LUISS

Degree Program in Data Science and Management

Course of Data Science In Action

Leveraging ML to fight back the cybersecurity risk of credential stuffing attacks

Prof. Alessio Martino

SUPERVISOR

Prof. Emilio Coppa

CO-SUPERVISOR

760361

CANDIDATE

Academic Year 2023/2024

Table of contents

1) Introduction

Credential stuffing is a cyberthreat in which the attacker gets control of a series of leaked credentials (e.g., login credentials such as username/email and password) and tries to get access to users' accounts so as to take over them, hoping that the user has chosen the same authentication credentials across multiple websites and services. Although it is often mistaken for a brute force attack, there are clear differences between the two threats: in credential stuffing, the attacker comes into possession of breached credentials which are usually compiled in lists containing billions of records and that are made available through hacker forums on the Deep/Dark Web. On the other hand, in a brute force attack, the attacker tries to guess the user's password, usually starting from a dictionary of most commonly used passwords or, in the worst case, by means of combinatorial trial-and-error routines. Overall, the key difference is that, in the former case, credentials (passwords) are already available once the attack is perpetrated.

Any credential stuffer's expectations of password reuse find evidence in real-world data: according to Bitwarden's (a well-known password management service) 2023 survey, as many as 85% of users tend to reuse the same password for multiple services, especially when such services constrain them to use (hence, remember) long and complicated passwords. Despite having a rate of success as low as 0.1%, the large volume of stolen credentials at the disposal of attackers makes credential stuffing worth it. This implies that, if millions of leaked credentials are available, an attacker could get hold of thousands of accounts, withdrawing all the funds associated with them or stealing users' identities to carry out other cybercrimes. The threat of credential stuffing is real: by the end of 2020, Akamai observed 193 billion credential stuffing attacks globally, of which 3.4 billion aimed at financial institutions, with a 45% increase on yearly basis. According to Okta's report of 2022, more than 10 billion credential stuffing attempts occurred on its platform in the first quarter of the year and in some countries the share of fraudulent login attempts exceeded that of legitimate attempts. The situation is made even more complicated by the difficulty of distinguishing benign from fraudulent login attempts: it is estimated that in some industries such as e-commerce, airline and consumer banking, the share of login attempts which are classified as credential stuffing can be as large as 90%. However, some characterizing elements can still be identified: large number of login attempts coming from the same IP address (although attackers can rely on IP proxy servers to hide their real IP), higher than average volume of FISPs (Foreign Internet Service Providers), changes in site traffic like multiple login attempts from multiple accounts within a limited timeframe, higher than usual failed login rate, downtimes due to increase in site traffic and anomalies in HTTP headers (user agent strings). It follows that these features could be leveraged to assess the credential stuffing risk of login attempts. So far, the literature has focused on device fingerprinting, using clickstream data to assess the credential stuffing risk of login attempts, leveraging differences between infrequent and engaged users [1] and detection protocols based on private set intersection to foster communication and coordination between servers [2] [3]. In the context of machine learning, [4] proposes a supervised learning model to detect credential stuffing attacks carried out using

bots, leveraging behavioral biometrics such as mouse and keystroke dynamics. Both [5] and [6] suggest the use of artificial neural networks, with [5] focusing on an optimization technique known as EWO (enhanced whale optimization) to select the best weights and train the neural network and [6] on a password-similarity model and a PPSM (personalized password strength meter), based on an embedding model, to warn users when they choose vulnerable credentials. [7] recommends a pattern-based algorithm that leverages login data to assign a score to each login event and to take preventive and corrective actions once some predefined thresholds are exceeded. It follows that, to the best of our knowledge, there are no previous studies that try to tackle credential stuffing by leveraging login-specific features to train a classifier that aims to predict the cybersecurity risk of login attempts in terms of account takeover and this is the research gap that this thesis is meant to address. Starting from a synthesized dataset, containing login data of millions of login attempts on a large-scale online service located in Norway, and used in the context of RBA (risk-based authentication) [8], we will build several classification algorithms (Naive Bayes, Logistic Regression, Decision Tree and ensemble models) whose performance will be evaluated in terms of robust classification metrics such as precision, recall, F1-score and the Matthews's correlation coefficient given the highly imbalanced nature of the target variable.

The rest of the work is organized as follows: section 2 will provide a detailed review of previous research in the context of credential stuffing and highly imbalanced data classification, as well as a brief review of the work in the context of RBA which represents the source of the dataset we used for the analysis, section 3 will present the methods developed to tackle the problem, whose results will be displayed and discussed in section 4. Finally, in section 5, we will wrap up by identifying the main implications of our work as well as future areas of potential research.

2) Theory/Literature review

2.1 Credential stuffing

Related literature tends to tackle the problem of credential stuffing from different perspectives. As stated in [6], credential stuffing attacks are the primary source of account takeover and credential stuffing attempts can account for up to 90% of login traffic at some of the world's largest websites. As humans are the weakest link in the cybersecurity chain, one of the key vulnerabilities that enable credential stuffing is the similarity of user-chosen passwords across multiple websites and services. Based on this consideration, the authors of [6] used 1.4 billion leaked emails and passwords to train a password similarity model, namely a model that takes as input a leaked authentication credential and produces as output credentials that are likely to be generated starting from the one that has been compromised. Furthermore, they suggested a PPSM (personalized password strength meter), using neural network-based word embedding techniques, as a defense to warn users when they are choosing vulnerable authentication credentials, considering past targeted attacks to which users have been exposed. According to [9], credential stuffing attacks are both hard to resolve and difficult to detect, with yearly business costs estimated at an average of $6 million. It is difficult to distinguish between benign login attempts and credential stuffing attacks, especially for websites that observe a large number of logins. However, [9] mentions several characterizing elements that should not be overlooked, such as large volume of wrong username or password errors, large numbers of requests coming from the same IP and rise in traffic coming from locations that break the regular usage pattern. Traffic spikes and hourly patterns of traffic are important elements to detect automated attacks. Based on these elements, the authors of [3] proposed a framework which leverages anomaly detection to differentiate between normal login behavior and credential stuffing attempts. Furthermore, they suggested a private membership protocol, relying on cuckoo filters, through which websites can coordinate to detect credential stuffing attacks, using MDPs (Markov Decision Processes) to evaluate the true detection rate and false detection rate of the framework. A similar approach is followed by [2] which proposes a detection framework based on cuckoo filters, with private set intersection being one of the key elements. Within the suggested framework, each of the two servers stores a set of suspicious credentials. One of the two servers issues a PSI request (query) to the other that is part of the protocol to determine the intersection of the two sets. Elements of the intersection will be reported as leaked credentials whereas all the remaining elements in the two sets are kept private.

An additional layer of complexity in detecting credential stuffing attacks is represented by the availability of botnets. According to [1], when carried out using bots and spread across several locations and days of the week, credential stuffing attempts are hard to distinguish from failed login attempts. The author of [1] proposed device

fingerprinting (by profiling each user's activity over time) as an effective measure against account takeover attempts through credential stuffing. When an attacker accesses a user's account through stuffed credentials, certain patterns of behavior can be identified. For example, users who are familiar with a system tend to exhibit goal-oriented behavior whereas fraudsters may tend to meander through screens, analyzing pages and paths before requesting an action. Based on this consideration, [1] mentions an experiment to predict fraud using "clickstream data" (user's learning behavior), with "IP Address" and "User Agent String" among the features leveraged for the predictive experiment. The use of botnets to automate the process of trying multiple authentication credentials across several websites is also highlighted in [4]. In this paper, the authors proposed a supervised learning model to detect credential stuffing bots using behavioral biometrics such as mouse and keystroke dynamics. Four different models (K-NN, SVM, Decision Tree and Random Forest) are trained and evaluated in terms of accuracy, precision, recall, F1-score and mean AUC score. The use of machine learning to counteract credential stuffing is also advocated by the authors of [7] who suggest a pattern-based machine learning model to detect credential stuffing as traditional mitigation strategies such as rate control checks on IP addresses, rate control checks on failed login attempts, bot and fraud detection systems can be easily bypassed. The proposed algorithm leverages IP address, ISP (Internet Service Provider), ASN (Autonomous System Number), Country, Username and an unsupervised anomaly detection model to establish thresholds for the aforementioned features. A risk score is assigned to each login event and corrective actions are taken for attempts that exceed predefined thresholds. Still in the context of machine learning, the authors of [5] suggest an ANN (Artificial Neural Network) model with the EWO (Enhanced Whale Optimization) technique that is used to train the neural network and find a near-optimal solution when computational resources are limited.

2.2 RBA (Risk-Based Authentication)

RBA (Risk-Based Authentication) represents an alternative to traditional methods such as 2FA (2-Factor Authentication) and MFA (Multi-Factor Authentication) to strengthen password-based authentication and protect users against cyberattacks that entail the use of stolen credentials. RBA aims to estimate whether a login is legitimate or an account takeover attempt by comparing feature values (ex related to the network or device) pertaining to the login attempt with those of previous ones in a login history, assigning a risk score to each login attempt. The main work that we will review in this subsection is [8], which represents the first in-depth analysis of RBA on a large-scale online service. The authors opted for the well-known Freeman et al. model, whose main equation describes the risk score that is assigned to each user based on a series of feature values. (Fig. 1)

$$S_u(FV) = \left(\prod_{k=1}^{d} \frac{p(FV^k)}{p(FV^k|u, legit)} \right) \frac{p(u|attack)}{p(u|legit)}.$$

Fig. 1 Freeman et al. model's main equation

Fig. 1 can be explained as follows: the risk score S of each user u, considering a set of feature values FV, is given by the sum all over the d features of the probability that the value of feature k is observed in the global login history divided by the probability that the value of feature k is observed in the legitimate user's login and $p(u|attack) = 1/|U|$ where U is the number of users or it can be estimated as the number of failed login attempts and $p(u|legit)$ is equal to the number of user logins divided by the total number of logins.

The dataset that the authors of [8] used for their analysis contains four main features: "Login Timestamp", "Round-Trip Time [ms]", "IP Address", and "User Agent String", with several sub-features that are derived from "IP Address" ("ASN", "Country", "Region", "City") and "User Agent String" ("Browser", "Operating System", "Device Type"). The Freeman et al. model was used to derive weights for each feature (or subfeature) and assign a larger weight to those features that are harder to spoof. Finally, the model was tested against four types of attacker: naive attacker (simulated using IP addresses of failed login attempts found in datasets of real-world attacks), VPN attacker (simulated using IP addresses located in the victim's main country), targeted attacker (simulated using IP addresses located in the victim's main country, sampling IP addresses and user agent strings from failed login attempts with the most occurrences), very targeted attacker (simulated using true account takeover instances contained in the dataset). The main implication of this work towards our goal of building a predictive model for credential stuffing using machine learning is that the dataset used for the RBA analysis contains many features that, as previously mentioned, have distinguishing values in the context of credential stuffing such as "IP Address", "Country", "ASN" (meant as Internet Service Provider) and "User Agent String" which can be used to build a fingerprint of the user's device. Furthermore, machine learning finds application in the context of risk-based authentication as well, where it can be used to establish a dynamic access threshold based on the user's login history size. Finally, the last research question that the authors of the paper asked themselves concerns the possibility of using "Round-Trip Time [ms]" (which measures the latency between client's request and server's response) as an alternative to "IP Address". By comparing them in terms of RSR (risk score relation, defined as the ratio of mean attacker risk score to mean legitimate user score), they concluded that the former can be used to verify geolocation information provided by the latter and is a good alternative to identify users. It follows the key importance of this feature towards our predictive goal.

2.3 Highly imbalanced data classification

Imbalanced data classification is a setting in which the binary/multiclass target variable is characterized by skewed data distribution, with unequally represented classes. In the context of binary classification, an imbalanced distribution implies that one of the two classes (referred to as the minority class) is severely under-represented. This makes it hard for machine learning models to learn the decision boundary in order to differentiate between the two classes since, intuitively, the larger the sample size, the higher the chances from the model to be able to characterize the class itself. The situation is made even more complicated by the concepts of class overlap and small disjuncts (classification rules that hold only for a limited number of training instances). It follows the need to find ways to cope with the class imbalance in order to improve any model's predictive performance, as well as choosing appropriate evaluation metrics that are not affected by the imbalanced scenario as traditional quality measures (ex. accuracy) can be misleading as they lead to "good" results but poor performance, as per [16]. In both [10] and [12], the authors present two main approaches to deal with the problem of class imbalance, cost-sensitive learning (which encapsulates the notion of cost matrix, namely setting a higher cost for the misclassification of instances belonging to the minority class) and sampling (undersampling of the majority class or oversampling of the minority one), with [10] discussing the performance of two variations of Random Forest (Weighted Random Forest and Balanced Random Forest) in terms of precision, recall and F1-score. The idea of resorting to an ensemble learning method is also discussed in [11] where the authors suggest the idea of starting with a "weak" base classifier and iteratively boosting its performance by generating synthetic examples for both the majority and minority classes, assigning a larger weight to examples that are harder to classify and a low weight to those that have been correctly labeled. This concept of "classification hardness" is also picked up by [20] in which the authors propose a framework that iteratively performs undersampling by selecting the most informative examples from the majority class according to their hardness distribution. The advantage of this resampling strategy is that it does not require a distance metric, unlike traditional distance-based methods which are computationally inefficient especially on very large datasets. The idea of adopting an hybrid strategy at data level (namely the combination of both undersampling and oversampling) is cited in [16] where the authors suggest weighted algorithmic-level approaches and ensemble-level approaches as well. In addition to the techniques we already mentioned, [13] presents alternative methods to tackle the problem of class imbalance, namely kernel-based learning methods, active learning methods, one-class learning and anomaly/novelty detection. With respect to the latter, a paper by Lee and Cho, [25], suggests that novelty detection methods are useful for highly imbalanced datasets whereas traditional discrimination-based classifiers are more suitable for moderately imbalanced datasets. To evaluate any classifier's performance, [13] proposes metrics such as precision, recall, F1-score, G-mean and area under the ROC curve. In addition to the evaluation metrics just mentioned, [20] suggests the use of MCC (Matthew's correlation

coefficient) and AUPRC (area under the precision-recall curve). The idea of framing the classification problem as an anomaly detection one is also discussed in [14] where the authors, with respect to the context of financial transactions, suggest to give a score in such a way that outliers are characterized by smaller values. However, treating the problem in this way would prevent us from computing any evaluation metric since data would not be labeled. Anomaly detection is also presented in [18] as a way to guide the resampling process to pay more attention to the minority class and class overlap. The idea of the authors is to give an anomaly score to each example, split the majority class into k bins according to the anomaly score and perform under-sampling from each bin to train each classifier in a more balanced scenario. The final output of the ensemble learning framework proposed in [18] is the output of each base classifier multiplied by a weight, with a larger weight that is assigned to classifiers with better recall or that are better at discovering class overlap. The use of a weighted scheme is also put forth by [19], in which the authors propose a framework called EASE which relies on undersampling to train each base classifier in a more balanced setting, as well as a weighted scheme in which G-mean scores on the original imbalanced dataset are used as weights of each base classifier to penalize more classifiers with high false positive rate.

In [15], the performance of different algorithms (Feed Forward Neural Network, SVM, Naive Bayes and Ripper with the latter that makes use of a series of rules to describe each class) is evaluated on several imbalanced datasets and on the same datasets after applying SMOTE (Synthetic Minority Oversampling Technique) to balance them. On average, while Naive Bayes performs better on the original datasets, FFNN, SVM and Ripper tend to perform better on the datasets after applying oversampling, suggesting the importance of the latter in the context of imbalanced data classification. Finally, [17] suggests the use of k-means clustering for imbalanced data classification, clustering instances belonging to the majority class and creating as many clusters as the number of instances in the minority class. The overall idea is to train the classifier on the minority class and a class consisting of cluster centroids so that a balanced setting can be achieved.

3) Methods

3.1 Dataset description

The starting point of our analysis is a dataset available on Kaggle, a well-known data science community which provides access to thousands of datasets for artificial intelligence and machine learning projects. As already mentioned in the literature review section, it was originally used in the field of RBA (risk-based authentication) which aims to assign a risk score to each login attempt based on the user's previous login history. The dataset contains login features of more than 31 million login attempts carried out at a SSO (single sign-on) online service located in Norway. Each login attempt is characterized by the following elements: "Login Timestamp" (timestamp associated with the login attempt), "User ID" (the unique identifier of each user's account), "Round-Trip Time [ms]" (which measures the time that elapses between client's request and server's response), "IP Address" (the IP address associated with the device from which the login attempt was carried out), "Country"/"Region"/"City" (self-explanatory and derived from "IP Address"), "ASN" (autonomous system number where an autonomous system is a collection of IP prefixes that are managed and controlled by the same entity or organization), "User Agent String" (the user agent string submitted by the client), "Browser Name and Version"/"OS Name and Version"/"Device Type" (self-explanatory and derived from "User Agent String"), "Login Successful" (whether the login attempt was successful or not), "Is Attack IP" (whether the IP address associated with the login attempt had already been found in known attacker datasets) and "Is Account Takeover" (whether the login was classified as an account takeover attempt or not). We decided to frame our problem as a classification one, treating "Is Account Takeover" as the binary target. After importing the dataset using "read_csv()" from "pandas", we started its exploration by displaying some general information such as each feature's data type, number of unique/missing values for each column, percentage of missing values for each feature. After realizing that the only columns containing missing values were "Round-Trip Time [ms]" (~95.9%), "Region" (~0.15%), "City" (~0.027%) and "Device Type" (~0.0048%), we decided to drop rows containing missing values for the latter three as the information loss was negligible. The "Round-Trip Time [ms]" feature, despite the large share of missing values, provides valuable information as it measures the latency between request and response and is hard to spoof as the attacker would have to have access to a device physically located near the victim's area. Therefore, the decision to drop or impute it will be taken only after carefully studying its association with "Is Account Takeover". The subsequent step of the exploratory phase was to visualize the distribution of boolean variables "Login Successful" (60% vs 40% distribution between failed and successful login attempts), "Is Attack IP" (large class imbalance in favor of false instances) and "Is Account Takeover" (characterized by only (4×10^{-4}) % of true instances). The target's high class imbalance will affect our choice of both predictive models and evaluation metrics. After acknowledging the imbalanced nature of the classification

task, we began to familiarize ourselves with the data through visualizations involving categorical variables: we plotted the most popular countries in terms of both total (Fig. 2) and failed (Fig. 3) login attempts (showing that in the former case Norway is the most popular country whereas US is in the latter), most popular IP addresses in terms of number of associated login attempts (Fig. 4, showing that 7 out of the top 8 most popular IP addresses have the first three octets in common, meaning that they belong to the same subnet and they can directly communicate with each other by exchanging packets without having to go through a router first) and the outcome of login attempts associated with the top 10 IP addresses (showing that they all have an unsuccessful outcome). To obtain the first three plots, we used "countplot" from "seaborn", choosing the "viridis" palette for "Country" and the "Accent" color palette for "IP Address". The chosen palette for each variable remained constant throughout the code.

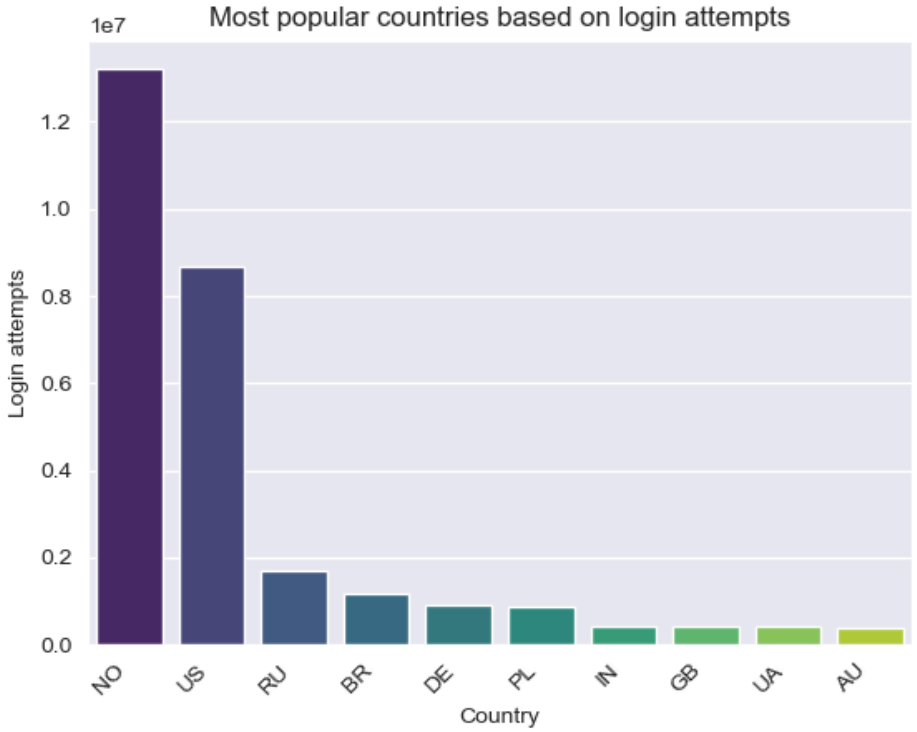


Fig. 2 Top 10 most popular countries based on total login attempts

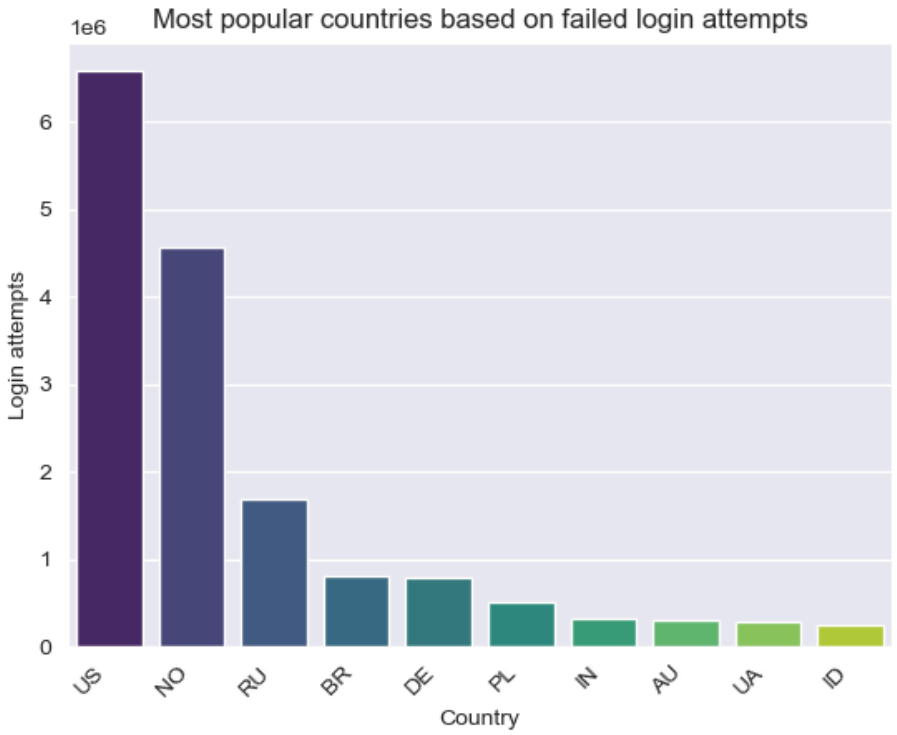


Fig. 3 Top 10 most popular countries based on failed login attempts

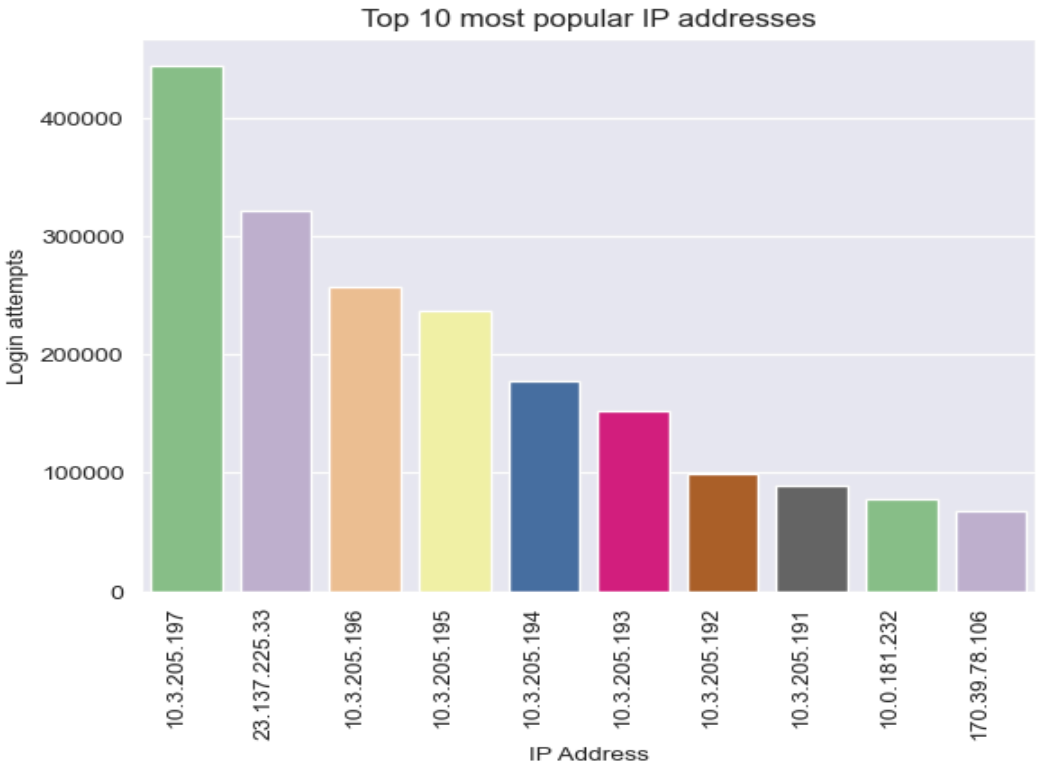


Fig. 4 Top 10 most popular IP addresses based on associated login attempts

We then decided to focus our attention on malicious IPs (Fig. 5), that is, IP addresses that have been previously found in attacker datasets.

