub [29].

4.2.1 How GitHub works

This subsection presents only the essential aspects of GitHub needed to understand the next steps.

GitHub [29] is a hosting platform used by developers and agencies to work on software projects. All projects are accessible to contributors via a URL that points to a *repository*. In other words, the *repository* is the virtual space where developers can work on a project. Each repository has an *owner*, who is the user that created the repository. To facilitate contributions from many users, GitHub has developed a sharing process based on *commits*. In GitHub, any version of a file is reachable by an URL with the following structure:
`https://github.com/{repo_owner}/{repo_name}/raw/{commit_sha}/{file_path}`

where:

- `repo_owner`: the GitHub user or organization owning the repository,
- `repo_name` : the name of the repository,
- `commit_sha`: the commit's identifier,
- `file_path`: the path of the file inside the repository.

The possibility of recovering previous versions of the blocklists is fundamental to trace how the blocklist was updated day by day during the period of collection data of *Pimcity*.

4.3 Blocklists' overview

Recovering data from limited number of blocklists, for instance 2 or 3, would not guarantee a correct evaluation of the accesses. Conversely, recovering data from a larger set of blocklists (43), we are able to compare different results and make the labeling phase

more accurate and reliable. For this purpose, we choose different blocklists from different projects. Most projects update their lists basing on a specific threat. For this reason, during data acquisition, we take into account the details of the threat, as this not only allow us to define a domain as malicious, but also helps us to determine what threat it poses to the user.

In total, we selected 10 different types of threats, plus 1 to classify all domains that do not fall into any other type.

- *Redirect*: The site uses redirect techniques to manipulate the user's request. In other words, the site is benign, but when a user points to it, it forces the user to make malicious access.

- *Crypto*: Websites related to cryptocurrency that may run cryptojacking script-hidden code that uses visitors' devices to mine cryptocurrency without their consent.

- *Ads*: Advertisement sites. Even if they are not malicious, they are very popular within blocklists because of their annoying behavior. However, ads blocklist usually are used to block phishing sites as well.

- *Spam*: A spam site is a website whose primary purpose is the widespread of unwanted content. The content may or may not be malicious; however, it is generally accepted that spam sites suggest malicious resources. To illustrate this point, consider a website that automatically sends fake promotions, which then redirect users to malicious web pages.

- *Scam*: Scam sites are designed to deceive the user by forcing them to provide personal or perform unwanted actions. For instance, Scam site may mimics legitimate site to obtain user's trust and force him to send money, personal data, or other type of sensitive information.

- *Ransomware*:Ransomware attacks involve the encryption of a device's resources by a virus, and the subsequent demand for a ransom to unlock them. A ransomware website hosts a specific virus that renders the victim's resources inaccessible until a sum of money is paid to the attacker.

- *Phishing*: Similar to scam, they deceive the user. The difference between phishing and scam is the utilize of social engineering methods to deceive the victim. Usually, phishing attacks start with an targeted email containing a link that redirects the receiver to a malicious server. In contrast, scam sites do not involve direct action by the attacker against the user, such as sending an email. Rather, they are malicious sites that deceive the user into believing they are legitimate in order to forward sensitive information to the attacker.

- *Abuse*: A list of sites that aim to mislead or deceive users. These sites may spread false information, or employing deceptive interfaces to deceive the user to taking unwanted actions

- *Malware*: A list of sites that host or distribute malicious software. These sites contain infected files or virus that are downloaded which execution on user's device results in a security and privacy compromise.

- *Fraud*: site created with the purpose to defraud users. Both *Scam* and *Fraud* sites are specifically designed to deceive the user, but *Fraud* refers to a more generic legal term.

- *Mixed*: If a blocklist do not specify the kind of malicious threat, it fall in the mixed type.

The Table 4.1 includes all blocklists used in the job. Next to the owner of each blocklist is the type of threat that each list blocks. If a blocklist represents multiple threats, it appears on multiple rows with the corresponding threats.

4.4 Data collection

This section presents the technical details for recovering blocklist data. To facilitate the reader, any subsection presents and explains a different part of the workflow. At a high level, the workflow for recovering blocklist data can be summarized as follows: first, we identify all relevant commits for each blocklist repository, extracting their unique identifiers and dates. Next, using these identifiers, we download all versions of the files of interest, storing them locally to ensure both computational efficiency and backup safety. Finally, we process and combine all downloaded file by extracting all the relevant domains to construct a reliable dataset, the *Unified Blocklist Dataset*.

STEP 1: Recovering of the useful sha's commits

API services of GitHub allow the programmer to recover all the commits and the sha associated between two dates. However, GitHub imposes a call limit for unauthenticated users. For this reason, the following code contains a token used to identify the require during the entire process.

Other information that the require must know to complete this phase is as follows:

- Name of the repository owner.

- Name of the repository.

- Path of the file. The simplest method to obtain the path of the file is to extract it from URL of the most recent version that appears in the repository.

The Script 4.1 aims to recover all the commits between two date by providing all the necessary elements. If the communication was successful and there are commits for that period, the commit's identifier (sha) and date are extracted from the details of each commit. Finally, everything is returned to the calling function.

Owner	Type	Citation
blocklistproject-abuse	abuse	[6]
blocklistproject-ads	ads	[7]
nocoin-adblock-list	ads	[50]
blocklistproject-crypto	crypto	[8]
blocklistproject-fraud	fraud	[9]
blocklist-malware	malware	[4]
blocklistproject-malware	malware	[10]
iam-py-test	malware	[35]
kitsapcreator-malware	malware	[41]
rpiList-malware	malware	[57]
tweet-feed-today	malware	[70]
blocklist-phishing	phishing	[5]
chainapsis-phishing-block-list	phishing	[18]
eth-phishing-detect	phishing	[26]
null-host-bad-domains	phishing	[51]
phishfort-domains	phishing	[53]
rpilist-phishing-angriffe	phishing	[58]
tweet-feed-today	phishing	[70]
blocklistproject-redirect	redirect	[12]
blocklistproject-ransomware	ransomware	[11]
tweet-feed-today	ransomware	[70]
antiscam-squad-cypto-scam	scam	[2]
discord-antiscam	scam	[22]
durablenapkin-scamblocklist	scam	[25]
global-anti-scam-org-scam-urls	scam	[30]
global-anti-scam-org-scam-urls-pihole	scam	[31]
blocklistproject-scam	scam	[13]
soteria	scam	[64]
tweet-feed-today	scam	[70]
kitsapcreator-spam	mixed	[42]
referer-spam	spam	[56]
search-engine-spam	spam	[60]
spam_404	spam	[65]
thiojoe-spamdomainslist	spam	[68]
azorult-tracker	mixed	[3]
cert-pl-domains	mixed	[17]
foxwallet-domains	mixed	[28]
inversions-dnsbl-blocklists	mixed	[37]
kadomains	mixed	[40]
stevenblack-hosts	mixed	[66]
tweet-feed-today	mixed	[70]
ultimate-host-blacklist-0	mixed	[71]
ultimate-host-blacklist-1	mixed	[72]
ultimate-host-blacklist-2	mixed	[73]
ultimate-host-blacklist-3	mixed	[74]

Table 4.1. Blocklists

Listing 4.1. Obtaining all commits between two dates.

```
def get_commits_between_dates(start_date, end_date, repo_owner,
    repo_name, filepath):

    url = f"https://api.github.com/repos/{repo_owner}/{repo_name}/
        commits"
    headers = {"Authorization": f"token {TOKEN}"}
    commits = []
    page = 1

    while True:
        params = {
            "since": start_date,
            "until": end_date,
            "file": filepath,
            "page": page,
            "per_page": 100,
        }

        response = requests.get(url, headers=headers, params=params
            )
        if response.status_code != 200:
            break

        data = response.json()

        if not data:
            break

        for commit in data:
            commit_date = commit["commit"]["committer"]["date"]
            commit_sha = commit["sha"]
            commits.append((commit_sha, commit_date))

        page += 1
    commits = list(set(commits))

    if len(commits) > 0:
        commits.sort(key=lambda x: x[1])

    return commits
```

STEP 2: Recovering all the versions of the file

Now we have all the elements to recover all the previous versions of the file. Even if the goal of this part is to generate a *Unified Blocklist Dataset* which contains all the input derived from all the selected blocklists. The code immediately saves the blocklist's file for two reasons:

- the operations performed are computationally heavy. For security reasons, saving the file before operating on it guarantees a form of backup.
- by atomizing the operations, it is possible to operate in parallel to optimize the work.

The Script 4.2 demonstrates how to construct a URL that directly points to the raw version of a file. This version is typically a .txt file, ready for download.

Listing 4.2. Downloanding files given its commits.

```
def download_file_from_commit(
    commit_sha, file_path, save_directory, filename, repo_owner,
        repo_name
):
    url = f"https://github.com/{repo_owner}/{repo_name}/raw/{
        commit_sha}/{file_path}"
    headers = {
        "Accept": "application/vnd.github.v3.raw",
        "Authorization": f"token {TOKEN}",
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        save_path = save_directory + filename
        with open(save_path, "w", encoding="utf-8") as file:
            file.write(response.text)
    else:
        print(f"Error occurs downloading file from: {url}")
    return response.status_code
```

STEP 3: Create a database by combing all the files

Finally, after downloading the files, it is possible to combine all the information to construct a reliable dataset to be used for the next steps. The function `find_url_in_line()` takes as input a line and searches through a regex if there is a domain. This function avoids storing unnecessary lines, such as comments, or explanations found in blocklists. N.B: The Script 4.3 uses the same terminology imposed by *Pimcity*. Therefore, the term URL stands for domain.

Listing 4.3. Inserting domains from a blocklist in Unified Blocklist Dataset.

```
def insert_in_unifiedBlocklistDataset(filepath, type, date, source)
    :
    output_lines = []
    with open(filepath, "r", encoding="utf-8") as infile:
        lines = infile.readlines()
        for line in lines:
          url = find_url_in_line(line)
          if url is not None:
              output_lines.append(
                  f"\n{url},{date},{type},{source}}"
```

```
10              )
11
12      with open("./unifiedBlocklisDataset.csv", "a", encoding="utf-8"
            ) as outfile:
13          for l in output_lines:
14              outfile.write(l)
```

4.5 Data exploration

Data exploration represents the first step in understanding of the new dataset called *Unified Blocklist Dataset*. Through a preliminary analysis of the variables and their relationships, it is possible to identify patterns, anomalies, and useful insights to guide the subsequent stages of modeling.

4.5.1 Initial presentation

The dataset is composed of the data recovered from 43 blocklists, for a total of 4,904,484 distinct domains and 365,619,022 rows. Each line consists of the following characteristics:

- *URL*: In accordance with the terminology already used in *Pimcity*, we refer to any domain stored in the dataset under the column *URL*.
- *Source*: Is the blocklist's name that has reported the domain.
- *Date*: The date of the report.
- *Type*: The type of threat that the domain represents

4.5.2 Blocklist coverage period

The first characteristic to consider is that the period covered by each list is not uniform. Every blocklist belongs to different projects that start or end at different times. To better analyze this condition, the Table 4.2 is presented showing all periods covered by each list. It is worth noting that, although some of the tables do not cover the entire *Pimcity* data acquisition period, by integration of several projects, we have sufficient information to label any access of the database.

4.5.3 Update of the blocklists

It principally depends on the blocklists' administrator, but it plays a key role in understanding how fast the blocklist is in the case of the discovery of a new malicious site. In *Unified Blocklist Dataset* the update times of each blocklist are shown in Table 4.3:

N.B: The `Average update` was calculated by dividing the number of updates in each list by the number of the coverage period. The closer the value is to 1, the more the list is updated (1 means that the list was updated every day). It is evident that no blocklists were updated more than one time per day.

