INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 04, April 2024, pp: 2503-2515

e-ISSN : 2583-1062

Impact Factor: 5.725

DETECTION OF PHISHING WEBSITES USING MACHINE LEARNING

Mr. V. Chandra Sekhar Reddy[1], Dasari Likhitha[2], Syed Nawaz Hussain[3], Varikuppala Vinod Kumar[4], Akhil Reddy Karnati[5]

[1]Associate. Professor, CSE Dept, ACE Engineering College, Hyderabad, India.

[2,3,4,5]Student, CSE Dept, ACE Engineering College, Hyderabad, India.

ABSTRACT

Phishing attack is a simplest way to obtain sensitive information from innocent users. Aim of the phishers is to acquire critical information like username, password and bank account details. Cyber security persons are now looking for trustworthy and steady detection techniques for phishing websites detection. This deals with machine learning technology for detection of phishing URLs by extracting and analyzing various features of legitimate and phishing URLs. Decision Tree, random forest and Support vector machine algorithms are used to detect phishing websites. Aim of the paper is to detect phishing URLs as well as narrow down to best machine learning algorithm by comparing accuracy rate, false positive and false negative rate of each algorithm. A web service is one of the most important Internet communications software services. Using fraudulent methods to get personal information is becoming increasingly widespread these days. However, it makes our lives easier, it leads to numerous security vulnerabilities to the Internet's private structure. Web phishing is just one of the many security risks that web services face. Phishing assaults are usually detected by experienced users however, security is a primary concern for system users who are unaware of such situations. Phishing is the act of portraying malicious web runners as genuine web runners to obtain sensitive information from the end-user. Phishing is currently regarded as one of the most dangerous threats to web security. Vicious Web sites significantly encourage Internet criminal activity and inhibit the growth of Web services. As a result, there has been a tremendous push to build a comprehensive solution to prevent users from accessing such websites. Our technology merely examines the Uniform Resource Locator (URL) itself, not the content of Web pages. As a result, it detects the fake or fraud websites. When compared to a blacklisting service, our approach performs better on generality and content since it uses learning techniques.

1. INTRODUCTION

The aim is to contribute to developing a more secure digital environment by offering an advanced approach to phishing site detection. By accurately identifying and mitigating phishing threats, the proposed model will enhance the safety and trustworthiness of online interactions, protecting users from falling victim to phishing attacks. Phishing is a fraudulent technique that uses social and technological tricks to steal customer identification and financial credentials.

early stages, making timely detection a critical factor in successful treatment.

Social media systems use spoofed e-mails from legitimate companies and agencies to enable users to use fake websites to divulge financial details like usernames and passwords. Hackers install malicious software on computers to steal credentials, often using systems to intercept username and passwords of consumers' online accounts. Phishers use multiple methods, including email, Uniform Resource Locators (URL), instant messages, forum postings, telephone calls, and text messages to steal user information. The structure of phishing content is similar to the original content and trick users to access the content in order to obtain their sensitive data. The primary objective of phishing is to gain certain personal information for financial gain or use of identity theft. Phishing attacks are causing severe economic damage around the world.

2. OBJECTIVES

To enhance the effectiveness of phishing website detection, several key considerations must be addressed. Firstly, accuracy is paramount; developing algorithms or models capable of accurately distinguishing phishing websites from legitimate ones is essential to minimize false positives and false negatives. Real- time or near-real-time detection mechanisms should be implemented to identify phishing websites promptly as they are created or accessed. Feature engineering plays a critical role, as identifying relevant features or indicators, such as URL structure, SSL certificates, and website content, can significantly aid in distinguishing phishing sites. Machine learning techniques, including supervised or unsupervised learning, can be employed to automatically classify websites based on these identified features. Scalability is crucial, ensuring that the detection system can efficiently handle large volumes of web traffic, as phishing attacks can scale rapidly.

User education is also vital, implementing measures to educate and alert users about potential phishing risks when they visit suspicious websites. Data sources must be collected and maintained, comprising up-to-date datasets of known phishing websites and legitimate websites to train and evaluate detection models effectively. Integration with web browsers, email clients, and other relevant software is necessary to provide proactive protection to users.

PROBLEM STATEMENT

Phishing attacks pose a significant threat to cybersecurity, exploiting both social engineering and technological vulnerabilities to steal sensitive information. This project aims to develop an advanced approach to detecting phishing sites, thereby bolstering the security of online interactions and safeguarding users from falling victim to fraudulent activities. By accurately identifying and mitigating phishing threats, the proposed model seeks to enhance the safety and trustworthiness of digital environments. Phishing tactics encompass a wide range of methods, including spoofed emails, fraudulent websites, and malicious software, all aimed at deceiving users into divulging personal and financial credentials. Through the utilization of innovative techniques and algorithms, this model aims to effectively distinguish between legitimate and phishing websites, thereby thwarting potential cyberattacks and minimizing economic damage on a global scale.

Phishing attacks continue to pose a significant threat to individuals, organizations, and online users worldwide. Phishing websites mimic legitimate websites to deceive users into disclosing sensitive information such as login credentials, financial data, or personal information. Traditional phishing detection methods often rely on manual inspection or signature- based approaches, which are limited in their ability to detect new and evolving phishing tactics.

