An Isolation Forest - based approach for brute force attack detection

Olha Mykhaylova [1,†], Andriy Shtypka [1,†] and Taras Fedynyshyn[1,*,†]

[1] *Lviv Polytechnic National University, 12 Stepan Bandera str., Lviv, 79000, Ukraine*

Abstract

In today's rapidly evolving digital landscape, characterized by dynamic cyber threats, brute force attacks persist as among the most prevalent and enduring security challenges. In the face of evolving cyber-attacks, conventional detection techniques frequently demonstrate inadequacies, prompting the exploration of innovative solutions like the integration of machine learning algorithms. This paper introduces a novel intelligent model utilizing decision trees, designed to detect anomalies in user behavior indicative of potential brute force attacks. Additionally, this paper includes the Python source code for implementing the model as well as presenting the obtained results. The paper explores the effectiveness of an intelligent decision tree-based model in detecting brute force attacks during system logins. It delves into the algorithmic underpinnings of these models, highlights their advantages over traditional detection methods, and examines practical considerations for their implementation and utilization. The proposed Isolation Forest-based model effectively detects brute force attacks with high accuracy, adaptability, and reduced false positives, but it requires careful tuning of anomaly thresholds, faces limitations in highly imbalanced datasets where attack instances are rare, and may be vulnerable to sophisticated adversarial tactics that mimic normal behavior, highlighting the need for further improvements in its robustness and sensitivity.

Keywords

intrusion detection model, Isolation Forest, brute force, machine learning

1. Introduction

In the modern digital world, where cyber threats evolve at an incredible pace, brute force attacks continue to remain one of the most common and persistent threats. These attacks, which involve the continuous guessing of user credentials, pose a serious threat to the security of systems and data. However, traditional methods of detection often prove to be insufficiently effective in the changing landscape of cyber-attacks, opening the door to innovative approaches such as the application of machine learning algorithms.

BAIT'2024: The 1st International Workshop on "Bioinformatics and applied information technologies", October 02-04, 2024, Zboriv, Ukraine

[*] Corresponding author.

[†] These authors contributed equally.

olha.o.mykhailova@lpnu.ua (O. Mykhaylova); andriyko7788@gmail.com (A. Shtypka); fedynyshyn.taras@gmail.com (T. Fedynyshyn)

0000-0002-3086-3160 (O. Mykhaylova); 0009-0005-1419-9051 (A. Shtypka); 0009-0006-8233-8057 (T. Fedynyshyn)

One such approach is the development of intelligent models based on decision trees, capable of identifying anomalies in user behavior that may indicate attempts of brute force attacks. These models, able to analyze large volumes of data and promptly detect potential threats, represent a significant advancement in countering these attacks. They not only enhance the accuracy of attack detection but also reduce the number of false positives, optimizing the performance of security teams.

The intelligent model for detecting brute force attacks is based on decision-making using decision trees as a machine learning algorithm to classify network traffic as either normal or anomalous. The model applies the decision tree algorithm for training based on a labeled dataset of network traffic, which includes examples of both successful login attempts or unsuccessful attempts within acceptable norms, as well as anomalous unsuccessful attempts that may indicate malicious unauthorized access attempts. During training, the model identifies criteria by which brute force attacks are most likely to occur.

Upon completion of training, the model can be utilized for classifying new network traffic, such as login attempts into the system. If the model identifies a segment of traffic as malicious, it is likely that this traffic is part of a brute force attack. This information can serve as the basis for taking actions such as blocking the traffic or notifying the network administrator.

Decision trees represent a type of machine learning algorithm that is optimal for classification tasks. One of the advantages of using decision trees for detecting brute force attacks is their high effectiveness in recognizing attacks that are not yet known to the system, thanks to their ability to learn from new data and adapt to changes in the attack landscape. It is also important to note that decision trees can be easily interpreted, facilitating understanding of the decisions made by the model.

However, there are some considerations when using decision trees for detecting brute force attacks. One such consideration is that decision trees require continual training to function correctly. Additionally, decision trees need to be adapted individually for each project, considering its specific characteristics and security requirements.

To enhance the accuracy and effectiveness of the intelligent intrusion detection model for system login, it is crucial to conduct performance evaluation of brute force attack detection using decision trees. Effective evaluation of brute force attack detection models is vital for ensuring system security. Decision trees, with their capability to identify complex patterns in data, offer a promising approach for this task.

The following metrics are used to evaluate the effectiveness of decision tree models in the context of brute force attack detection:

- True Positive Rate (TPR), also known as recall, measures the proportion of actual brute force attacks that were correctly identified. A high TPR indicates that the model effectively detects true attacks.
- False Positive Rate (FPR) assesses the proportion of normal login attempts that are incorrectly classified as brute force attacks. A low FPR ensures that the model minimizes unnecessary alarms.
- Accuracy measures the proportion of login attempts labeled as brute force attacks that are actually true attacks. A high accuracy value indicates that the model is precise in its classifications.

