IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

MALICIOUS URL DETECTION

[1]R. Indu, [2]M. Bhavya, [3]V. Pardhasaradhi, [4]Y. Sri Ram, [5]Dr. Y. Suresh

[1,2,3,4] B. Tech Students, Department of Information Technology

[5]Assistant Professor, Department of Information Technology

Prasad V. Potluri Siddhartha Institute of Technology, Vijayawada, Andhra Pradesh, India

Abstract: The importance of the World Wide Web has grown. Unfortunately, technical advancements have led to the emergence of new, highly complex techniques for user exploitation. In these attempts, users' computers may be infected with malware or utilize unfriendly websites to sell fake items or disclose private data that can be used to steal money and commit financial fraud. Malicious URLs can be particularly damaging to potential victims because they host a wide variety of unwanted content. So, a quick and efficient detection technique is needed. In this thesis, we focus on the problem of identifying dangerous URLs utilizing data from URLs and machine learning technology. We'll employ a variety of machine-learning techniques, including XGBoost, LightGBM, and Random Forest.

Index Terms **– Malicious URLs, XGBoost, LightGBM, Random Forest.**

I. INTRODUCTION

A website link that is intended to spread viruses, phishing scams, and other illegal actions is known as a malicious URL. A user can download computer viruses including trojan horses, ransomware, worms, and malware by clicking a malicious Link. Some infections' ultimate objectives include gaining access to personal data, harming the user's device, and generating revenue. They might also ruin the network of the business, causing losses. A malicious URL can also be employed to entice users to enter their personal data on a bogus website. Some folks are compelled to divulge their private information to strangers as a result. They make use of the data for a covert purpose. These rogue URLs have the potential to do a lot of damage.

II. PROPOSED SYSTEM

By extracting useful feature representations of URLs and training a prediction model on training data of both harmful and benign URLs, these approaches attempt to examine the information contained in a URL and its connected websites or webpages. Static and dynamic features can both be used. When performing static analysis, we use the information already in hand rather than actually running the URL. Lexical features from the URL string, host-related data, and occasionally even HTML and Javascript content are among the aspects that can be extracted. These techniques are safer than dynamic ones because execution is not necessary. The basic presumption is that malicious and benign URLs distribute these properties in different ways. This distribution data can be used to create a prediction model that can forecast new URLs. Static analysis techniques have been thoroughly investigated by applying machine learning techniques due to the comparatively safer environment for obtaining crucial information and the capacity to generalize to all forms of threats.

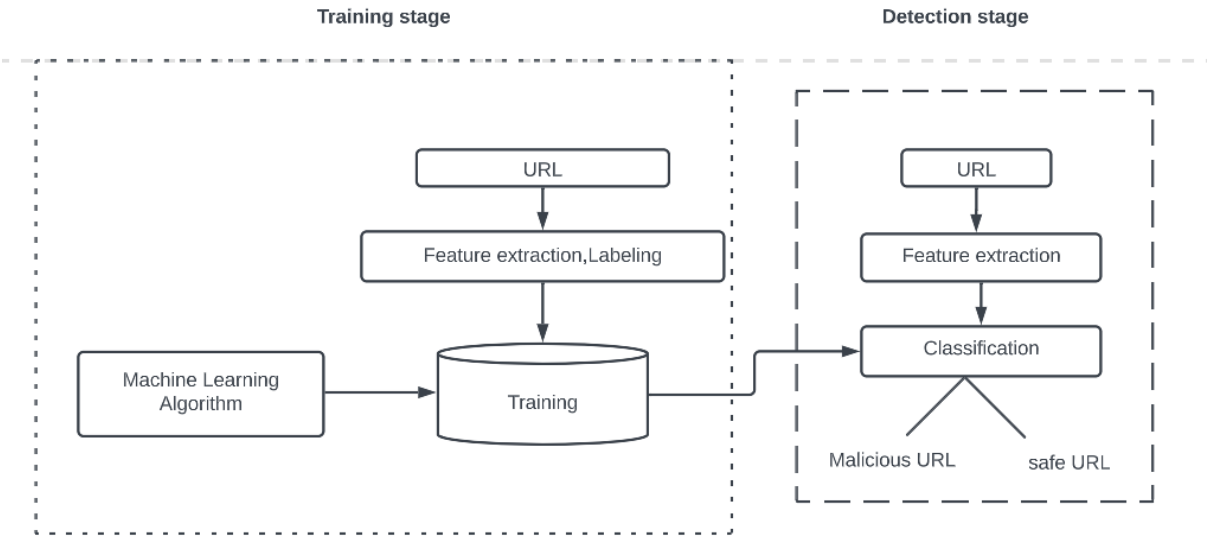

FIG 1.1 Proposed system model

2.1 Dataset Description

651,191 URLs total, including 428103 benign or safe URLs, 96457 defacement URLs, 94111 phishing URLs, and 32520 malware URLs, have been gathered into a sizable dataset by our team. As we all know, selecting the right dataset for a machine learning project is one of the most important responsibilities. This dataset is a compilation of five sources.