3. LITERATURE SURVEY

The detection of malicious URLs has become increasingly important in the realm of cybersecurity due to the rising prevalence of cyber threats such as phishing attacks, malware distribution, and website defacement. In recent years, researchers have employed various machine learning techniques to effectively classify URLs into different categories based on their malicious intent. This literature survey aims to provide an overview of existing methodologies and approaches utilized in the field of malicious URL detection.

Feature Engineering:

Feature engineering plays a crucial role in the successful detection of malicious URLs. Researchers have explored various lexical, structural, and semantic features derived from URLs to represent their characteristics effectively. Lexical features, such as domain length, presence of special characters, and domain age, have been widely used due to their simplicity and effectiveness in capturing malicious patterns.Reference:

Nappa, D., et al. "A Machine Learning Approach for Detection of Malicious URLs." IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 3, 2018, pp. 460-472.

Machine Learning Algorithms:

Supervised machine learning algorithms have been extensively employed for the classification of URLs into benign and malicious categories. Boosting algorithms, such as XGBoost, Light GBM, and Gradient Boosting Machines, have demonstrated superior performance in handling imbalanced datasets and achieving high accuracy in malicious URL detection tasks.References:

Tian, Y., et al. "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection." IEEE Transactions on Information Forensics and Security, vol. 14, no. 5, 2019, pp. 1175-1186.

Zhang, J., et al. "Malicious URL Detection Using Machine Learning: A Comparative Study." IEEE Access, vol. 8, 2020, pp. 17206-17219.

Dataset Curation:

Building a comprehensive and diverse dataset is essential for training robust machine learning models for malicious URL detection. Researchers have utilized a combination of publicly available datasets, domain blacklists, and crowdsourced repositories to collect a large corpus of URLs spanning different categories of malicious activities. Reference:

Ma, J., et al. "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs." Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 1245-1254.

Ensemble Techniques:

Ensemble learning techniques have been employed to improve the overall performance and generalization ability of malicious URL detection models. Ensemble methods combine predictions from multiple base classifiers to mitigate overfitting and enhance the robustness of the final classification.reference:

Wei, Q., et al. "Malicious URL Detection Based on Ensemble Learning." International Conference on Security and

Privacy in Communication Networks, 2017, pp. 161-176.In summary, the literature on malicious URL detection encompasses a wide range of approaches, including feature engineering, machine learning algorithms, dataset curation, and ensemble techniques. By leveraging these methodologies, researchers continue to advance the state-of-the-art in the field of cybersecurity, striving to develop more accurate and reliable solutions for identifying and mitigating malicious online threats.

4. PROPOSED SYSYTEM

The proposed system focuses on leveraging machine learning (ML) to automate the detection of phishing websites. It will extract relevant features like URL length, domain age, HTTPS presence, subdomain usage, sensitive keyword presence, HTML and JavaScript analysis, and website content analysis from each website in the dataset. Once the model achieves satisfactory performance, it will be deployed in a production environment to actively analyze incoming URLs in real-time. A user-friendly interface will be developed to present phishing detection results to users. To bolster user protection, the system can be integrated into popular web browsers as an extension or merged into existing security tools and antivirus software. This integration will provide users with real-time alerts when accessing potentially harmful websites. By integrating advanced algorithms with thorough

feature analysis, the system aims to provide an effective and precise solution for identifying phishing websites. It seeks to reduce the risks associated with phishing attacks and bolster overall cybersecurity measures.

5. HARDWARE AND SOFTWARE REQUIREMENTS

5.1 HARDWARE REQUIREMENTS:

- Processor – Pentium IV
- RAM – 4 GB (min)
- Hard Disk – 20 GB
- Key Board – Standard Windows Keyboard
- Mouse – Two or Three Button Mouse
- Monitor – SVGA

5.2 SOFTWARE REQUIREMENTS:

- Operating system – Windows 7, 8, 10, 11, Mac OS
- Coding Language – Python
- Back-End – Python.

PACKAGES USED

pandas (imported as pd): For data manipulation and analysis. **itertools:** For creating iterators for efficient looping. **sklearn. metrics**

classification_ report: For generating a classification report including precision, recall, F1-score, support. confusion_ matrix: For computing the confusion matrix to evaluate the accuracy of classification. Accuracy score: For computing the accuracy classification

score.

sklearn. model_ selection:

train_ test_ split: For splitting arrays or matrices into random train and test subsets.

numpy (imported as np): For numerical computing. **matplotlib. pyplot (imported as plt):** For creating static, interactive, and animated visualizations.

Xg boost: For implementing the XGBoost algorithm for gradient boosting.

lightgbm:

LGBM Classifier: For implementing the Light GBM algorithm, a gradient boosting framework that uses tree-based learning.

os: For interacting with the operating system, such as reading or writing files.

seaborn (imported as sns):

For statistical data visualization. **wordcloud:** For creating word clouds, which visualize word frequency in a given text.