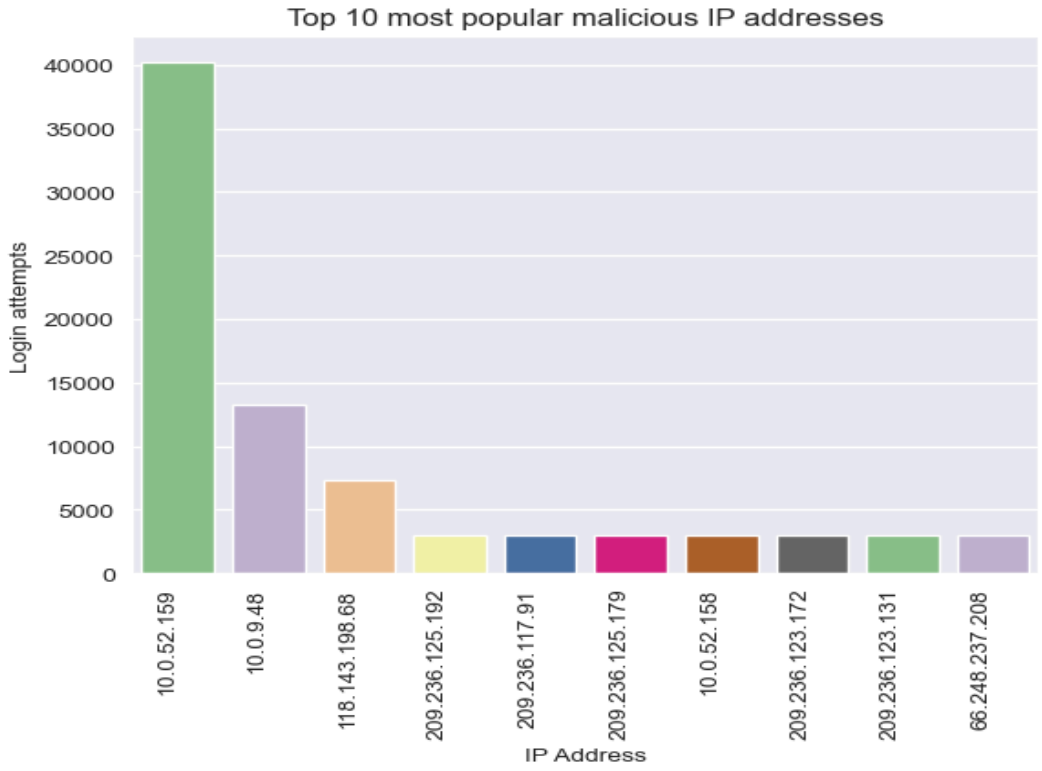


Fig. 5 Top 10 most popular IP addresses that have been found in leaked datasets

For this subset of IP addresses, we showed that the address "10.0.52.159" has been found in known attacker datasets way more often than others and that there is a common structure among some of the IPs, namely "10.0.52.159", "10.0.9.48" and "10.0.52.158" which belong to the same network having the first two octets in common, with the first and the third sharing the same subnet, with similar conclusions drawn for "209.236.125.192", "209.236.117.91", "209.236.125.179", "209.236.123.172" and "209.236.123.131". Moving on to the most commonly used browsers and Operating Systems, we acknowledged that "ZipppBot 0.11" and "Chrome Mobile 81.0.4044" are by far the most popular browsers, with the former associated only with unsuccessful login attempts, and "iOS" and "Mac OS X" are the most popular Operating Systems. In terms of "Device Type", ⅔ of logins are attempted from mobile devices whereas bot, unknown and tablet combine for ~10% of login attempts.

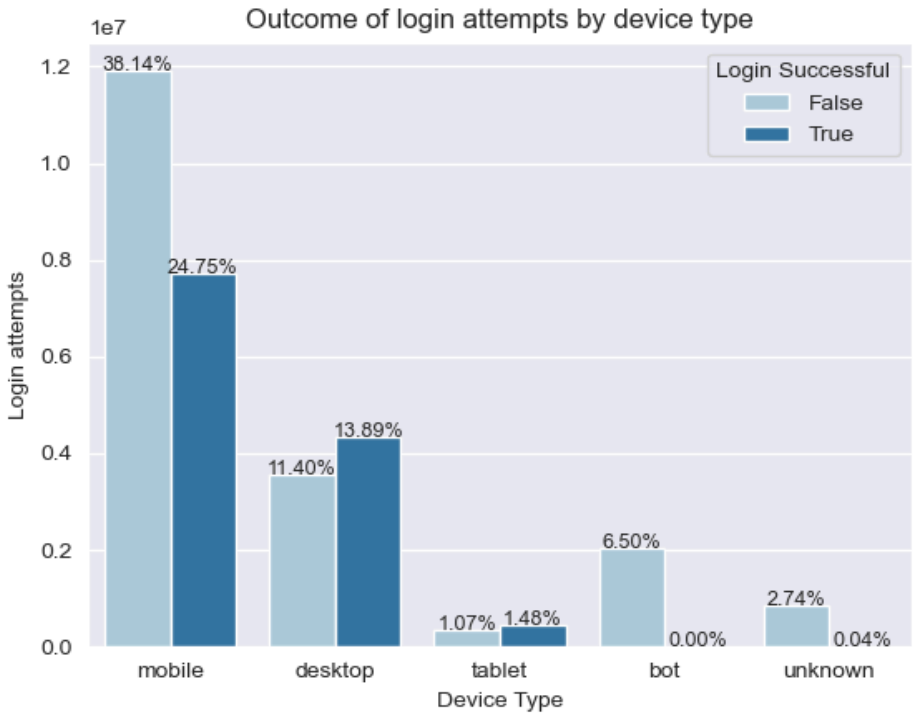


Fig. 6 Distribution of login attempts outcome across device types

When inspecting the distribution of logins outcome based on device type (Fig. 6), it emerged that all login attempts carried out using a bot/unknown device have an unsuccessful outcome, those attempted using a desktop/tablet device are more often successful than not whereas logins through mobile devices tend to follow the same distribution as the dataset's. To make the chart, we used "countplot" from "seaborn", specifying "Device Type" as x-axis, "Login Successful" as hue (the column that should be used for color encoding) and "Paired" as color palette. Each bar shows the percentage of successful/failed login attempts associated with the device.

When presenting some of the characterizing elements of credentials stuffing attacks in section 1, we pointed out that these kinds of attacks are usually associated with, among others, multiple login attempts from multiple accounts within a limited timeframe, higher than usual failed login rate and downtimes due to increases in site traffic. In order to verify these associations in the context of our dataset, we decided to plot the evolution of login attempts over time using timestamps. (Fig. 7)

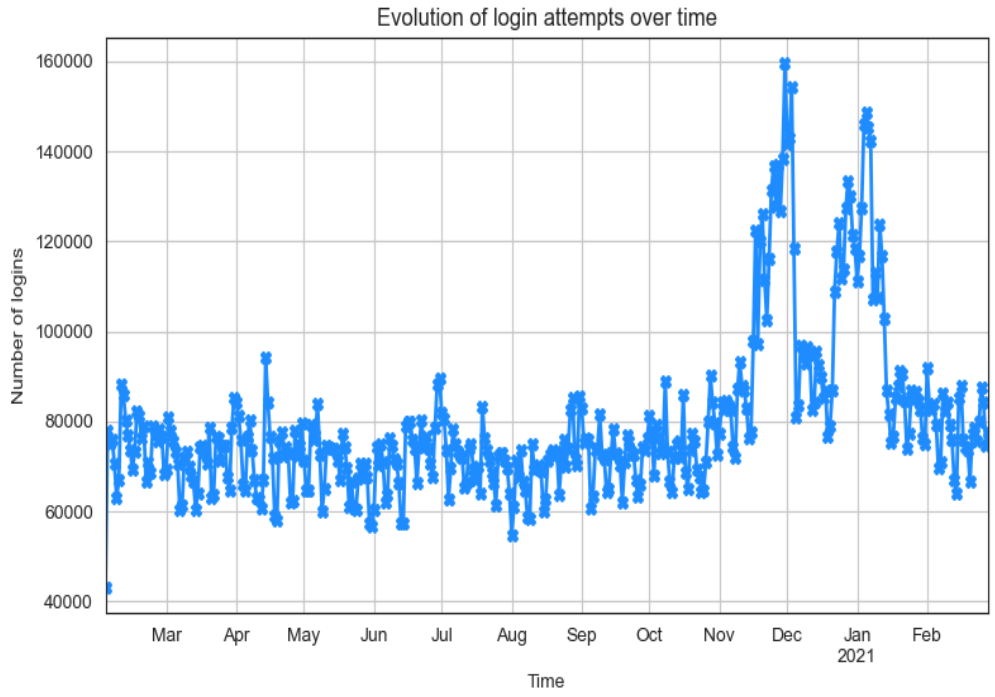


Fig. 7 Daily evolution of total login attempts over time

To make the chart, we used "plot" from "matplotlib.pyplot", specifying solid as linestyle, 2 as linewidth, "X" as marker and "dodgerblue" as color.

When plotting both successful and failed attempts together (Fig. 7), we observed a spike of traffic immediately before December 2020 and after January 2021, with fluctuations in the time period ranging from 15/11/2020 to 15/1/2021. When looking at the evolution of only failed login attempts over time (Fig. 8), it was clear that unsuccessful login attempts tend to increase in the same time period in which a spike in the total number of attempts can be witnessed. This time, we specified a different marker ("x") and color ("orangered").

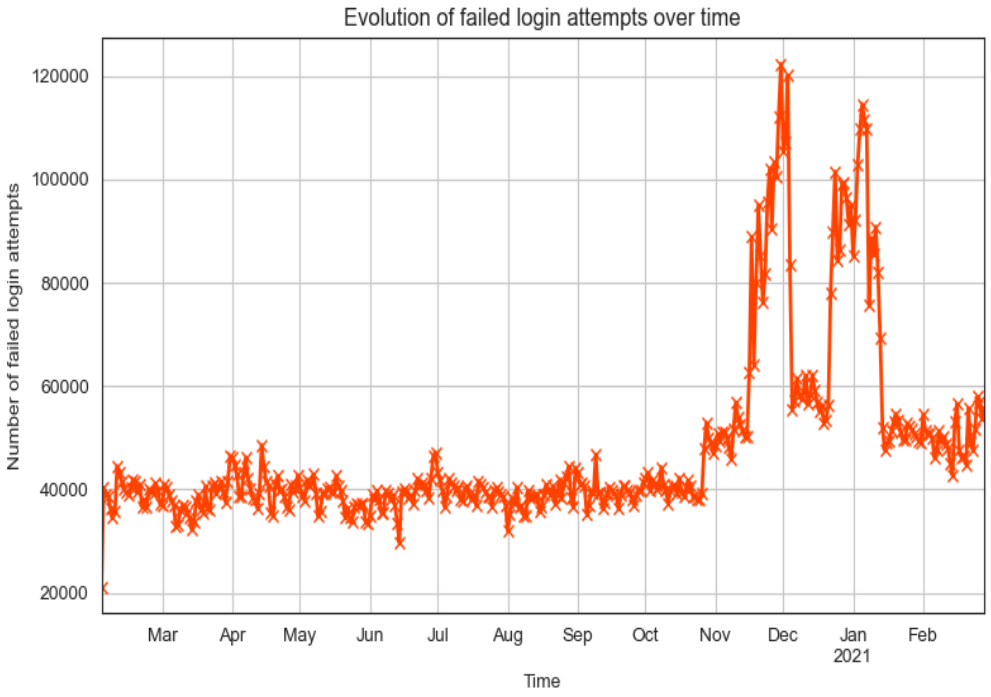


Fig. 8 Daily evolution of failed login attempts over time

Nevertheless, login attempts increased from ~80k at the beginning of November to ~160k at the end of November and from ~100k at the beginning of December to ~140k at the beginning of January whereas failed login attempts rose from ~50k at the beginning of November to ~120k at the end of the month and from ~60k at the beginning of December to ~110k at the beginning of January, suggesting that the increase in the number of failed login attempts was more than proportional.

When analyzing the distribution of login attempts across users, we realized that one user (whose –anonymized– "User ID" is -4324475583306591935) is associated with almost half of all login attempts (~14 millions). The next step of our analysis was to investigate the login behavior of the aforementioned user. In terms of login activity, we showed that login attempts carried out by the user are associated with more than 2 million different IP addresses (with the top 10 most popular that have never been found in attacker datasets), besides having an unsuccessful outcome. The last step of the exploratory phase consisted of examining the most popular ASNs in terms of login attempts (Fig. 9) and unique IP addresses associated with them. For Fig. 9, we used "countplot" from "seaborn", specifying "ASN" as x-axis, the index of the top 10 most popular ASNs by occurrences as order and "Reds_r" as color palette.

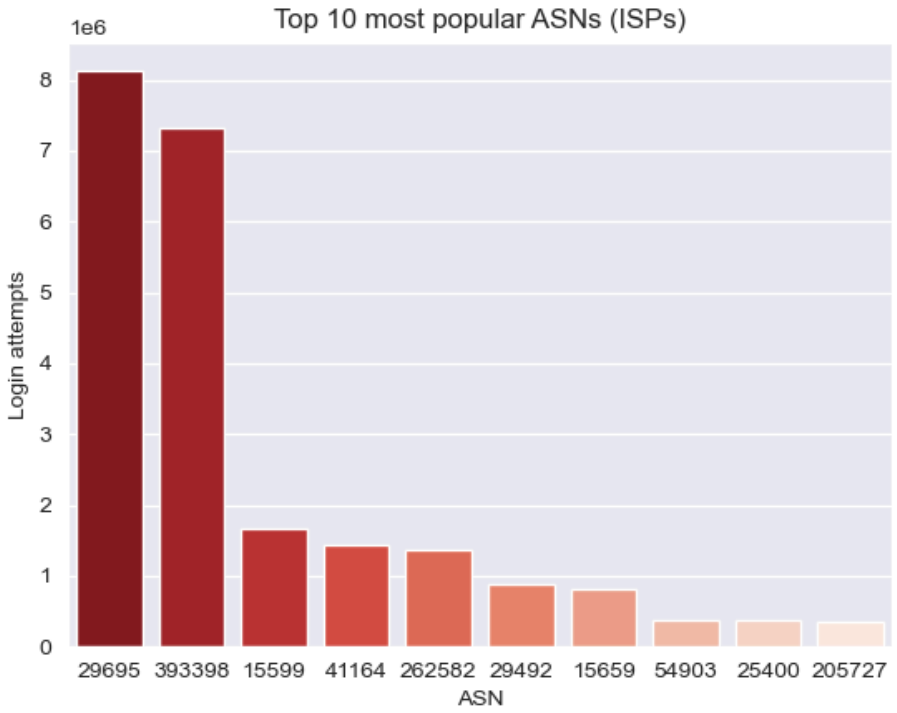


Fig. 9 Top 10 most popular ISPs by associated login attempts

We unveiled that the distribution of login attempts across ASNs is anything but uniform, with "29695" and "393398" characterized by ~8 million and ~7.2 million attempts respectively. Regarding the top two most popular ASNs, we showed that the former is also the most popular in terms of number of associated IPs whereas the latter does not appear in the top ranks. From now on, since according to the dataset's description "ASN" is derived from "IP Address", we will assume that the "ASN" feature corresponds to the ISP/FISP.

3.2 Analyzing the trend of failed/total login attempts in detail

Considering that credential stuffing attacks tend to exhibit a higher than usual failed login rate, we decided to investigate the trend of this rate over time (Fig. 10). In detail, we grouped the dataset by "Login Timestamp" and counted the number of failed login attempts, the number of failed login attempts coming from IP addresses that have been found in known attacker datasets and the number of logins that have been classified as account takeover events. In order to check whether the three plots displayed a similar trend, we opted for a lineplot with three overlapping subplots, specifying different "linewidth", "alpha" and "marker" values for each of the latter.

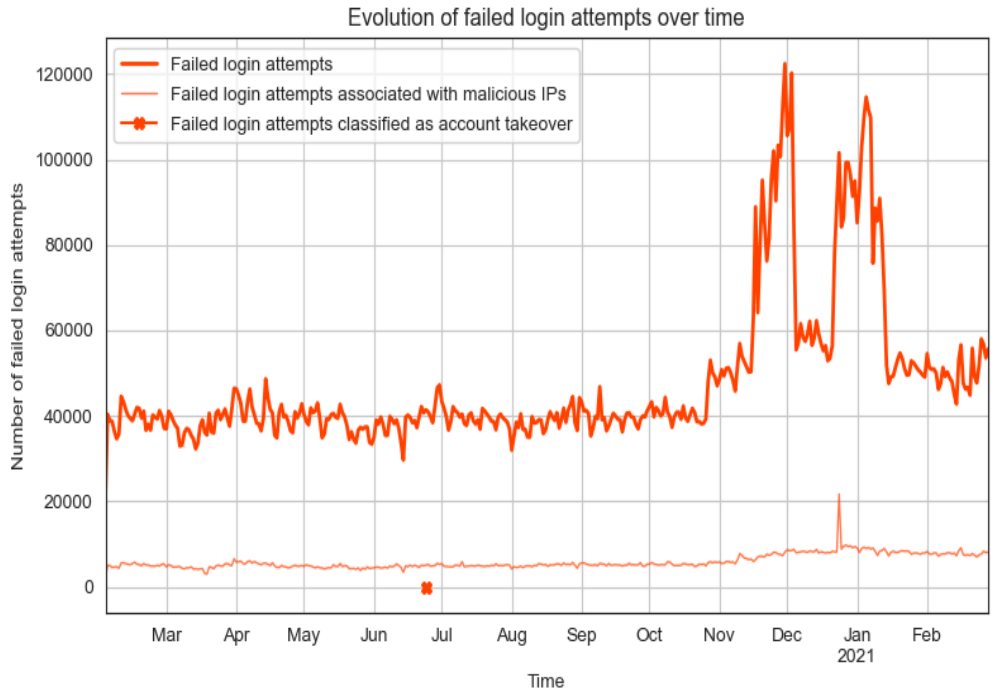


Fig. 10 Daily trend of failed login attempts, failed login attempts associated with IPs that have been found in leaked datasets and failed login attempts classified as account takeover

The line plot for failed login attempts exhibits fluctuations in the period ranging from 15/11/2020 to 15/1/2021 as we have already shown in 3.1 whereas the plot for failed login attempts associated with IPs found in attacker datasets is characterized by a constant trend, except for a spike that can be observed in late December. Since there was only one failed login attempt classified as account takeover and credential stuffing attacks are characterized by changes in site traffic like multiple login attempts from multiple accounts within a limited timeframe as well, we decided to reproduce the same line plots, considering both failed and successful logins to achieve a clearer visualization. (Fig. 11)