Blocklist	Start Date	End Date	Period (Days)
antiscam-squad-cypto-scam	2022-10-20	2022-11-13	25
azorult-tracker	2022-05-06	2022-11-01	180
blackbook-Stamparm	2022-05-06	2022-11-06	185
blocklist-malware	2022-05-15	2022-09-27	136
blocklistproject-abuse	2022-05-15	2022-09-27	136
blocklistproject-ads	2022-05-15	2022-09-27	136
blocklistproject-crypto	2022-05-15	2022-09-27	136
blocklistproject-fraud	2022-05-15	2022-09-27	136
blocklistproject-malware	2022-05-15	2022-09-27	136
blocklistproject-phishing	2022-05-15	2022-09-27	136
blocklistproject-redirect	2022-05-15	2022-09-27	136
blocklistproject-ransomware	2022-05-15	2022-09-27	136
blocklistproject-scam	2022-05-15	2022-09-27	136
chainapsis-phishing-block-list	2022-07-01	2022-11-12	135
cert-pl-domains	2022-05-06	2022-11-13	192
discord-antiscam	2022-05-06	2022-11-13	192
durablenapkin-scamblocklist	2022-05-07	2022-11-13	191
eth-phishing-detect	2022-05-08	2022-11-11	188
foxwallet-domains	2022-08-18	2022-09-07	21
iam-py-test	2022-05-07	2022-11-13	191
inversions-dnsbl-blocklists	2022-05-06	2022-11-13	192
kadomains	2022-05-06	2022-11-13	192
kitsapcreator-malware	2022-05-15	2022-09-25	134
kitsapcreator-spam	2022-05-15	2022-09-25	134
nocoin-adblock-list	2022-05-31	2022-09-06	99
null-host-bad-domains	2022-10-16	2022-11-13	29
phishfort-domains	2022-05-06	2022-11-13	192
referer-spam	2022-05-07	2022-10-27	174
rpiList-malware	2022-05-11	2022-11-13	187
rpilist-phishing-angriffe	2022-07-28	2022-11-13	109
search-engine-spam	2022-05-09	2022-11-09	185
soteria	2022-05-07	2022-11-12	190
spam_404	2022-05-29	2022-10-31	156
stevenblack-hosts	2022-05-11	2022-11-13	187
thiojoe-spamdomainslist	2022-05-31	2022-11-06	160
tweet-feed-today	2022-05-06	2022-11-13	192
ultimate-host-blacklist-0	2022-05-06	2022-11-13	192
ultimate-host-blacklist-1	2022-05-06	2022-11-13	192
ultimate-host-blacklist-2	2022-05-06	2022-11-13	192
ultimate-host-blacklist-3	2022-05-06	2022-11-13	192
global-anti-scam-org-scam-urls	2022-05-07	2022-11-13	191
global-anti-scam-org-scam-urls-pihole	2022-11-13	2022-11-13	1

Table 4.2. Time coverage per list

Table 4.3. Update frequency of each blocklist

Source	**Average update frequency (days)**
antiscam-squad-cypto-scam	1.0
azorult-tracker	0.15
blackbook-Stamparm	0.21
blocklist-malware	0.1
blocklist-phishing	0.1
blocklistproject-abuse	0.1
blocklistproject-ads	0.1
blocklistproject-crypto	0.1
blocklistproject-fraud	0.1
blocklistproject-malware	0.1
blocklistproject-phishing	0.1
blocklistproject-ransomware	0.1
blocklistproject-redirect	0.1
blocklistproject-scam	0.1
cert-pl-domains	1.0
chainapsis-phishing-block-list	0.24
discord-antiscam	0.98
durablenapkin-scamblocklist	0.87
eth-phishing-detect	0.7
foxwallet-domains	0.14
global-anti-scam-org-scam-urls	0.67
global-anti-scam-org-scam-urls-pihole	1.0
iam-py-test	0.57
inversions-dnsbl-blocklists	0.97
kadomains	1.0
kitsapcreator-malware	0.04
kitsapcreator-spam	0.04
nocoin-adblock-list	0.09
null-host-bad-domains	0.55
phishfort-domains	1.0
referer-spam	0.13
rpiList-malware	0.65
rpilist-phishing-angriffe	0.78
search-engine-spam	0.11
soteria	0.03
spam_404	0.01
stevenblack-hosts	0.43
thiojoe-spamdomainslist	0.24
tweet-feed-today	0.98
ultimate-host-blacklist-0	1.0
ultimate-host-blacklist-1	1.0
ultimate-host-blacklist-2	1.0
ultimate-host-blacklist-3	1.0

4.5.4 Number of reported domains

Finally, the number of reported domains is crucial to define a blocklist as reliable or not. If a blocklist contains only a small number of domains, users could be exposed to malicious websites. On the other hand, if the list flags even potentially suspicious domains as malicious, it risks blocking legitimate sites and unnecessarily restricting user access. The Table 4.1 shows the number of domains listed in each blocklist divided by the number of days covered by the list.

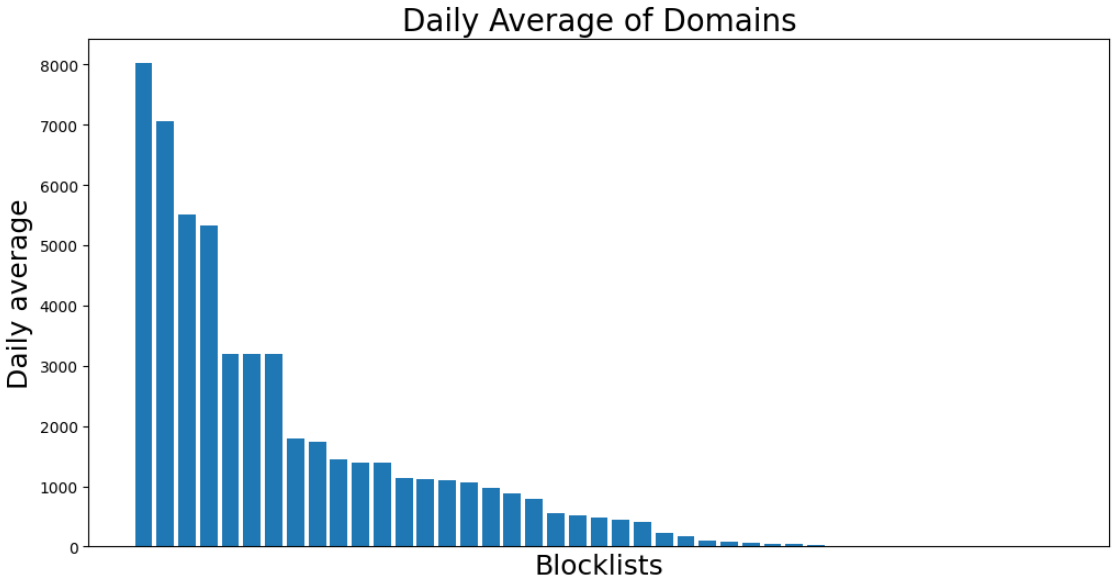

Figure 4.1. Bar chart of the daily average number of domains per blocklist (without names).

4.5.5 Domain identification by threat type

In the end to better understand the composition of *Unified Blocklist Dataset* we summarize the number of reported domains per their type.

The results 4.2 highlight that the majority of domains fall under the *mixed* and *malware* categories, together accounting for the largest share of the dataset. *Phishing* domains also represent a significant portion, followed by *abuse* and *spam*. Other categories such as *fraud*, *ads*, *redirect*, and *scam* contribute less but are still relevant for identifying specific attack vectors. Finally, categories like *crypto* and *ransomware* appear less frequent, but remain important due to the potential impact of these threats.

This classification provides a first overview of the threat landscape in the dataset and helps to prioritize the analysis of the most prevalent and dangerous categories.

N.B: The type depends on the nature of the list. If the same domain is contained in more lists with different type, the domain falls under different types too.

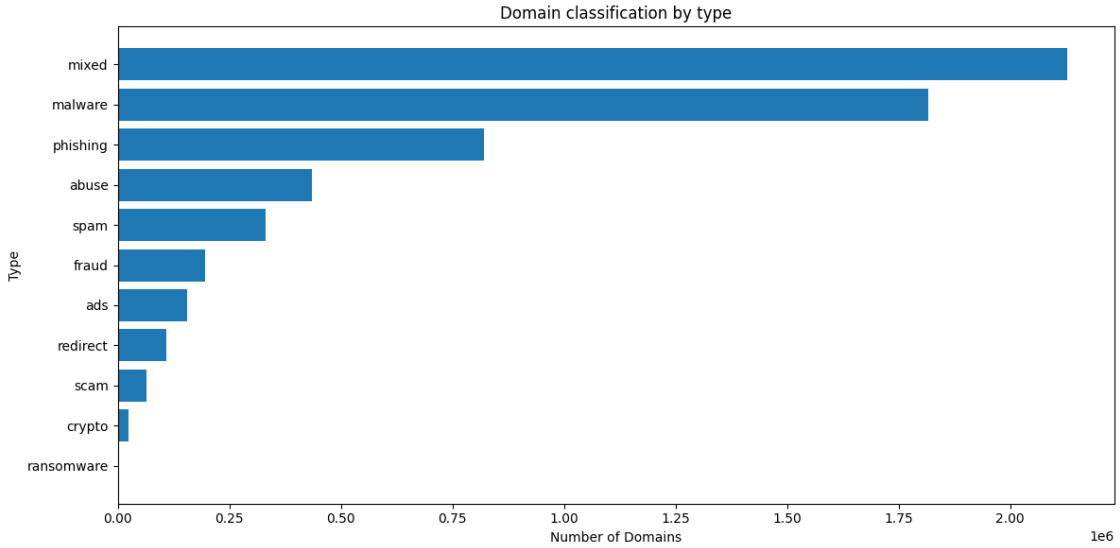

Figure 4.2. Domain classification by type of threat.

4.6 Data labeling

In the context of the evaluation of the classifier models, the choice of the metodology to label the accesses plays a key role. The labeling phase represents the objective reference with which the model compares the predicted results and calculates metrics such as precision, recall, and the F1-score. This leads to a correlation between labeling and model reliability. In particular, if the labeled accesses are wrongly classified, the model will use those data to calibrate its parameters and provide similar and inaccurate results. To choose the labeling methodology as accurate and reliable as possible, we explore different solutions.

4.6.1 Initial data labeling approach

The first approach consists of a preliminary analysis of the *Unified Blocklist Dataset*. A domain is considered malicious if it is reported by at least one blocklist on the same date that the user accesses it. At first glance, the number of exposures (accesses classified as malicious) is remarkably high, nearly 15.99% of all accesses are flagged as malicious. When distinguishing between direct and indirect accesses, 7.60% of direct accesses and 16.11% of indirect accesses are reported by at least one blocklist. This high proportion is primarily due to the use of domains as the unit of classification. It is worth noting that each domain is correlated with numerous URLs, some of which may be malicious and others not. Blocklists may label an entire domain as malicious based on a single suspected resource. This aggregation does not reflect normal web patterns. Different studies express a significantly lower number of malicious domains. For example, [67] confirms that the number of unique domains blocked by the company for malicious activity is 1.6% in

2023. While for third-parties additional resources, the study [36] labels only 1.2% of them as suspected of malicious activities. Both cases demonstrate an overestimation of the malicious domains. For this reason, we cannot consider this first approach useful to perform a valid and accurate labeling of the accesses.