In addition to these metrics, it is also important to evaluate the following characteristics of decision tree models:

- Detection time is a critical factor in real-time attack detection scenarios. Swift detection ensures that the system can promptly respond to an attack.
- Computational efficiency is also an important characteristic of decision tree models, especially when processing large volumes of data.
- Resistance to adversarial attacks is an important characteristic for any brute force attack detection model. Adversaries may employ various methods to evade detection, such as using unique login credentials or introducing subtle variations into their attack patterns.
- Interpretability of decision tree models is important for ensuring their credibility and interpretability. Interpretability methods can provide insight into the decision-making process of the model, allowing security analysts to verify its performance and identify potential biases.

Each performance evaluation metric corresponds to a specific task. For example, TPR is important to ensure that the model does not miss real attacks. FPR is important to ensure that the model does not generate too many false positives.

2. Related Works

In brute-force attacks, the assailant systematically submits every conceivable value as inputs for account credentials to gain unauthorized access to the system's account data. These attacks are characterized by two primary methodologies: dictionary attacks, which exhaustively test all entries in a predefined list, and random sequence methods, which systematically test all feasible string combinations in a sequential manner [1]. A prevalent strategy for mitigating brute-force attacks involves implementing access controls triggered by repeated incorrect password entries. For servers, this entails establishing a threshold for login failures via an account lock policy, which restricts access to the account once the threshold is surpassed [2].

This research paper delves into the creation of an Intrusion Detection System (IDS), specifically brute force attacks, with particular emphasis on the IDS's ability to withstand adversarial attacks and the reliability of explainable AI.

Over the past decade, numerous intrusion detection Systems have been developed to safeguard cyber networks against malicious attacks [3]. Notably, Machine Learning (ML)-based IDS have demonstrated outstanding performance owing to their capacity to learn vast numbers of parameters. However, these sophisticated models, often termed black-box models, lack interpretability [4], which contradicts the essential need for transparent decision-making in IDS operations. Given that even a single erroneous prediction by an IDS can expose the system to significant cyber threats, the integration of eXplainable Artificial Intelligence (XAI) is imperative in traditional IDS frameworks to enhance credibility and reliability.

Mane et al. [5] employed the NSL-KDD dataset alongside a Deep Neural Network (DNN)-based Machine Learning (ML) model for network intrusion detection. In an effort to enhance transparency, they employed five distinct XAI frameworks to illustrate the behavior of the trained model. Nevertheless, they did not leverage the explanations provided by any XAI framework to validate the credibility of the predicted outcomes.

Mahbooba et al. [6], similarly, addressed the explanation of individual predicted outcomes by deriving rules from the decision tree trained and assessed on the KDD dataset. These rules were exclusively employed to clarify each predicted outcome and the overall model response. However, their focus did not extend to adversarial attacks or enhancing Intrusion Detection Systems (IDS) using explanations from XAI tools

Fidel et al. [7] introduced a framework based on XAI signatures to distinguish between adversarial samples and normal network traffic. They assessed their approach using datasets typically employed in image recognition tasks and achieved an accuracy of approximately 97% in detecting adversarial attacks. Such defense mechanisms are essential in cyber networks. Hence, this paper suggests a novel intrusion detection approach aimed at verifying the credibility of machine learning model predictions and ensuring superior performance in both normal and adversarial scenarios. Moreover, it enhances transparency in the decision-making process, thereby bolstering user trust.

3. Proposed model

Isolation Forest is an innovative unsupervised learning method specifically designed for anomaly detection in data. A key feature of this method is its ability to effectively detect anomalies without the need for prior definition or labeling of normal data. Isolation Forest utilizes an ensemble of randomized decision trees to "isolate" observations, whereby anomalous data points are typically isolated in fewer steps than normal observations.

The key idea of the algorithm lies in the fact that anomalies typically represent a minority in the dataset and exhibit differences from normal observations, allowing them to be isolated more quickly. The process of partitioning the data is performed recursively until each instance is isolated in its own "leaf" of the tree. The number of splits required to isolate an instance is used as a measure of its anomaly score.

Isolation Forest demonstrates high effectiveness in various applications, including:

- Fraud detection: Identifying anomalous transactions or user behavior that may indicate financial fraud.
- Intrusion detection systems: Detecting anomalous patterns in network traffic that may indicate unauthorized access attempts.
- Recommendation systems: Identifying non-standard user behavior or anomalous purchasing patterns that may impact recommendation algorithms.
- Detection of anomalies in large datasets: Identifying anomalous records in large datasets that may indicate data errors, external interference, or other unforeseen phenomena.

The underlying principle of the Isolation Forest algorithm is that anomalous data points are more readily distinguishable from the rest of the dataset. To isolate a data point, the algorithm iteratively creates partitions within the dataset by randomly choosing an attribute and subsequently selecting a split value at random between the minimum and maximum values permitted for that attribute.