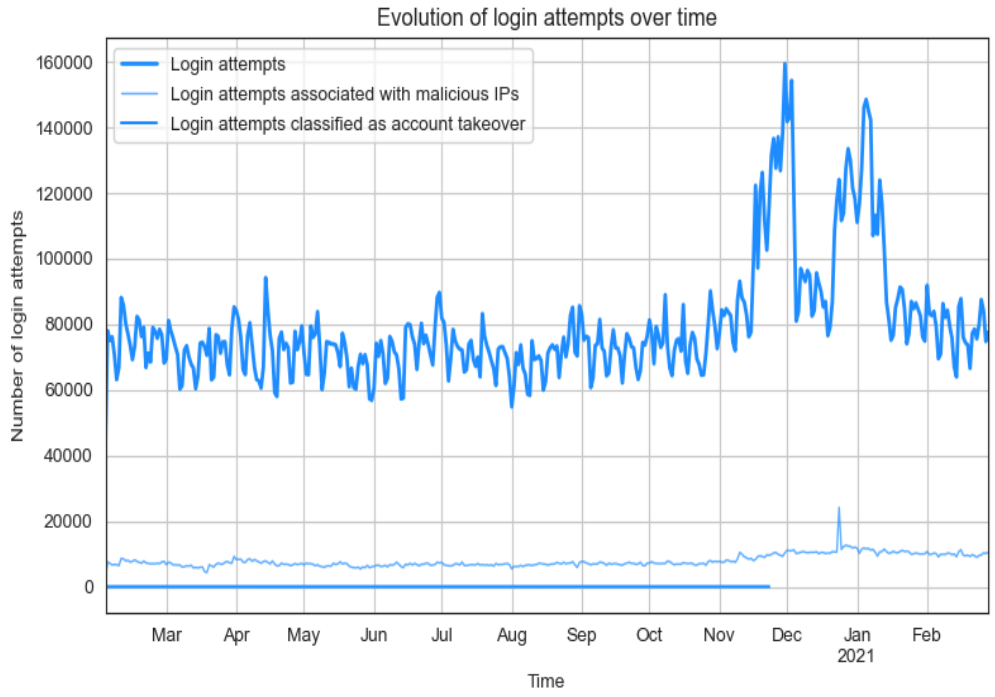


Fig. 11 Daily trend of total login attempts, total login attempts associated with IPs that have been found in leaked datasets and total login attempts classified as account takeover

Concerning login attempts classified as account takeover, Fig. 11 exhibits a constant trend, with no account takeover instances from the end of November 2020 onwards. The last step of this stage was to analyze the login activity of user "-4324475583306591935". (Fig. 12, Fig.13)

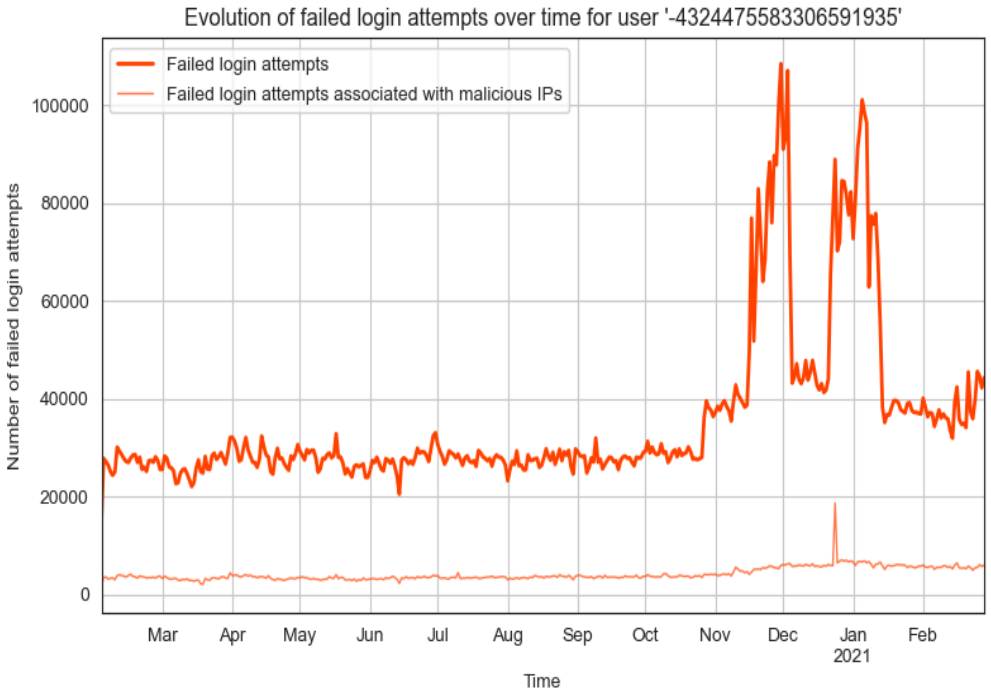


Fig. 12 Daily evolution of failed login attempts over time for the user associated with most login attempts

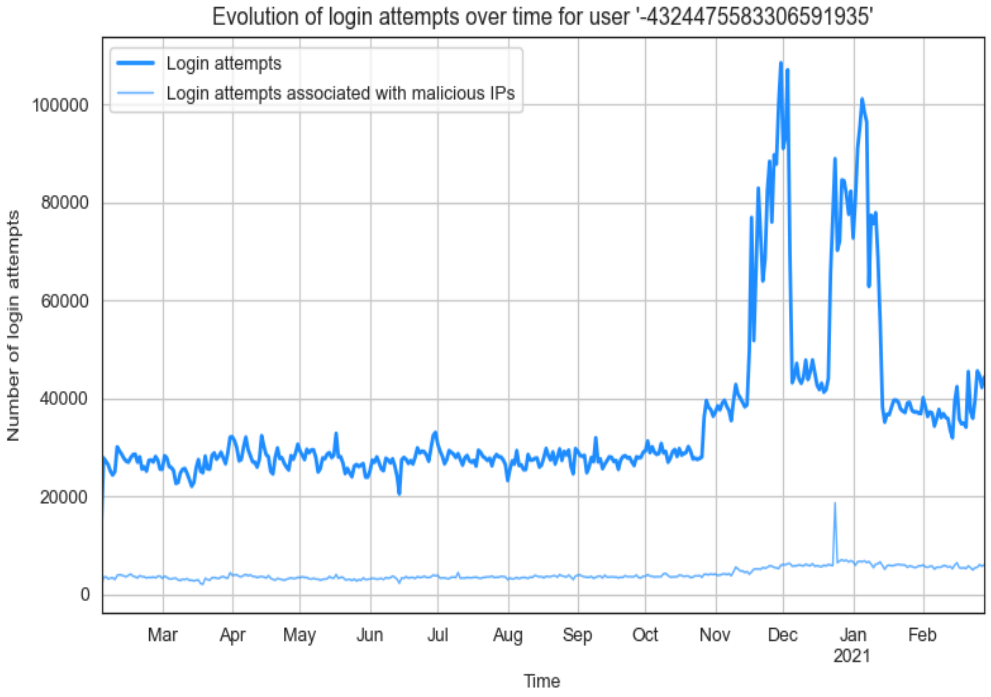


Fig. 13 Daily evolution of total login attempts over time for the user associated with the largest number of attempts

The line plots for this user exhibit a similar trend to those considering all users' login attempts except for login attempts classified as account takeover for which there are no instances.

3.3 Comparison between most popular countries in terms of both total/failed login attempts

When investigating the most popular countries in terms of login attempts/failed login attempts in 3.1, we noticed a geo-anomaly as Norway is the most popular country in the former case whereas the US is in the latter. The main question that this subsection is meant to address is the following: How do logins attempted from Norway and the US differ? Our first idea was to compare them in terms of their distribution over time (Fig. 14). To perform such comparison, we used a line plot, specifying different linewidths and alpha values for the two lines.

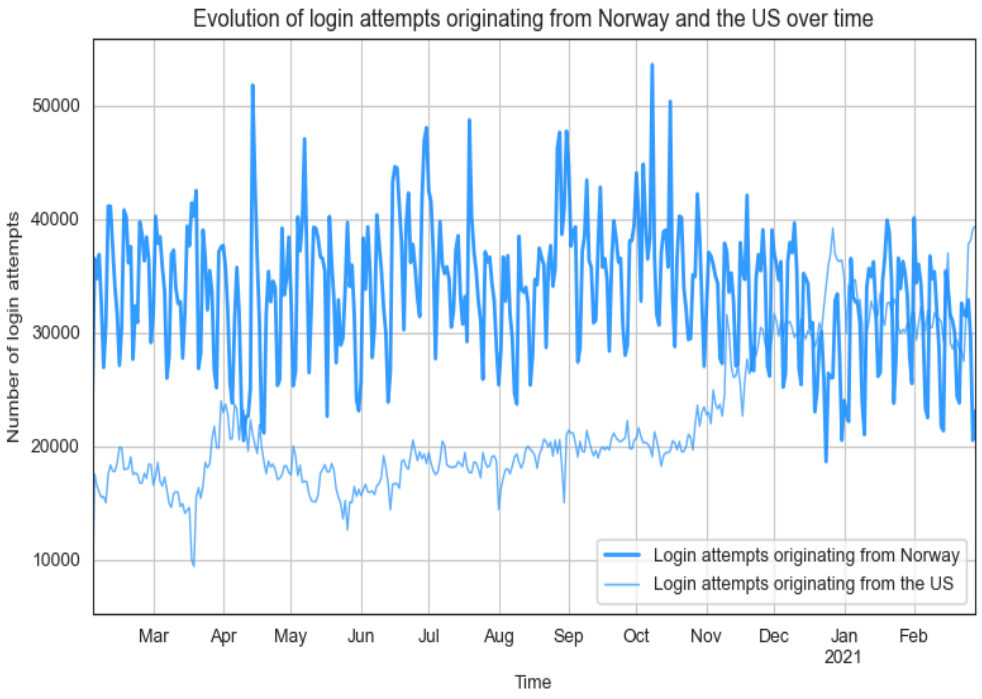


Fig. 14 Daily trend of total login attempts originating from Norway and the US

We can see that, nearly for the entire duration of 2020, the trend corresponding to login attempts originating from the US lies below the line representing Norway, suggesting that the latter was associated with more login attempts. The two lines are overlapping towards the end of 2020 and the beginning of 2021, except for short time periods in which the line representing the US is above the thicker line representing Norway.

Our next hint was to compare the two countries in terms of "Round-Trip Time [ms]" (Fig. 15), with the latter measuring the latency between client's request and server's response. After removing missing values and separating the dataset into Norway's and the US' login attempts, we realized two boxplots for comparison where a boxplot is a simple statistical plot that allows one to visualize the distribution of variables in terms of minimum, interquartile range, maximum and outliers.

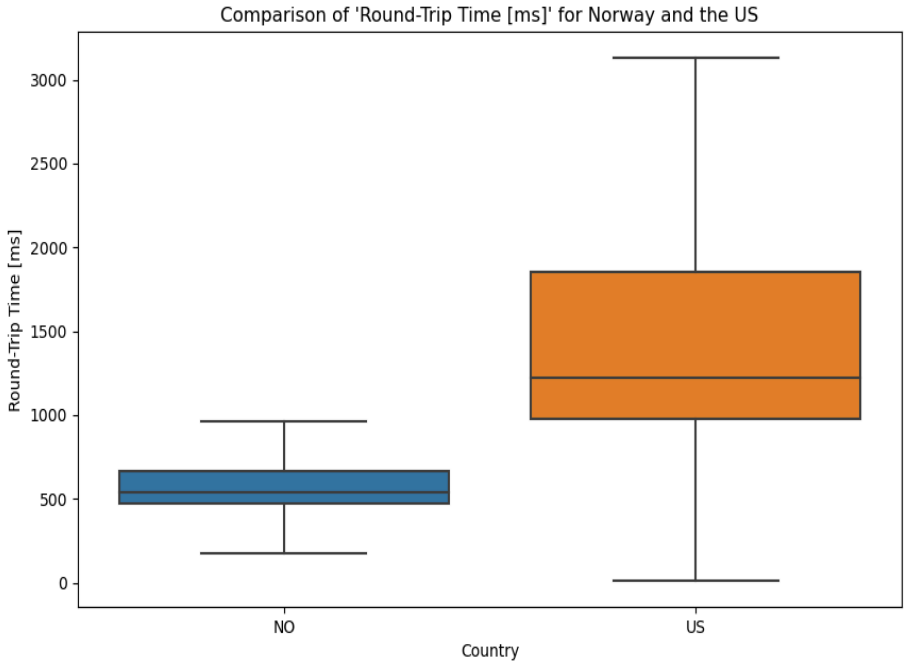


Fig. 15 Boxplot of "Round-Trip Time [ms]" for Norway and the US

The next step was to check whether the difference in mean RTT values for Norway and the US is statistically significant. To do so, we opted for a *t*-test which is used to compare means of two groups, specifying a null hypothesis H_0 (the two group means are equal) and an alternative hypothesis H_1 (difference between group means is non-zero). The choice of whether to reject or not the null hypothesis depends on the *p*-value and the chosen significance level. In our case, we opted for a standard level of statistical significance set to 0.05. As the *p*-value we obtained is zero, we rejected the null hypothesis and concluded that there is a statistically significant difference in mean RTT values for the two countries. The following step was to compare the two countries in terms of most popular user agent strings. (Fig. 16)

```
# check for matches between user agent strings.
matches = norway_top_user_agents.index.isin(us_top_user_agents.index)

# print the matches
print("Matching user agent strings:")
count_matches = 0
for user_agent, is_match in zip(norway_top_user_agents.index, matches):
    if is_match: # if there is a match
        print(f"Norway: {user_agent} | US: {user_agent}") # print the corresponding Norway and US user agent strings
        count_matches += 1 # increase the counter of matches.
    else:
        print(f"Norway: {user_agent} | US: No match")
print(f"Number of matches is the following: {count_matches}")
```

Fig. 16 Code snippet to check for matching user agent strings between Norway and the US

The flow of Fig. 16 is the following: first, we check for matches by making the intersection of the two lists "norway_top_user_agents.index" (which stores the top 10 most popular user agent strings for login attempts associated with Norway) and "us_top_user_agents.index" (which stores the same information for login attempts originating from the US). We initialize a counter variable to zero and use a for loop to check for matches. The code outputs the most popular user agent strings for Norway and if a match is found, it increases the counter by 1 and prints the corresponding user agent string for the US, otherwise it prints the string "US: No match". When the for loop is exited, the total number of matches is returned. (Fig. 17)

```
Norway: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.17.
23.92 Safari/537.36 | US: No match
Norway: Mozilla/5.0  (iPhone; CPU iPhone OS 9_3_3 like Mac OS X) AppleWebKit/533.1 (KHTML, like Gecko Version/4.0 M
obile Safari/533.1 variation/280613 | US: No match
Number of matches is the following: 0
```

Fig. 17 Sample output for "User Agent String" comparison between Norway and the US

To compare the two countries in terms of "Device Type", we used a countplot (Fig. 18) which shows device types on the x-axis and corresponding bars whose height represents the number of login attempts associated with each device type.

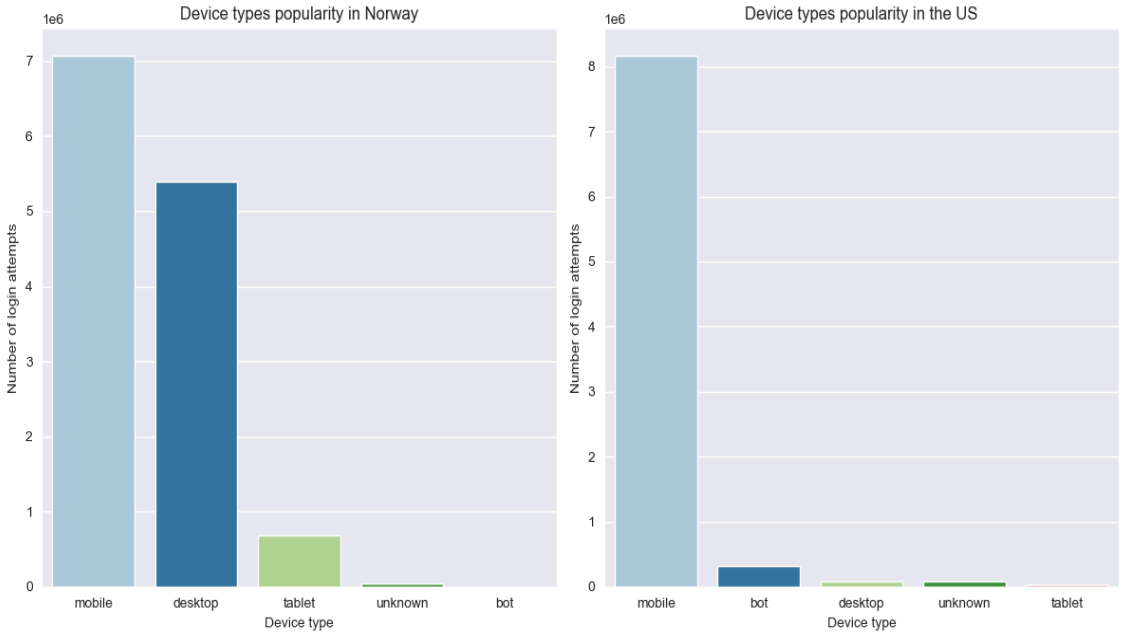


Fig. 18 Comparison of device types' popularity between Norway and the US

Although Fig. 18 agrees on the most commonly used device type (mobile), we can notice an interesting difference. In Norway, the second and third most popular device types are desktop and tablet whereas in the US the second most popular device type is bot (automated login attempts) and there are few login attempts that are carried out from a desktop/tablet device type.

A countplot was employed as well to relate the two countries based on popular IP addresses (Fig. 19). For Norway, we observed a rather uniform distribution of login attempts across most popular IP addresses, with 7 out of 10 of them (all those starting with 10.0.77) belonging to the same subnet and 8 of them (all those starting with 10.0) belonging to the same network. Regarding the US, login attempts tend to follow a power-law distribution, with few IP addresses associated with most login attempts and a large number of IPs connected to few login attempts. Looking at their common structure, we noticed that 6 out of the 10 most popular IP addresses belong to three different subnets (one for those starting with 170.39.78, one for those beginning with 170.39.76 and one for those starting with 199.26.84) and 7 out of 10 belong to two different networks (one for those starting with 170.39 and the other for those beginning with 23.137).

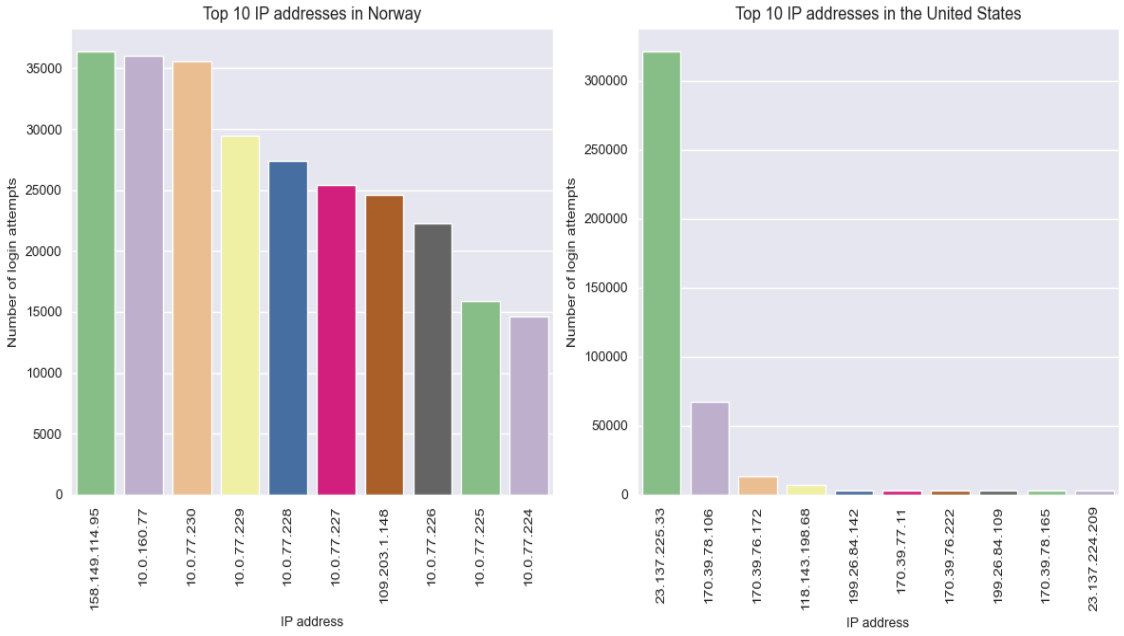


Fig. 19 Comparison of most popular IP addresses for Norway and the US

In terms of "ASN" (ISP), we compared Norway and the US by computing the proportion of login attempts for each of the top 10 most popular ISPs (Fig.20), setting ASNs as x-axis, proportion of login attempts as y-axis and "Reds_r" as color palette.

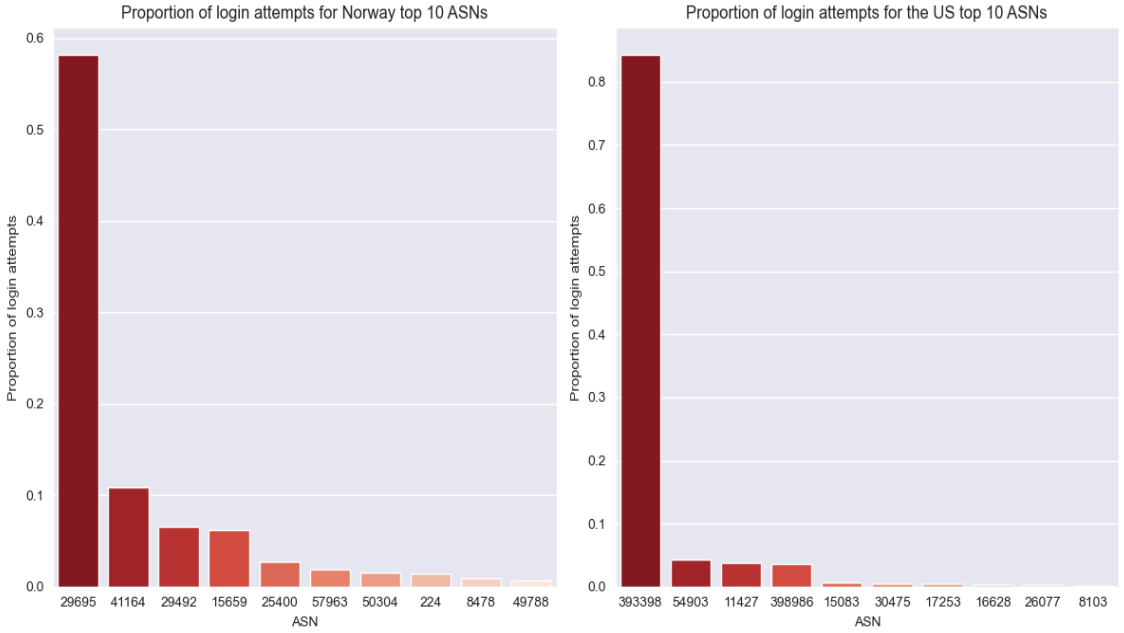


Fig. 20 "ASN" comparison between Norway and the US

ASNs were arranged in decreasing order of y-axis values. We can see that ~60% of Norway's login attempts are associated with ASN "29695" whereas ~85% of login attempts coming from the US are related to ASN "393398".

The last step of the country-level comparison consisted of inspecting login attempts in terms of "Login Successful", "Is Attack IP" and "Is Account Takeover". The main takeaways of the first analysis were that logins attempted from Norway tend to be successful ~⅔ times whereas for the US login attempts tend to fail ~75% of the times. Concerning the second analysis, Fig. 21 allowed us to assess that almost all login attempts originating from Norway are associated with benign IPs while in the US ~25% of login attempts are associated with malicious IP addresses.