4.6.2 Data labeling strategies

To establish the best approach to label the accesses, we considered two principal characteristics of domains. On the one hand, we have maliciousness that is the number of blocklists that have reported the domain. This feature is self-explanatory in terms of its importance for correctly determining whether a domain is malicious or not. On the other hand, we have the popularity of the domain. This characteristic is expressed in two terms: the number of visits and the number of different users who have access to it. In general, several studies have explored the idea of considering the popularity of a domain to establish if it may be considered malicious or not. For instance, Hu et al. [34] classifiers evaluated based on popularity and performance data to detect malicious and phishing domains. This relationship is further supported by [33, 75], who highlight owners of popular websites are more likely to employ robust and different security countermeasures to ensure high up-time, as increased traffic typically translates into higher revenue. Consequently, the adoption of these security systems establish a strong correlation between popularity in domains and its likelihood of being non-malicious. The goal of this section is to determine two cutoff values: one for the popularity and one for the maliciousness, which allow us to select a reliable labeling methodology for use in the subsequent steps of our work.

The Figure 4.3 shows every reported domain in three dimensions: number of distinct users who visited it (X axis), number of blocklist that reported it (Y axis) and number of visits (marker size).

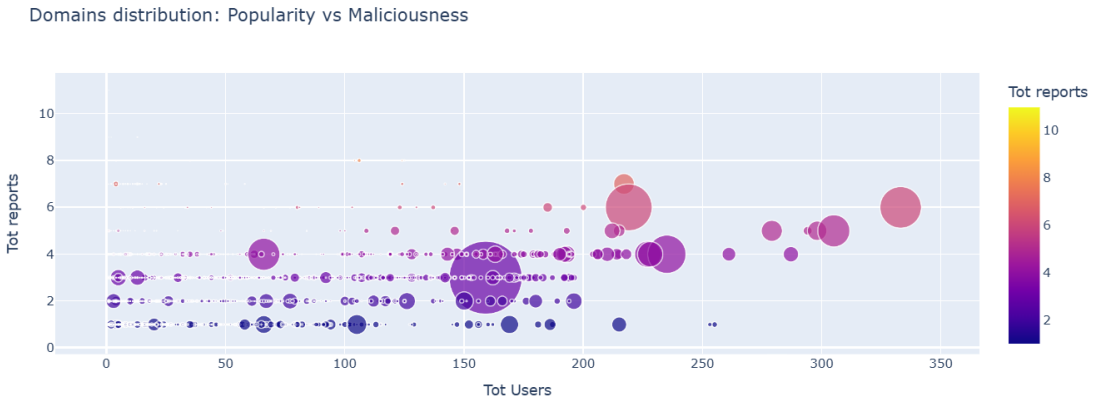

Figure 4.3. Popularity and Maliciousness of every reported domain

Given this image, it is possible to ideally divide it into four different sections:

- *Top right*:The section includes popular domains reported by many blocklists. Due to their popularity, they are unlikely to be truly malicious. Examples of domains contained in this section are `google.analytics` or `static.doubleclick.net`.
- *Top left*: All domains in this section are reported by many blocklists and are unpopular. Theoretically, there is a high probability that they are malicious since they satisfy both the constraints of popularity and maliciousness.
- *Bottom right*: This section is a gray zone, since the domains respect the popularity constraint, but they are rarely reported.
- *Bottom left*: This section includes all domains that we can consider misclassified as malicious. They do not respect popularity constraint, and they are reported by a few blocklists.

Since the domains in the top left section of the image are the better candidates, we have to isolate them.

Maliciousness Threshold

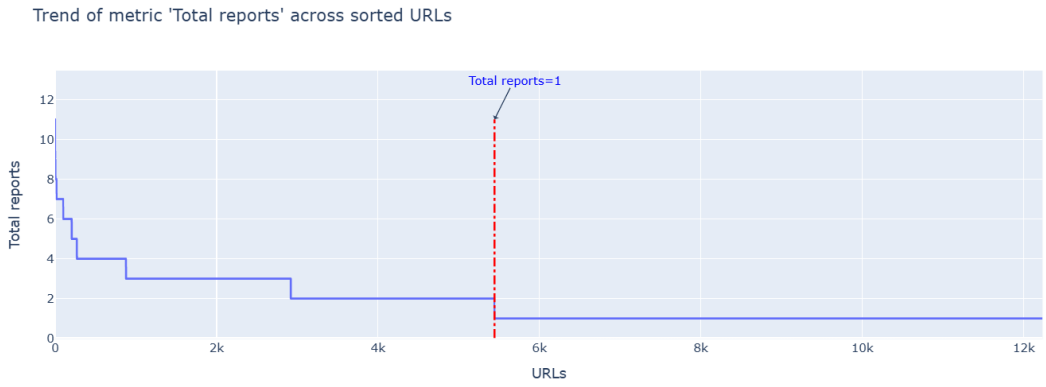

Figure 4.4. Ordered domains per number of reports (threshold between 1 and 2 reports).

The Figure 4.4 shows all domains, sorted by the number of blocklists that report them. Most of domains — 6,791 on 12,234 (55.51%)— have only one report. Although setting a threshold of at least two reports, causes a significant number of domains to be discarded, this step is necessary to ensure a more reliable labeling. By requiring multiple independent reports, we reduce the risk of including misclassified domains and focus on those that are more consistently recognized as suspicious.

Popularity Threshold

The Figure 4.5 below are shown the domains sorted by popularity (in terms of distinct users).

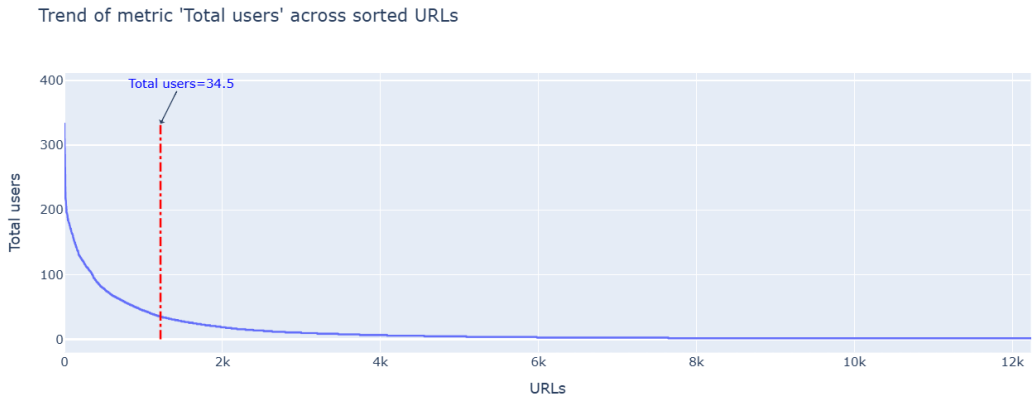

Figure 4.5. Ordered domains per number of distinct users who made access (threshold on 10% of total users).

Following the approach of the study conducted by Burton et al. [15], we can address the problem of discarding misclassified domains given popularity metrics. Similarly to the static method proposed in the paper, we consider the elbow of the curve as an eligible threshold. By approximating the threshold value at 10% of the total number of users, we can discard all the domains visited by more than that value. This filter discards only 1,216 on 12,234 domains, — almost 9.9% —, allowing us to filter out all popular and well-known domains.

4.6.3 Results and Statistics

In addition to the implementation of filters, we focus on the direct accesses represented by the *Pimcity-mainframe dataset*. This approach allows us to avoid the issue of indirect traffic by focusing solely on direct domain accesses for classification purposes. Finally, this additional precaution would further reduce the residual error, allowing us to work even more carefully on the data we have available. In light of the aforementioned considerations, after the application of the filter and the exclusive use of Pimcity-mainframe, the number of reported malicious accesses amounts to 382 distinct domains, constituting 0.74% of the total unique domains (50,976) registered with Pimcity-mainframe. Even if the percentage is lightly minor, the selection is in accordance with the behavior recorded by [67]. Since both thresholds are fixed, this selection cannot be considered valid in absolute sense. In particular, we selected a subset of domains which, although they would be classified as malicious with respect to our metrics, do not necessarily represent all of the threats in the entire Pimcity dataset. However, the Figure 4.6 shows in terms of

popularity and maliciousness the position of every reported domain. The visualization serves as practical tool for analyzing patterns in domain-level threats and supports the labeling of the accesses for subsequent classification tasks.

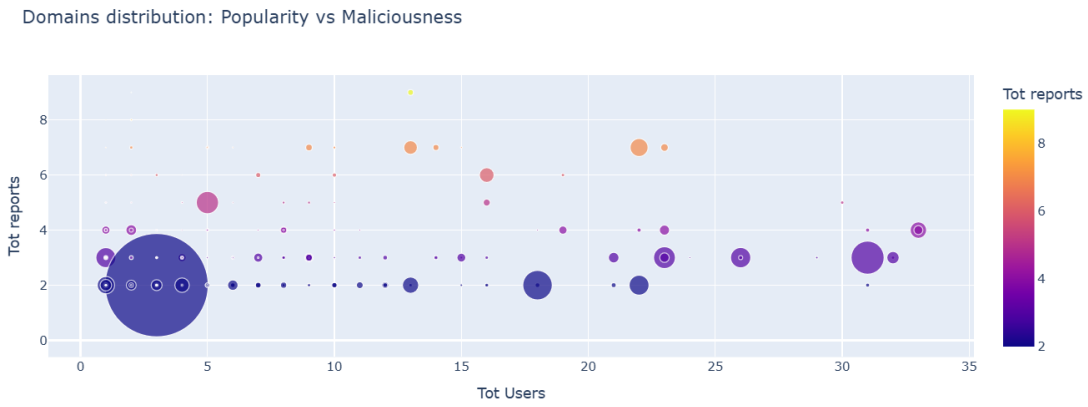

Figure 4.6. Maliciousness vs popularity after filters application

Chapter 5

Feature selection

It is understood that *features* represent all the characteristics that the classification model must take into account to determine whether an observed access is malicious or not. For *feature selection*, we mean the process of selecting, from among the possible features, those that we considered most important and explanatory for the model in function of the classification. We organized the features into four main groups: popularity features, which capture the web popularity of the domain; structural and lexical features, which reflect specific characteristics of the domain; categorical features, which describe the thematic area of the domain; and behavioral features, which are directly related to the individual user's browsing behavior.

5.1 Popularity features

We have already discussed the evidence showing that a domain's popularity and the likelihood of it being malicious are inversely proportional 4.6.2. Consequently, the first two features aim to materialize this idea by measuring two metrics: one capturing local popularity, that is, within the *Pimcity* dataset, and the other capturing global popularity, taking into account the domain's popularity on the Internet.

- *Traffic (10 bins)*: Firstly, domains are sorted according to the number of users accessing them. Then, the dataset is divided into ten different groups based on this order. Finally, each domain is assigned to the group to which it belongs.

- *PageRank*: *PageRank* is an algorithmic analysis tool developed by Google to rank websites based on the quality and quantity of their hyperlinks. We obtain the PageRank value using the API system provided by OpenPageRank [23]. Each domain that belongs to the top 10 million most popular domains is valued with a decimal score ranging from 0.0 to 10.0. Domains out of top 10 million are instead assigned a default value of 0.0.

5.2 Structural and lexical features

The initial features employed for training the model are based on the domain's structure. Numerous studies have demonstrated the enhanced classification value these features can offer [1, 20, 45, 55, 82]. Consequently, despite their non-strict relevance to user behavior, we have selected three features.

