URL Security Assessment Using Machine Learning

Sri Vijaya K[1] , Shravani V[2], Nikitha V[3], Chaitanya M[4], Mohitha M[5]

Department of Information Technology[1,2,3,4]
PVP Siddhartha Institute of Technology[1,2,3,4]

---***---

Abstract - The significance of the World Wide Web has expanded considerably over time. However, alongside technological progress, there has been a rise in the complexity of methods aimed at exploiting users. These efforts often involve infecting users' computers with malware or directing them to unfriendly websites for purposes such as peddling counterfeit goods or exposing sensitive information, leading to financial fraud. Malicious URLs pose a significant threat to potential victims by hosting a range of unwanted content. Therefore, there is a pressing need for a rapid and effective detection approach. This thesis addresses the challenge of identifying hazardous URLs by leveraging URL data and machine learning techniques. Various machine learning algorithms, including XGBoost, LightGBM, and Random Forest, will be utilized for this purpose.

***Key Words*:** Malicious URLs, XGBoost, LightGBM, Random Forest.

1. INTRODUCTION

A malicious URL refers to a web link designed to propagate viruses, phishing scams, and other illicit activities. Clicking on such a link can result in the download of various computer viruses, including trojan horses, ransomware, worms, and malware. The ultimate objectives of these infections often include accessing personal data, compromising the user's device, and generating illicit revenue. Moreover, they can disrupt business networks, leading to financial losses. Additionally, malicious URLs may lure users into entering their personal information on fraudulent websites, where individuals are coerced into disclosing sensitive data to strangers who exploit it for clandestine purposes. The repercussions of these rogue URLs can be substantial, causing significant harm.

1.2 EXISTING SYSTEM

The inability to maintain an exhaustive list of all possible malicious URLs, as new URLs can be easily generated daily, thus making it impossible for them to detect new threats.

1.2.1 DISADVANTAGES OF EXISTING SYSTEM

The following are the disadvantages of existing system:

- URL blacklisting is ineffective for new malicious URLs.
- It takes time to analyse malicious URLs and propagate a blacklist to end users.
- Suffers from nontrivial high false negatives.
- Blacklist features alone do not have as good performance as other features.

2. PROPOSED SYSTEM

The proposed system employs approaches that involve extracting valuable feature representations from URLs and training a predictive model using training data comprising both harmful and benign URLs. These methods aim to analyze the information encapsulated within a URL and its associated websites or webpages. Both static and dynamic features can be utilized in this process. Static analysis entails examining the available information without actually executing the URL. This includes extracting lexical features from the URL string, host-related data, and in some cases, even scrutinizing HTML and JavaScript content. Static
analysis techniques are considered safer than dynamic ones as they do not require execution. The underlying assumption is that malicious and benign URLs exhibit distinct distributions of these properties, which can be leveraged to develop a predictive model capable of forecasting the nature of new URLs. Due to the relatively secure environment for gathering crucial information and the ability to generalize across various threats, machine learning techniques have been extensively explored in conjunction with static analysis methods.

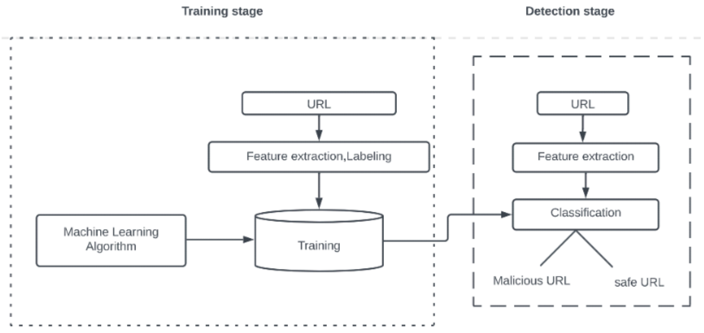


FIG 1.1 Proposed system model

2.1 Dataset Description

Our team has assembled a substantial dataset comprising a total of 651,191 URLs. Among these, there are 428,103 URLs classified as benign or safe, 96,457 as defacement URLs, 94,111 as phishing URLs, and 32,520 as malware URLs. As is widely recognized, selecting an appropriate dataset is a critical aspect of any machine learning project. This dataset was meticulously curated from five distinct sources. The ISCX-URL-2016 dataset was utilized to collect benign, phishing, malware, and defacement URLs. Additionally, leveraging information from the Malware Domain Blacklist facilitated the identification of phishing and malware URLs. Furthermore, we augmented the number of benign websites by incorporating data from the Faizan Git repository. To enrich the dataset with additional phishing URLs, we integrated datasets from Phishtank and PhishStorm. As indicated, multiple sources were amalgamated to obtain the dataset. To ensure consistency, we initially compiled the URLs and their corresponding class types into a separate dataframe.