ALGORITHM

- Datasets containing phishing and legitimate websites is collected from open-source platform Phish Tank.
- Write a code to extract the required features from the URL database.
- Analyse and preprocess the dataset by using EDA techniques.
- Divide the dataset into training and testing sets.
- Run selected machine learning and deep neural network algorithms on the dataset like Decision Tree, Random Forest,
- MultilayerPerceptron's, XGBoost, Autoencoder Neural Networks and Support Vector Machines on the dataset.
- Write a code for displaying the evaluation result considering accuracy metrics.
- Compare the obtained results for trained models and specify which is better.

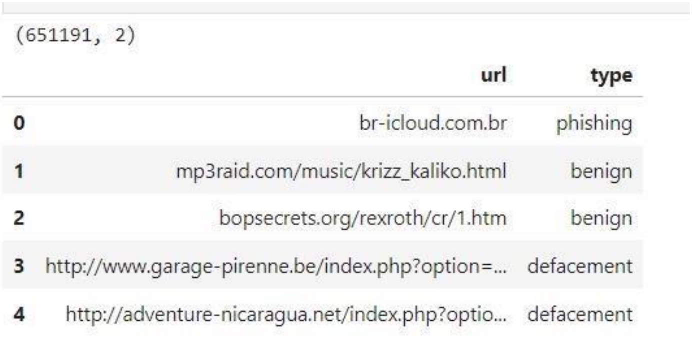
SOURCE CODE:

```
import pandas as pd import itertools
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score from sklearn.model_selection
import train_test_split import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import xgboost as xgb
from lightgbm import LGBMClassifier import os
import seaborn as sns
from wordcloud import WordCloud df=pd.read_csv('malicious_phish.csv')
print(df.shape) df.head() df.type.value_counts()
df_phish = df[df.type=='phishing'] df_malware = df[df.type=='malware'] df_deface = df[df.type=='defacement']
df_benign = df[df.type=='benign'] phish_url = " ".join(i for i in df_phish.url) wordcloud = WordCloud(width=1600,
height=800,colormap='Paired').generate(phish_url) plt.figure( figsize=(12,14),facecolor='k') plt.imshow(wordcloud,
interpolation='bilinear') plt.axis("off")
plt.tight_layout(pad=0) plt.show()
malware_url = " ".join(i for i in df_malware.url) wordcloud = WordCloud(width=1600,
height=800,colormap='Paired').generate(malware_url) plt.figure( figsize=(12,14),facecolor='k') plt.imshow(wordcloud,
interpolation='bilinear') plt.axis("off")
plt.tight_layout(pad=0) plt.show()
deface_url = " ".join(i for i in df_deface.url) wordcloud = WordCloud(width=1600,
height=800,colormap='Paired').generate(deface_url) plt.figure( figsize=(12,14),facecolor='k') plt.imshow(wordcloud,
interpolation='bilinear') plt.axis("off")
plt.tight_layout(pad=0) plt.show()
benign_url = " ".join(i for i in df_benign.url) wordcloud = WordCloud(width=1600,
height=800,colormap='Paired').generate(benign_url) plt.figure( figsize=(12,14),facecolor='k') plt.imshow(wordcloud,
interpolation='bilinear')
 plt.axis("off") plt.tight_layout(pad=0) plt.show()
import re
#Use of IP or not in domain def having_ip_address(url):
match = re.search(
'(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-
4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
'([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
'((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-
fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4 in hexadecimal
'(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
if match:
```

```
# print match.group() return 1
else:
# print 'No matching pattern found' return 0
df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i)) from urllib.parse import urlparse
def abnormal_url(url):
hostname = urlparse(url).hostname hostname = str(hostname)
match = re.search(hostname, url) if match:
# print match.group() return 1
else:
# print 'No matching pattern found' return 0
df['abnormal_url'] = df['url'].apply(lambda i: abnormal_url(i)) from googlesearch import search
def google_index(url): site = search(url, 5) return 1 if site else 0
df['google_index'] = df['url'].apply(lambda i: google_index(i)) def count_dot(url):
count_dot = url.count('.') return count_dot
df['count.'] = df['url'].apply(lambda i: count_dot(i)) df.head()
def count_www(url): url.count('www')
return url.count('www')
df['count-www'] = df['url'].apply(lambda i: count_www(i)) def count_atrate(url):
return url.count('@')
df['count@'] = df['url'].apply(lambda i: count_atrate(i)) def no_of_dir(url):
urldir = urlparse(url).path return urldir.count('/')
df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i)) def no_of_embed(url):
urldir = urlparse(url).path return urldir.count('//')
df['count_embed_domian'] = df['url'].apply(lambda i: no_of_embed(i))
def shortening_service(url):
match =
re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tiny url|tr\.im|is\.gd|cli\.gs|'
'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\
.nl|snipurl\.com|'
'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snip r\.com|fic\.kr|loopt\.us|'
'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\
.do|t\.co|lnkd\.in|'
'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly| bit\.ly|ity\.im|'
'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt
\.us|u\.bb|yourls\.org|'
'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\. net|1url\.com|tweez\.me|v\.gd|'
'tr\.im|link\.zip\.net', url)
if match:
return 1 else:
return 0
df['short_url'] = df['url'].apply(lambda i: shortening_service(i)) def count_https(url):
return url.count('https')
df['count-https'] = df['url'].apply(lambda i : count_https(i)) def count_http(url):
return url.count('http')
df['count-http'] = df['url'].apply(lambda i : count_http(i)) def count_per(url):
return url.count('%')
df['count%'] = df['url'].apply(lambda i : count_per(i)) def count_ques(url):
```

```
return url.count('?')
df['count?'] = df['url'].apply(lambda i: count_ques(i)) def count_hyphen(url):
return url.count('-')
df['count-'] = df['url'].apply(lambda i: count_hyphen(i)) def count_equal(url):
return url.count('=')
df['count='] = df['url'].apply(lambda i: count_equal(i)) def url_length(url):
return len(str(url)) #Length of URL
df['url_length'] = df['url'].apply(lambda i: url_length(i)) #Hostname Length
def hostname_length(url):
return len(urlparse(url).netloc)
df['hostname_length']          =          df['url'].apply(lambda          i: hostname_length(i))
df.head()
def suspicious_words(url):