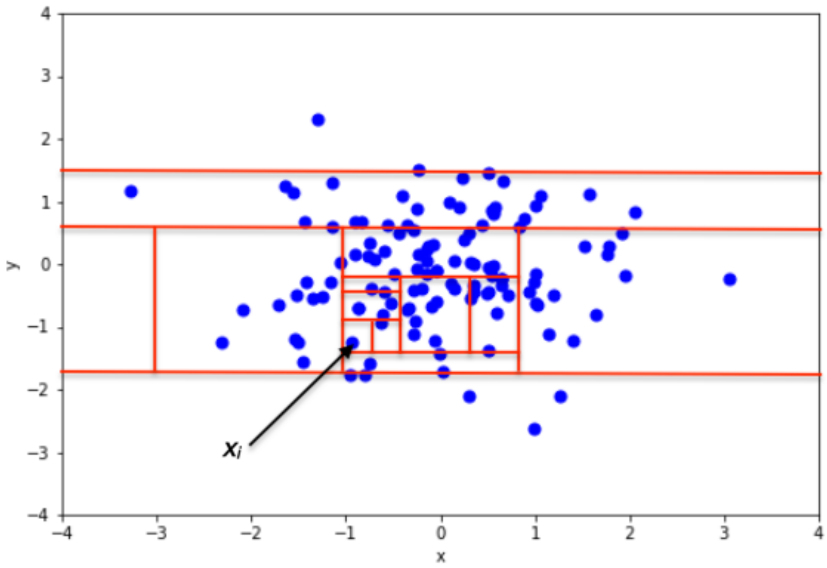


Figure 1: An example of isolating an anomalous point in a 2D Gaussian distribution [8].

Figure 1 illustrates random partitioning in a 2D dataset of normally distributed points for a non-anomalous point, while Figure 2 depicts a point that is more likely to be an anomaly. It is evident from the visuals that anomalies necessitate fewer random partitions for isolation compared to normal points.

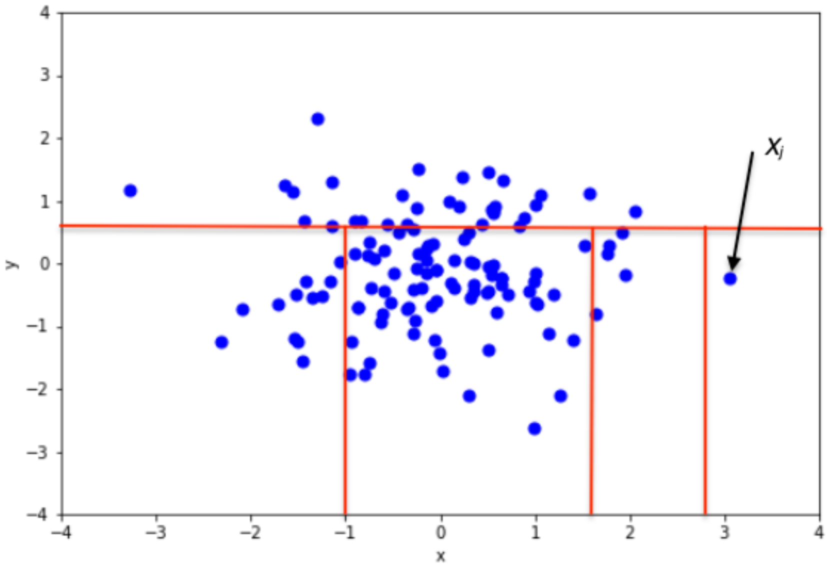


Figure 2: An example of isolating a non-anomalous point in a 2D Gaussian distribution [8].

The process of recursive partitioning can be visualized through a tree structure called the Isolation Tree. The number of partitions needed to isolate a point can be understood as the length of the path within this tree, starting from the root and terminating at a leaf node. For instance, in Figure 1, the path length of point x_i is longer than that of x_j in Figure 2.

Let $X = \{x_1, ..., x_n\}$ be [8] a set of d-dimensional points and $X' \subset X$. An Isolation Tree (iTree) is defined as a data structure with the following properties:

1. for each node T in the Tree, T is either an external-node with no child, or an internal-node with one "test" and exactly two child nodes (T_l and T_r).
2. a test at node T consists of an attribute q and a split value p such that the test $q < p$ determines the traversal of a data point to either T_l or T_r.

To build an iTree, the algorithm recursively divides X' by randomly selecting an attribute q and a split value p, until either:

1. the node has only one instance, or
2. all data at the node have the same values.

When the iTree is fully grown, each point in X is isolated at one of the external nodes. Intuitively, the anomalous points are those (easier to isolate, hence) with the smaller path length in the tree, where the path length $h(x_i)$ of point $x_i \in X$ is defined as the number of edges x_i traverses from the root node to get to an external node.

To detect anomalies using IForest, two main steps need to be executed:

1. Training the model. The model is built by taking a partial sample of the dataset.
2. Testing the model. The test dataset is fed into the model to compute anomaly scores for each point.