- *Domain level*: A *subdomain* is the extension of a domain created to organize or identify specific services. For instance, in `mail.google.com`, "mail" is a subdomain of `google.com`. The *domain level* corresponds to the number of subdomains present in the domain in according to the number of dots used to separate all the subdomains. In *mail.google.com* are presents 3 dots, then it is a level-3 domain. According to article [20] legitimate domains tend to have 2 or 3, sub-levels. Domains with many sub-levels are often considered suspicious. This feature is taken into account in other articles dealing with the classification of malicious domains [45, 55].

- *Shannon entropy*: To bypass blocklists, many malicious domains are generated automatically using a Domain Generation Algorithm (DGA), which produces a large number of new domains. These domains often exhibit complex and seemingly random structures.

  One way to detect such domains is by using *Shannon entropy*, a mathematical measure of the degree of randomness or disorder in a string, defined as follows. For a domain name with characters from an alphabet $\mathcal{A}$, Shannon entropy is defined as:

  $$H(X) = -\sum_{x \in \mathcal{A}} p(x) \log_2 p(x)$$

  Where $p(x)$ is the ratio between the number of occurrences of character x in the string's length of the string. Finally, we add up all the probabilities of each letter to define the value of the *Shannon entropy* of the entire domain. The underlying idea is that a domain with high Shannon entropy is more likely to have been automatically generated by a DGA for malicious purposes.

- *Levenshtein distance*: Another method used by attackers to deceive the victim is the *cybersquatting*. *Cybersquatting* is the process of creation of a malicious domain by making a small change on a popular and legitimate one. To better illustrate this definition a malicious domain may be generated by changing the letter L in `netflix.com` in the letter I (`netiix.com`). Finally, the attacker may send a phishing email to the user with a link to the malicious `netiix.com` that is managed by the attacker. Although both the *domain level* and *Shannon entropy* provide a measure of the complexity of the domain's structure, the *Levenshtein distance* measures the grade of similarity between two strings. For two strings a and b, the Levenshtein

distance $d(a, b)$ can be formally defined as:

$$d(a,b) = \begin{cases} |b|, & \text{if } |a| = 0, \\ |a|, & \text{if } |b| = 0, \\ d(\text{tail}(a), \text{tail}(b)), & \text{if head}(a) = \text{head}(b), \\ 1 + \min \begin{cases} d(a, \text{tail}(b)), \\ d(\text{tail}(a), b), \\ d(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

where head is the first character of the string, tail is the string but the first character, and the absolute value of a string is its length. In other words, the *Levenshtein distance* corresponds to the minimum number of insertions, replacements, deletions to obtain the string b given the string a. The Levenshtein distance is measured between each access and all domains with a PageRank value greater than 6. cutoff value of 6 was chosen taking into account taking into account the computational effort required to calculate this metric. Finally, the minimum Levenshtein distance is taken into account as a feature to detect possible cybersquatted domain.

5.3 Categorical features

To better understand the behavior and interests of a user, and establish if a new access is concordant with the user's navigational patterns, we need to classify any domain for the thematic area that the web page belongs to. On the basis of [38] we use the classifier provided by Google Topics API [32], which has been used in the advertising sector to identify user interests across website topics [76].

In particular, the classifier takes as input the domain of a visited website and, based on its lexical structure (i.e. the words contained in the domain), assigns it one to three topics from a predefined hierarchical taxonomy [24], along with a confidence score for each classification.

The taxonomy is organized into main categories and derived subcategories, allowing the model to capture both general and specific user interests.

For instance, a site about cars may be classified under the main topic `/Autos & Vehicles`, or under a sub-topic like `/Autos & Vehicles/Classic Vehicles` that better describes the site. Nevertheless, in order to reduce the number of potential topics, we take in consideration exclusively the primary topics,and we ignore the remaining ones.

Basically, after the classification procedure, up to three main categories and their confidence values are associated with each domain.

5.4 Behavioral features

In order to provide further clarification regarding the selection of behavioral features, it is essential to revisit the specific goals of our classification model. The primary objective of this thesis is to demonstrate the feasibility of predicting malicious accesses through the

analysis of a series of previous accesses. To this end, we introduce the concept of *session*, defined as a period of web activity that is followed and preceded by a minimum of 20 minutes of inactivity. It is important to note that user behavior differs depending on the activity [52]. For example, sending an e-mail is characterized by highly targeted behavior in terms of the accessed sites, in contrast to web research, which is exploratory. The concept of a session enables us to circumvent the distinction between potential behaviors, given that the period during which we want to extract features is not fixed and may vary in number. Finally, we selected five primary features to characterize each session. Each feature was computed using only the accesses preceding the current one, ensuring that the classification can be performed in real time.

- *Session depth*: number of accesses executed by the user (until access).
- *Session depth (distinct domains)*: number of distinct domains in the session (until access).
- *Session time*: duration of the session (until access).
- *New domain in session*: binary feature that indicates if the user has already accessed the domain during the current session.
- *Hour (4 bins)*: The day is divided into four periods (morning 6—11, afternoon 12—17, evening 18—21, night 22—5), and this feature stores, as a vector, which period the domain was accessed.

Nevertheless, the concept of a session could prove to be a limitation in the pursuit of our objectives. In the event that our goal is to detect malicious sessions by comparing information with previous sessions, the classifier may not predict abnormal behavior in the case of session begins as malicious or suspicious. For example first access of the session was caused by a click of the user on a phishing link sent by an attacker. Consequently, certain features are designed to encompass not only the current session, but also memory for up to *100 previous access*, independently of the session. We assume that over 100 accesses, the behavior is not more useful to establish if the next access will be malicious or not.

In conclusion, the following features are also extracted from the knowledge of the session at the time of access.

- *Count visits*: The feature is calculated by counting how many times the domain you want to access appears in the previous 100 accesses
- *Coherency index*: The feature combines the concept of category and previous accesses by calculating the number of times the category of the site the user wants to access appears in the previous 100 accesses. Only the category with the highest confidence of each access is taken into account for the calculation.

Finally, we also took into account descriptive features that may be useful for the model to capture temporal patterns or network delays caused by low-quality services, typical of malicious domain.

- *Hour*: hour of access

- *Weekend*: binary features that highlights the accesses that occurs during the weekend
- *Difference from the previous access*: measure of the temporal difference from the current access and the previous one

Chapter 6

Models and evaluation metrics

The purpose of this chapter is to illustrate the choice, development and results of different classification models. For our convenience, we have decided to divide the possible classification models into two groups according to their unit of classification. The first set comprehends the *non-sequential models*. They receive as input a set of features (corresponding to the features extracted from each individual access) and, based on the comparison and values of the features, they find recurring patterns that can predict a classification. The term ”non-sequential” is due to the fact that they do not consider ordered set of accesses as a whole, but only one access that is classified through comparison with other accesses taken randomly from the dataset.

Sequential models, on the other hand, tend to classify accesses by considering entire ordered set of accesses. The literature shows that of all the models, LSTMs are the most suitable for the task. They consider a sequence of accesses and attempt to classify the last access, not only by analyzing its features, but also by taking into account the features of previous accesses, relating the entire sequence to other sequences. It is worth noting that the selected features, and in particular the behavioral ones, already contain information about session or about the 100 previous access. Therefore, whether it is a sequential model or a non-sequential model, both represent a valid choice to establish whether a malicious access can be expected given a sequence of user accesses.

6.1 Non - sequential models

The literature shows how these models have been used to estimate a user's risk of accessing malicious sites [16]. The substantial difference between our work and that proposed by the cited paper is the granularity of the classification, i.e. the ability of the model to predict the individual access and not only the overall risk dictated by the user's behavior.

Illustrated by several studies [36, 67], and also found in our dataset, the concept of malicious access is comparable to a rare event with respect to the total number of benevolent accesses. It follows that the class of benevolent accesses is disproportionately large compared to the class of malicious accesses. A classifier that takes the entire dataset as input could be subject to the problem of *class imbalance*, in other words, it could happen that during testing, it would label all accesses as benign and still achieve a high

degree of accuracy. The adoption of algorithms such as SMOTE, which artificially insert elements of the minority class to re-balance the dataset, could alter the information related to the real behavior of the user. Our solution is therefore to force an *unbalance ratio* between the malevolent and benevolent classes. The *unbalance ratio* (UR) can be formally defined as:

$$UR = \frac{N_{begnin}}{N_{malicious}}$$

where N_{benign} represents the number of benign accesses and $N_{malicious}$ the number of malicious accesses in the dataset. In other words, it indicates that for each malicious access, UR benevolent accesses are taken. Finally, in order to ensure a clear separation between classes, the training class is composed of the 80% of users to which the selected accesses belong, while the test class is composed of the remaining 20%.

We decided to focus our efforts on two classification models: Random Forest and Multilayer Perceptron (MLP).

6.1.1 Random Forest Classifier

The *Random Forest Classifier* is a learning model based on multiple distinct decision trees. It works by combining multiple decision trees to improve predictive performance and reduce overfitting. The main functioning can be summarized as follows.

1. *Bootstrap Aggregating (Bagging):* N decision trees are created on random samples with replacement (bootstrap) from the training dataset.
2. *Random Feature Selection:* during the construction of each tree, a random subset of features is considered at each split, increasing diversity among the trees.
3. *Voting:* for classification, each tree casts its vote for the class of the observation, and the final class is chosen by majority voting.
4. *Handling Class Imbalance:* using the hyperparameter `class_weight="balanced"`, the model automatically weights classes according to their frequency in the dataset.

Hyperparameters

Table 6.1.1 summarizes the main hyperparameter used.

Hyperparameter	Value	Description
`n_estimators`	100	Number of trees in the forest
`random_state`	RANDOM_STATE	Seed for reproducibility
`class_weight`	"balanced"	Automatic class weighting

Table 6.1. Hyperparameters of the Random Forest Classifier

6.1.2 Multilayer perceptron (MLP)

The *Multi-Layer Perceptron (MLP)* is a feedforward neural network composed of multiple fully-connected layers. Hyperparameter are summarized in Table 6.1.2

1. *Input Layer:* accepts a feature vector of size $X\_train.shape[1]$.
2. *Hidden Layers:*
   - First hidden layer: 256 neurons, `ReLU` activation, followed by `BatchNormalization` and `Dropout` with a rate of 0.3.
   - Second hidden layer: 64 neurons, `ReLU` activation, followed by `BatchNormalization` and `Dropout` with a rate of 0.2.
3. *Output Layer:* 1 neuron with `sigmoid` activation, suitable for binary classification tasks.
4. *Model Compilation:* the model is trained by minimizing `binary_crossentropy`, using the `Adam` optimizer with a learning rate of 1×10^{-3}, and monitoring the `accuracy` metric.

Hyperparameters

Hyperparameter	Value	Description
`layers`	[256, 64, 1]	Number of neurons per layer
`activations`	["relu", "relu", "sigmoid"]	Activation functions for each layer
`dropout_rates`	[0.3, 0.2]	Dropout rate for regularization
`batch_normalization`	True	Batch normalization to stabilize training
`loss`	"binary_crossentropy"	Loss function
`optimizer`	Adam	Optimizer with learning rate $1e-3$
`metrics`	["accuracy"]	Metrics monitored during training

Table 6.2. Hyperparameters of the MLP model

6.2 Sequential Model

The analysis of the behavior requires the ability of the model to take in consideration not the single event, but also the ordered sequence of preceding event. Many academic works [39, 49, 63, 81], probe that LSTM is a valide choice, since that it receives as input a sequence of events and try to predict the class of the last one. In this case, the unit to take into account for the classification is not the single access (even if the model classify only the last access of the sequence), but the entire sequence of access. For a first try, we select 20 accesses, considering it as a good tradeoff between a limited number of

accesses which do not provide enough information about the context and consequently of the behavior; and a larger set of accesses that could have led the model to accentuate the correlation of features occurring at a high temporal distance from current access. Finally, we define a malicious sequence as any sequence that ends with a malicious access. Conversely, a benign sequence corresponds to any sequence of twenty accesses that ends with a legitimate one. Since any sequence is composed of twenty accesses, to avoid overfitting, we chose not to intersect the sequences.