match                                                                    =
re.search('PayPal|login|signin|bank|account|update|free|lucky|ser vice|bonus|ebayisapi|webscr',
url)
if match:
return 1 else:
return 0
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i)) def digit_count(url):
digits = 0 for i in url:
if i.isnumeric(): digits = digits + 1
return digits
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()
#Importing dependencies
from urllib.parse import urlparse from tld import get_tld
import os.path
#First Directory Length def fd_length(url):
urlpath= urlparse(url).path try:
return len(urlpath.split('/')[1]) except:
return 0
df['fd_length'] = df['url'].apply(lambda i: fd_length(i)) #Length of Top Level Domain
df['tld'] = df['url'].apply(lambda i: get_tld(i,fail_silently=True)) def tld_length(tld):
try:
return len(tld) except:
return -1
df['tld_length'] = df['tld'].apply(lambda i: tld_length(i)) df = df.drop("tld",1)
df.columns df['type'].value_counts() import seaborn as sns sns.set(style="darkgrid")
ax = sns.countplot(y="type", data=df,hue="use_of_ip") sns.set(style="darkgrid")
ax = sns.countplot(y="type", data=df,hue="abnormal_url") sns.set(style="darkgrid")
ax = sns.countplot(y="type", data=df,hue="google_index") sns.set(style="darkgrid")
ax = sns.countplot(y="type", data=df,hue="short_url") sns.set(style="darkgrid")
ax = sns.countplot(y="type", data=df,hue="sus_url") sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="count.", kind="box", data=df) sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="count-www", kind="box", data=df) sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="count@", kind="box", data=df)\ sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="count_dir", kind="box", data=df) sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="hostname_length", kind="box", data=df)
sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="fd_length", kind="box", data=df) sns.set(style="darkgrid")
ax = sns.catplot(x="type", y="tld_length", kind="box", data=df) from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
df["type_code"] = lb_make.fit_transform(df["type"]) df["type_code"].value_counts()
#Predictor Variables
# filtering out google_index as it has only 1 value
X = df[['use_of_ip','abnormal_url', 'count.', 'count-www', 'count@',
'count_dir', 'count_embed_domian', 'short_url', 'count-https', 'count-http', 'count%', 'count?', 'count-', 'count=',
'url_length', 'hostname_length', 'sus_url', 'fd_length', 'tld_length', 'count-
digits',
count-letters']] #Target Variable
y = df['type_code'] X.head() X.columns
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle=True, random_state=5)
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=['beni gn',
'defacement','phishing','malware']))
cm = confusion_matrix(y_test, y_pred_rf) cm_df = pd.DataFrame(cm,
index = ['benign', 'defacement','phishing','malware'], columns = ['benign',
'defacement','phishing','malware']) plt.figure(figsize=(8,6)) sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix') plt.ylabel('Actal Values') plt.xlabel('Predicted Values')
plt.show()
feat_importances = pd.Series(rf.feature_importances_, index=X_train.columns)
feat_importances.sort_values().plot(kind="barh",figsize=(10, 6)) lgb
LGBMClassifier(objective='multiclass',boosting_type= 'gbdt',n_jobs = 5, random_state=5)
LGB_C = lgb.fit(X_train, y_train) y_pred_lgb = LGB_C.predict(X_test)
 print(classification_report(y_test,y_pred_lgb,target_names=['ben ign', 'defacement','phishing','malware']))
cm = confusion_matrix(y_test, y_pred_lgb) cm_df = pd.DataFrame(cm,
index = ['benign', 'defacement','phishing','malware'], columns = ['benign',
'defacement','phishing','malware']) plt.figure(figsize=(8,6)) sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix') plt.ylabel('Actal Values') plt.xlabel('Predicted Values')
plt.show()
feat_importances = pd.Series(lgb.feature_importances_, index=X_train.columns)
feat_importances.sort_values().plot(kind="barh",figsize=(10, 6)) xgb_c = xgb.XGBClassifier(n_estimators= 100)
xgb_c.fit(X_train,y_train)
y_pred_x = xgb_c.predict(X_test) print(classification_report(y_test,y_pred_x,target_names=['benig n',
'defacement','phishing','malware']))
cm = confusion_matrix(y_test, y_pred_x) cm_df = pd.DataFrame(cm,
index = ['benign', 'defacement','phishing','malware'], columns= ['benign',
'defacement','phishing','malware']) plt.figure(figsize=(8,6)) sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix') plt.ylabel('Actal Values') plt.xlabel('Predicted Values')
plt.show()
feat_importances = pd.Series(xgb_c.feature_importances_, index=X_train.columns)
feat_importances.sort_values().plot(kind="barh",figsize=(10, 6)) def main(url):
status = [] status.append(having_ip_address(url)) status.append(abnormal_url(url)) status.append(count_dot(url))
status.append(count_www(url)) status.append(count_atrate(url)) status.append(no_of_dir(url))
status.append(no_of_embed(url)) status.append(shortening_service(url)) status.append(count_https(url))
status.append(count_http(url)) status.append(count_per(url)) status.append(count_ques(url))
status.append(count_hyphen(url)) status.append(count_equal(url)) status.append(url_length(url))
status.append(hostname_length(url)) status.append(suspicious_words(url)) status.append(digit_count(url))
status.append(letter_count(url)) status.append(fd_length(url))
tld = get_tld(url,fail_silently=True)
status.append(tld_length(tld)) return status
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="SAFE" return res
elif int(pred[0]) == 1.0:
res="DEFACEMENT"
return res
elif int(pred[0]) == 2.0:
res="PHISHING"
return res
elif int(pred[0]) == 3.0: res="MALWARE"
return res
urls = ['titaniumcorporate.co.za','www.youtube.com'] for url in urls:
print(get_prediction_from_url(url))
```