The diagram in Figure 3 illustrates the algorithmic steps of applying the IForest model.

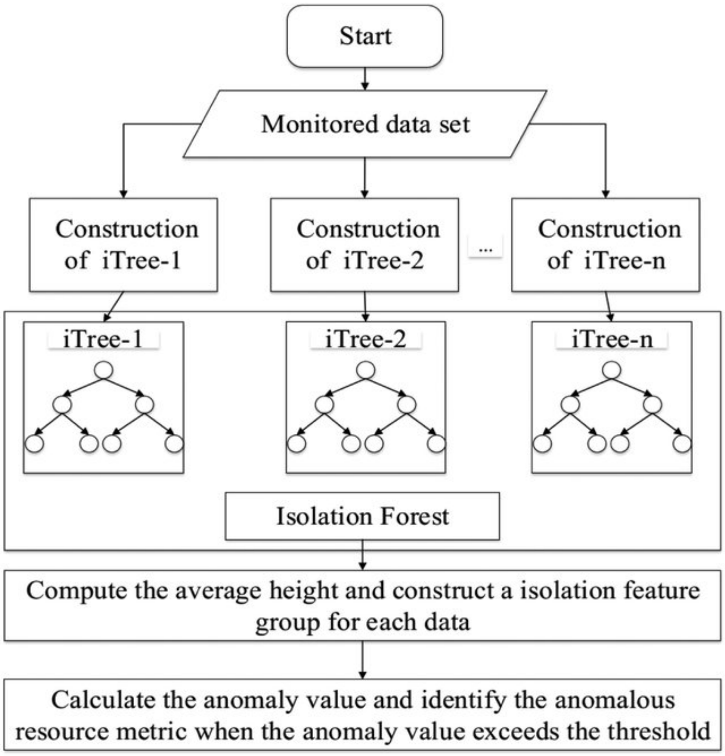

Start
Monitored data set
Construction of iTree-1
Construction of iTree-2
...
Construction of iTree-n
iTree-1
iTree-2
iTree-n
Isolation Forest
Compute the average height and construct a isolation feature group for each data
Calculate the anomaly value and identify the anomalous resource metric when the anomaly value exceeds the threshold

Figure 3: Isolation forest construction process [9].

The IForest algorithm can be used to detect brute force attacks on a login system. If an attacker attempts to guess the password, they will make many incorrect login attempts. The IForest algorithm can detect and label such attempts as anomalies. The process of applying the Isolation Forest algorithm for a login system attack detection model consists of several steps.

The first step is to gather data about login attempts to the system. This data may include characteristics such as:

- IP address
- Username
- Password
- Date

The second step is to create an IForest model. This can be done using the Isolation Forest library from the scikit-learn package.

The third step is to train the IForest model on the gathered data. This process may take some time depending on the size of the dataset.

After training the IForest model, it's necessary to determine the anomaly threshold. This threshold value determines which login attempts are considered anomalous.

The fifth step is to use the IForest model to assess anomalies for new login attempts. If a login attempt has an anomaly score lower than the anomaly threshold, it is considered anomalous.

Brute force attacks on login systems typically have a short path to the root of the tree. This is because the attacker attempting to crack the password will try many incorrect passwords. If a login attempt has an anomaly score below the anomaly threshold, it is considered anomalous. This login attempt can be classified as a brute force attack. The anomaly threshold can be optimized to achieve an optimal balance between sensitivity and specificity. Sensitivity is the probability that the model correctly classifies an anomalous login attempt as an anomaly. Specificity is the probability that the model correctly classifies normal login attempts as normal. A too low anomaly threshold will result in high sensitivity but also a high number of false positive results. A too high anomaly threshold will result in high specificity but also a high number of false negative results.

IForest is an effective method for detecting brute force attacks on login systems. It has several advantages, such as:

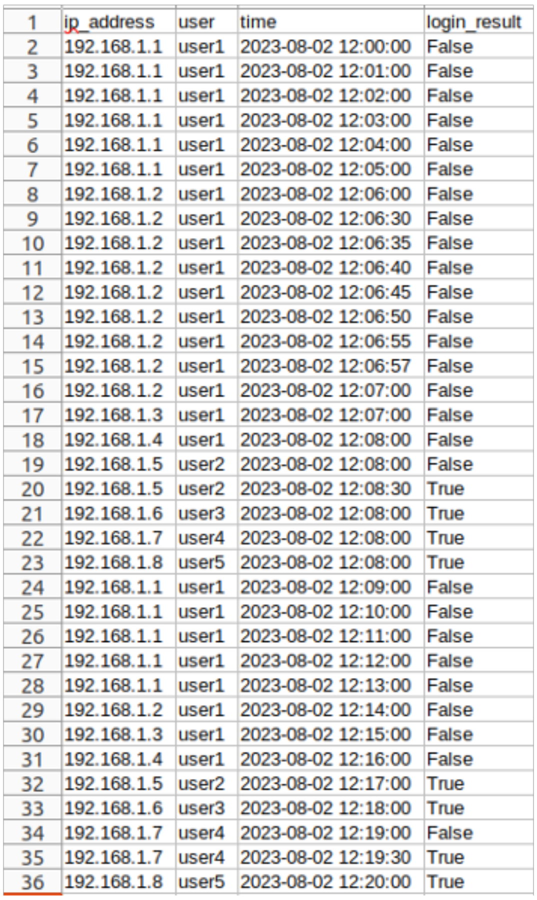
- Speed
- Accuracy
- Robustness to noise
- Adaptability to changes