- **Random Forest :**

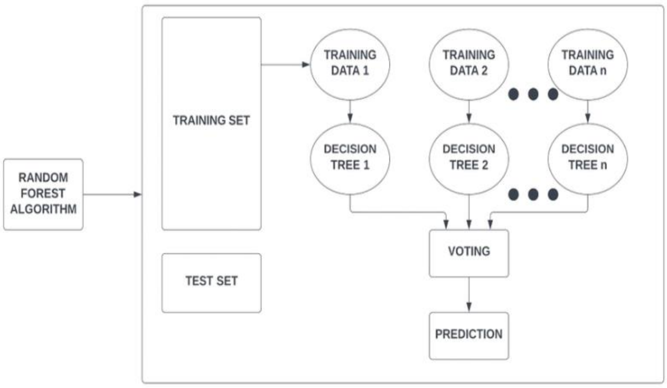


Fig 1.2 Working principle of Random forest classifier

The above diagram depicts the operation of a random forest classifier. The training dataset is divided into n subsets, each of which contains n decision trees that predict an output. The final output is predicted by voting on the output of each decision tree[3].

- **XG Boost :**

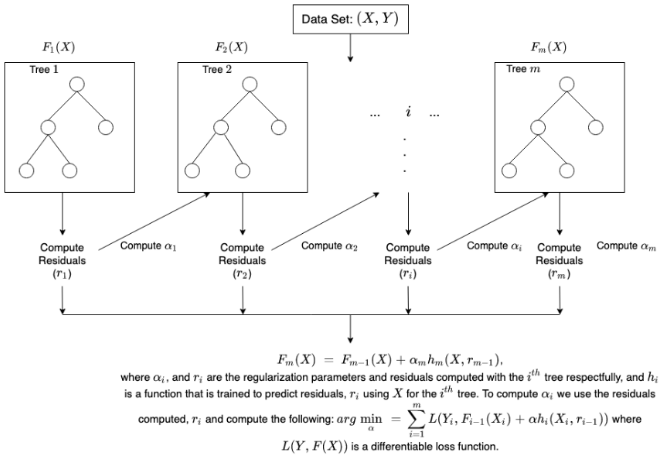

Data Set: (X, Y)
$F_1(X)$
$F_2(X)$
$F_m(X)$
Tree 1
Tree 2
Tree m
... i ...
Compute Residuals (r_1)
Compute α_1
Compute Residuals (r_2)
Compute α_2
Compute Residuals (r_i)
Compute α_i
Compute Residuals (r_m)
Compute α_m
$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1})$,
where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

FIG 1.3 Working principle of XGBoost classifier

The above figure depicts the working principle of the xgboost classifier. It is a popular and efficient open-source implementation of the gradient-boosted trees algorithm [4].

- **Light GBM classifier :**

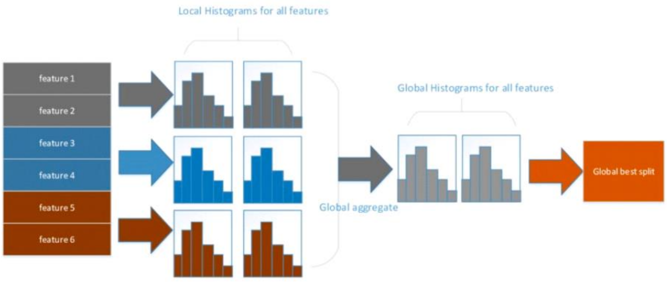


FIG 1.3 Working principle of XGBoost classifier

The above diagram describes the working principle of the LightGBM classifier. LightGBM is a gradient-boosting framework based on decision trees [5].