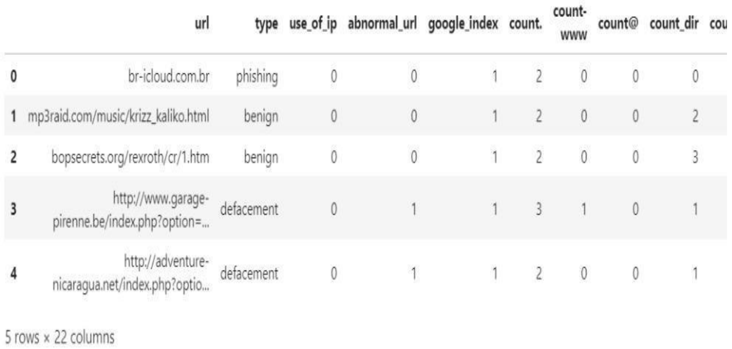
6. OUTPUT



```
(651191, 2)
```

	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

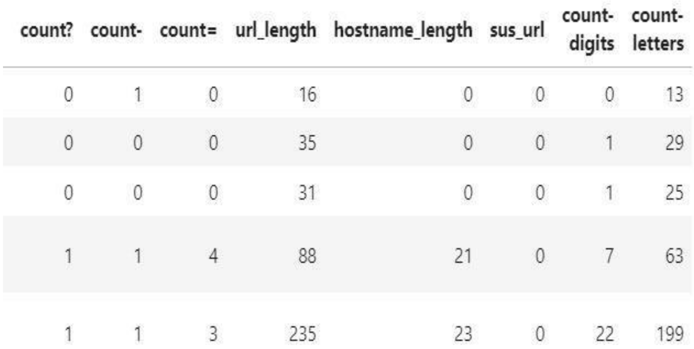
Fig .1 import the dataset using the pandas library and check the sample entries in the dataset.



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

5 rows × 22 columns

Fig .2 after creating features the dataset



count?	count-	count=	url_length	hostname_length	sus_url	count-digits	count-letters
0	1	0	16	0	0	0	13
0	0	0	35	0	0	1	29
0	0	0	31	0	0	1	25
1	1	4	88	21	0	7	63
1	1	3	235	23	0	22	199