4. Results

In this section, the Python code implementation is demonstrated to create an intelligent intrusion detection system (for brute force attacks) on the login system. After displaying each written part of the code, the execution result and its description is provided.

The working principle of the intelligent system model involves detecting unsuccessful login attempts to specific accounts within a system or software application based on login history and providing information about each anomaly (users and anomalous IP addresses). This enables the individual or the monitoring model itself to take appropriate measures to ensure integrity, confidentiality, and availability.

Before writing the code, we need to first obtain and save the login data into a CSV file for training and applying the model. The CSV file should have 4 columns (IP-address, user, time, login result), containing data about the IP addresses from which login attempts were made, the users whose accounts were attempted to be logged into, the time, and the login result, as shown in Figure 4.



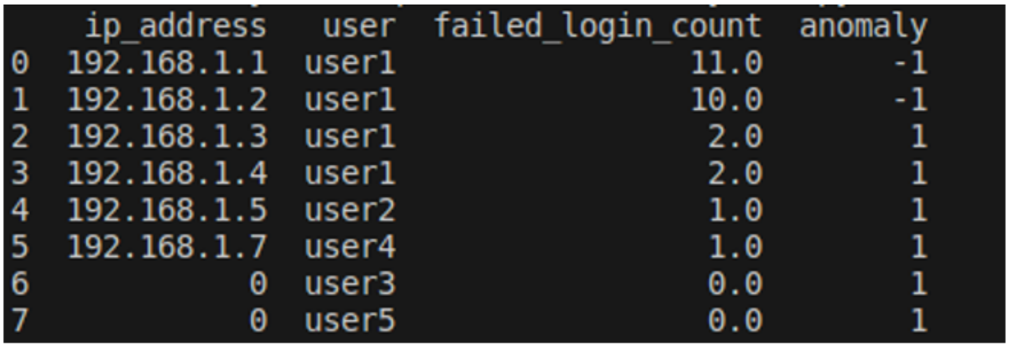
1	ip_address	user	time	login_result
2	192.168.1.1	user1	2023-08-02 12:00:00	False
3	192.168.1.1	user1	2023-08-02 12:01:00	False
4	192.168.1.1	user1	2023-08-02 12:02:00	False
5	192.168.1.1	user1	2023-08-02 12:03:00	False
6	192.168.1.1	user1	2023-08-02 12:04:00	False
7	192.168.1.1	user1	2023-08-02 12:05:00	False
8	192.168.1.2	user1	2023-08-02 12:06:00	False
9	192.168.1.2	user1	2023-08-02 12:06:30	False
10	192.168.1.2	user1	2023-08-02 12:06:35	False
11	192.168.1.2	user1	2023-08-02 12:06:40	False
12	192.168.1.2	user1	2023-08-02 12:06:45	False
13	192.168.1.2	user1	2023-08-02 12:06:50	False
14	192.168.1.2	user1	2023-08-02 12:06:55	False
15	192.168.1.2	user1	2023-08-02 12:06:57	False
16	192.168.1.2	user1	2023-08-02 12:07:00	False
17	192.168.1.3	user1	2023-08-02 12:07:00	False
18	192.168.1.4	user1	2023-08-02 12:08:00	False
19	192.168.1.5	user2	2023-08-02 12:08:00	False
20	192.168.1.5	user2	2023-08-02 12:08:30	True
21	192.168.1.6	user3	2023-08-02 12:08:00	True
22	192.168.1.7	user4	2023-08-02 12:08:00	True
23	192.168.1.8	user5	2023-08-02 12:08:00	True
24	192.168.1.1	user1	2023-08-02 12:09:00	False
25	192.168.1.1	user1	2023-08-02 12:10:00	False
26	192.168.1.1	user1	2023-08-02 12:11:00	False
27	192.168.1.1	user1	2023-08-02 12:12:00	False
28	192.168.1.1	user1	2023-08-02 12:13:00	False
29	192.168.1.2	user1	2023-08-02 12:14:00	False
30	192.168.1.3	user1	2023-08-02 12:15:00	False
31	192.168.1.4	user1	2023-08-02 12:16:00	False
32	192.168.1.5	user2	2023-08-02 12:17:00	True
33	192.168.1.6	user3	2023-08-02 12:18:00	True
34	192.168.1.7	user4	2023-08-02 12:19:00	False
35	192.168.1.7	user4	2023-08-02 12:19:30	True
36	192.168.1.8	user5	2023-08-02 12:20:00	True

Figure 4: Implemented model input file.