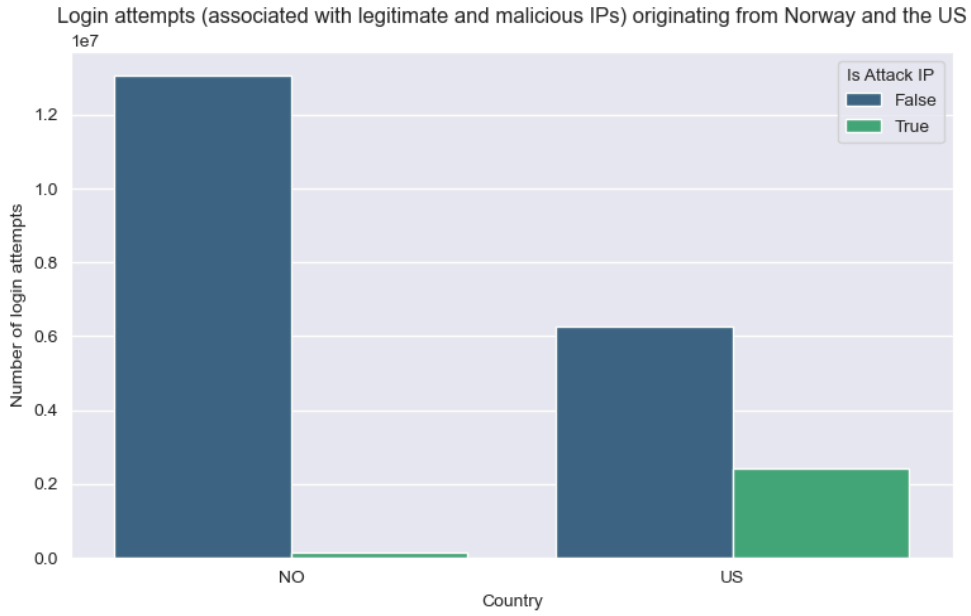


Fig. 21 Comparison of login attempts originating from Norway and the US by "Is Attack IP"

Finally, we established that there are only 10 login attempts coming from Norway that have been classified as account takeover whereas there are no account takeover instances associated with the US.

3.4 Inspecting the "Round-Trip Time [ms]" feature in more detail

When analyzing the differences between the most popular countries in terms of login attempts/failed login attempts in 3.3, we proved a statistically significant association with "Round-Trip Time [ms]". This feature, as we highlighted in 3.1, provides valuable information and is hard to spoof as the attacker would have to have access

to a device situated near the victim's location. It follows that it is of utmost importance as it could be used to verify geolocation information provided by "IP address". However, we also pointed out that it contains ~96% of missing values and imputing it using traditional strategies such as mean or median wouldn't make much sense considering the large share of missing values and that the feature's value should reflect the physical distance between client and server. Given these considerations, we decided to investigate its relationship with other variables (particularly our target) before deciding on whether to keep it or discard it. The first step of this analysis was to study the relationship with "Country". As the latter has 229 unique values, we extracted the top 15 countries in terms of login attempts and made a boxplot of "Round-Trip Time [ms]" for each of them. (Fig. 22)

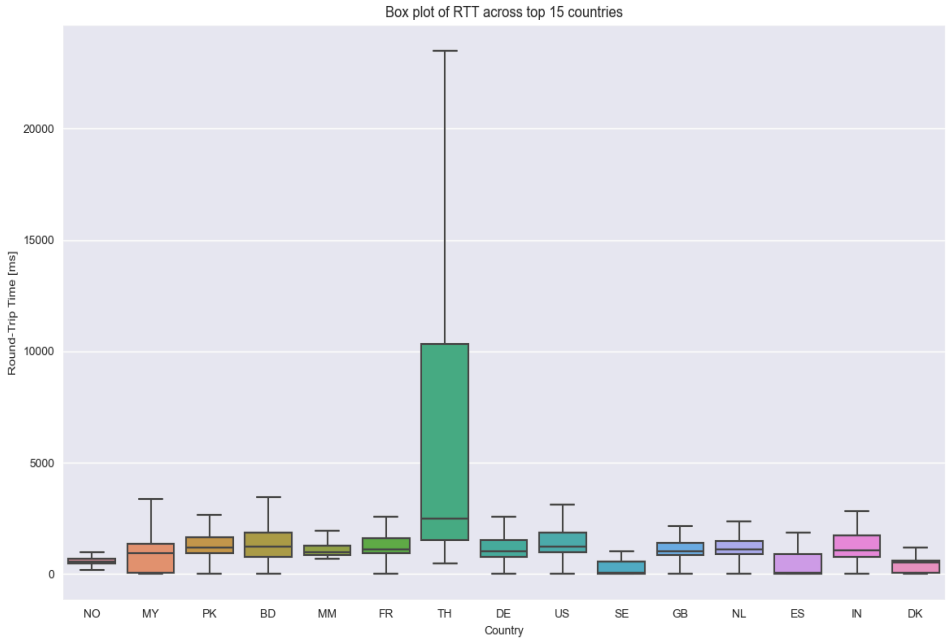


Fig. 22 Boxplot of "Round-Trip Time [ms]" for the top 15 most popular countries based on total login attempts

To determine whether the two features are associated, we imported "f_oneway" from "scipy.stats" to perform an analysis of variance (ANOVA). The latter is a statistical test that allows to investigate the variance across multiple group means, whose results shall be interpreted in a similar way as we did for the t-test. We performed the test on all countries and obtained a p-value of 0. Based on the latter result, we rejected the null hypothesis and concluded that there is a statistically significant difference in mean "Round-Trip Time [ms]" values across groups. However, the ANOVA test tells us whether results are significant overall but it does not tell us where

differences lie. This is why it is usually a good idea to run a post-hoc test. To do so, we opted for the Tukey's test that enables us to identify which specific groups' means are different. (Fig. 23, Fig. 24)

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
AE	AF	-295.6	1.0	-5006.9178	4415.7178	False
AE	AT	-282.7429	1.0	-2801.0481	2235.5624	False
AE	AU	209.5667	1.0	-2394.7122	2813.8456	False
AE	BD	706.657	1.0	-1217.9822	2631.2963	False
AE	BE	-598.513	1.0	-2720.6914	1523.6653	False

Fig. 23 Tukey's HSD test sample results

Fig. 23 should be read in the following way: for the country whose abbreviation is "AE" (United Arab Emirates), there is not a statistically significant difference in mean "Round-Trip Time [ms]" values with respect to countries labeled as "AF" (Afghanistan), "AT" (Austria), "AU" (Australia), "BD" (Bangladesh), "BE" (Belgium) and vice versa (the test is symmetric). The chosen significance level is 0.05 and since the *p*-value is 1.0, there is not enough evidence to reject the null hypothesis H_0 (there is no difference across groups' means).

NO	TW	1058.9148	1.0	-3241.912	5359.7415	False
NO	UA	436.4148	1.0	-1714.0012	2586.8307	False
NO	US	1712.4924	0.0	1587.5485	1837.4362	True
NO	VN	1913.9148	1.0	-2386.912	6214.7415	False
NO	ZA	2754.9148	0.0	1129.3512	4380.4784	True

Fig. 24 Tukey's HSD test sample results for Norway

Fig. 24 suggests that there is enough evidence to conclude that there is a statistically significant difference in mean "Round-Trip Time [ms]" values across Norway ("NO") and the US/South Africa ("ZA") and vice versa. The post-hoc test confirms the results of the *t*-test that we performed between Norway and the US, providing additional evidence of a significant difference in mean "Round-Trip Time [ms]" for the two countries. In order to check the feature's relationship with "Is Attack IP" and "Is Account Takeover" we resorted to the *t*-test, obtaining evidence of a significant association in both cases. Our final decision as to whether to keep or discard "Round-Trip Time [ms]" was based on the variable's share of missing values for true instances of "Is Account Takeover". Since 139 out of 140 true instances of the target exhibit missing values for "Round-Trip Time [ms]", we decided to discard the latter since it does not allow us to discern whether the login attempt is benign or a credential stuffing attack.

3.5 Geo-distribution of account takeover instances

After comparing the most popular countries in terms of login attempts/failed login attempts, we decided to investigate the geo-distribution of account takeover instances. We filtered the original dataset to include only login attempts with "Is Account Takeover" = True and acknowledged that account takeover instances are distributed across 14 countries. In order to plot the map, we first created a dictionary to store the mapping between country abbreviations and full country names as "Country" stores the short version of each country's name. We then replaced country abbreviations with full country names and computed the number of account takeover instances by country. The first step of building the map consisted of importing "geopandas", a library that allows to work with geospatial data in an intuitive way. We loaded the dataframe containing geospatial data representing world countries ("naturalearth_lowres") using "read_file" from "geopandas" and merged it with the dataframe storing information about countries associated with true instances of "Is Account Takeover".

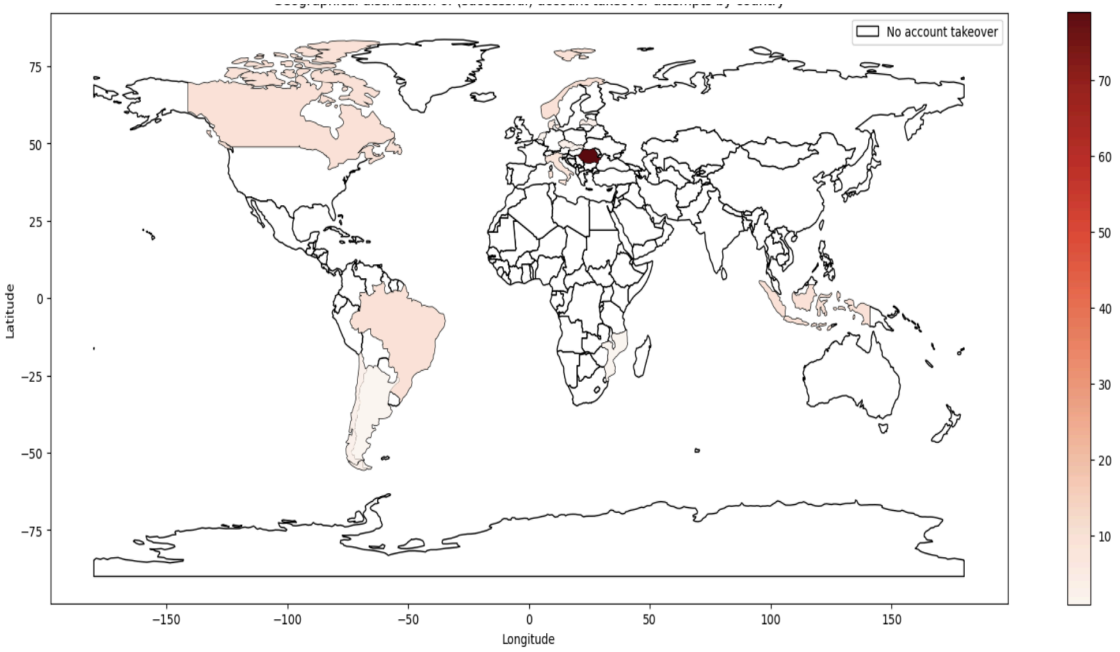


Fig. 25 World map for the geographic distribution of true account takeover instances

Fig. 25 is plotted with black borders and a white background to enable a clearer visualization and a white patch is added to the plot's legend to represent countries with no successful account takeover attempts. Fig. 25 clearly shows that account takeover instances tend to be concentrated in the following geographic areas: United States, South America, Southern Africa, Southeast Asia, Northern and Central-Eastern Europe.

3.6 Relationship between the target and other predictors

Our next step towards building a predictive model for "Is Account Takeover" was to plot the distribution of independent variables for true and false values of the target to check whether true instances have characterizing values for some of the predictors. To investigate the relationship between "Is Account Takeover" and "Device Type" visually, given the target's large class imbalance, we chose to represent two separate countplots (Fig. 26) to ensure a clearer visualization. Fig. 26 shows that neither bot nor unknown device types are used to carry out login attempts classified as account takeover. Furthermore, the most commonly used device type for true instances of "Is Account Takeover" is desktop which accounts for ~86% of login attempts followed by mobile and tablet whereas mobile is the most widely used device type to carry out benign login attempts, followed by desktop and bot.

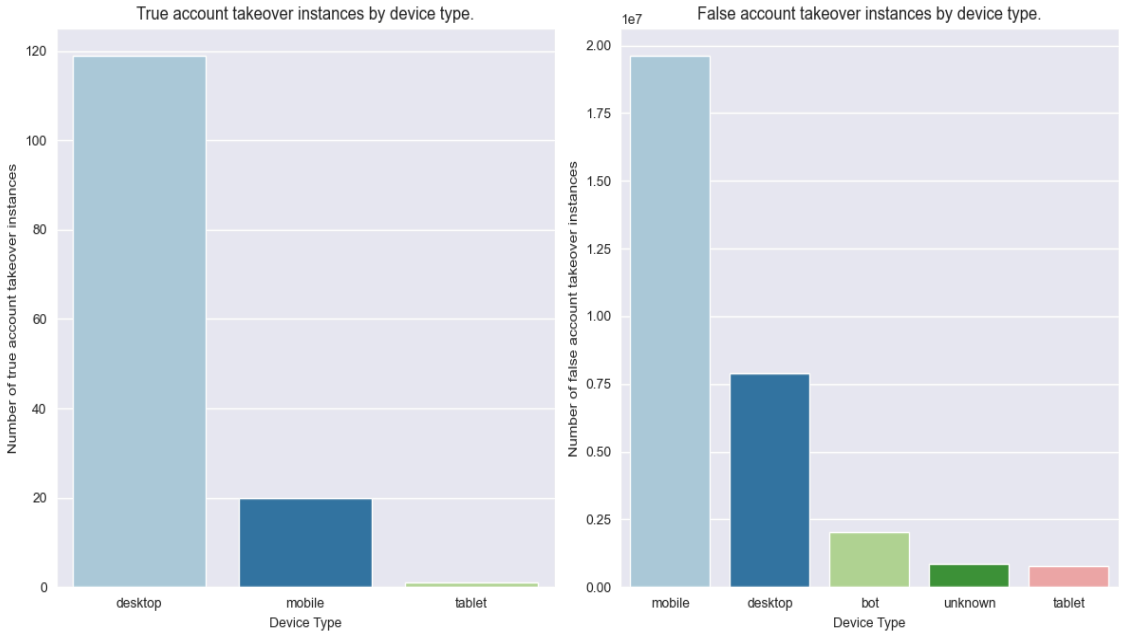


Fig. 26 "Device Type" comparison between true and false account takeover instances

In order to check the statistical magnitude of the association between the two variables, we opted for a chi2 test, which verifies whether the frequency distribution of categorical variables differs significantly from the expected distribution under the hypothesis of independence. To perform the test, we resorted to "chi2_contingency" from "scipy.stats". (Fig. 27)

```
chi2 test statistic:  264.77575960970137
chi2 p-value for device type:  4.263739858185625e-56
```

Fig. 27 Chi2 test results for "Device Type" and "Is Account Takeover"

As the *p*-value is almost zero, we concluded that there is strong evidence of a significant association between "Device Type" and the target.

In order to explore the relationship between "Is Account Takeover" and "Login Timestamp", we first created a new column called "Time Of Day" which takes on value "Morning" if the hour contained in the login timestamp is between 6 and 12 (excluded), "Afternoon" if it is between 12 and 18 (excluded), "Evening" if the hour of the day is in between 18 and 24 (excluded) and "Night" for the remaining hours. The subsequent step was to plot the distribution of true and false instances of "Is Account Takeover" during the day to verify whether there were any time slots in which they concentrated. (Fig. 28)

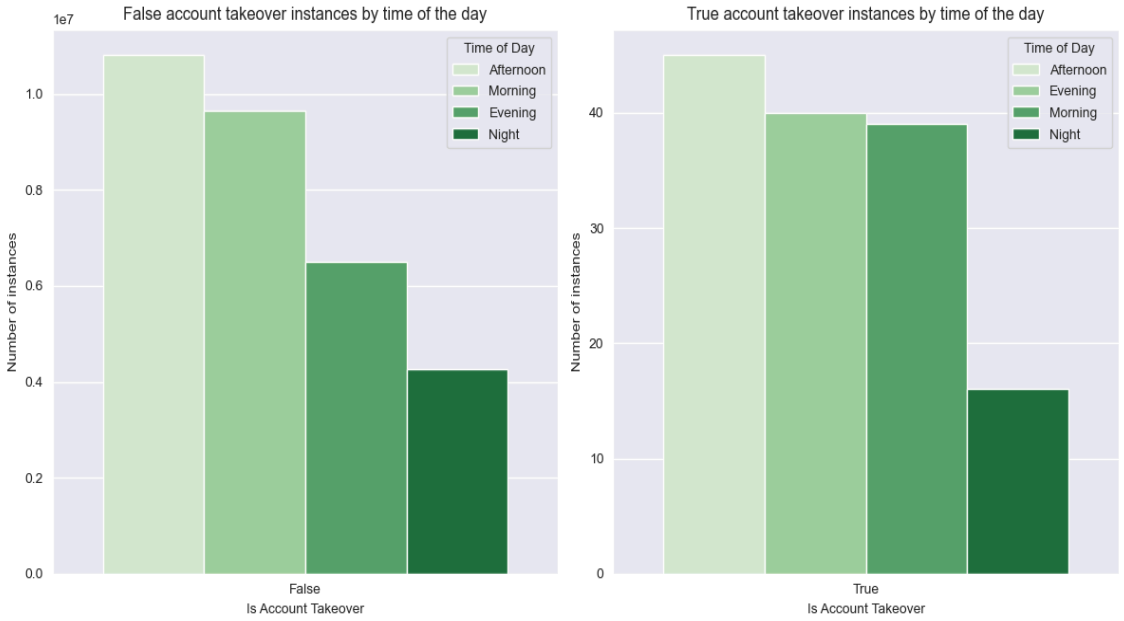


Fig. 28 Distribution of true and false account takeover instances by "Time of Day"

According to Fig. 28, it is clear that in both cases the time of the day in which most login attempts are concentrated is "Afternoon" whereas "Night" is the time slot in which the least amount of login attempts are reasonably carried out. Looking at the distribution of false account takeover instances, we can notice that there is quite a large difference between logins attempted in "Evening" and "Morning" whereas the difference is much less obvious for true instances. Once again, we resorted to "chi2_contingency" to quantify the association between the two variables but unlike "Device Type", there was not enough evidence to reject the null hypothesis and we thus had to conclude that there is no association between the time of the day in which the login attempt is carried out and its classification as an account takeover. The next step in studying the association between the target and "Login Timestamp" was to extract from the latter the following self-explanatory variables: "Day", "Month", "Year". To investigate the relationship with each of the latter, we performed three different chi2 tests

(each between the target and one of the time-related independent variables) which provided evidence of statistical significance for the association with "Month" and "Year" unlike "Day".

Our next intention was to check whether there was any difference between most popular countries associated with true and false instances of "Is Account Takeover". (Fig. 29)

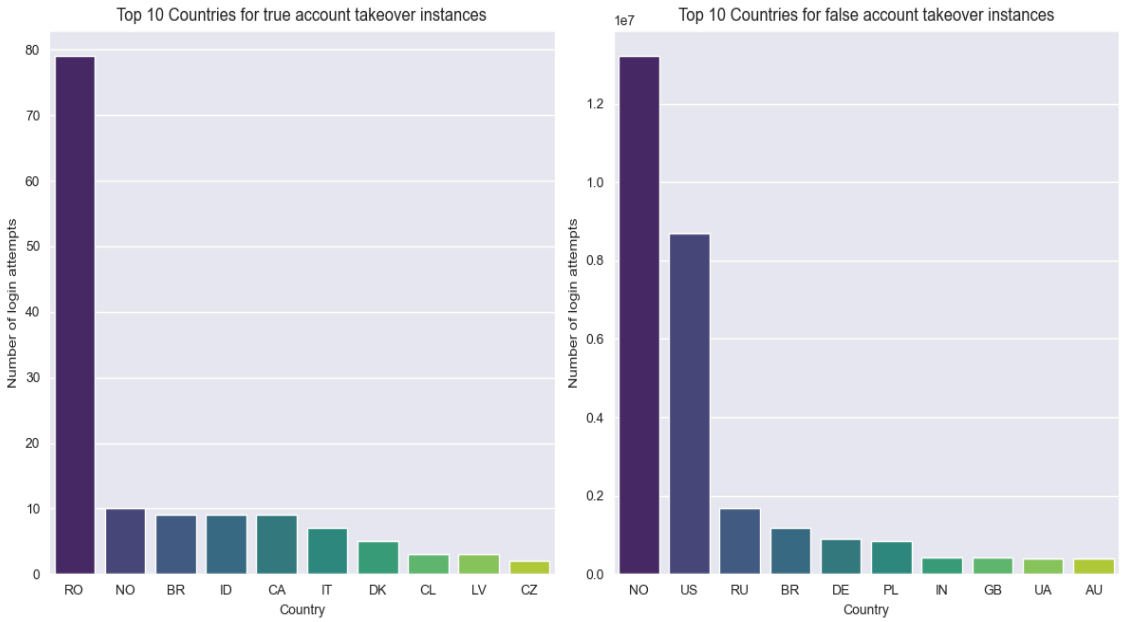


Fig. 29 "Country" comparison between true and false instances of "Is Account Takeover"

Looking at the top 10 countries for true and false instances of "Is Account Takeover", we can see that, in the latter case, Norway, United States and Russia ("RU") are the most popular countries in terms of login attempts. Interestingly, Fig. 29 shows that most login attempts classified as account takeover originate from Romania ("RO"), with Indonesia ("ID), Canada ("CA"), and Italy ("IT") also being quite popular, despite not appearing in the top 10 countries for legitimate login attempts. We then created a contingency table between "Country" and "Is Account Takeover" and performed a chi2 test to assess the relationship between the two variables. The close to zero p-value allowed us to conclude that there is a statistically significant association between them. To simplify the complexity of our analysis, with a view to subsequent steps, we used "value_counts" from "pandas" to visualize the distribution of login attempts across countries to determine whether the cardinality of "Country" could be reduced. After acknowledging that the top 10 most frequent countries account for ~90% of all login attempts, to reduce the number of unique values for "Country", we selected the top 10 most popular countries, created a mask to store them and used the latter to convert less frequent categories to "Other". This will turn out to be useful to simplify the variable's encoding procedure in the pre-processing step prior to the model building phase.

The subsequent step in studying the relationship between the target and predictors was to plot the most popular IP addresses (Fig. 30) and ASNs (Fig. 31) to check whether there was any noticeable difference in their patterns. Looking at the right side of Fig. 30, 7 out of the 10 most popular IP addresses (all those starting with 10.3.205) belong to the same subnet as they have the first three octets in common. For what concerns true account takeover instances, 4 out of the 10 most commonly used IPs (all those beginning with 91.240.236) are part of the same subnet. For Fig. 30 and Fig. 31, we specified the number of login attempts on the y-axis and used the "Accent" and "Reds_r" palettes respectively to ensure consistency of the color encoding scheme.