For LSTM, the *balance problem* is further aggravated since the input unit is not a single access, but a sequence of 20 accesses.

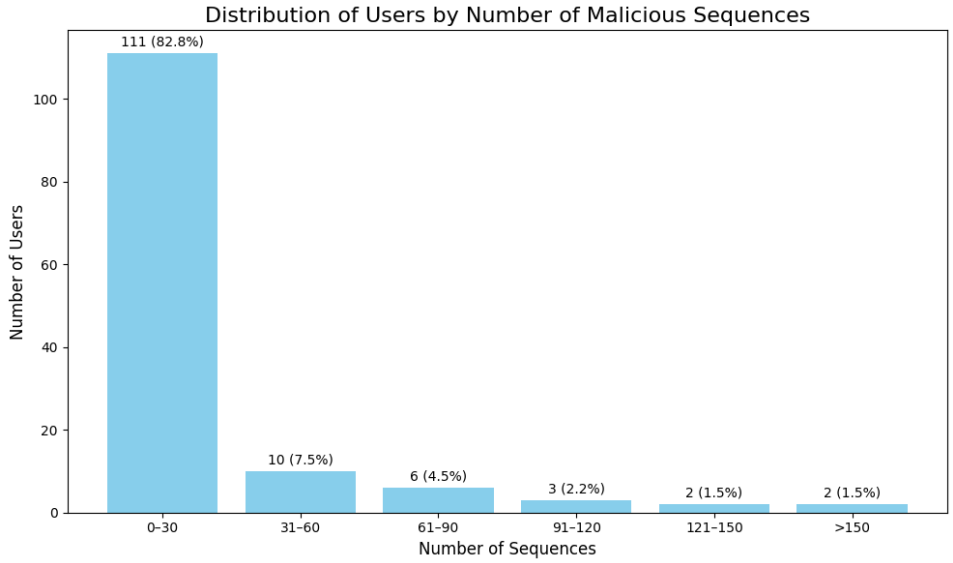

Figure 6.1. Distribution of users by number of malicious sequences.

As shown in the Figure 6.1, many of the malicious accesses (and consequently sequences) are executed by a minority of the total users. If these users were selected for the test phase, the model would lose much of the information needed for proper training. We therefore define N as the number of sequences to be extracted from the histories of each user. If a user has less than N sequences of a given class, then the maximum number of available sequences closest to N will be extracted. This system allows us to consider many users and once we perform an 80/20 split on the users, it ensures that the training and test classes have enough sequences. Finally, we train the model on the accesses performed by the 80% of the users and test it on the remaining 20% user base's accesses.

6.2.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) are a kind of *Recurrent Neural Network (RNN)* specialized to address the *vanishing gradient problem*. Recurrent Neural Networks (RNNs) use a loss function to quantify the error, or the difference between the classification of the network and the *ground truth*. Subsequently, the gradient of the loss with respect to the

weights of the RNN is calculated, which is then backpropagated to update the weights and reduce the error in future iterations. As the gradient is the product of several derivations, it is possible for the gradient to take on extremely small values if the influence of a weight on the loss is negligible. In this cases, the weights do not change, and the network is subject to difficulties in learning long-term dependencies. Long Short-Term Memory (LSTM) address the problem by introducing memory cells that facilitate the flow of the gradient over time, preventing it from becoming too small even after multiple time steps.

Table 6.2.1 summarizes the parameters.

- `Sequential()`: Creates a linear stack of layers for the model.
- `Input(shape=(sequence_length, input_shape))`: Specifies the input shape of the sequences, where `sequence_length` is the number of timesteps and `input_shape` is the number of features per timestep.
- `Masking(mask_value=0.0)`: Skips timesteps that are entirely zeros, useful for variable-length sequences.
- `LSTM(100)`: Adds an LSTM layer with 100 units to capture temporal dependencies in the data.
- `Dense(1, activation="sigmoid")`: Adds an output layer with 1 neuron and sigmoid activation for binary classification.
- `compile()`: Configures the learning process with `binary_crossentropy` loss, `Adam` optimizer with learning rate $1e-3$, and monitors `accuracy` during training.

Hyperparameters

Hyperparameter	Value	Description
`sequence_length`	variable	Length of input sequences
`input_shape`	variable	Number of features per timestep
`mask_value`	0.0	Value to ignore for variable-length sequences
`LSTM_units`	100	Number of LSTM units in the cell
`loss`	"binary_crossentropy"	Loss function
`optimizer`	Adam	Optimizer with learning rate $1e-3$
`metrics`	["accuracy"]	Metrics monitored during training

Table 6.3. Hyperparameters of the LSTM model

6.3 Evaluation Metrics

To evaluate the effectiveness of a classification model, several metrics are used based on four fundamental variables:

- *True Positive (TP)*: the number of elements correctly classified as belonging to the positive class.
- *False Positive (FP)*: the number of elements incorrectly classified as belonging to the positive class, but actually belonging to the negative class.
- *True Negative (TN)*: the number of elements correctly classified as belonging to the negative class.
- *False Negative (FN)*: the number of elements incorrectly classified as belonging to the negative class, but actually belonging to the positive class.

Usually, counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class are expressed in an appropriate table known as *Confusion Matrix*. It provides a complete view of the comparison between the instances correctly classified by the model and their true class. A typical confusion matrix for a binary classification looks like this:

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

Each row represents the actual class, while each column represents the predicted class.

Based on these variables, the other crucial metrics are defined:

- *Precision*: indicates the proportion of instances classified as positive that are actually positive. It is defined as:

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

- *Recall*: measures how many true positive instances the model has recognized in comparison to the total number of instances that were designed for classification.

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

- *F1-score*: is the harmonic mean of *precision* and *recall*. It allows a single metric to summarize the properties of the two metrics.

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

- *Accuracy*: This metric is indicative of the number of instances that have been correctly classified, in comparison to the total number of instances.

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

- *Support*: refers to the number of occurrences of each class in the dataset. It is used to weight other metrics when computing averages.

These metrics allow the model's performance to be evaluated from different perspectives, depending on the application context and class distribution.

Chapter 7

Results

This chapter presents and discusses the results obtained by the various selected models. The analysis is based on a unbalanced dataset with a *unbalance ratio of 2* and, only for the LSTM dataset, the number of benign and malicious sequences taken by each user corresponds to 6.

Firstly, the classification results are presented, with a discussion of the metrics: *Precision*, *Recall*, *F1-score*, and *Support*. Next, we analyze the variation of two main factors: the unbalance ratio, and the variation of the classifier performances in a scenario closer to a real context. Finally, we compare classifiers and blocklists to study the ability of classifiers to anticipate the behavior of blocklists.

7.1 Classifier performances

As a first approach, we have evaluated the performances of the three classifiers: *Random Forest*, *Multi-Layer Perceptron* (MLP) e *Long Short-Term Memory* (LSTM). The metrics adopted by three evaluation include: *Precision*, *Recall*, *F1-Score*,and *Support*. In addition, for each classifier, its *confusion matrix* and a summary table are presented. The dataset used for multi-layer perceptron (MLP) and random forest (RF), consists of 247 unique users who have made a total of 8,550 malicious accesses and 17,100 benign accesses. Upon implementing an *80/20 split* on the user base, the test set comprises 50 users. The support for the malicious class is found to be 2,407, while the benign class has a support of 3,118.

The dataset used to train and evaluate LSTM was built using a different approach to the MLP and Random Forest datasets. For each user, up to six sequences of the malicious and benign classes are taken. Consequently, if the benign class is larger than the malicious class by a factor greater than the unbalance ratio of 2, random sequences of the benign class are eliminated from the dataset to normalize the proportion of the classes. At the end of the aforementioned operations, we perform an *80/20 split* on the users to guarantee the impossibility of data leakage between the classes. In conclusion, the LSTM's dataset includes 1,731 benign sequences. Of these, 1,379 are used to train the classifier and the remaining 352 are used as the test set. Finally, the test set consists of 223 benign sequences and 129 malicious ones.

Random Forest

Performance is high overall, with good *precision* values for both classes. The only exception is represented by *recall* on the malicious class. The model aims to classify as benign some accesses who are malicious. This phenomenon suggests that the conditions to classify an access as malicious are stringent, and the result is an increase of the false negative class. The Table 7.1 and the Figure 7.1 illustrate, respectively, the performances and the confusion matrix of RF classifier.

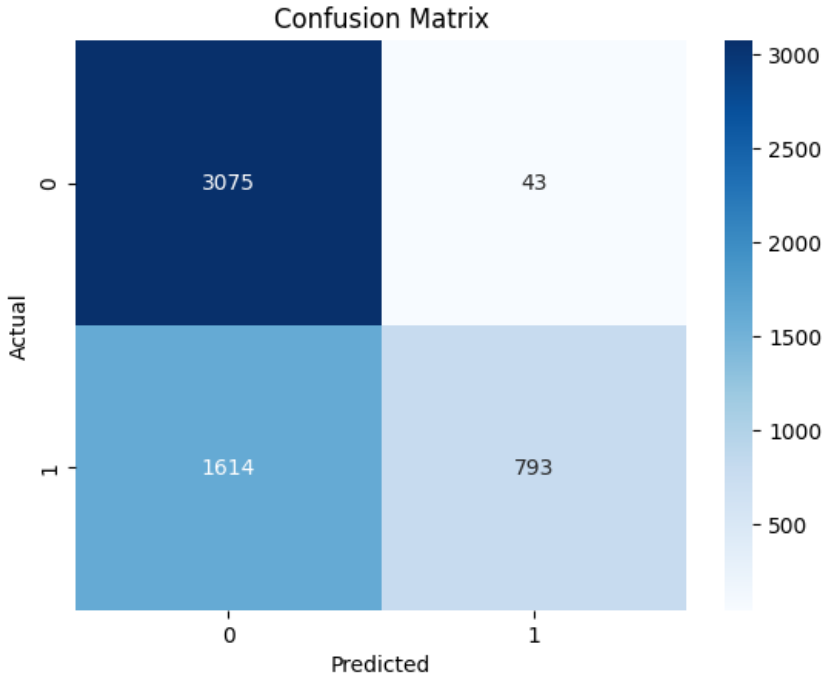

Figure 7.1. Confusion matrix - Random Forest classifier

Class	Precision (%)	Recall (%)	F1-score (%)	Support
Benign	65.58	98.62	78.78	3118
Malicious	94.86	32.95	48.91	2407

Table 7.1. Classification Report - Random Forest

MLP

MLP obtains balanced and good results in terms of *precision* and *recall* for both the classes, with an overall *accuracy* of 81.11%. The Table 7.2 and the Figure 7.2 illustrate, respectively, the performances and the confusion matrix of MLP classifier.