3. TECHNOLOGIES USED

3.1 Jupyter Notebook

Jupyter Notebook serves as the foundational platform for creating and disseminating computational documents. Renowned as the original web application designed for this purpose, Jupyter Notebook offers a user-friendly and document-centric interface. It facilitates a seamless and interactive experience, making it an indispensable tool for the development and presentation of data science projects.

3.2 Python Libraries

3.2.1 NumPy

NumPy facilitates a broad range of mathematical operations on arrays, enhancing Python with efficient array and matrix computations. It boasts an extensive library of high-level mathematical functions tailored for array and matrix manipulation.

3.2.2 Pandas

Pandas serves as a versatile Python library tailored for working with datasets. Its functionalities encompass data analysis, cleaning, exploration, and manipulation, making it indispensable for data-centric tasks.

3.2.3 Scikit-learn (Sklearn)

Scikit-learn, or Sklearn, stands out as a highly valuable and robust machine learning library in Python. Offering a diverse array of tools, it empowers users with efficient solutions for various machine learning and statistical modeling tasks, including classification, regression, clustering, and dimensionality reduction, all via a consistent Python interface.

3.2.4 Itertools

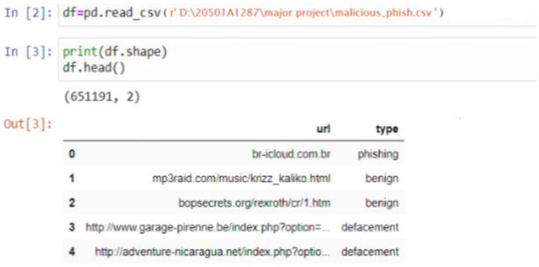
Itertools, a Python module, facilitates iteration over data structures that can be traversed using a for-loop. Known as iterators, these data structures are efficiently managed by functions within the Itertools module, making it an invaluable resource for computational tasks.

3.2.5 Matplotlib

Matplotlib is an extensive Python library renowned for generating static, animated, and interactive visualizations. It simplifies the creation of publication-quality plots and enables the development of interactive figures with features like zooming, panning, and real-time updates

4. RESULTS

4.1 Loading Database:



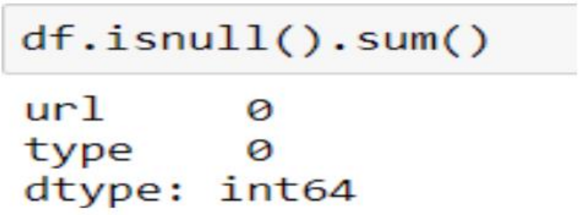
```
In [2]: df=pd.read_csv(r'D:\20501A1287\major project\malicious_phish.csv')

In [3]: print(df.shape)
        df.head()

        (651191, 2)

Out[3]:
```

	url	type
0	br-icloud.com.br	phishing
1	mp3raid.com/music/krizz_kaliko.html	benign
2	bopsecrets.org/rexroth/cr/1.htm	benign
3	http://www.garage-pirenne.be/index.php?option=...	defacement
4	http://adventure-nicaragua.net/index.php?optio...	defacement

FIG 1.5 First five rows of the dataset

The above figure gives an overview of the columns of the dataset that is used in the detection of malicious URLs.

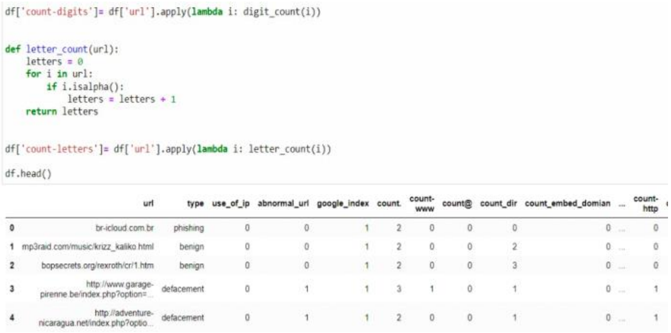
4.2 Data Preprocessing



```
df.isnull().sum()

url      0
type     0
dtype: int64
```

The above figure shows that the data has no more null and missing values. The data is cleaned using the Python library pandas