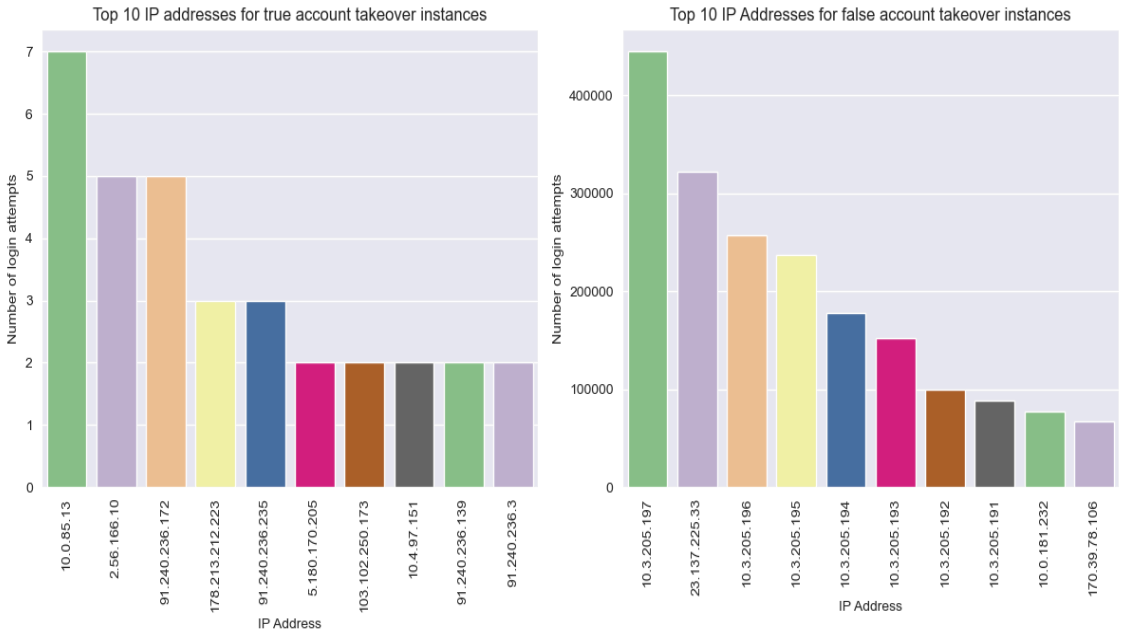


Fig. 30 Top 10 most popular IP addresses for true and false account takeover instances

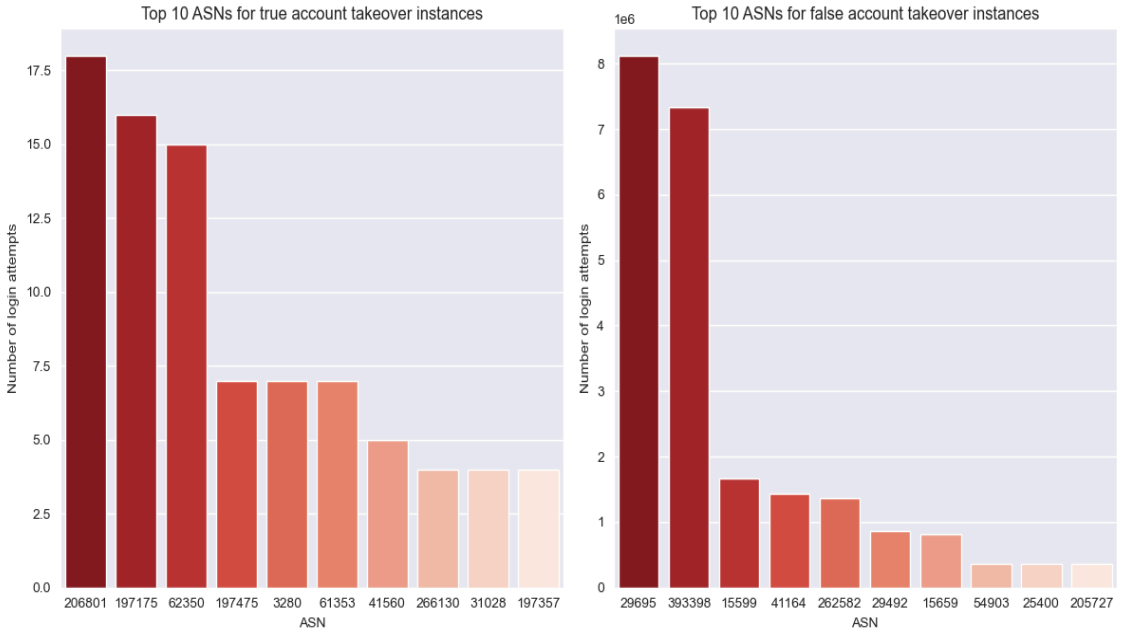


Fig. 31 "ASN" comparison between true and false instances of "Is Account Takeover"

To assess the relationship between the target and "ASN", we followed a similar approach as for "Country", creating a contingency table between the variable and "Is Account Takeover" before applying "chi2_contingency". Once again, it turned out that there is a significant association between the two variables. Given the feature's large number of unique values, we used "value_counts" to acknowledge that the top 15 ASNs account for ~75% of login attempts. We created a mask to store indices of the most popular ASNs and used it to group less frequent categories into "Other".

In the context of having distinguishing values for true instances of the target, user agent strings may provide especially valuable information in relation to credential stuffing attacks. In particular, they can be used for device fingerprinting. The user agent header includes information such as operating system, browser and language that can be leveraged to fingerprint a device. This fingerprint can then be matched against any browser attempting to log in into the user's account and if it doesn't match, the user may be prompted for additional authentication. A fingerprint different from the commonly used device's may provide evidence that a credential stuffing attack is taking place. Given these considerations, it is of utmost importance to investigate the relationship between "Is Account Takeover" and "User Agent String" to check whether there is any interesting pattern especially for true instances of the target. Plotting the distribution of most common user agent strings allowed us to establish that 85/140 login attempts classified as account takeover are associated with the same user agent string and that there are no common strings between true and false instances. This suggests that the former may be associated with distinguishing values of "User Agent String".

After this investigation, the remaining independent variables whose association with the target still needed to be tested were “Login Successful” (Fig. 32) and “Is Attack IP” (Fig. 33).

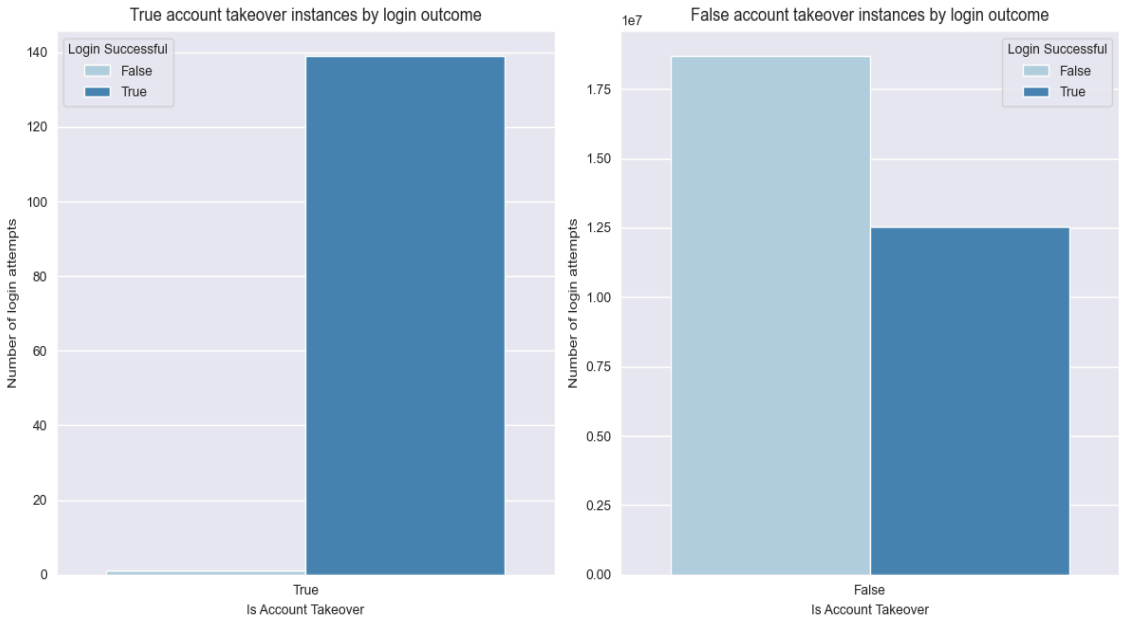


Fig. 32 Comparison of login outcome between true and false account takeover instances

From Fig. 32, we can see that 139/140 login attempts classified as account takeover are associated with a successful login outcome whereas the distribution for false instances of account takeover is ~60% (failed login outcome) vs ~40% (successful outcome).

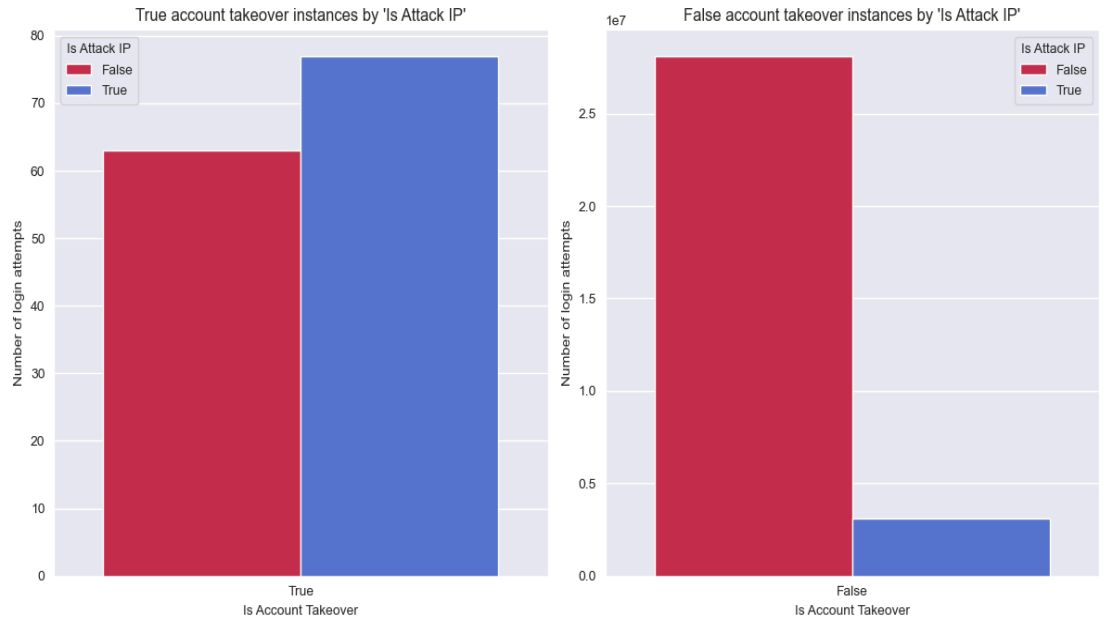


Fig. 33 "Is Attack IP" comparison between true and false account takeover instances

As we may have expected, the majority of login attempts classified as account takeover have been carried out using IP addresses that have been previously found in attacker datasets whereas the opposite holds for legitimate login attempts.

For both "Login Successful" and "Is Attack IP", a chi2 test provided strong evidence to reject the null hypothesis H_0 (meaning that the observed frequency was different from the expected one) and to conclude that they are both associated with "Is Account Takeover". To wrap up, the main takeaways of this subsection, with respect to our goal of building a predictive model for the target, are the following:

- Concerning the association between "Device Type" and "Is Account Takeover", we concluded that there is strong evidence of a significant relationship between the two variables.
- Regarding the association between "Time of Day"and "Is Account Takeover", we proved that there is no association between the time of the day in which the login attempt is carried out and its classification as account takeover.
- For what concerns the association between the target and the three time-related variables extracted from "Login Timestamp" ("Day", "Month", "Year"), we showed that there is no evidence of statistical significance for the association with "Day" unlike that with "Month" and "Year".
- Concerning the relationship between "Round-Trip Time [ms]" and "Is Account Takeover", there are 139 out of 140 missing values for true account takeover instances. We will not consider this variable in the following steps of our analysis since it does not allow us to distinguish between true and false instances of account takeover.
- When studying the association between "Country" and "Is Account Takeover", we saw that there are some peculiar countries (ex. Romania) that are associated with the majority of true account takeover instances. We performed a chi2 test which showed a strong level of statistical significance and converted less frequent categories to "Other".
- For the association between "ASN" and "Is Account Takeover", we followed an analogous approach as for "Country" which led to similar conclusions.
- Looking at the most popular user agent strings for both true and false account takeover instances, we noticed that there are no matches and 85/140 true instances are associated with the same user agent string.
- Regarding the association between "Login Successful" and "Is Account Takeover", we proved that there is strong evidence of statistical significance.
- Finally, concerning the relationship between "Is Attack IP" and "Is Account Takeover", as we expected, we showed that there is strong evidence of a statistically significant association.

3.7 Is there any hidden information behind "User ID" that could have an impact on the target?

In this subsection, we decided to investigate "User ID" in more detail as there might be some hidden information behind it that could have an association with the target. After proving that 74/140 (~50%) of true account takeover instances and 2 million/31 million (~6.5%) of false instances are associated with an ID that starts with a minus sign, we verified that ~40% of users (1.7 million) are associated with a single login attempt and, considering login attempts classified as account takeover, 134/137 User IDs are associated with a single login instance. Of these 137 IDs, 123 are shared by both true and false account takeover instances whereas 14 of them are associated only with true instances of the target. Inspecting the latter, we could notice that 4/14 IDs are associated with IP addresses that belong to the same subnet (they all start with 5.180.170) and 2/14 with IPs that are part of the same network (as they both begin with 10.4). Moreover, 13/14 IDs are associated with the level "Other" for both "Country" and "ASN", suggesting that they are related to less frequent categories of both variables. All login attempts carried out by users whose IDs are associated only with true instances of account takeover are associated with either a desktop or mobile device type and 10/14 share the same user agent string, namely the one starting with "Mozilla/Macintosh/Intel/Mac OS X/10_14_6".

Looking at the features in our dataset, we supposed that the only variable that could have had an association with "User ID" was "Login Timestamp". We made the hypothesis that the ID could have been assigned according to the account's creation date, with smaller values associated with older accounts whose information could be more likely to have ended up in datasets of leaked authentication credentials. To test this hypothesis, we first extracted a column called "User Account Creation Date", assuming that the user's account creation date corresponds to the timestamp associated with the user's first login attempt. We grouped by "User ID" and for each ID we applied the "min" function to "Login Timestamp". (Fig. 34)

User ID	Round-Trip Time [ms]	IP Address	Country	Region	City	ASN	User Agent String	Browser Name and Version	OS Name and Version	User Account Creation Date
-4324475583306591935	NaN	10.0.65.171	NO	-	-	29695	Mozilla/5.0 (iPhone; CPU iPhone OS 13_4 like ...	Firefox 20.0.0.1618	iOS 13.4	2020-02-03 12:43:30.772
-4324475583306591935	NaN	194.87.207.6	AU	-	-	60117	Mozilla/5.0 (Linux; Android 4.1; Galaxy Nexus...	Chrome Mobile 46.0.2490	Android 4.1	2020-02-03 12:43:30.772

Fig. 34 Dataset after adding "User Account Creation Date"

Fig. 34 shows two login attempts by the user whose ID is "-4324475583306591935". We can see that "User Account Creation Date" takes on value "2020-02-03 12:43:30", meaning that the user created his/her account on the 3rd February 2020 at 12:43:30. The account's creation date column was then converted into seconds elapsed since 1st January 1970. The next step in the analysis of "User ID" was to check the correlation between the latter and "User Account Creation Date". This information may turn out to be useful in predicting the target as older accounts may be more likely to be the target of credential stuffing attacks as it is more likely that the user's credentials have been exposed in data leaks. Since the "User ID" column contains both positive and negative values, we decided to compute two different correlations, namely one between positive values and time since epoch and another between negative values and seconds elapsed since 1st January 1970. We first made two separate hexbin plots (Fig. 35, Fig. 36) to verify whether we could notice any pattern/trend graphically. A hexbin plot divides the space into hexagonal bins and counts the number of points that fall within each bin. It mitigates the issue of overlapping points as each hexagon represents the density of points within that region, providing a clear indication of where the majority of data points lie.

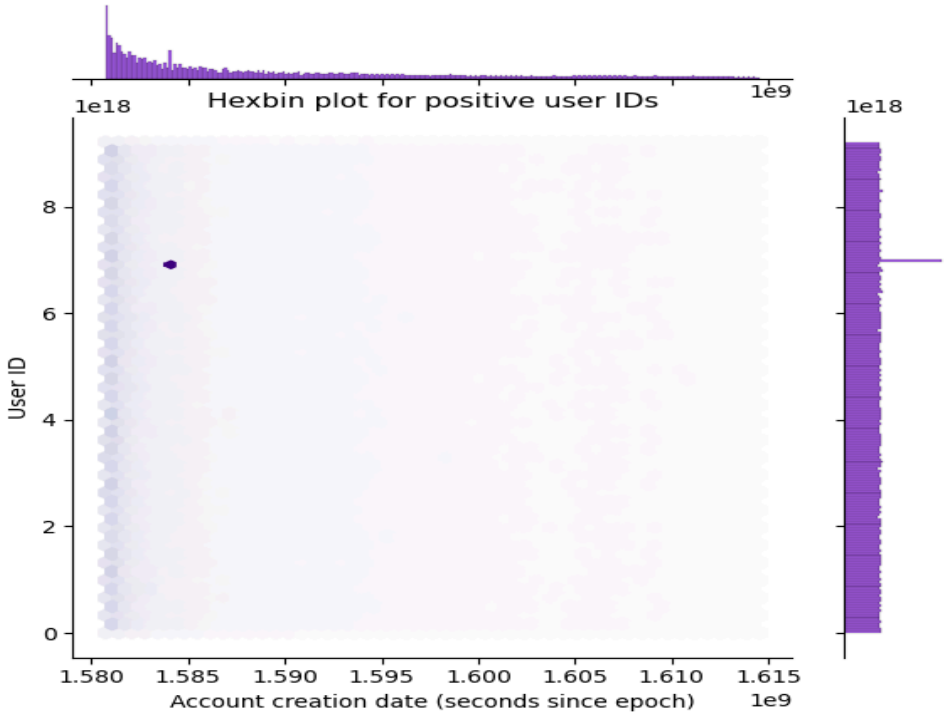


Fig. 35 Hexbin plot between "User ID" (>0) and "User Account Creation Date"

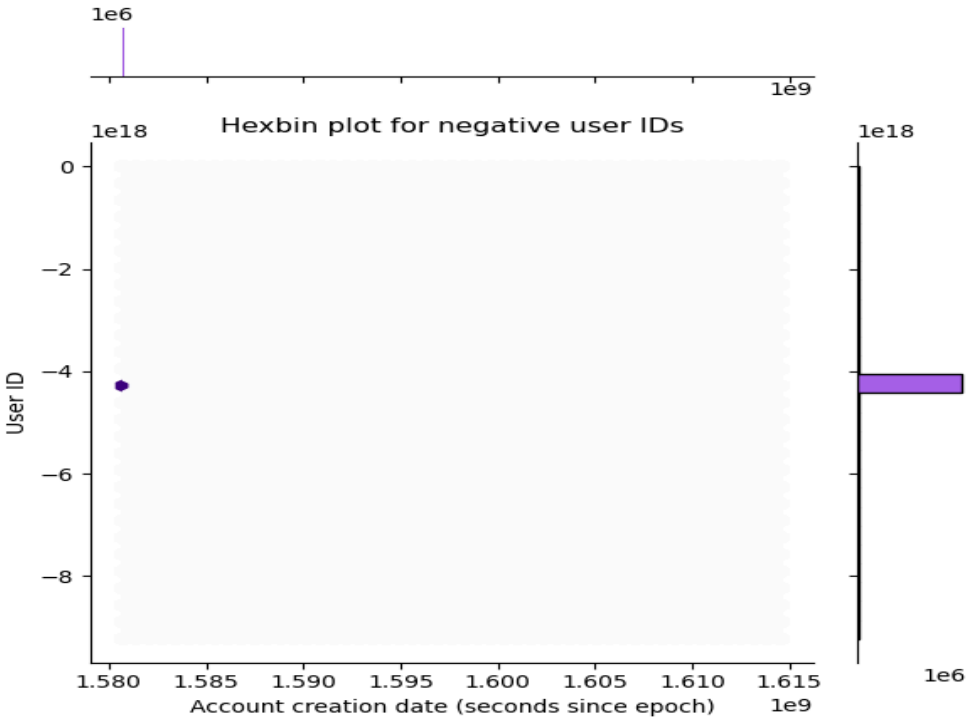


Fig. 36 Hexbin plot between "User ID" (<0) and "User Account Creation Date"

Hexagons represent bins and the darkness of each hexagon represents the density of points within the bin. The histograms on the top and right sides show the distribution of data along the x-axis ("User Account Creation Date") and y-axis ("User ID"), providing additional information on the marginal distributions of both variables. From Fig. 35 and Fig. 36, it is clear that there is no relationship between "User Account Creation Date" and "User ID". Furthermore, there is not any discernible trend or pattern in the distribution of "User ID" relative to "User Account Creation Date". As commonly employed Pearson's correlation assumes that the relationship between continuous variables is linear and that both variables are normally distributed, to verify the second assumption, we decided to plot the distribution of "User ID" (Fig. 37) and "User Account Creation Date" (Fig. 38).