We used the URL dataset to gather benign, phishing, malware, and defacement URLs (ISCX-URL-2016) Using the Malware domain blacklist information, we have seen an increase in phishing and malware URLs. Using the Faizan git repository, we have enhanced the number of benign Websites.

The Phishtank dataset and the PhishStorm dataset have finally allowed us to increase the number of phishing URLs. As we have mentioned, various sources were used to get the dataset. In order to preserve only URLs and their class type, we first collected the URLs from various sources into a separate data frame.

2.2 Random Forest Classifier

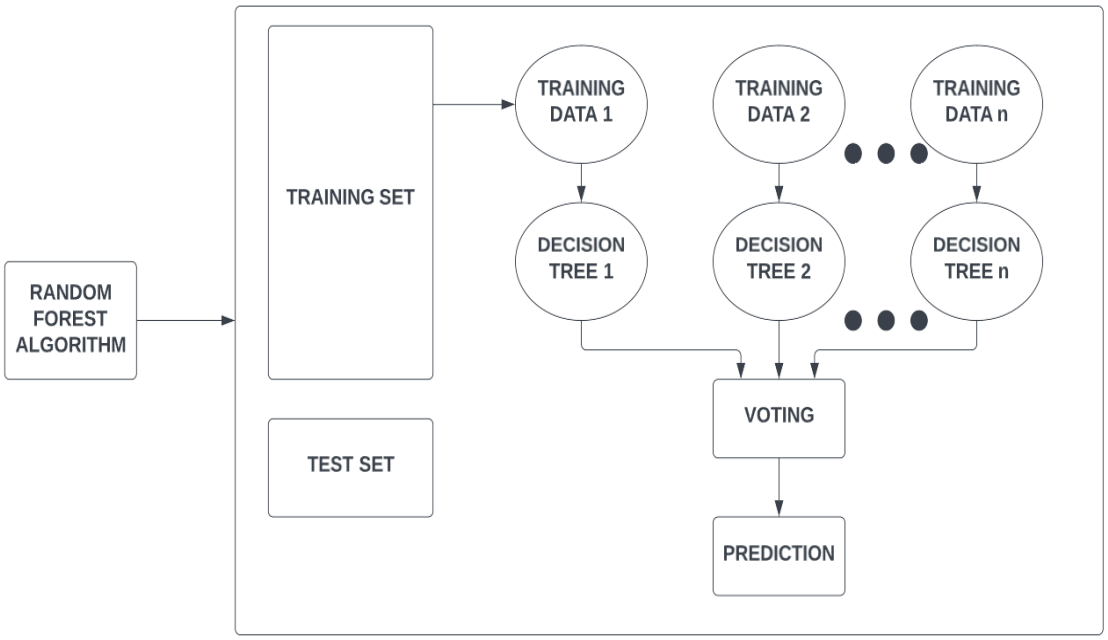

Fig 1.2 Working principle of Random forest classifier

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].