The next step is to install and utilize the Scikit-learn library along with Isolation Forest for training and application, Pandas for file operations, NumPy for numerical functions, and Matplotlib for creating plots. See Appendix A for references [10, 11, 12].

To filter the data, we create two DataFrames, failed logins and success logins, where we store the data and count the number of failed and successful login attempts, respectively. Then, we merge them into one DataFrame, grouped data, where we store data about users, IP addresses, and the count of failed and successful logins.

To train the anomaly detection model for the login system, we need to utilize the grouped data DataFrame containing information about the count of failed and successful login attempts. For this purpose, we create our model (model) and specify parameters (contamination='auto', which denotes the percentage of anomalous points from the total by isolating normal points from anomalies, random state=42, defining the initial state of the random number generator). Then, based on the count of failed and successful attempts for each IP address, the model predicts whether there is anomaly, labeling each row accordingly: -1 if an anomaly is detected and 1 if not, as showed on Figure 5.



```
    ip_address   user  failed_login_count  anomaly
0  192.168.1.1  user1                11.0       -1
1  192.168.1.2  user1                10.0       -1
2  192.168.1.3  user1                 2.0        1
3  192.168.1.4  user1                 2.0        1
4  192.168.1.5  user2                 1.0        1
5  192.168.1.7  user4                 1.0        1
6            0  user3                 0.0        1
7            0  user5                 0.0        1
```

Figure 5: Trained model output.

To visually represent the model's output, we need to use the Matplotlib library. The plot is constructed based on information about failed attempts and is displayed corresponding to each user from the "Logs.csv" file. The visualized results presented on Figure 6.

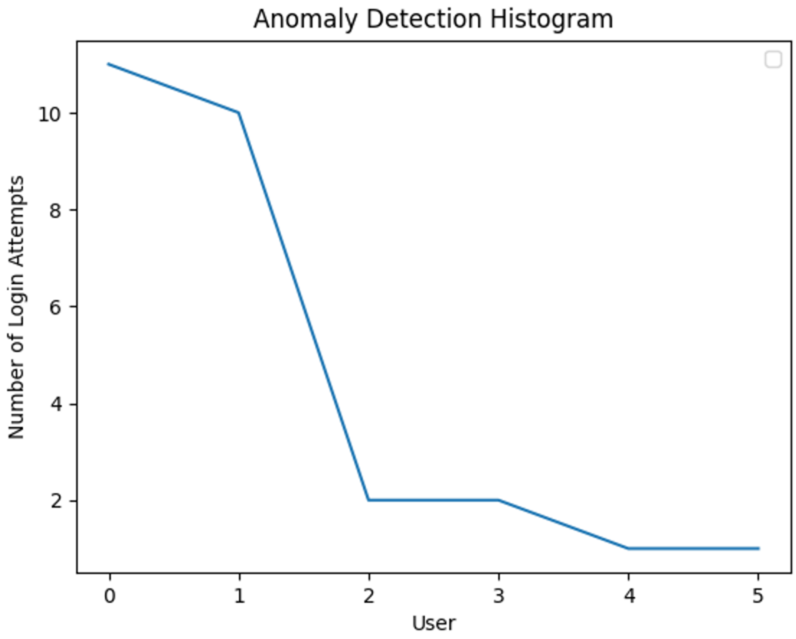


Figure 5: The resulting visualization of failed login attempts per user.

To obtain information about suspicious IP addresses from which a specific user experienced an anomalous number of login attempts, you need to filter IP addresses from the DataFrame “grouped_data” based on the "anomaly" index and present it as a message.

The development of an intelligent intrusion detection system for identifying brute force attacks on login systems demonstrates significant potential in ensuring digital security. The application of the Isolation Forest algorithm in the context of unsupervised learning enables effective identification of anomalies in large datasets, particularly in login histories.

5. Discussion

The proposed Isolation Forest-based approach effectively detects brute force attacks with high accuracy and reduced false positives compared to traditional methods. Its ability to isolate anomalous login attempts in real-time enhances its utility in large-scale systems, offering both speed and adaptability as it learns from evolving data patterns. One of the model’s major strengths is its unsupervised nature, allowing it to identify anomalies without the need for labeled datasets, making it suitable for dynamic environments. However, the model’s performance heavily depends on setting an optimal anomaly threshold, which requires fine-tuning to balance sensitivity and specificity. A key limitation is that in cases of highly imbalanced datasets, where brute force attacks are rare, the model may struggle with sensitivity, potentially missing some attacks. Additionally, the model can be vulnerable to adversarial attacks that are designed to mimic normal login behavior, reducing its effectiveness in more sophisticated threat scenarios. Despite these limitations, the Isolation Forest method shows promise as a scalable and efficient solution for brute force detection. Future work could explore hybrid approaches or enhancements to further improve the model’s resilience against adversarial tactics and improve its performance in imbalanced data sets.