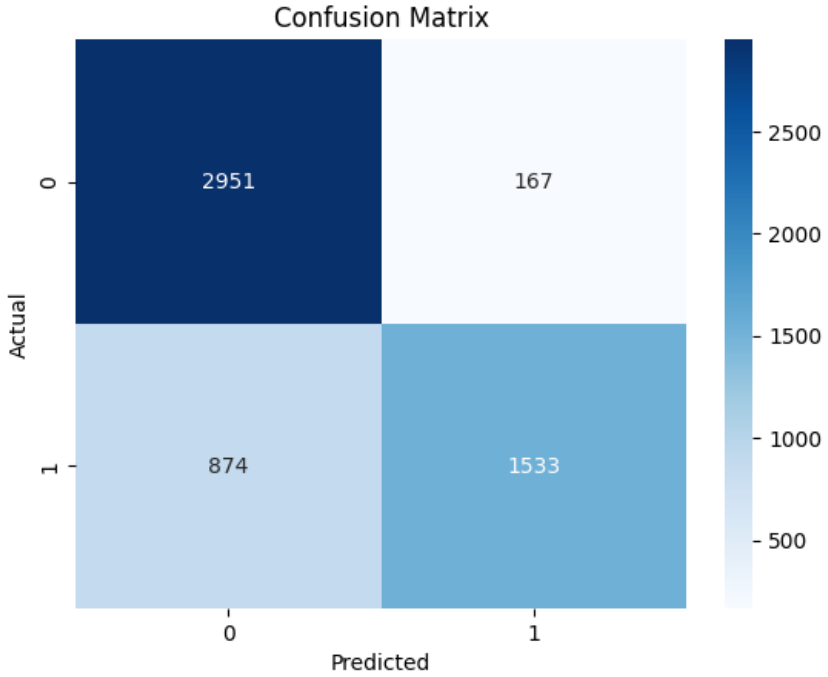

Figure 7.2. Confusion matrix - MLP classifier

Class	Precision	Recall	F1-score	Support
Benign	77.15%	94.64%	85.01%	3118
Malicious	90.18%	63.69%	74.65%	2407

Table 7.2. Classification Report - MLP classifier

LSTM

Among all classifiers, LSTM shows the best performances for every metric. This is due to its ability to exploit the sequential nature of the data, confirming the model as particularly suited for the classification of temporal sequences of accesses. The Table 7.3 and the Figure 7.3 illustrate, respectively, the performances and the confusion matrix of LSTM classifier.

Discussion of the preliminary results

In summary, preliminary results indicate that all the classifiers tested are able to identify and classify malicious accesses, either by considering single accesses (MLP and Random Forest) or by analyzing sequences of accesses (LSTM).

The first consideration suggests that neural networks, in particular MLP and LSTM, are particularly suited to this purpose. Moreover, the context of malicious accesses plays a crucial role: the LSTM classifier, which inherently incorporates information on previous

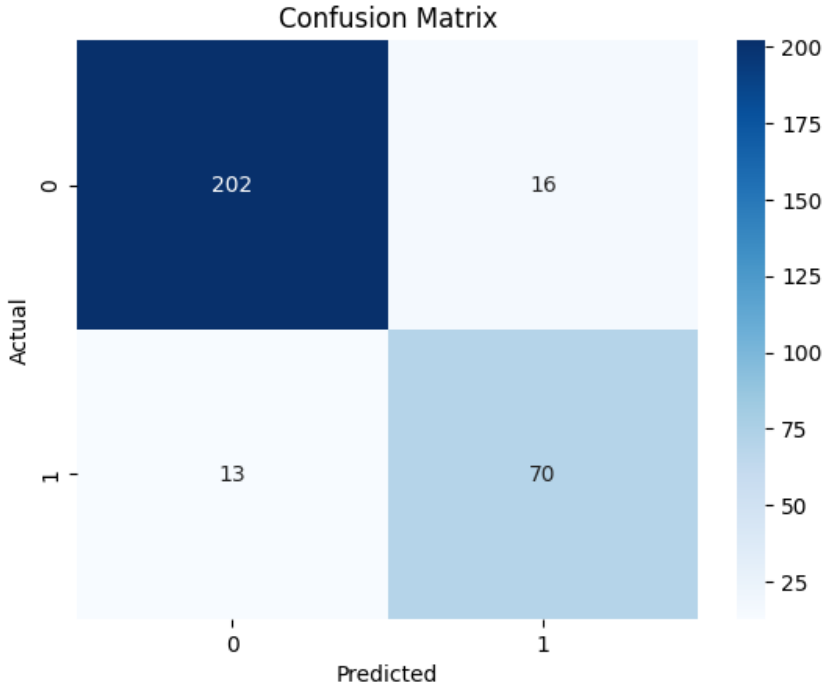

Figure 7.3. Confusion matrix -LSTM classifier

Class	Precision (%)	Recall (%)	F1-score (%)	Support
Benign	93.95	92.66	93.30	218
Malicious	81.40	84.34	82.84	83

Table 7.3. Classification Report - LSTM classifier

accesses by effectively exploiting the temporal dimension, shows superior performance in the classification phase compared to MLP and Random Forest. From this first analyses, it is clear that it is possible to detect malicious access by observing local user behavior.

7.2 Other results

Following the initial results, a further analysis was conducted to explore how the performance of the models changes when unbalance ratio or the method of splitting are varied. The metric utilized to illustrate the trend of performance as these factors change is the F1-score on the malicious class. F1-score is a suitable metric due to the fact that it combines both precision and recall in a single measure, thus capturing not only the proportion of correctly detected malicious instances among those classified as malicious (precision) but also the proportion of actual malicious instances that are correctly identified (recall).

This is of particular importance in unbalanced scenarios, such as the detection of malicious accesses, where a focus on accuracy alone could be misleading due to the dominance of the benign class.

7.2.1 Unbalance ratio variation

Figure 7.4 shows how an increase in the unbalance ratio corresponds to a performance degradation for all classifiers. However, the correlation between performance and unbalance ratio was predictable, since an increase in the unbalance ratio corresponds to an increase in benign accesses, and consequently an unbalanced dataset. In an unbalanced dataset, in fact, one class (in this case benign accesses) dominates numerically over the other (malicious accesses). This can lead the model to favor the majority class, reducing the ability to correctly recognize the minority class. Even in this scenario, the LSTM classifier proves to be the best-performing model. Although its performance decreases, it still shows a notable robustness, achieving, in the worst analyzed case, an F1-score on the malicious class higher than 60%, in contrast to roughly 20% for the RF and MLP classifiers. It is worth noting that Random Forest classifier demonstrates a drop of the performances when the unbalance ratio is set to 2. This unanticipated outcome indicates that the dataset generated with this unbalance ratio may present challenges for the Random Forest in differentiating between malicious and benign accesses.

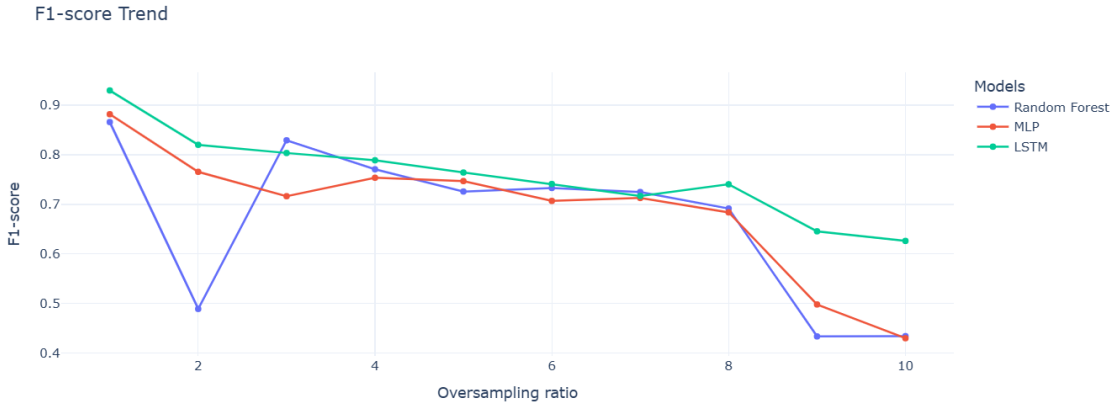

Figure 7.4. Unbalance ratio trend

7.2.2 Leave one user out

Another useful approach is what we have called *Leave One User Out (LOUO)*. The methodology involves training the model on all users except one whose accesses will be used in the test set. To ensure more robust and generalizable results, the procedure was repeated for each user in the dataset. In other words, for each user its accesses were

tested, while with all the accesses of all other users was in training set of the classifier. In this way, we ensure that the classes of the model are balanced only in the training phase, while, in the testing phase, the model is tested on a simulation of random accesses (or sequences) of a user that could have only benign, only malicious or mixed. This metric would not only prove that the model is able to predict malicious access through user behavior, but would also make the model applicable in a context closer to the real one, in which the number of benigns and malicious accesses is strongly disproportionate. For each model, the performances obtained by each user were collected. Subsequently, arithmetic average of the performances was then applied in order to normalize the number and compare them with the performance obtained with the 80/20 model.

Random Forest

Class	**Leave One User Out (%)**			**80/20 Split (%)**		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Malicious	83.33	68.44	71.71	94.86	32.95	48.91
Benign	94.08	98.79	95.60	65.58	98.62	78.78

Table 7.4. Comparison of Random Forest Classification Metrics: Leave One User Out vs 80/20 Split

The Table 7.4 shows as the *Random Forest*'s performances improve significantly. This finding, combined with the study on the variation of unbalance ratio, indicates that the initial results, which were considered the worst performances between the three model, are probably caused by a problematic 80/20 splitting method. This approach has been found to provide inadequate conditions for the model to learn effectively.

MLP

As shown in Table 7.5 show a degradation in performances, specifically the *precision* of the malicious class of the model decreases by almost 30 percentage points.

This decline can be explained by the fact that, in the Leave One User Out setting, the model is trained on a dataset that differs from the context of the test user, making generalization more challenging. In contrast, using an 80/20 random split ensures that the training and test sets share a similar distribution, which generally leads to better performance metrics.

If, in terms of performances, MLP has demonstrated itself as the better choice during 80/20 splitting, the Random Forest classifier is the most suitable in a real-world scenario where the malicious accesses are sporadic events.

LSTM

As illustrated in Table 7.6, there is a clear decline in performance for the malicious class. LSTM displays a behavior close to the MLP, especially with a reduction in the *recall* value for the malicious class. However, the overall performance remains the best between

Class	Leave One User Out (%)			80/20 Split (%)		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Malicious	61.00	60.26	58.02	90.18	63.69	74.65
Benign	92.22	92.67	91.49	77.15	94.64	85.01

Table 7.5. Comparison of MLP Classification Metrics: Leave One User Out vs 80/20 Split

the three models (all metrics above 70%), suggesting that the LSTM remains the most suitable option for our purpose.

Class	Leave One User Out (%)			80/20 Split (%)		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Malicious	80.37	77.49	77.68	81.40	84.34	82.84
Benign	95.43	94.84	94.40	93.95	92.66	93.30

Table 7.6. Comparison of LSTM Classification Metrics: Leave One User Out vs 80/20 Split

7.3 Classifiers and blocklists Comparison

Another desirable outcome would be the ability of the classifiers to anticipate the identification of malicious domains before the blocklists are updated. A preliminary study of false positives generated by classifiers showed that some accesses would have been reported as malicious by a blocklist within a few days. This result highlights the potential of behavioral recognition models in anticipating threats.

The analysis was conducted on the false positive set since it comprises the accesses that are classified as malicious by the classifiers, but benign during the labeling phase. Our models were trained on a relatively small dataset, which limits the precision with which we can quantify the anticipatory capability of the classifiers against blocklists. Nevertheless, each model demonstrated the ability to predict at least one malicious access ahead of its official classification. The analysis encompasses three principal aspects: the rational behind the accesses are not labeled as malicious, the proportion of anticipated malicious accesses, and the number of days in which the blocklist was anticipated. It should be noted that the access classification method, better described in Chapter 4.6.2, considers an access to be malicious solely if the domain was reported by at least two blocklists on the day of access and it is not utilized by more than 10% of the population. None of the reported false positives violate the popularity constraints, with the exceptions of `ad.doubleclick.net` (141 users) and `login.live.com` (84 users), reported by both MLP and RF. For definition, even if they these domains were reported by several blocklists, an access to them is not labeled as malicious. For this reason we cannot consider them as a result of the antcipatory capability of the classifiers.