2.3 XGBoost Classifier

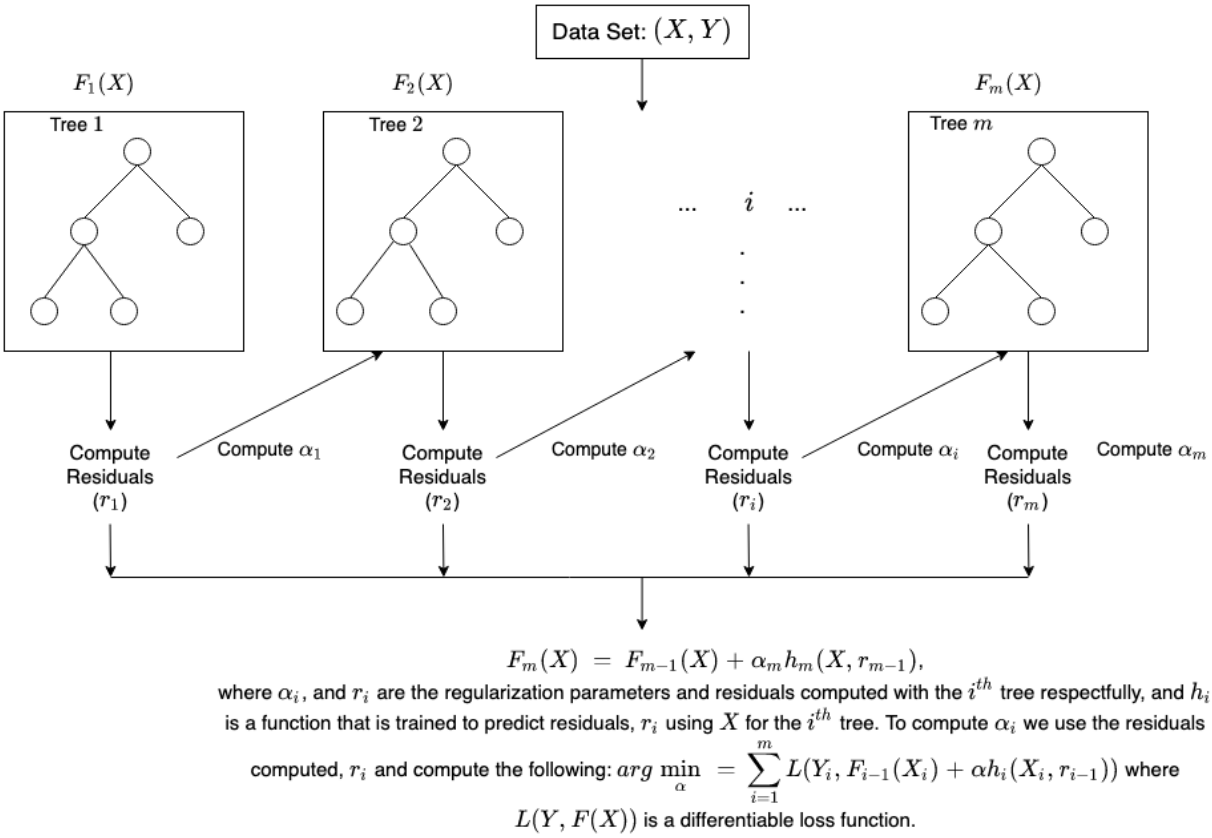

FIG 1.3 Working principle of XGBoost classifier

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].

2.4 Light GBM Classifier

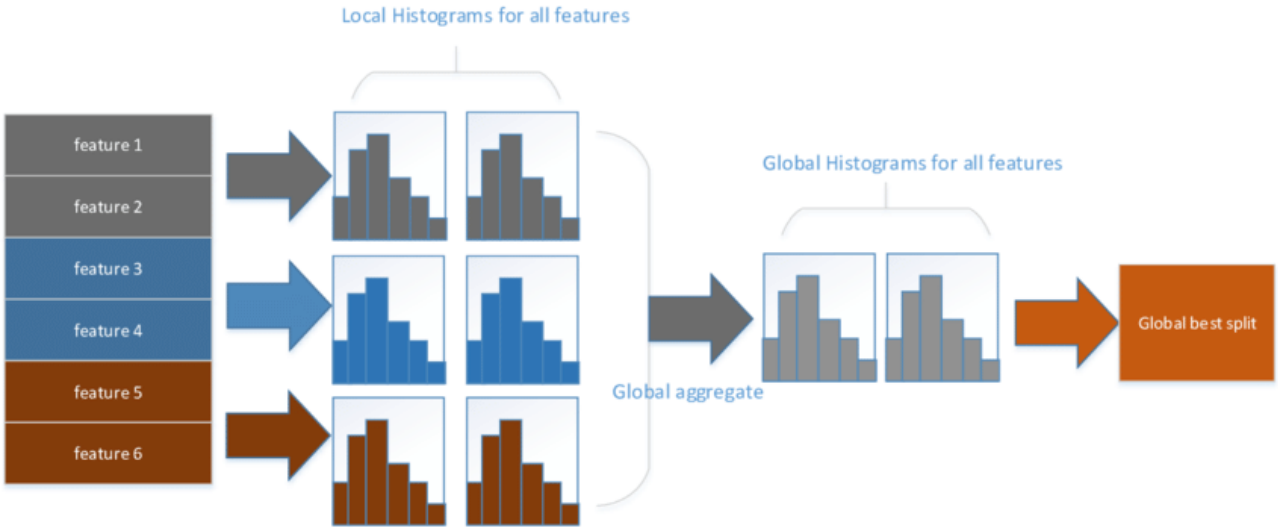

FIG 1.4 Working principle of LightGBM classifier

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

III. TECHNOLOGIES USED

3.1 Jupyter Notebook

The original web application for creating and sharing computational documents is Jupyter Notebook. It provides a straightforward, streamlined, document-centric experience. The Jupyter Notebook is an extremely powerful tool for developing and presenting data science projects interactively.