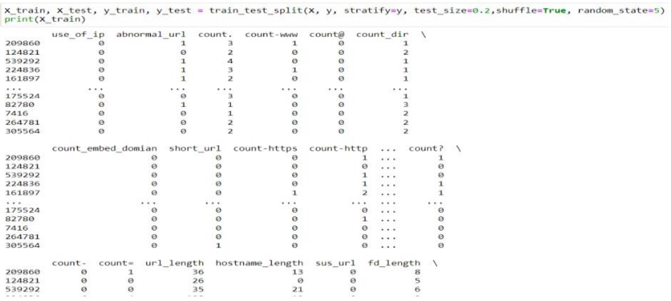
4.3 Extracting lexical features



```
df['count-digits']= df['url'].apply(lambda i: digit_count(i))


def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters


df['count-letters']= df['url'].apply(lambda i: letter_count(i))

df.head()
```

	url	type	use_of_ip	abnormal_url	google_index	count.	count-www	count@	count_dir	count_embed_domian	...	count-http
0	br-icloud.com.br	phishing	0	0	1	2	0	0	0	0	...	0
1	mp3raid.com/music/krizz_kaliko.html	benign	0	0	1	2	0	0	2	0	...	0
2	bopsecrets.org/rexroth/cr/1.htm	benign	0	0	1	2	0	0	3	0	...	0
3	http://www.garage-pirenne.be/index.php?option=...	defacement	0	1	1	3	1	0	1	0	...	1
4	http://adventure-nicaragua.net/index.php?optio...	defacement	0	1	1	2	0	0	1	0	...	1

4.4 Splitting the dataset

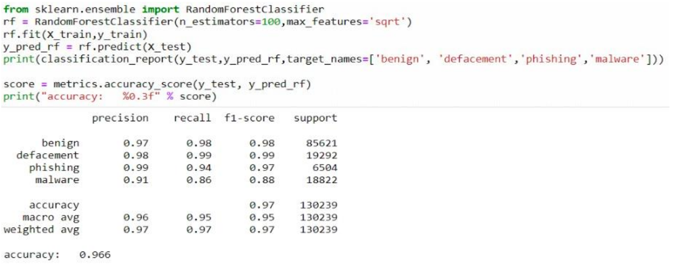


```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle=True, random_state=5)
print(X_train)
```

```
        use_of_ip  abnormal_url  count.  count-www  count@  count_dir  \
209860          0             1       3          1       0          1
124821          0             0       2          0       0          2
539292          0             1       4          0       0          1
224836          0             1       3          1       0          1
161897          0             1       2          0       0          1
...           ...           ...     ...        ...     ...        ...
175524          0             0       3          0       0          1
82780           0             1       1          0       0          3
7416            0             0       1          0       0          2
264781          0             0       2          0       0          2
305564          0             0       2          0       0          2

        count_embed_domian  short_url  count-https  count-http  ...  count?  \
209860                   0          0            0           1  ...       1
124821                   0          0            0           0  ...       0
539292                   0          0            0           1  ...       0
224836                   0          0            0           1  ...       1
161897                   0          0            1           2  ...       1
...                    ...        ...          ...         ...  ...     ...
175524                   0          0            0           0  ...       0
82780                    0          0            0           1  ...       0
7416                     0          0            0           0  ...       0
264781                   0          0            0           0  ...       0
305564                   0          1            0           0  ...       0

        count-  count=  url_length  hostname_length  sus_url  fd_length  \
209860       0       1          36               13        0          8
124821       0       0          26                0        0          5
539292       0       0          35               21        0          6
```

The above figure describes that the dataset is split into the training dataset and testing dataset by using the train_test_split which is imported from the sklearn. The training data is 80% and the testing data is 20% [2].