As illustrated in Table 7.7, all classifiers have reported at least one access that has been identified by a blocklist within the following days. The number of accesses anticipated

with respect to at least one blocklist is relatively low with a value of 11.67% for Random Forest, 4.91% for MLP, and 6.25% for LSTM.

Finally, if we focus on the only domains that are not considered malicious, but subsequently they respect both maliciousness and popularity constraints.

- The RF model anticipates the blocklist `rpiList-malware` by classifying `s.optnx.com` 3 days earlier.
- The MLP model anticipates the blocklist `rpiList-malware`, classifying `blockadsnot.com` and `s.optnx.com` approximately one and a half months and 3 days earlier, respectively.
- The LSTM model anticipates the blocklists `iam-py-test` and `rpiList-malware`, classifying `ptaimpeerte.com` 4 days earlier for both blocklists.

It is worth noting that only LSTM was able to anticipate both blocklists that would render the access malicious after 4 days from the time of access. However, each classifier has at least one instance of malicious classified access that anticipates the classification of one or more blocklists. In addition, all classifiers anticipate accesses by only a few days with respect to the blocklist.

Model	URL	Time of accesses	Report date	Blocklist
Random Forest	login.live.com	2022-07-03 16:21:14	2022-08-12	tweet-feed-today
	jrpkizae.com	2022-06-20 00:11:54	2022-08-12	rpiList-malware
	effusedprankle.com	2022-08-06 23:30:19	2022-08-12	rpiList-malware
	rtbrvdirect.com	2022-07-11 22:10:59	2022-08-12	rpiList-malware
	s.optnx.com	2022-08-09 20:55:46	2022-08-12	rpiList-malware
	www.safestgatetocontent.com	2022-07-11 22:11:09	2022-08-12	rpiList-malware
MLP	blockadsnot.com	2022-06-24 01:27:36	2022-08-12	rpiList-malware
	login.live.com	2022-06-07 08:23:03	2022-06-10	tweet-feed-today
	login.live.com	2022-06-10 07:52:33	2022-06-27	tweet-feed-today
	login.live.com	2022-07-03 16:21:14	2022-08-12	tweet-feed-today
	grandsupple.com	2022-09-16 23:29:00	2022-09-18	rpiList-malware
	jrpkizae.com	2022-06-20 00:11:54	2022-08-12	rpiList-malware
	effusedprankle.com	2022-08-06 23:30:19	2022-08-12	rpiList-malware
	snoreempire.com	2022-07-05 02:36:24	2022-08-12	rpiList-malware
	go.xlviirdr.com	2022-06-23 23:50:14	2022-07-01	ultimate-host-blacklist-1
	download-ready.net	2022-06-24 02:01:57	2022-08-12	rpiList-malware
	rtbrvdirect.com	2022-07-11 22:10:59	2022-08-12	rpiList-malware
	s.optnx.com	2022-08-09 20:55:46	2022-08-12	rpiList-malware
	ad.doubleclick.net	2022-06-16 13:34:04	2022-08-12	rpiList-malware
	ad.doubleclick.net	2022-07-14 03:11:22	2022-08-12	rpiList-malware
LSTM	ptaimpeerte.com	2022-07-08 21:07:44	2022-07-12	iam-py-test
	ptaimpeerte.com	2022-07-08 21:07:44	2022-08-12	rpiList-malware

Table 7.7. Summary Table of the anticipated domains.

7.4 Discussion

The experimental results demonstrate the varying behaviors of the evaluated classifiers under different conditions. Firstly, in the 80/20 split, all models (with the exception

of Random Forest) achieved good performance in distinguishing between benign and malicious accesses. LSTM achieved the best overall performance thanks to its ability to capture temporal dependencies in sequential accesses, confirming its suitability for sequential data modeling. The MLP also demonstrated balanced precision and recall, while the Random Forest exhibited a tendency to misclassify malicious accesses as benign, but the cause can be attributed to a malicious 80/20 split as it later showed improvements in both unbalance ratio and LOUO, bucking the trend of the remaining models.

In LOUO evaluation, all models experienced a decrease in performance, especially in the recall of the malicious class. This is an expected outcome, as the LOUO scenario introduces a more realistic condition where the model must generalize to users whose behavior it has never seen before. Among the three models, the LSTM maintained the most stable performance, showing that temporal models can better generalize across users sequences of accesses by learning behavioral patterns rather than access-specific features. The analysis of the unbalance ratio confirmed that the disproportion between classes significantly affects classifier behavior. Excessive oversampling of the legitimate class leads to an artificial bias that reduces the model's ability to detect rare malicious accesses. Finally, it is important to note that every model has demonstrated anticipatory capabilities by detecting malicious accesses before the report date of the access domain in a blocklist.

The results demonstrate that:

- *LSTM* is the most robust classifier across different testing conditions, maintaining acceptable performance even in realistic scenarios.
- *MLP* provides a good trade-off between computational efficiency and accuracy, but is more sensitive to data distribution variations.
- *Random Forest.* The preliminary results are not satisfactory, but , subsequently, we have demonstrated that Random Forest is a valid solution that can be used to train on a bigger set of data, because its light computational weight.

These findings suggest that temporal and sequential models such as LSTM are preferable for real-world applications of malicious access detection, where user behavior and access patterns play a crucial role.

Chapter 8

Conclusions and future works

The objective of this thesis was to detect malicious web accesses through the analysis of users' browsing data. To this end, we label the *Pimcity dataset*'s accesses by limning them with malicious domains reported by various open-source blocklists. Next, we researched for relevant features in the literature, and validates the proposed approach through multiple experiments. To ensure a combination of results from different models, and to guarantee a more reliable and accurate result, a total of three distinct machine learning classifiers were developed based on *Random Forest*, *Multilayer Perceptron (MLP)*, and *Long-Short Term Memory (LSTM)* respectively. The experimental results demonstrated that each model achieved a satisfactory level of accuracy in classifying malicious accesses. In particular, the LSTM model outperformed both the MLP and the Random Forest classifiers, highlighting the importance of navigational patterns in such tasks. Subsequently, the classifiers were examined under different conditions to make the classification task more challenging. The results on the oversampling variations revealed a strong correlation between label imbalance and the classifiers' ability to correctly detect malicious activity.

Moreover, we create a more challenging dataset (*Leave One User Out (LOUO) dataset*) to test all the classifiers. The *LOUO dataset* was constructed by removing all accesses (or sequences) belonging to a single user from the original dataset and utilizing the remaining accesses for the training process. The objective of the experiment is to simulate the behavior of a model that is installed on a machine as a countermeasure to malicious accesses. Although performance experienced a slight decrease, experiments conducted using the *LOUO dataset* consistently demonstrated that training on a proportionally imbalanced dataset was sufficient to detect malicious accesses in an strongly unbalanced test set.

Despite the promising results, limitations exist, and the outcomes could be further improved. Firstly, analyzing and identifying an access only by the domain imposes a restrictive condition. As discussed in Chapter 2.1.3, many hosts require additional information, resulting in extensive communication between clients and servers. The incorporation of more detailed access information would therefore be of crucial importance, as it would enable the application of the classifier to the entire Internet communication, and not only to direct requests. In addition, data on user-browser interactions — such

as clicks, cursor movements, and information about open tabs — could provide a more accurate representation of user behavior. Moreover, larger and updated datasets could better investigate the anticipatory capabilities of the classifiers.

Encompassing the various limitations, however, the study shows that it is possible to use users' navigational patterns to achieve satisfactory levels of accuracy in the classification of web accesses. The application of classifiers is multiple. For instance, the classifier could be integrated with other security mechanisms for improving reliability during communication. An Intrusion Detection System (IDS) could monitor the traffic of a network through a behavioral recognition classifier to detect suspicious behavioral patterns and select them for further analysis. In conclusion, this thesis represents a preliminary step towards future research on adaptive, behavior-based cybersecurity systems.

Bibliography

[1] Areej Alhogail and Isra Al-Turaiki. Improved detection of malicious domain names using gradient boosted machines and feature engineering. *Information Technology and Control*, 51(2):313–331, 2022.

[2] antiscam-squad-cypto scam. antiscam-squad-cypto-scam. https://raw.githubusercontent.com/AntiScam-Squad/pi-hole/main/crypto_scam.txt. Accessed: 2025-09-19.

[3] azorult tracker. azorult-tracker. https://azorult-tracker.net/api/list/domain?format=plain. Accessed: 2025-09-19.

[4] blocklist malware. blocklist-malware. https://raw.githubusercontent.com/blocklistproject/Lists/refs/heads/master/alt-version/malware-nl.txt. Accessed: 2025-09-19.

[5] blocklist phishing. blocklist-phishing. https://blocklistproject.github.io/Lists/alt-version/phishing-nl.txt. Accessed: 2025-09-19.

[6] blocklistproject abuse. blocklistproject-abuse. https://blocklistproject.github.io/Lists/abuse.txt. Accessed: 2025-09-19.

[7] blocklistproject ads. blocklistproject-ads. https://blocklistproject.github.io/Lists/ads.txt. Accessed: 2025-09-19.

[8] blocklistproject crypto. blocklistproject-crypto. https://blocklistproject.github.io/Lists/crypto.txt. Accessed: 2025-09-19.

[9] blocklistproject fraud. blocklistproject-fraud. https://blocklistproject.github.io/Lists/fraud.txt. Accessed: 2025-09-19.

[10] blocklistproject malware. blocklistproject-malware. https://blocklistproject.github.io/Lists/malware.txt. Accessed: 2025-09-19.

[11] blocklistproject ransomware. blocklistproject-ransomware. https://blocklistproject.github.io/Lists/ransomware.txt. Accessed: 2025-09-19.

[12] blocklistproject redirect. blocklistproject-redirect. https://blocklistproject.github.io/Lists/redirect.txt. Accessed: 2025-09-19.

[13] blocklistproject scam. blocklistproject-scam. https://blocklistproject.github.io/Lists/scam.txt. Accessed: 2025-09-19.

[14] Google Safe Browsing. Google safe browsing.

[15] Renée Burton and Laura Rocha. Whitelists that work: Creating defensible dynamic whitelists with statistical learning. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–10. IEEE, 2019.

[16] Davide Canali, Leyla Bilge, and Davide Balzarotti. On the effectiveness of risk

prediction based on users browsing behavior. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 171–182, New York, NY, USA, 2014. Association for Computing Machinery.

[17] cert-pl domains. cert-pl-domains. https://hole.cert.pl/domains/domains.txt. Accessed: 2025-09-19.

[18] chainapsis-phishing-block list. chainapsis-phishing-block-list. https://raw.githubusercontent.com/chainapsis/phishing-block-list/main/block-list.txt. Accessed: 2025-09-19.

[19] Kyle Crichton, Nicolas Christin, and Lorrie Faith Cranor. How do home computer users browse the web? *ACM Trans. Web*, 16(1), September 2021.

[20] Sumitra Das Guptta, Khandaker Tayef Shahriar, Hamed Alqahtani, Dheyaaldin Alsalman, and Iqbal H Sarker. Modeling hybrid feature-based phishing websites detection using machine learning techniques. *Annals of Data Science*, 11(1):217–242, 2024.