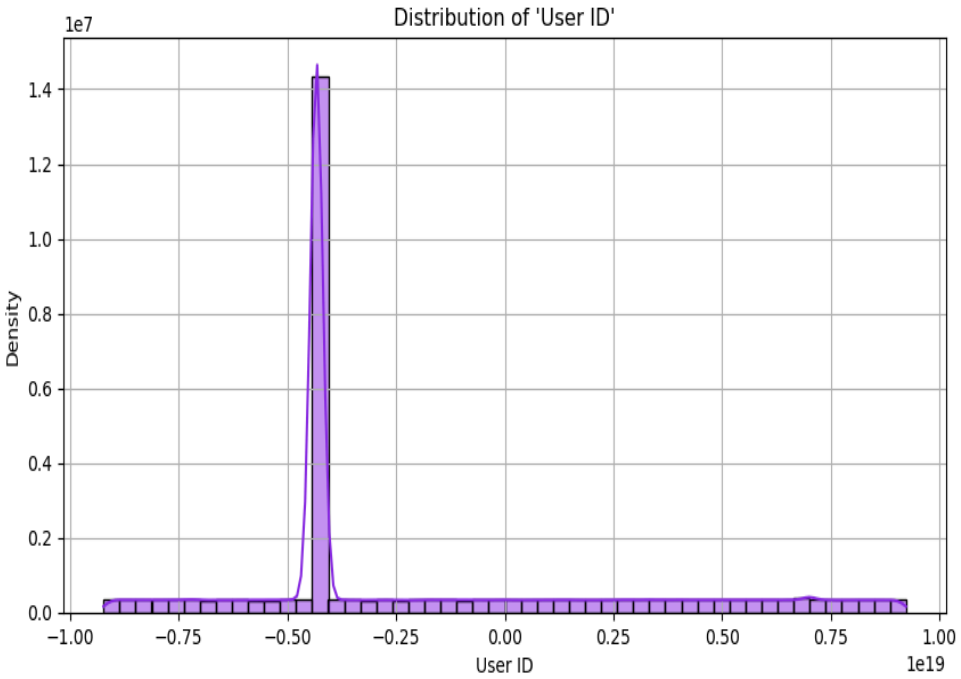


Fig. 37 Histogram showing the distribution of "User ID"

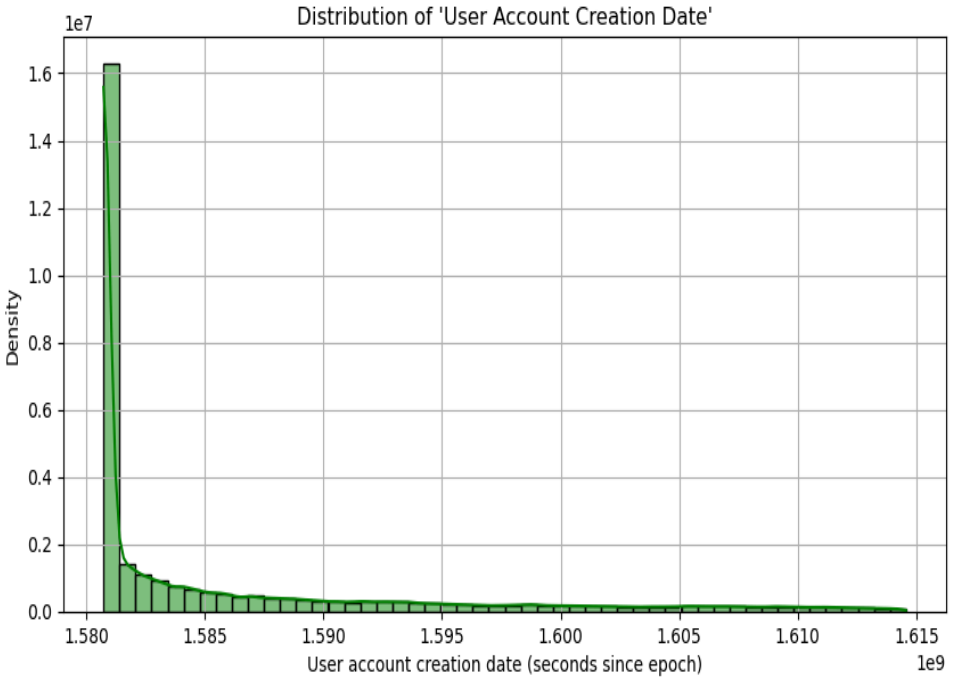


Fig. 38 Histogram showing the distribution of "User Account Creation Date"

Both assumptions of linearity and normality are clearly violated. Considering this, Spearman's correlation coefficient is a more reasonable choice to investigate the correlation between the two variables. This coefficient quantifies the degree of monotonicity of the relationship between two variables and determines whether they tend to increase or decrease together, without assuming neither linearity nor normality. Furthermore, it is robust to outliers and nonlinear relationships.

The Spearman's correlation coefficient that we computed is almost zero, which allowed us to conclude that there is no tendency for the two variables to change in tandem, suggesting no relationship. There is no statistical evidence that accounts have been assigned user IDs depending on the account's creation date. Given these considerations, we can discard "User ID" as it has probably been assigned random values that do not influence the outcome of login attempts in terms of credential stuffing risk. Finally, we verified whether there is a significant association between "User Account Creation Date" and "Is Account Takeover", plotting the former as a function of the latter. (Fig. 39)

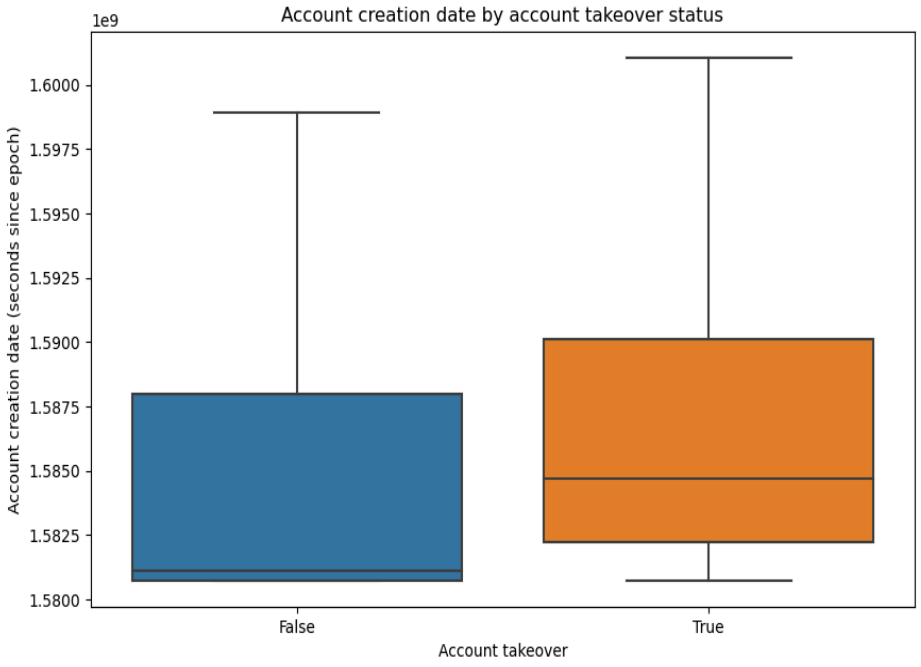


Fig. 39 Boxplot of "User Account Creation Date" for true and false account takeover instances

We decided to perform a *t*-test using "ttest_ind" from "scipy.stats", running the test after extracting the account's creation date of both true and false instances of "Is Account Takeover". Based on a p-value of ~0.11, we concluded that there is no statistically significant difference in mean "User Account Creation Date" values depending on "Is Account Takeover" status. The next step was to compute the point-biserial correlation

coefficient between the two variables, a coefficient which measures the strength and direction of the association between a continuous variable ("User Account Creation Date") and a binary variable ("Is Account Takeover"). The correlation coefficient's computation confirmed the results of the t-test, allowing us to conclude that the account's creation date (namely how old the account is) does not influence the credential stuffing risk of login attempts carried out through the user's account.

3.8 Clustering the dataset after removing "Is Account Takeover"

Our next hint towards building a predictive model was to perform clustering on the original dataset after removing the target variable. Our idea was to explore whether remaining independent features formed clusters that tended to align with true instances of "Is Account Takeover". This will also turn out to be useful in dealing with the target's large class imbalance. This subsection contains a detailed description of our modus operandi. The first step was to drop the following variables: "Region" and "City" (to simplify the complexity of our analysis given the large number of unique values, considering that geo-location information is usually provided at country/macro-area level), "Browser Name and Version", "OS Name and Version" (they provide redundant information which is already contained in "User Agent String"), "Round-Trip Time [ms]" (it contains 139 out of 140 missing values for true instances of "Is Account Takeover"), "Time of Day", "Day", "Month", "Year" and "User Account Creation Date" (they are extracted from "Login Timestamp" and they are not included in the original dataset). As clustering algorithms incorporate the notion of distance between data points, the subsequent step consisted of converting categorical and boolean columns to numeric using appropriate encoding schemes. For "Login Timestamp", we decided to convert it to "time since epoch", namely seconds that elapsed since 1st January 1970. (Fig. 40)

	Login Timestamp	User ID	IP Address	Country	ASN	User Agent String	Device Type	Login Successful	Is Attack IP	Is Account Takeover
0	1580733810	-4324475583306591935	10.0.65.171	NO	29695	Mozilla/5.0 (iPhone; CPU iPhone OS 13_4 like ...	mobile	False	False	False
1	1580733823	-4324475583306591935	194.87.207.6	AU	60117	Mozilla/5.0 (Linux; Android 4.1; Galaxy Nexus...	mobile	False	False	False
2	1580733835	-3284137479262433373	81.167.144.58	NO	29695	Mozilla/5.0 (iPad; CPU OS 7_1 like Mac OS X) ...	mobile	True	False	False

Fig. 40 Dataset after encoding "Login Timestamp"

We can see that the encoding has worked as intended and that "Login Timestamp" is now represented as a numeric value. Moving on in the embedding process, we decided to handle "User Agent String" by first applying TF-IDF vectorization to obtain the sparse TF-IDF matrix and then reducing its dimensionality using TruncatedSVD (Latent Semantic Analysis), selecting a number of components that is sufficient to explain

~80/85% of the variance. TF-IDF vectorization is an embedding technique widely used in NLP (Natural Language Processing) that converts each word into a number, by multiplying TF (term frequency, meaning the number of times that the word appears in the document divided by the total number of words in the document) by IDF (inverse document frequency, namely the log of the number of documents in the corpus divided by the number of documents in the corpus in which the term appears). The key idea is to assign a large score to terms that appear frequently in a document but rarely in the corpus. LSA (latent semantic analysis) is another popular technique in the field of NLP which takes as input a matrix where rows represent terms and columns documents and aims to reduce the number of rows while preserving documents similarity by using a mathematical technique named SVD (Singular Value Decomposition). To apply both techniques, we resorted to "sklearn", in particular to "TfidfVectorizer" from "sklearn.feature_extraction.text" and "TruncatedSVD" from "sklearn.decomposition". (Fig. 41)

```
user_agent_strings = X["User Agent String"].tolist()

# define a custom tokenizer (we want to consider only words and not numbers).
def custom_tokenizer(text):
    # use a regular expression to do so.
    tokens = re.findall(r'\b[^\d\W]+\b', text)
    return tokens

# initialize the tf-idf vectorizer.
vectorizer = TfidfVectorizer(tokenizer = custom_tokenizer)

# obtain the tf-idf matrix.
tfidf_matrix = vectorizer.fit_transform(user_agent_strings)

# save the numpy array to a file using the npy file format.
np.save("matrice_tf_idf.npy", tfidf_matrix)
```

Fig. 41 TF-IDF vectorization process

The first step was to convert "User Agent String" to a list. Then, we defined a customized tokenizer, using a regular expression, to consider only words and not numbers in order to reduce the dimensionality of the TF-IDF matrix. The latter is generated through "fit_transform" and saved to a ".npy" file to apply LSA on an external server and overcome memory usage constraints on the local machine. Finally, the LSA model was trained on 100 components.

The following step was to compute the LSA model's cumulative explained variance through "cumsum" applied on the percentage of variance explained by each component. (Fig. 42)

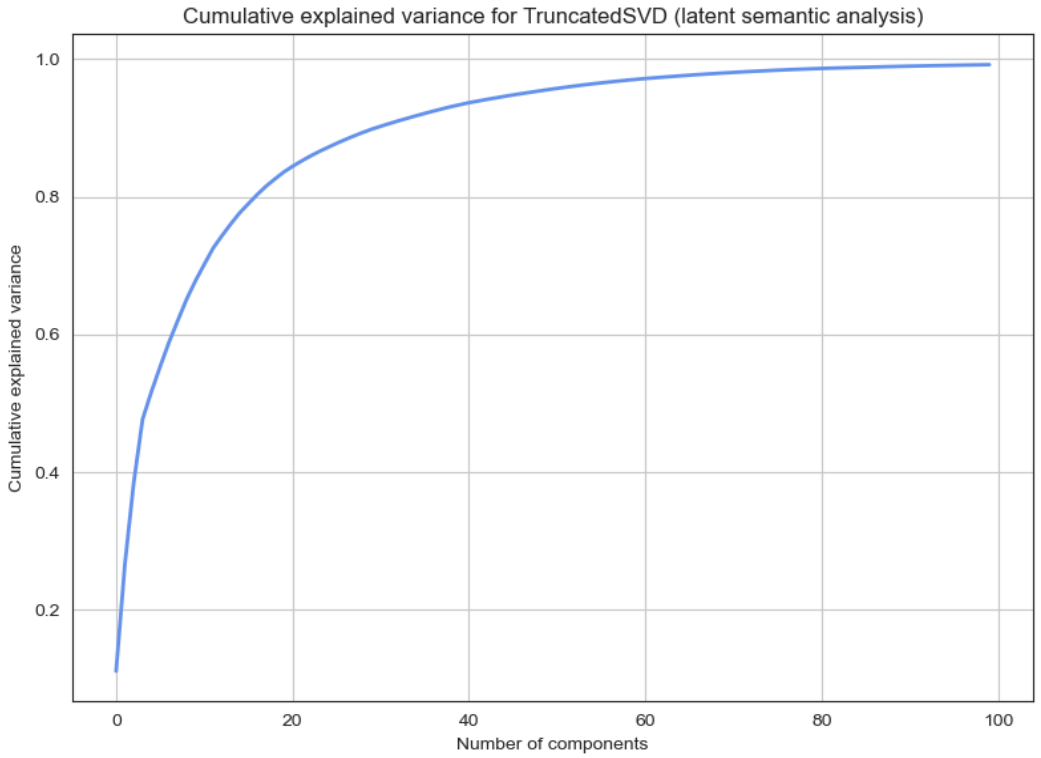


Fig. 42 Cumulative explained variance as a function of number of LSA components

Looking at Fig. 42, it is clear that 20 components are enough to explain ~85% of the variance. Therefore, we selected the first 20 components of the LSA matrix and saved them to a ".npz" file named "components_array". We then removed "User Agent String" from the original dataset and replaced it with the LSA components. Regarding the encoding of the remaining variables, we opted for WOEE (Weight Of Evidence Encoding) which does not assume any ordering among the categories and does not increase the complexity of the analysis by generating additional columns. (Fig. 43)

ASN	Country	User ID	Device Type	Login Successful	Is Attack IP	IP Address
-2.223563	-1.697898	-4.152713	-1.447501	-3.749341	-0.692508	7.194642
-0.250004	-0.580680	-4.152713	-1.447501	-3.749341	-0.692508	7.767988
-2.223563	-1.697898	9.256065	-1.447501	0.898347	-0.692508	9.815681
-3.506667	-3.676601	-4.152713	-1.447501	-3.749341	-0.692508	4.372181

Fig. 43 Weight of Evidence Encoding applied to "ASN", "Country", "User ID", "Device Type", "Login Successful", "Is Attack IP", "IP Address"

WOEE works in the following way: for each level x of a categorical feature, it computes the percentage of rows containing positives (compared to the whole set), the percentage of rows containing negatives (compared to the whole set) and takes the natural logarithm of the ratio. WOE(x) will be greater than or smaller than 0 depending on whether the level is more representative of positives or negatives. The results of the encoding process were saved to a ".npz" file named "WOE_embedding_array". Finally, "Login Timestamp" was saved to "numerical_array.npz". The three ".npz" files "numerical_array", "components_array" and "WOE_embedding_array" were concatenated on an external server to obtain the whole encoded dataset on which k-means clustering was applied.

k-means is a clustering algorithm which allows us to partition n data points into a pre-specified number of clusters k. These points are grouped so that the squared Euclidean distance of points within a cluster to the cluster's centroid (WCSS or within cluster sum of squares) is minimized. The results of k-means were stored in "clusteringSolution.pkl", a pickle file which contains an array consisting of four dictionaries, for each of which keys represent the number of clusters (from 2 to 50) whereas values are fitted k-means objects, Silhouette score [23], Calinski-Harabasz [24] and Davies-Bouldin [22] indices respectively. The latter three measures were used to choose the appropriate value of "k", namely a suitable number of clusters to ensure reasonable separation between the groups. Given the quadratic complexity of computing traditional Silhouette score, we relied on a "simplified Silhouette" which does not consider all pairwise distances between points but rather distances between data points and the centroid of the cluster to which they belong: this yields a sub-optimal solution, which is computationally feasible for large datasets and still retains the original mathematical properties of the original Silhouette score [21]. In general, the higher the Silhouette score, the better. Regarding the Calinski-Harabasz index, it is the ratio of between cluster dispersion (we want this to be large) to within cluster dispersion (we want this to be small). Larger values of the index suggest that clusters are dense and well separated. Finally, the Davies-Bouldin index measures the average similarity of each cluster with its most similar cluster where similarity is defined as the ratio of WCDs (within cluster distances, we want this to be as small as possible in order to have dense clusters) and BCDs (between cluster distances, we want this to be as large as possible to have clusters which are well separated). The minimum value that this index can assume is 0 and smaller values correspond to better clustering results. Each of these indices was plotted separately as a function of the number of clusters and a vertical dashed line was added in correspondence of the value of k that maximizes Silhouette score (Fig. 44) and Calinski-Harabasz index (Fig. 45) and minimizes Davies-Bouldin index (Fig. 46).

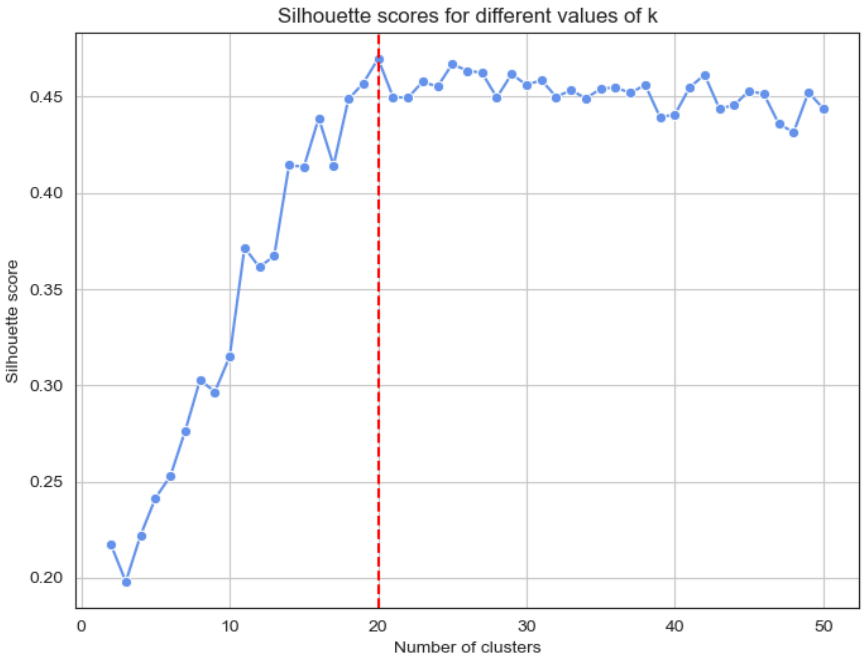


Fig. 44 Silhouette score as a function of the number of clusters k

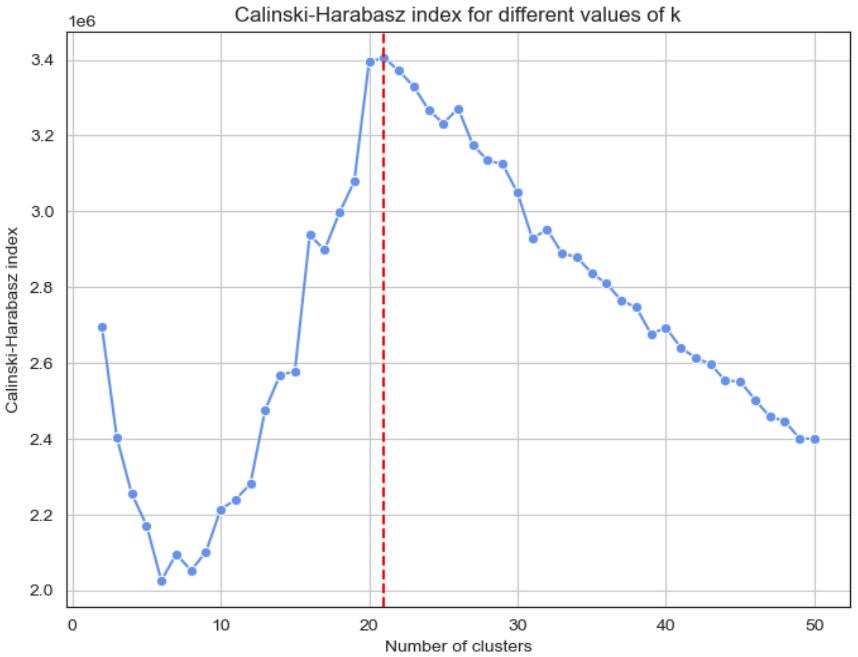


Fig. 45 Calinski-Harabasz index as a function of the number of clusters k

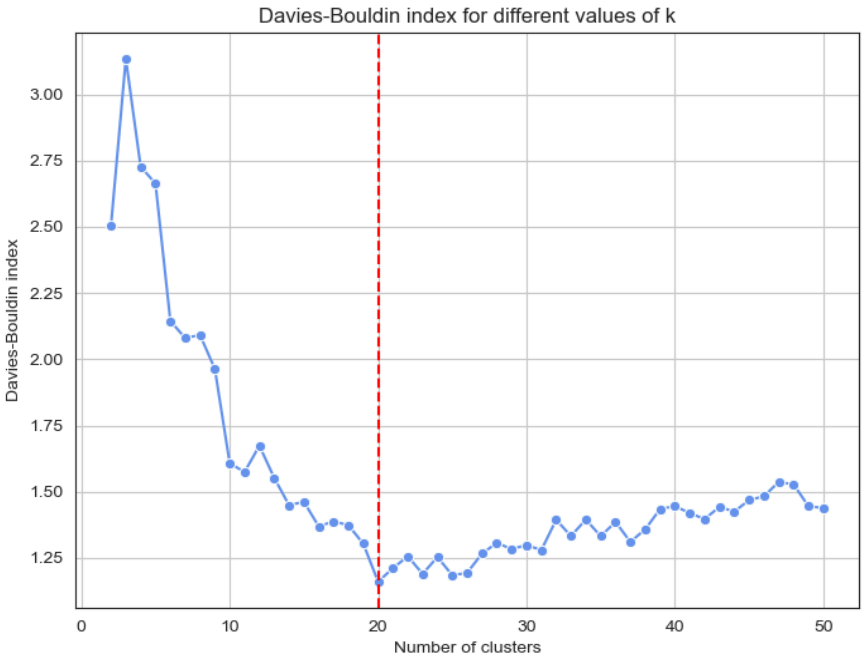


Fig. 46 Davies-Bouldin index as a function of the number of clusters k

For k = 20, Silhouette score and Davies-Bouldin index are maximized and minimized respectively whereas the Calinski-Harabasz index achieves its maximum value at k = 21. Since two out of the three criteria agree on k = 20 and for the latter the Calinski-Harabasz index is slightly below the maximum, we concluded that 20 was a reasonable choice for the appropriate number of clusters. After deciding on the optimal k, we investigated the cluster distribution for true instances of "Is Account Takeover" to check whether they tend to fall within the same or a few clusters (Fig. 47). To do so, we created and added a new column "Cluster label" to the original dataset and filtered the latter to include only true account takeover instances.

```
Cluster distribution for true account takeover instances is:
0     82
10    29
18     6
9      5
6      5
4      4
19     3
15     2
11     1
2      1
17     1
16     1
```