Fig .3 dataset looks like above

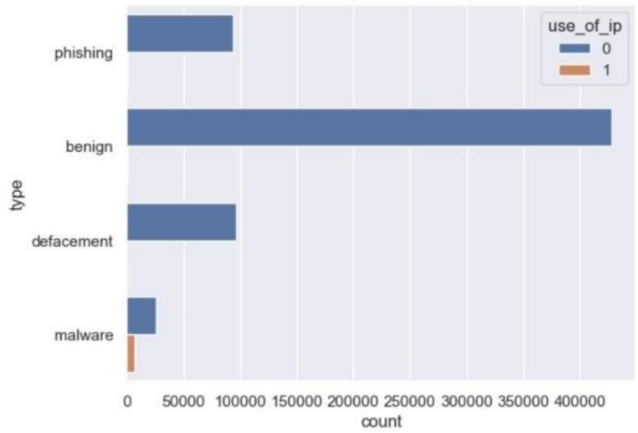


Fig.4 Distribution of use of IP

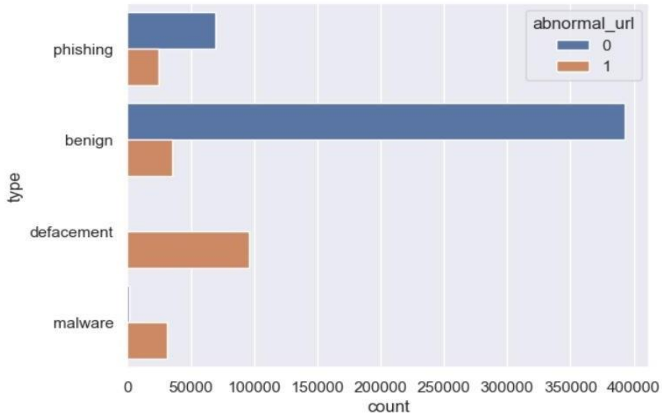


Fig.5 Distribution of Abnormal URL

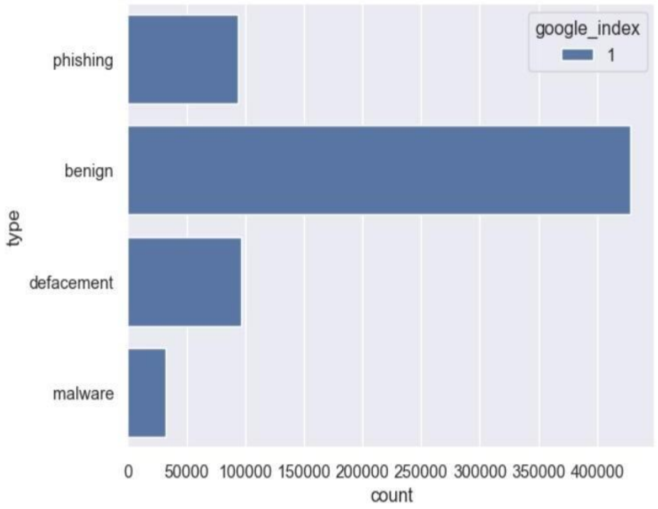


Fig .6 Distribution of Google Index

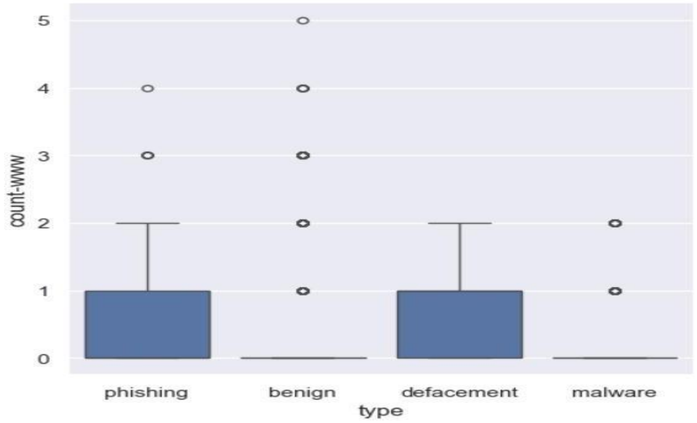


Fig .7 Distribution of Short URL's

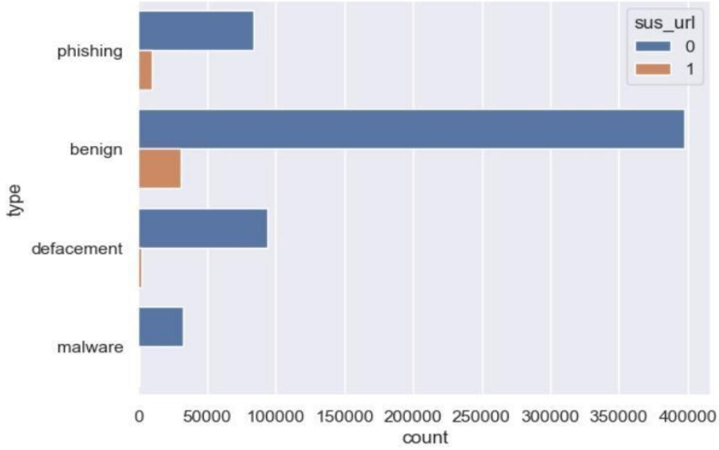


Fig .8 Distribution of Suspicious URL

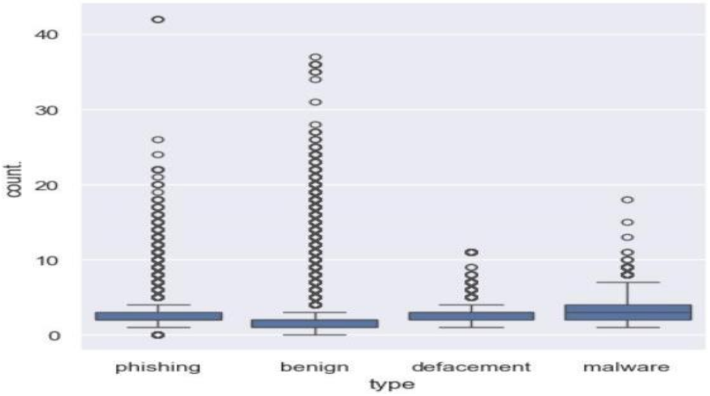


Fig .9 Distribution of count of [.] dot