3.2 Python libraries

3.2.1 NumPy
NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

3.2.2 Pandas
Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

3.2.3 Sklearn
Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistency interface in Python.

3.2.4 Itertools
Itertools is a module in Python, it is used to iterate over data structures that can be stepped over using a for-loop. Such data structures are also known as itertools. This module incorporates functions that utilize computational resources efficiently.

3.2.5 Matplotlib
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication-quality plots. Make interactive figures that can zoom, pan, and update.

IV. RESULTS

4.1 Loading Dataset

```
In [2]: df=pd.read_csv(r'D:\19501a1295\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[2].

4.2 Data Preprocessing

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

url     0
type    0
dtype: int64
```

FIG 1.6 Data without null values

The above figure shows that the data has no more null and missing values. The data is cleaned using the Python library pandas [2].

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	c
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	

FIG 1.7 Lexical features are extracted from the raw URLs

The above diagram describes about the lexical features which are extracted from the raw URLs [2].

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
```

FIG 1.8 Splitting the data into training data and testing data

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].

```
lgb = LGBMClassifier(objective='multiclass',boosting_type= 'gbdt',n_jobs = 5,
          silent = True, 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=['benign', 'defacement','phishing','malware']))

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

```
C:\Users\HP\anaconda3\lib\site-packages\lightgbm\sklearn.py:598: UserWarning: 'silent' argument is deprecated and will be remov
ed in a future release of LightGBM. Pass 'verbose' parameter via keyword arguments instead.
  _log_warning("'silent' argument is deprecated and will be removed in a future release of LightGBM. "
```

```
              precision    recall  f1-score   support

      benign       0.97      0.99      0.98     85621
  defacement       0.96      0.99      0.98     19292
    phishing       0.97      0.90      0.93      6504
     malware       0.90      0.83      0.86     18822

    accuracy                           0.96    130239
   macro avg       0.95      0.93      0.94    130239
weighted avg       0.96      0.96      0.96    130239

accuracy:   0.959
```

FIG 1.9.2 Training the model with Light GBM classifier

The above diagram describes that the model is trained using a light gbm classifier [2].

```
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=['benign', 'defacement','phishing','malware']))


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

```
              precision    recall  f1-score   support

      benign       0.97      0.99      0.98     85621
  defacement       0.97      0.99      0.98     19292
    phishing       0.98      0.92      0.94      6504
     malware       0.91      0.83      0.87     18822

    accuracy                           0.96    130239
   macro avg       0.96      0.93      0.94    130239
weighted avg       0.96      0.96      0.96    130239

accuracy:   0.962
```

FIG 1.9.3 Training the model with XGBoost classifier

The above diagram describes that the model is trained using the xgboost classifier [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 [2].

4.7 Final 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 [2].

V. FUTURE SCOPE

The Future Scope of this work would be training the Machine Learning model with more data and more URL features for more accurate and improved results. A model can be further trained to detect Dark Websites. Moreover, a Browser Extension can also be made so that the process can run in the background continuously to dynamically filter Malicious Websites.

CONCLUSION

Therefore, we have used Random Forest, Light GBM, and XGBoost ML classifiers in order to detect malicious URLs. Out of all these three ML classifiers Random Forest has the highest accuracy of 97%. The feature importances count-www, hostname_length, count_dir, fd_length, abnormal_url, count-http, etc., are used as the major data in detecting malicious URLs. So, by detecting the malicious URLs we could prevent cyber-attacks in the future.

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-random-forest-algorithm**
[4] **https://www.geeksforgeeks.org/xgboost/**
[5] **https://www.geeksforgeeks.org/lightgbm-light-gradient-boosting-machine/**
[6] **https://www.geeksforgeeks.org/generating-word-cloud-python/**
[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/docs/reports/istr-24- 2019-en.pdf [Last accessed 10/2019].