Fig. 47 Cluster distribution of true instances of "Is Account Takeover"

It is clear that the distribution is anything but uniform. We can see that 82 instances tend to fall within cluster 0 whereas 29 within cluster 10. To assess the absolute and relative importance of each cluster, we decided to compute two indices: one referred to as "sensitivity" (dividing the number of true account takeover instances that fall within the cluster by the cluster's size) and the other as "specificity" (dividing the number of true account takeover instances that fall within the cluster by the total number of true account takeover instances). These two indices were then normalized using min-max scaling (subtracting the minimum value and dividing by the difference between the maximum and minimum values) to bring them to the same [0,1] range and enable a comparison across different clusters. (Fig. 48)

cluster	size	sensitivity	specificity	normalized_sensitivity	normalized_specificity
0	5846624	2.157324e-05	0.585714	1.000000	1.000000
10	976060	1.099519e-05	0.207143	0.509668	0.353659
18	612141	9.801663e-06	0.042857	0.454344	0.073171
9	1098278	5.122636e-06	0.035714	0.237453	0.060976
6	1598650	3.400227e-06	0.035714	0.157613	0.060976
15	708927	2.971115e-06	0.014286	0.137722	0.024390
19	495943	1.593689e-06	0.021429	0.073873	0.036585
16	673148	1.409956e-06	0.007143	0.065357	0.012195
17	642995	1.314952e-06	0.007143	0.060953	0.012195
4	1882425	6.841555e-07	0.028571	0.031713	0.048780
2	2637518	6.255278e-07	0.007143	0.028996	0.012195
11	883394	4.169425e-07	0.007143	0.019327	0.012195

Fig. 48 Absolute and relative importance of each cluster. 8 missing clusters are not shown since they do not contain any true account takeover instance

As we may have reasonably expected, clusters 0 and 10 have the largest "sensitivity" and "specificity" values among all clusters. To inspect 0 and 10 in more detail and to compare them with other clusters, after adding back "Time of Day", "Day", "Month" and "Year" to the original dataset, we chose to perform two statistical analyses, one for intra-cluster and the other for inter-cluster difference, using "chi2_contingency". Concerning the intra-cluster difference, we tested the association of the following variables with "Is Account Takeover": "Country", "ASN", "Device Type", "Login Successful", "Is Attack IP", "Time of Day", "Day", "Month", "Year". (Fig. 49)

```
from scipy.stats import chi2_contingency # function to perform chi2 test.
# iterate over cluster labels.
for cluster_label in X["Cluster label"].unique():
    print(f"The following are the results for cluster {cluster_label}:")
    print("\n")
    # use a dictionary to store the results for each cluster.
    cluster_results = {}
    # filter the original dataframe to include only rows related to the cluster.
    cluster_data = X[X["Cluster label"] == cluster_label]
    # check if the cluster contains any true instance of account takeover.
    if cluster_data["Is Account Takeover"].any():
        # iterate over the categorical variables whose association with the target has to be verified.
        for col in cols_to_test:
            # create a contingency table between the categorical variable and the target.
            contingency_table = pd.crosstab(cluster_data[col], cluster_data["Is Account Takeover"])
            # perform the chi2 test.
            _, p_value, _, _ = chi2_contingency(contingency_table)
            # store the p-value for each variable in the cluster results dictionary.
            cluster_results[col] = p_value

        # print the p-value for each variable.
        print("p-values: ")
        for variable, p_value in cluster_results.items(): # iterate over the dictionary that stores variable:p-value
            print(f"{variable}: {p_value}")

        # print variables which are significant (p-value < 0.05) for the cluster.
        significant_vars = [var for var, p_value in cluster_results.items() if p_value < 0.05]
        print("\nSignificant Variables (p-value < 0.05) for cluster", cluster_label, ":")
        print(significant_vars)
        print("\n")
    else:
        # if there are no true instances within the cluster, print the following line.
        print("No true instances of account takeover within this cluster.\n")
```

Fig 49. Code snippet for intra-cluster difference analysis

Fig. 49 can be explained as follows: we imported "chi2_contigency" from "scipy.stats" and started iterating over cluster labels. For each unique value of "Cluster label", we filtered the original dataset to include only rows pertaining to the cluster. After checking whether the cluster contained any true instance (if not, the code would return a string warning that there are no true instances of account takeover within the cluster), we iterated over the categorical variables (stored in "cols_to_test") whose association with the target we wanted to investigate. For each of these variables, we created a contingency table between the latter and the target, performed a chi2 test and stored the results for each cluster in terms of *p*-value in a dictionary named "cluster_results". Finally, the

code iterates over the dictionary storing variable:*p*-value pairs and prints, for each cluster, significant variables where statistical significance is interpreted as *p*-value < alpha where alpha = 0.05. (Fig. 50, Fig. 51)

```
The following are the results for cluster 0:


p-values:
Device Type: 0.9926813833516646
Country: 0.0
ASN: 2.4185977848284032e-45
Login Successful: 0.0013172405316633092
Is Attack IP: 0.0
Time of Day: 2.7026023533389134e-05
Day: 0.125564443655384
Month: 0.002928182967040647
Year: 0.00049048154163227

Significant Variables (p-value < 0.05) for cluster 0 :
['Country', 'ASN', 'Login Successful', 'Is Attack IP', 'Time of Day', 'Month', 'Year
```

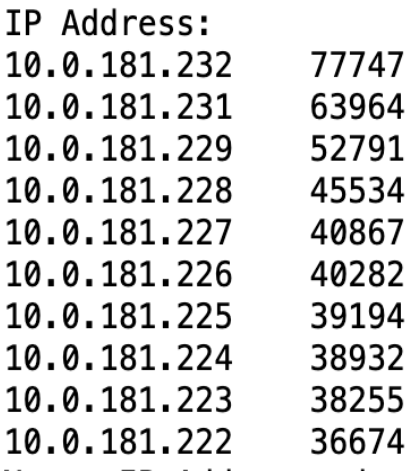
Fig. 50 Intra-cluster difference analysis results for cluster 0

```
The following are the results for cluster 18:


p-values:
Device Type: 1.0
Country: 6.277196358840015e-09
ASN: 0.5824047727980789
Login Successful: 6.217196677140092e-05
Is Attack IP: 0.018853099468014764
Time of Day: 0.03601943865083781
Day: 0.20122462650277814
Month: 0.2471389946244257
Year: 0.7552148267189377

Significant Variables (p-value < 0.05) for cluster 18 :
['Country', 'Login Successful', 'Is Attack IP', 'Time of Day']
```

Fig. 51 Intra-cluster difference analysis results for cluster 18

We can see that "Country", "ASN", "Login Successful", "Is Attack IP", "Time of Day", "Month" and "Year" are significant for cluster 0, meaning that these variables allow us to differentiate between true and false instances of "Is Account Takeover" that fall within the cluster. Overall, the results for the intra-cluster difference analysis show that "Country" is significant for all clusters except 15 and 16. Moreover, cluster 0 (which has both maximum "sensitivity" and "specificity") has a significant association with all variables except "Device Type" and "Day" whereas cluster 10 (that contains the second largest number of true account takeover instances) is associated with "Country", "ASN", "Login Successful", "Is Attack IP" and "Year". As the last step of our first

statistical analysis, we decided to print a frequency table for each cluster, showing the top 10 most frequent values for each categorical variable. Looking at the frequency table for clusters 0 and 10, we could notice that for the former, 9 out of the top 10 most popular IP addresses belong to the same network, with six of them being part of the same subnet, ~90% of login attempts are carried out from Norway, ~50% are associated with ASN "29695", all top 10 most popular user agent strings are related to Mac OS devices and almost all login attempts have been carried out from a desktop device. Inspecting the frequency table for cluster 10, we could observe that ~93% of login attempts are associated with the User ID "-4324475583306591935", all the top 10 most popular IP addresses belong to the same subnet (and same network as the 9 out of 10 most frequent addresses for cluster 0) (Fig. 52), ~50% of logins are associated with ASN "262582", all top 10 most popular user agent strings are associated with Mac OS devices and the vast majority of logins have been attempted from a desktop device.



```
IP Address:
10.0.181.232    77747
10.0.181.231    63964
10.0.181.229    52791
10.0.181.228    45534
10.0.181.227    40867
10.0.181.226    40282
10.0.181.225    39194
10.0.181.224    38932
10.0.181.223    38255
10.0.181.222    36674
```

Fig. 52 Frequency of "IP Address" for cluster 10

In relation to the inter-cluster difference, we decided to perform an analysis at three different levels: considering all 20 clusters (Fig. 53), collapsing clusters 0 and 10 into a single cluster and the remaining clusters into another, comparing clusters 0 and 10.

```
from scipy.stats import chi2_contingency
# use a list to store significant variables.
significant_variables = []
cols_to_test = ["Country", "ASN", "Device Type", "Login Successful", "Is Attack IP",
                "Is Account Takeover", "Time of Day", "Day", "Month", "Year"]
# iterate over each categorical variable whose association we want to test.
for col in cols_to_test:
    contingency_table = pd.crosstab(X_inter_difference[col], X_inter_difference["Cluster label"])
    chi2_statistic, p_value, _, _ = chi2_contingency(contingency_table)
    print(f"variable: {col}")
    print(f"p-value: {p_value}")
    if p_value < 0.05:
        significant_variables.append(col)
# print significant variables.
print("\nSignificant Variables (p-value < 0.05):")
print(significant_variables)
```

Fig. 53 Code snippet for inter-cluster difference analysis at the first level

This time, we wanted to test the association between variables stored in "cols_to_test" and "Cluster label" to determine which features are significant in helping us to distinguish between different clusters. Fig. 53 iterates over each categorical variable, builds a contingency table, performs a chi2 test, prints the variable and the associated p-value and if the latter is < 0.05, it appends the variable to "significant_variables", a list that is used to store significant variables for the inter-cluster difference. (Fig. 54)

```
Significant Variables (p-value < 0.05):
['Country', 'ASN', 'Device Type', 'Login Successful', 'Is Attack IP', 'Is Account Takeover', 'Time of Day', 'Day',
'Month', 'Year']
```

Fig. 54 Results of inter-cluster difference analysis at the first level

Regarding the second inter-cluster analysis, we made a copy of the original dataset and created a new column called "Account Takeover Cluster" which takes on value 1 if the login attempt instance belongs to either cluster 0 or 10, 0 otherwise. We dropped the original "Cluster label" variable and applied a similar reasoning as in the first analysis, this time treating "Account Takeover Cluster" as values to group by in the columns in "pd.crosstab". The second analysis led to the same conclusions in terms of significant variables as the first.

With respect to the last inter-cluster analysis, we asked ourselves: What are the differences between clusters 0 and 10? How come as many as 29 true account takeover instances tend to fall within cluster 10 rather than 0? In order to answer these questions, we filtered the original dataset to include only data points whose cluster label is either 0 or 10 and performed a chi2 test to verify the association of each categorical variable with "Cluster label". The results allowed us to draw similar conclusions as for the other inter-cluster difference analyses. Overall, the main takeaways of the clustering analysis were that all categorical variables ("Country", "ASN", "Device Type", "Login Successful", "Is Attack IP", "Time of Day", "Day", "Month", "Year") are significant for the inter-cluster differences considering all 20 clusters and only clusters 0 and 10, with the target "Is Account Takeover" being significant for the analysis performed after collapsing clusters 0 and 10 into a single cluster and the remaining clusters into another. Furthermore, "Country", "ASN", "Login Successful", "Is Attack IP", "Time of Day", "Month" and "Year" are statistically significant for cluster 0, meaning that these variables allow us to distinguish between true and false account takeover instances within cluster 0 (that 82/140 true instances belong to) whereas "Country", "ASN", "Login Successful", "Is Attack IP" and "Year" are significant for cluster 10, implying that they can help us in differentiating between true and false instances of "Is Account Takeover" within cluster 10 (that 29/140 true instances belong to). Finally, we saved the updated dataset to a ".csv" file named "X_model".

To wrap up, the main takeaways of previous subsections towards our goal of building a predictive model for the binary target "Is Account Takeover" are the following:

- we should keep the following variables: "Login Timestamp", "IP Address" (encoding it using a method that allows to preserve the relationship between addresses belonging to the same subnet/network), "Country", "ASN" (there is a strong level of statistical significance for the association with the target), "User Agent String" (we used it to perform clustering after removing the target), "Device Type", "Login Successful", "Is Attack IP", "Month" (encoding the latter using a method that allows to account for the variable's cyclic nature) and "Year".
- we should discard the following features: "User ID" (there is not any hidden information behind it so we can drop it as it has been probably assigned at random), "User Account Creation Date" (we showed that there is no association with "Is Account Takeover" by performing a t-test followed by the calculation of the point-biserial correlation coefficient).
- "Time of Day" is not associated with the target (we proved it by running a chi2 test) but it is statistically significant for the three inter-cluster differences and for the intra-cluster difference (it allows to distinguish between true and false instances of "Is Account Takeover) for clusters 0 and 18 (which combine for 88/140 of true instances). We will keep the variable as it is significant for the clustering results that we will use as reference to perform under-sampling of the original dataset as a strategy to deal with the target's high degree of class imbalance. A similar reasoning applies to "Day" which is not associated with the target but is significant for all inter-cluster differences.

3.9 Data preprocessing

In order to carry out the data preprocessing and model building steps, we decided to create two folders: "model_building_entire", containing scripts where models are trained on the entire dataset and "model_building_under", hosting scripts where models are trained using the under-sampled dataset according to the clustering's results. The data-preprocessing operations are the same for all scripts within a folder and they consisted of the following steps:

1) We started by reading the ".csv" file containing the dataset that has to be preprocessed which consists of ~31 million rows and 16 columns. We took a glimpse at it using "head" from "pandas". Furthermore, in "model_building_under", we selected only instances whose cluster label is either 0 or 10 (the two clusters characterized by largest "sensitivity" and "specificity" values), reducing the number of rows in the starting dataset to ~6.5 millions.

2) Considering the main takeaways of previous subsections, we got rid of variables that are not associated with the target: "User Account Creation Date" (t-test and point-biserial correlation coefficient confirmed to us that the account's age does not influence the credential stuffing risk of logins attempted from the user's account) ,

"User ID" (we proved that there is not any hidden information behind it since it has been probably assigned randomly), "User Agent String" (it will be replaced with the first 20 components of the LSA matrix).

3) After removing variables that do not have any statistical association with the target, we started encoding those that do. We began by converting "Login Timestamp" first to "datetime" format and then to time since epoch, that is, seconds that elapsed since 1st January 1970. The next step was to import "ipaddress", a manipulation library that allows to work with both IPv4 and IPv6 addresses. This library has a built-in function "IPv4Address" that enables the conversion of IPv4 addresses to integers. The IP address is split into octets and the first octet is multiplied by 256^3, the second by 256^2, the third by 256^1 and the fourth by 256^0 (i.e., 1). This encoding scheme allows us to preserve the relationship between addresses that belong to the same network and subnet since the latter will be associated with close integer values. (Fig. 55)

Login Timestamp	IP Address	Country	ASN	Device Type	Login Successful	Is Attack IP	Is Account Takeover	Cluster label	Time of Day	Day	Month	Year
1580733859	1558015394	NO	29695	desktop	True	False	False	0	Afternoon	3	2	2020
1580733859	1412447623	NO	15659	desktop	True	False	False	0	Afternoon	3	2	2020
1580733864	2620665939	NO	29695	desktop	True	False	False	0	Afternoon	3	2	2020
1580733871	1369861544	NO	29695	desktop	True	False	False	0	Afternoon	3	2	2020
1580733883	167801130	NO	Other	desktop	True	False	False	0	Afternoon	3	2	2020

Fig. 55 Dataset after "Login Timestamp" and "IP Address" encoding

Both "Login Timestamp" and "IP Address" now take on integer values. In order to encode cyclic features like "Time of Day", "Day" and "Month", we decided to resort to cyclic encoding. Considering that the former takes on values "Morning", "Afternoon", "Evening" and "Night", we first defined a mapping from strings to integers ("Night" to 0, "Morning" to 1, "Afternoon" to 2, "Evening" to 3). Concerning "Month" and "Day", we first checked how the variables are encoded and realized that January was encoded as 1, February as 2, the 1st day of the month as 1, the second as 2 and so on and so forth while integer values should start from 0. We subtracted 1 from each integer value in "Month" and "Day" and defined a function "cyclic_encoding" which takes as input a dataframe, a specific column and the latter's maximum value and returns as output the updated dataframe after computing sine and cosine coordinates of the column. For example, "Month" takes on values 0, 1, 2 ,3 (maximum value is 4), which means that 0 will be represented as 0pi, 1 as pi/2, 2 as pi and 3 as 3pi/2. Sine and cosine coordinates will be extracted from angles in radians. (Fig. 56)

IP Address	Country	ASN	Device Type	Login Successful	Is Attack IP	Is Account Takeover	Cluster label	Year	Time of Day_sin	Time of Day_cos	Month_sin	Month_cos	Day_sin	Day_cos
1558015394	NO	29695	desktop	True	False	False	0	2020	1.224647e-16	-1.0	0.5	0.866025	0.394356	0.918958
1412447623	NO	15659	desktop	True	False	False	0	2020	1.224647e-16	-1.0	0.5	0.866025	0.394356	0.918958
2620665939	NO	29695	desktop	True	False	False	0	2020	1.224647e-16	-1.0	0.5	0.866025	0.394356	0.918958
1369861544	NO	29695	desktop	True	False	False	0	2020	1.224647e-16	-1.0	0.5	0.866025	0.394356	0.918958
167801130	NO	Other	desktop	True	False	False	0	2020	1.224647e-16	-1.0	0.5	0.866025	0.394356	0.918958

Fig. 56 Dataset after applying "cyclic_encoding" to "Time of Day", "Day" and "Month"

"Time of Day", "Day" and "Month" have been replaced with columns containing sine and cosine coordinates of angles in radians.

4) In order to replace "User Agent String" with the first 20 components of the LSA matrix, we started by loading the ".npz" file "components_array" (which stores the aforementioned components in the form of a "numpy" array that can be accessed through the key "arr_0") and created a dataframe "components_dataframe" to store them. In "model_building_under", we also extracted row indices of the under-sampled dataset in order to merge the latter with corresponding rows of "components_dataframe". The next step was to concatenate "X_model" (dataset after all preprocessing steps have been applied) with "components_dataframe" ("components_dataframe_selected" in "model_building_under") to obtain "X_model_final". (Fig. 57)

```
Index(['Login Timestamp', 'IP Address', 'Country', 'ASN', 'Device Type',
       'Login Successful', 'Is Attack IP', 'Is Account Takeover',
       'Cluster label', 'Year', 'Time of Day_sin', 'Time of Day_cos',
       'Month_sin', 'Month_cos', 'Day_sin', 'Day_cos', 'LSA_component_1',
       'LSA_component_2', 'LSA_component_3', 'LSA_component_4',
       'LSA_component_5', 'LSA_component_6', 'LSA_component_7',
       'LSA_component_8', 'LSA_component_9', 'LSA_component_10',
       'LSA_component_11', 'LSA_component_12', 'LSA_component_13',
       'LSA_component_14', 'LSA_component_15', 'LSA_component_16',
       'LSA_component_17', 'LSA_component_18', 'LSA_component_19',
       'LSA_component_20'],
```

Fig. 57 "X_model_final" columns

5) After taking a glimpse at "X_model_final", we decided to discard "Cluster label" since it does not provide any useful information towards our goal of building a predictive model for "Is Account Takeover". The remaining preprocessing steps prior to obtaining our final train and test sets are shown in Fig. 58.

```
from sklearn.model_selection import train_test_split # to split dataset into train and test sets.
# employ robust scaler as it uses statistics that are less sensitive to outliers.
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import OrdinalEncoder # to encode binary (boolean) variables.
from category_encoders import WOEEncoder # to perform weight of evidence encoding.
from sklearn.compose import ColumnTransformer # to apply transformers to multiple columns simultaneously.
# split the dataset into predictors and target
predictors = X_model_final.drop("Is Account Takeover", axis=1)
target = X_model_final["Is Account Takeover"]
# split data into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(predictors, target, test_size=0.2,
                                                    random_state=42, stratify = target)
# define different encoding schemes for specific columns.
num_columns = X_train.drop(["Country", "ASN", "Device Type", "Year", "Login Successful", "Is Attack IP"], axis=1)
num_attributes = list(num_columns)
woe_attributes = ["Country", "ASN", "Device Type", "Year"]
le_attributes = ["Login Successful", "Is Attack IP"]
# apply weight of evidence encoding
woe_encoder = WOEEncoder(cols=woe_attributes)
X_train_woe = woe_encoder.fit_transform(X_train[woe_attributes], y_train)
X_test_woe = woe_encoder.transform(X_test[woe_attributes])
# concatenate the weight of evidence encoded columns with the remaining columns.
X_train_concat = pd.concat([X_train_woe, X_train.drop(woe_attributes, axis=1)], axis=1)
X_test_concat = pd.concat([X_test_woe, X_test.drop(woe_attributes, axis=1)], axis=1)
ct = ColumnTransformer([
    ("num", RobustScaler(), num_attributes), # scale numerical features.
    ("label", OrdinalEncoder(), le_attributes) # encode remaining categorical (boolean) columns.
], remainder = "passthrough") # remainder set to "passthrough" to leave the remaining columns untouched.
X_train_final = ct.fit_transform(X_train_concat)
X_test_final = ct.transform(X_test_concat)
y_train = y_train.values
y_test = y_test.values
```

Fig. 58 Encoding and scaling remaining features to obtain "X_train_final" and "X_test_final"

We started by importing "train_test_split" from "sklearn.model_selection" (to split the dataset into train and test sets), "RobustScaler" and "OrdinalEncoder" from "sklearn.preprocessing" (the former to scale numeric variables, by subtracting the median and dividing by the interquartile range, namely the difference between 3rd and 1st quartile, and the latter to apply label encoding to multiple binary/boolean columns simultaneously), "WOEEncoder" from "category_encoders" (to apply weight of evidence encoding to "Country", "ASN", "Device Type" and "Year", considering that there is not a natural ordering among the categories and that this encoding scheme does not generate additional columns) , "ColumnTransformer" from "sklearn.compose" (to apply transformations to multiple columns simultaneously and in a compact way). After splitting the dataset into train and test sets (stratifying by "Is Account Takeover" and using a "test_size" of 0.2), we specified a different encoding scheme depending on the column: weight of evidence for "Country", "ASN", "Device Type", "Year" and ordinal for boolean variables "Login Successful" and "Is Attack IP". All remaining numeric features were scaled using "RobustScaler". The next step was to apply weight of evidence encoding to the specified columns and concatenate them back with the original train and test sets. After scaling numeric features, encoding boolean variables and leaving the remaining columns untouched setting the "remainder" parameter of "ColumnTransformer" to "passthrough", we came up with the final version of our train and test sets, calling "fit_transform" in the former case and "transform" in the latter.