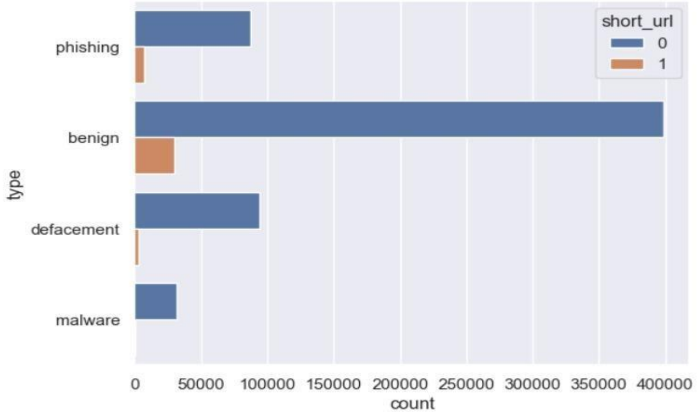


Fig .10 Distribution of count-www

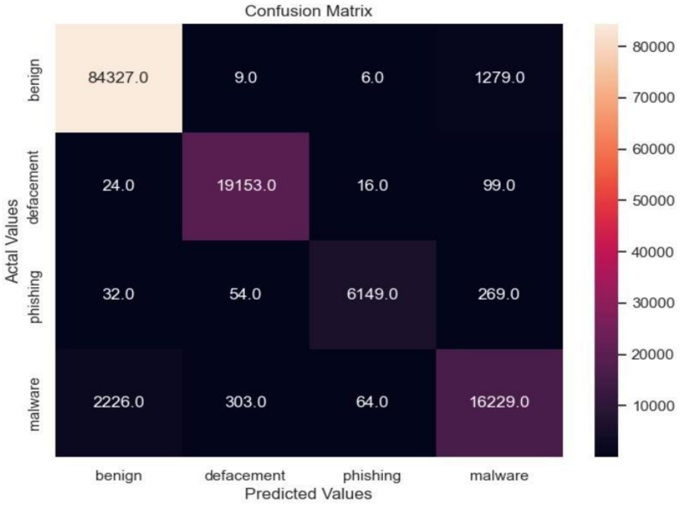


Fig .11 Distribution of hostname length

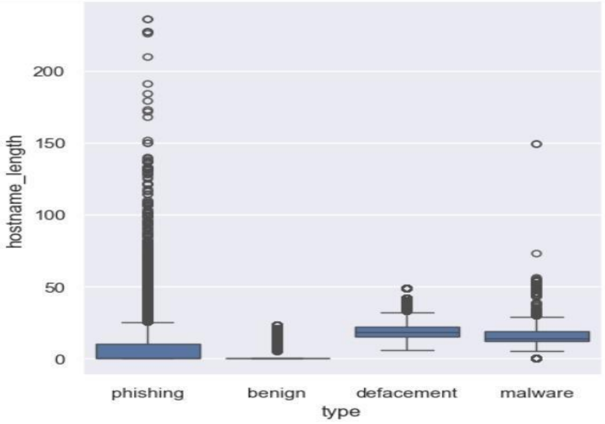
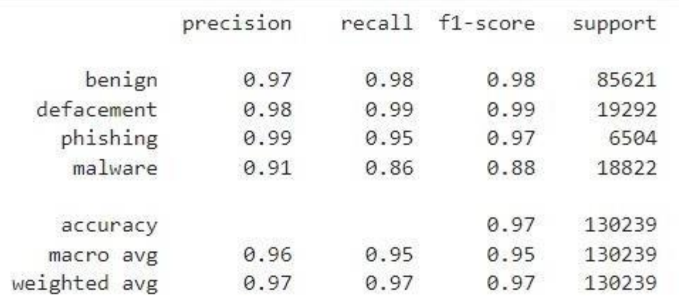


Fig .12 Random Forest Classifier



```
              precision    recall  f1-score   support

      benign       0.97      0.98      0.98     85621
  defacement       0.98      0.99      0.99     19292
    phishing       0.99      0.95      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
```