4.5 Training the model



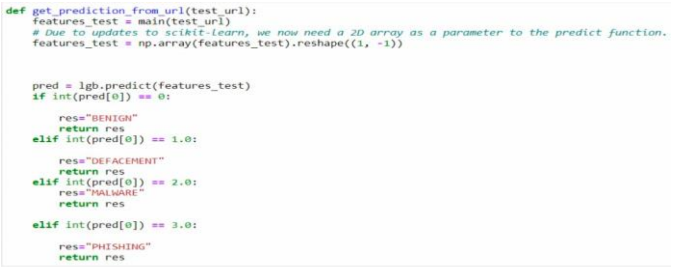
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100,max_features='sqrt')
rf.fit(X_train,y_train)
y_pred_rf = rf.predict(X_test)
print(classification_report(y_test,y_pred_rf,target_names=['benign', 'defacement','phishing','malware']))

score = metrics.accuracy_score(y_test, y_pred_rf)
print("accuracy:   %0.3f" % score)
```

```
              precision    recall  f1-score   support

      benign       0.97      0.98      0.98     85621
  defacement       0.98      0.99      0.99     19292
    phishing       0.99      0.94      0.97      6504
     malware       0.91      0.86      0.88     18822

    accuracy                           0.97    130239
   macro avg       0.96      0.95      0.95    130239
weighted avg       0.97      0.97      0.97    130239

accuracy:   0.966
```

FIG 1.9.1 Training the model with random forest algorithm

The above diagram describes that the model is trained using a random forest algorithm [2].

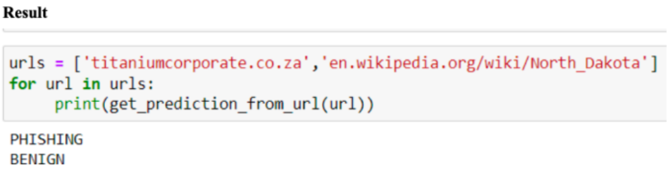
4.6 Testing the model



```
def get_prediction_from_url(test_url):
    features_test = main(test_url)
    # Due to updates to scikit-learn, we now need a 2D array as a parameter to the predict function.
    features_test = np.array(features_test).reshape((1, -1))


    pred = lgb.predict(features_test)
    if int(pred[0]) == 0:

        res="BENIGN"
        return res
    elif int(pred[0]) == 1.0:

        res="DEFACEMENT"
        return res
    elif int(pred[0]) == 2.0:
        res="MALWARE"
        return res

    elif int(pred[0]) == 3.0:

        res="PHISHING"
        return res
```

FIG 2.0 Testing the model.

RESULT



Result

```
urls = ['titaniumcorporate.co.za','en.wikipedia.org/wiki/North_Dakota']
for url in urls:
    print(get_prediction_from_url(url))
```

```
PHISHING
BENIGN
```

FIG 2.1 Real-time output .

5. FUTURE SCOPE

The future scope of this work entails enhancing the Machine Learning model by incorporating additional data and URL features to yield more accurate and refined results. There is potential for training the model to detect Dark Websites, thereby broadening its applicability. Additionally, the development of a Browser Extension could facilitate continuous background processing to dynamically filter out Malicious Websites, thus augmenting user security.

CONCLUSION

In conclusion, this study employed Random Forest, Light GBM, and XGBoost ML classifiers for the detection of malicious URLs. Among these classifiers, Random Forest exhibited the highest accuracy, reaching 97%. Key features such as count-www, hostname_length, count_dir, fd_length, and abnormal_url played significant roles in identifying malicious URLs. Through the detection of such URLs, we contribute to the prevention of future cyber-attacks.

REFERENCES

[1] Dataset: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

[2] Notebook: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

[3] https://www.javatpoint.com/machine-learning-randomforest-algorithm

[4] https://www.geeksforgeeks.org/xgboost/

[5] https://www.geeksforgeeks.org/lightgbm-light-gradientboosting-machine/

[6] https://www.geeksforgeeks.org/generating-word-cloudpython/

[7] https://www.section.io/engineering-education/detecting-malicious-url-using-machine-learning/

[8] https://thesai.org/Downloads/Volume11No1/Paper_19-Malicious_URL_Detection_based_on_Machine_Learning.pdf

[9] D. Sahoo, C. Liu, S.C.H. Hoi, “Malicious URL Detection using Machine Learning: A Survey”. CoRR, abs/1701.07179, 2017.

[10] R. Heartfield and G. Loukas, “A taxonomy of attacks and a survey of defence mechanisms for semantic social engineering attacks,” ACM Computing Surveys (CSUR), vol. 48, no. 3, p. 37, 2015 [

[11] Internet Security Threat Report (ISTR) 2019–Symantec. https://www.symantec.com/content/dam/symantec/docsreports/istr24- 2019-en.pdf [Last accessed 10/2019].