6. Conclusion

The developed algorithm demonstrates high effectiveness in anomaly detection, especially in contexts where anomalies are relatively rare compared to the overall number of observations. Its ability to quickly isolate anomalous data points makes it an ideal tool for detecting brute force attack attempts.

Usage of this method may automate the process of identifying suspicious login attempts, reducing the need for constant monitoring by security analysts and increasing the speed of response to potential threats. This approach can be utilized on web and mobile application services and may be highly valuable if applied on critical infrastructure computer systems [13]. The unsupervised learning usage helps reduce the number of false positives, as the model adapts to changes in data behavior, ensuring more accurate detection of real anomalies.

The developed method seamlessly integrates with existing login systems and databases, enabling the quick implementation of an additional layer of security without the need for significant modifications to the existing infrastructure. It also provides information about users and IP addresses associated with anomalous access attempts, which can be used for further analysis and the development of countermeasures.

References

[1] Park, J., Kim, J., Gupta, B.B., Park, N. (2021). Network log-based SSH brute-force attack detection model. Computers, Materials & Continua, 68(1), 887-901. https://doi.org/10.32604/cmc.2021.015172.

[2] P. S. Abril, R. Plant, The patent holder's dilemma: Buy, sell, or troll?, Communications of the ACM 50 (2007) 36–44. https://doi.org/10.1145/1188913.1188915.

[3] Abid Salih, A. and Abdulazeez, A.M. (2021) "Evaluation of Classification Algorithms for Intrusion Detection System: A Review", Journal of Soft Computing and Data Mining, 2(1), pp. 31–40. Available at: https://publisher.uthm.edu.my/ojs/index.php/jscdm/article/view/7982 (Accessed: 29 April 2024).

[4] Rai, A. Explainable AI: from black box to glass box. J. of the Acad. Mark. Sci. 48, 137–141 (2020). https://doi.org/10.1007/s11747-019-00710-5.

[5] S. Mane and D. Rao, "Explaining network intrusion detection system using explainable ai framework," arXiv preprint arXiv: 2103.07110 , 2021. https://doi.org/10.48550/arXiv.2103.07110.

[6] Mahbooba, Basim & Timilsina, Mohan & Sahal, Radhya & Serrano, Martin. (2021). Explainable Artificial Intelligence (XAI) to Enhance Trust Management in Intrusion Detection Systems Using Decision Tree Model. Complexity. 2021. 11. https://doi.org/10.1155/2021/6634811.

[7] G. Fidel, R. Bitton, and A. Shabtai, "When explainability meets adversarial learning: detecting adversarial examples using shap signatures," in Proceedings of 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-8.

[8] Isolation forest, Available at: https://en.wikipedia.org/wiki/Isolation_forest.

[9] Zou, Zhuping & Xie, Yulai & Huang, Kai & Xu, Gongming & Feng, Dan & Long, Darrell. (2019). A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest. IEEE Transactions on Cloud Computing. PP. 1-1.