[21] Politecnico di Torino. Pimcity - building the next generation personal data platforms. https://www.pimcity-h2020.eu/, 2020. Accessed: 2025-09-30.

[22] discord antiscam. discord-antiscam. https://github.com/Discord-AntiScam/scam-links/blob/main/list.txt. Accessed: 2025-09-19.

[23] DomCop. Openpagerank, 2025. Accessed: 2025-09-29.

[24] Patcg Individual Drafts. Taxonomy v2. https://github.com/patcg-individual-drafts/topics/blob/main/taxonomy_v2.md, 2025. Accessed: 2025-09-29.

[25] durablenapkin scamblocklist. durablenapkin-scamblocklist. https://raw.githubusercontent.com/durablenapkin/scamblocklist/master/hosts.txt. Accessed: 2025-09-19.

[26] eth-phishing detect. eth-phishing-detect. https://raw.githubusercontent.com/MetaMask/eth-phishing-detect/master/src/hosts.txt. Accessed: 2025-09-19.

[27] World Economic Forum. 72% of cyber leaders say cybersecurity risks are rising, 2025. Accessed: 2025-09-16.

[28] foxwallet domains. foxwallet-domains. https://raw.githubusercontent.com/foxwallet/blacklist/master/domain.json. Accessed: 2025-09-19.

[29] GitHub, Inc. Github, 2025.

[30] global-anti-scam-org-scam urls. global-anti-scam-org-scam-urls. https://raw.githubusercontent.com/elliotwutingfeng/GlobalAntiScamOrg-blocklist/main/global-anti-scam-org-scam-urls.txt. Accessed: 2025-09-19.

[31] global-anti-scam-org-scam-urls pihole. global-anti-scam-org-scam-urls-pihole. https://raw.githubusercontent.com/elliotwutingfeng/GlobalAntiScamOrg-blocklist/main/global-anti-scam-org-scam-urls-pihole.txt. Accessed: 2025-09-19.

[32] Google. Google topics api. https://developers.google.com/topics. Accessed: 29 settembre 2025.

[33] Muhammad Yasir Muzayan Haq, Mattijs Jonker, Rol Van Rijswijk-Deij, Kimberly C Claffy, Lambert JM Nieuwenhuis, and Abhishta Abhishta. No time for downtime: understanding post-attack behaviors by customers of managed dns providers. In

2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pages 322–331. IEEE, 2022.

[34] Zhongyi Hu, Raymond Chiong, Ilung Pranata, Willy Susilo, and Yukun Bao. Identifying malicious web domains using machine learning techniques with online credibility and performance data. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 5186–5194. IEEE, 2016.

[35] iam-py test. iam-py-test. https://raw.githubusercontent.com/iam-py-test/my_filters_001/main/Alternative%20list%20formats/antimalware_domains.txt. Accessed: 2025-09-19.

[36] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, and Roya Ensafi. A study of third-party resources loading on web. *arXiv preprint arXiv:2203.03077*, 2022.

[37] inversions-dnsbl blocklists. inversions-dnsbl-blocklists. https://raw.githubusercontent.com/elliotwutingfeng/Inversion-DNSBL-Blocklists/main/Google_hostnames.txt. Accessed: 2025-09-19.

[38] Nikhil Jha, Martino Trevisan, Emilio Leonardi, and Marco Mellia. On the robustness of topics api to a re-identification attack. *arXiv preprint arXiv:2306.05094*, 2023.

[39] Wei Jiang, Yuan Tian, Weixin Liu, and Wenmao Liu. An insider threat detection method based on user behavior analysis. In *International conference on intelligent information processing*, pages 421–429. Springer, 2018.

[40] kadomains. kadomains. https://raw.githubusercontent.com/FiltersHeroes/KADhosts/master/KADomains.txt. Accessed: 2025-09-19.

[41] kitsapcreator malware. kitsapcreator-malware. https://raw.githubusercontent.com/ClovisWebDev/pihole-blocklists/refs/heads/master/malware-malicious.txt. Accessed: 2025-09-19.

[42] kitsapcreator spam. kitsapcreator-spam. https://blocklists.kitsapcreator.com/scam-spam.txt. Accessed: 2025-09-19.

[43] Geza Kovacs. Reconstructing detailed browsing activities from browser history. *arXiv preprint arXiv:2102.03742*, 2021.

[44] Juhi Kulshrestha, Marcos Oliveira, Orkut Karaçalık, Denis Bonnay, and Claudia Wagner. Web routineness and limits of predictability: investigating demographic and behavioral differences using web tracking data. In *Proceedings of the international AAAI conference on web and social media*, volume 15, pages 327–338, 2021.

[45] L Lakshmi, M Purushotham Reddy, Chukka Santhaiah, and U Janardhan Reddy. Smart phishing detection in web pages using supervised deep learning classification and optimization technique adam. *Wireless Personal Communications*, 118(4):3549–3564, 2021.

[46] Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 20–39. Springer, 2007.

[47] Sylvain Lugeon, Tiziano Piccardi, and Robert West. Homepage2vec: Language-agnostic website embedding and classification. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 16, pages 1285–1291, 2022.

[48] Trend Micro. Trend 2025 cyber risk report, 2025. Accessed: 2025-09-16.

[49] Rida Nasir, Mehreen Afzal, Rabia Latif, and Waseem Iqbal. Behavioral based insider threat detection using deep learning. *IEEE Access*, 9:143266–143274, 2021.

[50] nocoin-adblock list. nocoin-adblock-list. https://raw.githubusercontent.com/hoshsadiq/adblock-nocoin-list/master/hosts.txt. Accessed: 2025-09-19.

[51] null-host-bad domains. null-host-bad-domains. https://raw.githubusercontent.com/kioan/null-hosts/main/hosts. Accessed: 2025-09-19.

[52] Changkun Ou, Daniel Buschek, Malin Eiband, and Andreas Butz. Modeling web browsing behavior across tabs and websites with tracking and prediction on the client side. *arXiv preprint arXiv:2103.04694*, 2021.

[53] phishfort domains. phishfort-domains. https://raw.githubusercontent.com/phishfort/phishfort-lists/master/blacklists/domains.json. Accessed: 2025-09-19.

[54] PIMCity Project. Personal privacy preserving analytics (p-ppa). https://gitlab.com/pimcity/wp2/personal-privacy-preserving-analytics, 2020. Accessed: 2025-09-30.

[55] A Saleem Raja, R Vinodini, and A Kavitha. Lexical features based malicious url detection using machine learning techniques. *Materials Today: Proceedings*, 47:163–166, 2021.

[56] referer spam. referer-spam. https://raw.githubusercontent.com/desbma/referer-spam-domains-blacklist/master/spammers.txt. Accessed: 2025-09-19.

[57] rpiList malware. rpilist-malware. https://raw.githubusercontent.com/RPiList/specials/master/Blocklisten/malware. Accessed: 2025-09-19.

[58] rpilist-phishing angriffe. rpilist-phishing-angriffe. https://raw.githubusercontent.com/RPiList/specials/master/Blocklisten/Phishing-Angriffe. Accessed: 2025-09-19.

[59] Marc Schmitt and Ivan Flechais. Digital deception: Generative artificial intelligence in social engineering and phishing. *Artificial Intelligence Review*, 57(12):324, 2024.

[60] search-engine spam. search-engine-spam. https://raw.githubusercontent.com/no-cmyk/Search-Engine-Spam-Domains-Blocklist/master/blocklist.txt. Accessed: 2025-09-19.

[61] SentinelOne. Key cyber security statistics for 2025, 2025. Accessed: 2025-09-16.

[62] Mahmood Sharif, Jumpei Urakawa, Nicolas Christin, Ayumu Kubota, and Akira Yamada. Predicting impending exposure to malicious content from user behavior. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1487–1501, New York, NY, USA, 2018. Association for Computing Machinery.

[63] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting security events through deep learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 592–605, 2018.

[64] soteria. soteria. https://raw.githubusercontent.com/soteria-nou/domain-list/master/fake.txt. Accessed: 2025-09-19.

[65] spam_404. spam_404. https://raw.githubusercontent.com/Spam404/lists/

master/main-blacklist.txt. Accessed: 2025-09-19.
[66] stevenblack hosts. stevenblack-hosts. https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts. Accessed: 2025-09-19.
[67] DNSFilter Team. Dnsfilter finds rise in malicious domains underscores importance of cybersecurity awareness at all levels, 2024. Accesso: 25 settembre 2025.
[68] thiojoe spamdomainslist. thiojoe-spamdomainslist. https://raw.githubusercontent.com/ThioJoe/YT-Spam-Lists/main/SpamDomainsList.txt. Accessed: 2025-09-19.
[69] Virus Total. Virus total.
[70] tweet-feed today. tweet-feed-today. https://raw.githubusercontent.com/0xDanielLopez/TweetFeed/master/today.csv. Accessed: 2025-09-19.
[71] ultimate-host-blacklist 0. ultimate-host-blacklist-0. https://raw.githubusercontent.com/Ultimate-Hosts-Blacklist/Ultimate.Hosts.Blacklist/refs/heads/master/superhosts.deny/superhosts0.deny. Accessed: 2025-09-19.
[72] ultimate-host-blacklist 1. ultimate-host-blacklist-1. https://raw.githubusercontent.com/Ultimate-Hosts-Blacklist/Ultimate.Hosts.Blacklist/refs/heads/master/superhosts.deny/superhosts1.deny. Accessed: 2025-09-19.
[73] ultimate-host-blacklist 2. ultimate-host-blacklist-2. https://raw.githubusercontent.com/Ultimate-Hosts-Blacklist/Ultimate.Hosts.Blacklist/refs/heads/master/superhosts.deny/superhosts2.deny. Accessed: 2025-09-19.
[74] ultimate-host-blacklist 3. ultimate-host-blacklist-3. https://raw.githubusercontent.com/Ultimate-Hosts-Blacklist/Ultimate.Hosts.Blacklist/refs/heads/master/superhosts.deny/superhosts3.deny. Accessed: 2025-09-19.
[75] Hai Truong Van Thanh. Relationships between web traffic ranks and online sales revenue of e-retailers in australia. *Unpublished manuscript, Central Queensland University. https://doi. org/10.13140/RG*, 2(20883.37924), 2018.
[76] Alberto Verna, Nikhil Jha, Martino Trevisan, and Marco Mellia. A first view of topics api usage in the wild. In *Proceedings of the 20th International Conference on emerging Networking EXperiments and Technologies*, pages 48–54, 2024.
[77] Jialei Wang, Ji Wan, Yongdong Zhang, and Steven CH Hoi. Solar: Scalable online learning algorithms for ranking. ACL, 2015.
[78] IBM X-Force. Ibm x-force 2025 threat intelligence index, 2025. Accessed: 2025-09-16.
[79] Wei Xu, Kyle Sanders, and Yanxin Zhang. We know it before you do: predicting malicious domains. In *Virus Bulletin Conference*, pages 73–77, 2014.
[80] Yuanhui Yu. Web page classification algorithm based on deep learning. *Computational intelligence and neuroscience*, 2022(1):9534918, 2022.
[81] Fangfang Yuan, Yanan Cao, Yanmin Shang, Yanbing Liu, Jianlong Tan, and Binxing Fang. Insider threat detection with deep neural network. In *International Conference on Computational Science*, pages 43–54. Springer, 2018.

[82] Hong Zhao, Zhaobin Chang, Weijie Wang, and Xiangyan Zeng. Malicious domain names detection algorithm based on lexical analysis and feature quantification. *IEEE Access*, 7:128990–128999, 2019.