3.10 Model building

After obtaining our final train and test sets, we decided to implement the following models: Naive Bayes, Logistic Regression, Decision Tree and Random Forest. To evaluate their performance, we used precision, recall, F1-score and Matthew's correlation coefficient. We will now provide a brief explanation of each algorithm and evaluation metric.

3.10.1 Naive Bayes

It is a very simple algorithm which is widely used for binary classification tasks. Like the name suggests, it is based on Bayes theorem with the "naive" assumption of independence between features. Despite being simple, it often performs well and is commonly used in a wide range of domains such as text classification and spam filtering. The algorithm assumes that features are conditionally independent given the class label (ex. for three features $P(x1,x2,x3|y) = P(x1|y) * P(x2|y) * P(x3|y)$) and for each new observation, the predicted class will be that associated with the highest posterior probability. Its main advantages are that it is relatively simple and it tends to work well with large datasets whereas the main limitation is the independence assumption. To implement Naive Bayes, we imported "GaussianNB" from "sklearn.naive_bayes", a Naive Bayes model which assumes that features are independent and normally distributed.

3.10.2 Logistic Regression

Logistic regression is a statistical method used in binary classification. It outputs values in the range [0,1] as it models the probability that an observation belongs to a given class using the logistic function. The predicted probability can be interpreted as the likelihood that the observation belongs to the positive class (1) and a decision is made according to a threshold whose default value is usually 0.5: if the predicted probability is >= 0.5, the observation will be assigned to class 1, 0 otherwise. Its main advantages are computational efficiency for large datasets and interpretability in terms of probabilities. Its main limitation is that it assumes linearity and it might not be a suitable choice when complex non-linear interactions between features and the target have to be modeled. To implement Logistic Regression, we imported "SGDClassifier" from "sklearn.linear_model", a linear model which optimizes the specified loss function by "SGD" (stochastic gradient descent) for faster convergence. According to sklearn's documentation, specifying "log_loss" as the loss function that has to be minimized gives logistic regression, a probabilistic classifier.

3.10.3 Decision Tree

Decision Tree is a supervised learning (data are labeled) model that is used in both regression and classification tasks. In the context of binary classification, data are subsequently split based on the values of input features with the goal of creating decision rules. In a decision tree, nodes represent features, edges represent decision

rules and leaves represent class labels. The main advantages of decision tree are its interpretability and the fact that it does not make any assumption about the underlying distribution of the data, making it suitable to model both linear and nonlinear relationships. On the other hand, it is prone to overfitting and unstable as small changes in the training data can lead to significantly different results. To implement this model, we relied on "DecisionTreeClassifier" from "sklearn.tree".

3.10.4 Random Forest

Random Forest is an ensemble learning technique that brings together multiple decision trees. The predictions of the latter are combined to improve the model's performance and robustness. Each tree is trained independently on a subset of the training data and the final prediction is made considering the votes of all trees. Its main advantages are that it is robust to overfitting and in general it leads to more accurate results compared to single decision trees. This comes at the cost of increased computational complexity (especially for large datasets) and reduced interpretability with respect to decision tree. To implement this robust classifier, we imported "RandomForestClassifier" from "sklearn.ensemble".

3.11 Performance metrics

3.11.1 Precision

The following is the formula for precision score:

$$\text{precision} = TP/\ TP + FP$$

where TP = true positives (number of true instances that are correctly classified) and FP = false positives (number of negative instances that are mislabeled as positive). It takes on values in the range [0,1] and the higher the value the better is the classifier at avoiding false positives. It should be used in contexts where minimizing the false positives rate is crucial but always in conjunction with other evaluation metrics.

3.11.2 Recall

The following is the formula for recall score:

$$\text{recall} = TP/TP + FN$$

where TP = true positives (number of correctly classified true instances) and FN = false negatives (number of true instances that are misclassified as negative). It takes on values in the [0,1] range and the higher the value the greater the ability of the classifier at avoiding false negatives. It should be used in contexts where minimizing the false negatives rate (correctly identifying all positive instances) is of utmost importance.

3.11.3 F1-score

F1-score combines both precision and recall into a single score. Its formula is the following:

F1-score = 2 * precision * recall / (precision + recall)

where precision and recall are the classifier's. It takes on values in the [0,1] range as well and it represents a balance between the two evaluation metrics.

3.11.4 MCC

MCC (Matthew's correlation coefficient) is an evaluation metric which is used in the context of imbalanced data classification. (Fig. 59)

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Fig. 59 Matthew's correlation coefficient formula

where TP = true positives (number of correctly classified true instances), TN = true negatives (number of correctly labeled false instances), FP = false positives (number of false instances misclassified as true), FN = false negatives (number of true instances mislabeled as false). It takes on values in the range [-1,1] where 1 is a perfect prediction, 0 represents a random prediction and -1 indicates complete disagreement between the prediction and the observation.

4) Results and Discussion

We started by importing all required evaluation metrics ("precision_score", "recall_score", "f1_score", "matthews_corrcoef") from "sklearn.metrics", creating the dictionaries "results" (to store results of non-optimized models trained on the whole dataset) and "results_optimized" (to store performance of optimized algorithms trained on the entire dataset) for "model_building_entire" and the equivalent "results_under" and "results_under_optimized" for "model_building_under". Results will be presented and discussed in more detail in the following subsections.

4.1 Model results on the entire dataset

4.1.1 Non-optimized models

The following results were obtained by fitting the models on "X_train_final" and "y_train", using "sklearn" default parameter values. To output each model's predictions on unseen data, we called "predict" on "X_test_final". The following are the parameter values that we used to train each of the chosen models.

"GaussianNB" -> "var_smoothing" = 1e-9

"SGDClassifier" -> "penalty" = "l2", "alpha" = 0.0001 , "l1_ratio" = 0.15

"DecisionTreeClassifier" -> "criterion" = "gini", "max_depth" = None, "min_samples_split" = 2, "min_samples_leaf" = 1

"RandomForestClassifier" -> "n_estimators" = 100, "criterion" = "gini", "max_depth" = None, "min_samples_split" = 2, "min_samples_leaf" = 1, "class_weight" = None

	precision	recall	f1-score	mcc
naive bayes	0.000000	0.000000	0.000000	0.000000
logistic regression	0.000003	0.178571	0.000005	-0.000569
decision tree	0.151515	0.178571	0.163934	0.164484
random forest	0.333333	0.071429	0.117647	0.154302

Fig. 60 Results of non-optimized models (trained on the entire dataset)

4.1.2 Optimized models

In order to carry out hyperparameter tuning for each of the selected models, we decided to perform Bayesian Optimization, relying on "BayesSearchCV" from "scikit-optimize". Bayesian optimization is more efficient than traditional methods like "GridSearchCV" and "RandomizedSearchCV" as it adjusts the search space dynamically based on both prior knowledge and unexplored promising regions where optimal hyper-parameters are more likely to be found. This combination of exploitation and exploration leads to fewer evaluations and faster convergence. Furthermore, it can handle high-dimensional search spaces efficiently, making it suitable for when there are numerous hyper-parameters that have to be optimized. Concerning "GaussianNB", we defined the following search space, where "var_smoothing" is the parameter's name and "Real" specifies an interval of floating point numbers that ranges from 1e-9 to 1e-1 (both included), with points that are sampled uniformly between log(lower bound) and log(upper bound) where log has base 10. This allows us to explore values of different orders of magnitude. (Fig. 61)

```
nb_param_space = {"var_smoothing": Real(1e-9, 1e-1, "log-uniform")}
```

Fig. 61 Search space for Naive Bayes

Furthermore, in "BayesSearchCV", we specified 200 as the number of iterations of the optimization process, "precision"/"f1"/"mcc_scorer" as the evaluation metric that we wish to optimize and "StratifiedKFold" with 5 splits as desired cross-validation strategy.
Concerning the optimization of "SGDClassifier", we defined the parameter space shown in Fig. 62:

```
lr_param_space = {
    "penalty": Categorical(["l1", "l2", "elasticnet"]),
    "alpha": Real(1e-6, 1e-2, "log-uniform"),
    "l1_ratio": Real(0.05, 0.3, "uniform")
}
```

Fig. 62 Search space for Logistic Regression

where "penalty", "alpha" and "l1_ratio" are the parameters we wish to optimize, "Categorical" is used to define a search space that takes on categorical values, and "Real" is used to define a search space that takes on floating point numbers between a lower bound and an upper bound, with points that are sampled log-uniformly for "alpha" and uniformly for "l1_ratio". As usual, in BayesSearchCV, we specified 200 as the number of iterations, "precision"/"f1"/"mcc_scorer" as the evaluation metric to be optimized and "StratifiedKFold" with 5 splits as cross-validation strategy.

To optimize "DecisionTreeClassifier", we chose to define the search space displayed in Fig. 63:

```
dt_param_space = {
    "criterion": Categorical(["gini", "entropy", "log_loss"]),
    "max_depth": Integer(1, 10, "uniform"),
    "min_samples_split": Integer(1, 10, "uniform"),
    "min_samples_leaf": Integer(1, 10, "uniform")
}
```

Fig. 63 Search space for Decision Tree

where "criterion", "max_depth", "min_samples_split", "min_samples_leaf" are the parameters we want to optimize. For "criterion", we defined a "Categorical" space that can take on three values ("gini", "entropy", "log_loss") whereas for the remaining three parameters, we defined an "Integer" interval, with points that are sampled uniformly from an interval of integer values between 1 and 10 (both included).

Finally, for the optimization process of "RandomForestClassifier", we defined the parameter space of Fig. 64:

```
rf_param_space = {
    "n_estimators": Integer(50, 150, "uniform"),
    "criterion": Categorical(["gini", "entropy"]),
    "max_depth": Integer(1, 10, "uniform"),
    "min_samples_split": Integer(1, 10, "uniform"),
    "min_samples_leaf": Integer(1, 10, "uniform"),
    "class_weight": Categorical(["balanced", "balanced_subsample", None])
}
```

Fig. 64 Search space for Random Forest

where keys of the dictionary are the model's parameters we wish to optimize. For "criterion" and "class_weight", we defined a "Categorical" space as the two parameters take on categorical values. For the number of estimators, we specified an "Integer" interval ranging from 50 to 150, with points sampled uniformly. Concerning the remaining parameters, we opted for an interval between 1 and 10, with uniformly sampled integer values. Once again, in "BayesSearchCV", we specified 200 as number of iterations, "precision"/"f1"/"mcc_scorer" as scoring criterion and "StratifiedKFold" as cross-validation strategy.

In the end, each model was fitted on the optimized hyperparameters (that we retrieved through the "best_params_" attribute of each "BayesSearchCV" object), obtaining the results that are shown in the next three subsections.

4.1.2.1 Optimized models (scoring = "precision")

	precision	recall	f1-score	mcc
naive bayes	0.000000	0.000000	0.000000	0.000000
decision tree	0.250000	0.035714	0.062500	0.094490

Fig. 65 Results of optimized models (trained on the entire dataset) when "precision" is set as scoring criterion for Bayesian Optimization

4.1.2.2 Optimized models (scoring = "f1")

	precision	recall	f1-score	mcc
naive bayes	0.000000	0.000000	0.000000	0.000000
decision tree	1.000000	0.035714	0.068966	0.188982

Fig. 66 Results of optimized models (trained on the entire dataset) when "f1" is set as scoring criterion for Bayesian Optimization

4.1.2.3 Optimized models (scoring = "mcc_scorer")

	precision	recall	f1-score	mcc
naive bayes	0.000000	0.000000	0.000000	0.000000
decision tree	0.000000	0.000000	0.000000	0.000000

Fig. 67 Results of optimized models (trained on the entire dataset) when "mcc_scorer" is set as scoring criterion for Bayesian Optimization

4.2 Model results on the under-sampled dataset

4.2.1 Non-optimized models

	precision	recall	f1-score	mcc
naive bayes	0.000161	0.863636	0.000322	0.011066
logistic regression	0.000000	0.000000	0.000000	0.000000
decision tree	0.000000	0.000000	0.000000	-0.000018
random forest	0.000000	0.000000	0.000000	-0.000005

Fig. 68 Results of non-optimized models (trained on the under-sampled dataset)

4.2.2 Optimized models

For the scripts contained in "model_building_under", the optimization process was the same as for those hosted in "model_building_entire", since we defined an analogous search space for each model, as well as parameter values of "BayesSearchCV". Results are displayed in the upcoming subsections.

4.2.2.1 Optimized models (scoring = "precision")

	precision	recall	f1-score	mcc
naive bayes	0.017621	0.363636	0.033613	0.079985
logistic regression	0.000000	0.000000	0.000000	0.000000
decision tree	0.000000	0.000000	0.000000	-0.000009
random forest	0.000000	0.000000	0.000000	0.000000

Fig. 69 Results of optimized models (trained on the under-sampled dataset) when "precision" is set as scoring criterion for Bayesian Optimization

4.2.2.2 Optimized models (scoring = "f1")

	precision	recall	f1-score	mcc
naive bayes	0.018927	0.272727	0.035398	0.071792
logistic regression	0.000000	0.000000	0.000000	0.000000
decision tree	0.000000	0.000000	0.000000	-0.000009
random forest	0.000000	0.000000	0.000000	0.000000

Fig. 70 Results of optimized models (trained on the under-sampled dataset) when "f1" is set as scoring criterion for Bayesian Optimization

4.2.2.3 Optimized models (scoring = "mcc_scorer")

	precision	recall	f1-score	mcc
naive bayes	0.014754	0.409091	0.028481	0.077619
logistic regression	0.000000	0.000000	0.000000	0.000000
decision tree	0.000000	0.000000	0.000000	-0.000010
random forest	0.001738	0.681818	0.003467	0.034198

Fig. 71 Results of optimized models (trained on the under-sampled dataset) when "mcc_scorer" is set as scoring criterion for Bayesian Optimization

4.3 Results Discussion

When models are trained on the entire dataset but not optimized, Fig. 60 displays that Decision Tree outperforms the remaining models in terms of all metrics except precision, for which Random Forest stands out. Concerning each model's optimization process, we set a watchdog timer of 7 days to prevent excessive use of

computational resources. This can be helpful when working with large datasets and/or large search spaces as the optimization process is automatically stopped once the time threshold is exceeded. Fig. 65, Fig, 66 and Fig. 67 show results for only two of the four models, implying that the optimization procedure of Logistic Regression and Random Forest did not converge within the 7-day time limit. Nevertheless, among the algorithms whose optimization process has been successfully completed, Decision Tree clearly outperforms Naive Bayes in terms of all evaluation criteria, regardless of the metric that is used as scoring criterion for Bayesian Optimization. When "f1" is set as scoring metric, Decision Tree displays improved precision and Matthew's correlation coefficient while having comparable results in terms of remaining performance metrics as to when "precision" is used to guide the optimization process.

When working on the under-sampled dataset, Fig. 68 shows that, when models are non-optimized, Naive Bayes is the best performing algorithm in terms of all evaluation metrics. Once the models have been optimized, Naive Bayes still performs better in terms of all metrics, both when "precision" and "f1" are set as scoring criteria, as evidenced by Fig. 69 and Fig. 70. When "mcc_scorer" (a wrapper around the "matthews_corrcoef" performance metric) is chosen as scoring criterion, Fig. 71 shows that Random Forest performs better in terms of recall, whereas Naive Bayes is still the best performing model in terms of all remaining evaluation criteria.

After each model's optimization process is successfully completed, we can see that all best performing models yield comparable results in terms of all metrics except recall, for which Random Forest performs better when "matthews_corrcoef" is selected as scoring criterion. As we previously mentioned in 3.11.2, recall should be carefully taken into account when minimizing false negatives is a primary concern. In the context of our problem, a false negative is a login attempt which is labeled as benign while it is actually an account takeover attempt. On the other hand, a false positive is a login instance which is classified as account takeover whereas it is actually a legitimate attempt. Of course, both errors come at a cost: for instance, in the case of a false positive, the user's account might be temporarily locked or the user might be prompted for additional verification steps, with a negative impact on the overall user experience. However, failing to detect a malicious login attempt is associated with a higher cost, as the attacker might be granted access to the user's account, as well as the financial resources that are stored there. This implies that recall should be carefully considered, without, of course, neglecting other evaluation metrics. Based on these considerations, when working on the under-sampled dataset, Fig. 71 shows that Random Forest (and "mcc_scorer" as scoring criterion for the model's optimization process) are the overall best choices in the context of our binary classification problem.

5) Conclusion

In spite of the low rate of success, credential stuffing is a cyber attack which shouldn't be underestimated, considering its large socio-economic impact and people's password reuse habits. After acknowledging how related literature has traditionally tackled the problem, we identified a research gap insofar that no previous study had leveraged login-related features to predict the credential stuffing (account takeover) risk of login attempts. Starting from this consideration, we reviewed related literature in the fields of credential stuffing, RBA (risk-based authentication, the source of the dataset that we used to carry out our analysis) and imbalanced data classification (given the problem's imbalanced nature with most login attempts classified as benign and a few labeled as malicious). After obtaining the dataset from Kaggle, we selected the binary variable "Is Account Takeover" as our target. The vast majority of the remaining work was devoted to assessing the relationship between the target and independent variables, using plots and statistical tests. Furthermore, we clustered the dataset after removing the target variable, to check whether true instances tended to form clusters and the dataset's large dimensionality could be reduced. Based on the latter results, we implemented different classification algorithms, both on the full and under-sampled datasets, whose performance was evaluated using robust metrics such as precision, recall, F-1 score and Matthew's correlation coefficient. The best performing model was identified for both settings, taking into account problem-specific characteristic and error costs.

Overall, the work was very challenging. In spite of having some characterizing elements, credential stuffing attacks are hard to distinguish from bening login attempts, especially when the former are carried out using automated bots. Moreover, account takeover attempts represent a small fraction of all logins that are performed. This makes it hard for machine learning algorithms to detect distinguishing feature values and to differentiate between legitimate and fraudulent login activity, as confirmed by the performance of our chosen models. Future research should take into account the extreme degree of class imbalance that characterizes the context of credential stuffing, suggesting methods that can reduce it while still not neglecting the concept of class overlap. The problem could also be framed as an anomaly detection one or as a combination of both anomaly detection and binary classification.

Finally, our research was conducted on an online service located in Norway, with the vast majority of login attempts coming from either Norway or the US. Future research may expand this geographic boundary, considering online services with a more heterogeneous source of login attempts.

References

[1] McElroy, S. A. (2021). Learning from learning: detecting account takeovers by identifying forgetful users. *Computer Fraud & Security*, *2021*(6), 11-17.

[2] Zhang, Q. (2021). Detecting Credential Stuffing Between Servers. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage: SpaCCS 2020 International Workshops, Nanjing, China, December 18-20, 2020, Proceedings 13* (pp. 454-464). Springer International Publishing.

[3] Wang, K. C., & Reiter, M. K. (2020). Detecting stuffing of a {User's} credentials at her own accounts. In *29th USENIX Security Symposium (USENIX Security 20)* (pp. 2201-2218)

[4] Barkworth, A., Tabassum, R., & Habibi Lashkari, A. (2022, December). Detecting IMAP Credential Stuffing Bots Using Behavioural Biometrics. In *Proceedings of the 2022 12th International Conference on Communication and Network Security* (pp. 7-15).

[5] Jyothi, K. K., Borra, S. R., Srilakshmi, K., Balachandran, P. K., Reddy, G. P., Colak, I., ... & Khan, B. (2024). A novel optimized neural network model for cyber attack detection using enhanced whale optimization algorithm. *Scientific Reports*, *14*(1), 5590.

[6] Pal, B., Daniel, T., Chatterjee, R., & Ristenpart, T. (2019). Password similarity models using neural networks.

[7] Credential stuffing attack: countermeasures using patterns and machine learning, Sandeep Saxena, International Research Journal of Engineering and Technology, Volume: 09 Issue: 09 | Sep 2022.

[8] Wiefling, S., Jørgensen, P. R., Thunem, S., & Iacono, L. L. (2022). Pump up password security! Evaluating and enhancing risk-based authentication on a real-world large-scale online service. *ACM Transactions on Privacy and Security*, *26*(1), 1-36.

[9] Rees-Pullman, S. (2020). Is credential stuffing the new phishing?. *Computer Fraud & Security*, *2020*(7), 16-19.

[10] Chen, Chao & Breiman, Leo. (2004). Using Random Forest to Learn Imbalanced Data. University of California, Berkeley.

[11] Guo, H., & Viktor, H. L. (2004). Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *ACM Sigkdd Explorations Newsletter*, *6*(1), 30-39.

[12] Ganganwar, V. (2012). An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, *2*(4), 42-47.

[13] He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, *21*(9), 1263-1284.

[14] Bouguessa, M. (2012, December). Unsupervised Anomaly Detection in Transactional Data. In *2012 11th International Conference on Machine Learning and Applications* (Vol. 1, pp. 526-531). IEEE.

[15] Oreški, G., & Oreški, S. (2014, March). An experimental comparison of classification algorithm performances for highly imbalanced datasets. In *25th Central European Conference on Information and Intelligent Systems* (pp. 1-4).

[16] Kulkarni, Ajay & Chong, Deri & Batarseh, Feras. (2021). Foundations of data imbalance and solutions for a data democracy. (Chapter 5)

[17] Shukla, P., & Bhowmick, K. (2017, March). To improve classification of imbalanced datasets. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)* (pp. 1-5). IEEE.

[18] Liang, X., Gao, Y., & Xu, S. (2024). ASE: Anomaly scoring based ensemble learning for highly imbalanced datasets. *Expert Systems with Applications*, *238*, 122049.

[19] Ren, J., Wang, Y., Mao, M., & Cheung, Y. M. (2022). Equalization ensemble for large scale highly imbalanced data classification. *Knowledge-Based Systems*, *242*, 108295.

[20] Liu, Z., Cao, W., Gao, Z., Bian, J., Chen, H., Chang, Y., & Liu, T. Y. (2020, April). Self-paced ensemble for highly imbalanced massive data classification. In *2020 IEEE 36th international conference on data engineering (ICDE)* (pp. 841-852). IEEE.

[21] Martino, A., Rizzi, A., & Frattale Mascioli, F. M. (2018, July). Distance matrix pre-caching and distributed computation of internal validation indices in k-medoids clustering. In *2018 international joint conference on neural networks (IJCNN)* (pp. 1-8). IEEE.

[22] Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2), 224-227.

[23] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53-65.

[24] Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, *3*(1), 1-27.

[25] Lee, H. J., & Cho, S. (2006, October). The novelty detection approach for different degrees of class imbalance. In *International conference on neural information processing* (pp. 21-30). Berlin, Heidelberg: Springer Berlin Heidelberg.