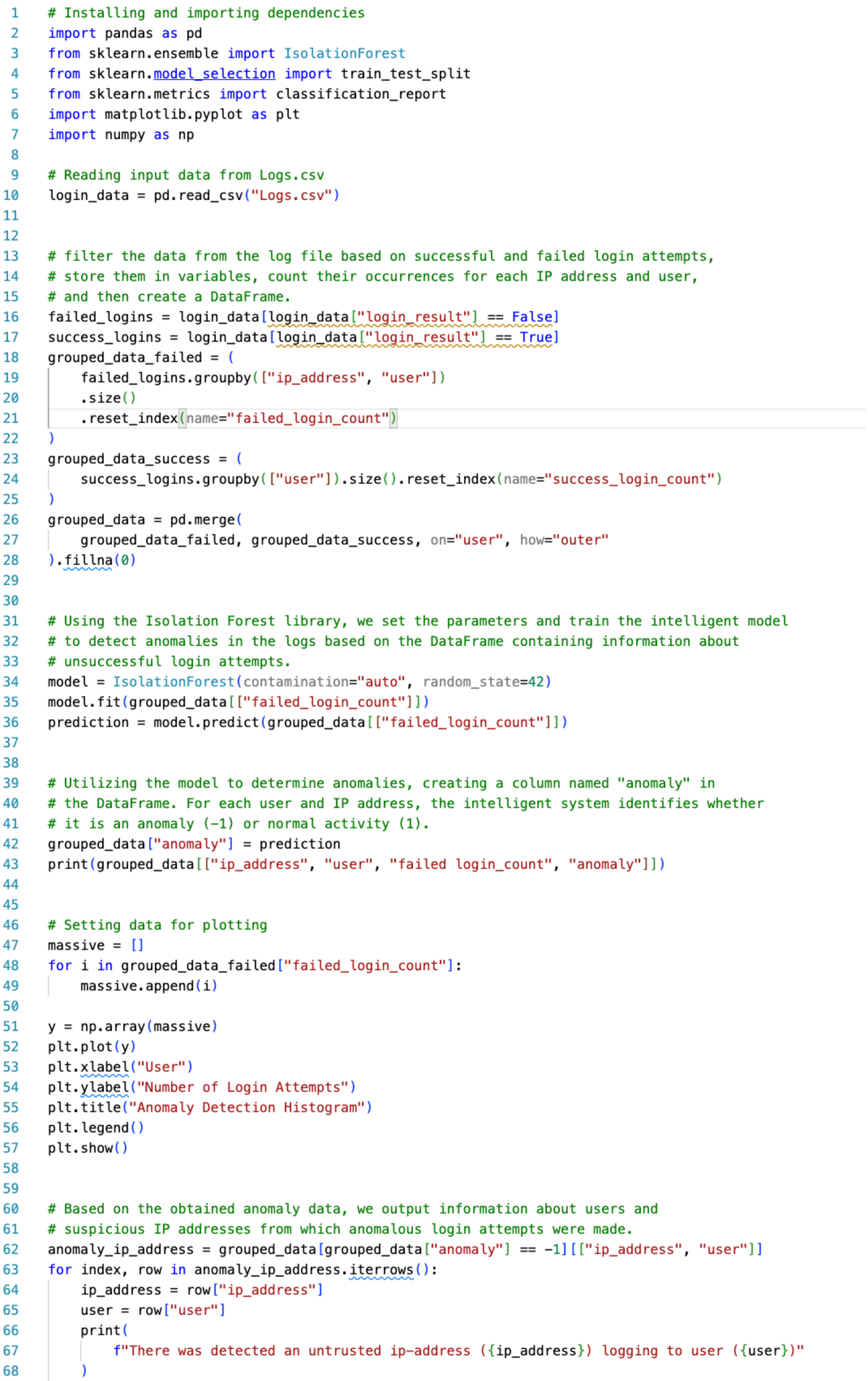
[10] Python documentation, Available at: https://www.python.org/doc/ (Accessed: 11 April 2024).

[11] Scikit-learn Isolation Forest documentation. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[12] Matplotlib 3.8.4 documentation. Available at: https://matplotlib.org/stable/index.html (Accessed: 11 April 2024).

[13] Olha Mykhaylova, Taras Fedynyshyn, Anastasiia Datsiuk, Bohdan Fihol, Hennadii Hulak: Mobile Application as a Critical Infrastructure Cyberattack Surface. Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II co-located with International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023, CEUR-WS.org/Vol-3550, urn:nbn:de:0074-3550-0.

A. Algorithm implementation python code



```python
# Installing and importing dependencies
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import numpy as np

# Reading input data from Logs.csv
login_data = pd.read_csv("Logs.csv")


# filter the data from the log file based on successful and failed login attempts,
# store them in variables, count their occurrences for each IP address and user,
# and then create a DataFrame.
failed_logins = login_data[login_data["login_result"] == False]
success_logins = login_data[login_data["login_result"] == True]
grouped_data_failed = (
    failed_logins.groupby(["ip_address", "user"])
    .size()
    .reset_index(name="failed_login_count")
)
grouped_data_success = (
    success_logins.groupby(["user"]).size().reset_index(name="success_login_count")
)
grouped_data = pd.merge(
    grouped_data_failed, grouped_data_success, on="user", how="outer"
).fillna(0)


# Using the Isolation Forest library, we set the parameters and train the intelligent model
# to detect anomalies in the logs based on the DataFrame containing information about
# unsuccessful login attempts.
model = IsolationForest(contamination="auto", random_state=42)
model.fit(grouped_data[["failed_login_count"]])
prediction = model.predict(grouped_data[["failed_login_count"]])


# Utilizing the model to determine anomalies, creating a column named "anomaly" in
# the DataFrame. For each user and IP address, the intelligent system identifies whether
# it is an anomaly (-1) or normal activity (1).
grouped_data["anomaly"] = prediction
print(grouped_data[["ip_address", "user", "failed login_count", "anomaly"]])


# Setting data for plotting
massive = []
for i in grouped_data_failed["failed_login_count"]:
    massive.append(i)

y = np.array(massive)
plt.plot(y)
plt.xlabel("User")
plt.ylabel("Number of Login Attempts")
plt.title("Anomaly Detection Histogram")
plt.legend()
plt.show()


# Based on the obtained anomaly data, we output information about users and
# suspicious IP addresses from which anomalous login attempts were made.
anomaly_ip_address = grouped_data[grouped_data["anomaly"] == -1][["ip_address", "user"]]
for index, row in anomaly_ip_address.iterrows():
    ip_address = row["ip_address"]
    user = row["user"]
    print(
        f"There was detected an untrusted ip-address ({ip_address}) logging to user ({user})"
    )
```