Fig .13 Confusion Matrix for Random Forest Classifier

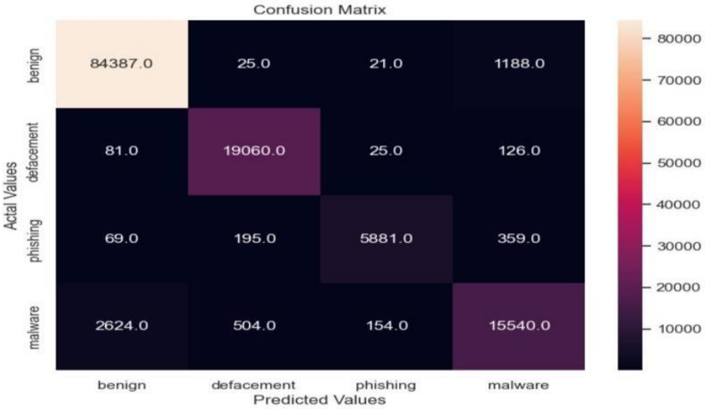


Fig .14 Confusion Matrix for Light BGM Classifier

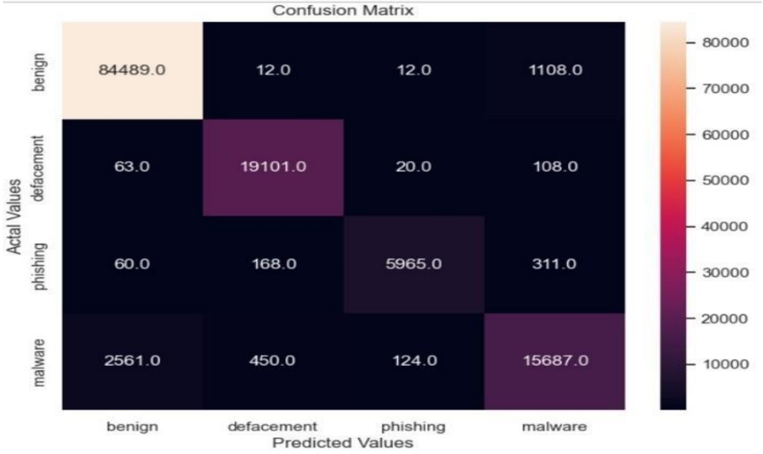
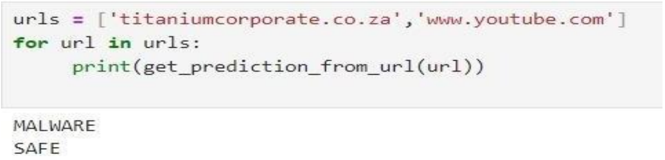


Fig .15 Confusion Matrix for XGBoost Classifier



```
urls = ['titaniumcorporate.co.za','www.youtube.com']
for url in urls:
     print(get_prediction_from_url(url))

MALWARE
SAFE
```

Fig .16 Final Output of Predection of the given URL

7. TESTING

We have split the dataset into 80:20 ratio i.e., 80% of the data was used to train the machine learning models, and the rest 20% was used to test the model.As we know we have an imbalanced dataset. The reason for this is around 66% of the data has benign URLs, 5% malware, 14% phishing, and 15% defacement URLs. So after randomly splitting the dataset into train and test, it may happen that the distribution of different categories got disturbed which will highly affect the performance of the machine learning model. So to maintain the same proportion of the target variable stratification is needed .This stratify parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to the parameter stratify.**X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle=True, random_state=5)**

8. CONCLUSION

In conclusion, the Phishing Website Detection system represents a significant advancement in cybersecurity, harnessing the power of cutting-edge technologies such as Artificial Intelligence (AI) and Machine Learning (ML). This innovative platform not only addresses the pressing need to combat the escalating threat of phishing attacks but also provides a robust solution for safeguarding users' sensitive information in the digital landscape. By integrating sophisticated AI algorithms, the system excels in analysing complex web data and identifying subtle indicators of phishing activities with remarkable accuracy. Through predictive modelling and pattern recognition, the Phishing Website Detection system empowers users to anticipate and thwart malicious attempts in real-time, thereby mitigating potential risks and preserving online security. The adaptability and continuous learning capabilities of the system ensure its relevance and effectiveness in dynamically evolving cyber threats cape. As new phishing tactics emerge, the system autonomously adapts its detection mechanisms, staying ahead of adversaries and providing users with proactive protection. Moreover, rigorous testing methodologies, including unit testing, integration testing, and security testing, validate the system's reliability, performance, and resilience against cyber threats. By prioritizing user experience through intuitive user interfaces and seamless integration with existing software environments, the system ensures accessibility and usability for both cybersecurity experts and end-users. The cross-disciplinary applicability of the Phishing Website Detection system underscores its significance in diverse sectors, including finance, e-commerce, healthcare, and government. Its ability to detect and prevent phishing attacks effectively serves as a critical line of defence against financial fraud, data breaches, and identity theft, thereby safeguarding individuals and organizations worldwide. As the system undergoes continuous refinement and enhancement, fueled by ongoing research and feedback from cybersecurity professionals and end-users alike, it remains poised to redefine the landscape of online security and shape the future of cybersecurity technologies. In essence, the Phishing Website Detection system stands as a testament to the transformative potential of AI and ML in fortifying digital defenses and preserving trust in the interconnected world. Through its implementation

9. FUTURE SCOPE

Machine Learning Model Refinement: Continuously train and refine the machine learning model used for phishing website detection to improve its accuracy and efficiency over time. Incorporate more sophisticated algorithms or techniques such as deep learning to enhance detection capabilities.

Real-Time Detection: Implement real-time detection capabilities to promptly identify and block phishing websites as they emerge. This could involve integrating the detection system with web browsers or network security solutions to provide immediate protection to users.

Multi-Layered Detection Approach: Develop a multi- layered detection approach that combines various detection methods such as heuristic analysis, URL analysis, content analysis, and behavioral analysis. This approach can improve the robustness of the detection system and reduce false positives.

10. REFERENCES

[1] https://blog.keras.io/building-autoencoders-in-keras.html

[2] https://en.wikipedia.org/wiki/Autoencoder

[3] https://mc.ai/a-beginners-guide-to-build-stacked-autoencoder- and-tying-weights-with-it/

[4] https://machinelearningmastery.com/save-gradient-boosting- models-xgboost-python/