An Adversarial Attack Analysis on Malicious Advertisement URL Detection Framework

Ehsan Nowroozi†, Senior Member, IEEE, Abhishek‡, Member, IEEE, Mohammadreza Mohammadi§, Member, IEEE, and Mauro Conti‡, Fellow, IEEE

Abstract—Malicious advertisement URLs pose a security risk since they are the source of cyber-attacks, and the need to address this issue is growing in both industry and academia. Several attempts have been made in recent years for malicious URL detection using machine learning (ML). The most widely used techniques extract linguistic features of URL string to extract features like bag-of-words (BoW) before applying ML model. Existing malicious URL detection techniques require effective manual feature engineering that can handle unseen features and generalise to test data. In this study, we extract a novel set of lexical and Web-scrapped features and employ ML techniques for fraudulent advertisement URL detection. The combination set of six different kinds of features precisely overcomes the obfuscation in fraudulent URL classification. Based on distinct statistical properties, we use twelve differently formatted datasets for detection, prediction and classification task. We extend our prediction analysis for mismatched and unlabelled datasets. For this framework, we analyze the performance of four ML techniques: Random Forest, Gradient Boost, XGBoost and AdaBoost in the detection part. With our proposed method, we achieve a false negative rate up to 0.0037 while maintaining high detection accuracy of 99.63%. Moreover, we employ an unsupervised learning technique for data clustering using the K-Means algorithm for the visual analysis. This paper analyses the vulnerability of decision tree-based models using the limited knowledge attack scenario. We considered the exploratory attack during the test phase and implemented Zeroth Order Optimization adversarial attack on the detection models.

Index Terms—Malicious advertisement URL, cybersecurity, machine learning, Web-scrapped features, classification, clustering, adversarial attack.

I. INTRODUCTION

IN RECENT years, the network of Web pages has grown faster with the expansion of the Internet. Online services, business, banking, and online marketing have made the Internet an integral part of our lives. Despite the numerous advantages of this platform for online advertising, it has also become a primary source of malicious activities. Attackers deliberately put malicious links in online advertisements that, when visited, redirect the users to unauthorised websites. Attackers make it easy for people to be steered to phishing or malware websites to steal their confidential data, make a fast buck, or defraud them by injecting harmful code into these websites. Such illegal activities cost billions of dollars each year [1]. Most of these harmful websites seem identical to genuine websites, and the user cannot distinguish between them easily. To minimise the effects of this scam, organisations and enterprises are investing significant money in keeping their systems secure against these harmful links and URLs.

The researchers made several attempts to distinguish the malicious advertisement URL, also referred to as malicious URL in this study, using statistical analysis and the popularity feature of the domain name [2]. Though most phishing and malicious sites have a short lifespan, these features may not be available to them for analysis. As a result, the rate of misclassification rises. The creation of new URLs daily makes the job more challenging. There are currently two primary trends in identifying malicious URLs: the first is detecting malicious URLs based on the rule’s set, and the second is based on a behavioural analysis approach. The approach based on rules set can efficiently and precisely detect harmful URLs. However, this strategy cannot recognise the latest malicious advertisement URLs which do not match any of the specified indicators. Deep learning (DL) and ML algorithms have wide use in categorising malicious URLs based on their behaviour analysis techniques. Classification tasks across a wide range of domains have been found to perform better with DL approaches than with feature-based learning approaches. Nevertheless, the performance of every DL model is determined by a variety of aspects, such as hyper-parameter tuning, neural network (NN) design, and more. In non-complex problems, recursive reinforcement learning (RL) may lead to overload of states which may show the adverse outcomes.

This paper investigates the use of ML-based approaches for malicious advertisement URL detection and the performance analysis of ML models against the adversarial attack. For this, we systematically investigated novel feature extraction from URLs, supervised and unsupervised ML techniques for the classification of malicious and benign URLs, i.e., ensemble trees and K-Means, respectively. We consider a new research area in ML termed adversarial machine learning (Adv-ML),...
which illustrate the detection of adversarial attack or inputs made specifically to fool classifiers, and the suitable defences methodologies. Here, we restrained this study to malicious URL detection framework and examined the robustness of our classifiers against the exploratory attack. In future, we will discover suitable defence strategies against various exploratory and causative adversarial attacks on the ML models. During the testing phase, we considered exploratory attacks, also called evasion attacks, due to their popularity and being more straightforward to employ for adversaries [3]. Most of the attacks Adv-ML ever developed belong to the group of exploratory attacks. Evasion attacks limit the attacker’s ability to modify test data but disallow modification of training data. In causative attack scenarios, the attack can disrupt the training phase, also known as poisoning attack [3].

A. Contribution

The following is a summary of our contributions:

- We perform our experiment on twelve diversified malicious and non-malicious advertisement URL datasets that differ in structure and orientation and have a range of statistical properties. For further evaluation, we examine two distinctive and innovative classes of features, namely: lexical and Web-scrapped feature groups.
- We utilize supervised ML to evaluate the efficacy of four distinct classifiers, namely Random Forest, Gradient Boost, AdaBoost, and XGBoost, while taking various parameters into account. We also employ K-Means clustering algorithm of unsupervised ML to integrate the result in a visual comprehension with the formation of clusters. The K-Means clustering easily adapts to new examples and scales to large datasets. We provide a comprehensive study for the first time to acknowledge which classifiers best fit the model and have the highest detection accuracy and least FPR and FNR.
- We provide a comprehensive analysis by examining the performance of supervised and unsupervised classifiers on balanced matched and mismatched datasets. We achieved the highest detection accuracy up to 99.63% and a clear distinction between malicious and benign classes of URLs in the clustering process.
- We evaluate the robustness of the classifiers, also referred to as detectors, against recent attack methodology based on gradient attack. We are the first group to study the effect of ZOO adversarial attack on ensemble trees for the malicious URL detection. We considered the ZOO attack from the black-box setting, that proposed so far to apply the attack on ensemble trees and employed it during the testing phase on 3000 manipulated examples in an account for the limited knowledge scenario. We systematically examine how partial adversarial control and knowledge affect the susceptibility of ML models to test-time attacks.

B. Organisation

The rest of our paper is organised as follows. We overview the related works discussing malicious URL detection and adversarial attack in Section II. Then, we provide the statistical properties of our datasets, background information on the classification approaches, and adversarial attack in Section III. Afterwards, we provide our methodology and the experimental setup in Section IV. Section V presents the experimental results, followed by the discussion evaluating the effectiveness of our classifiers and adversarial attack. Finally, we conclude our paper and provide promising future research work in Section VI. Our code and all the datasets are available on GitHub [4].

II. RELATED WORK

Significant research has been conducted earlier to detect malicious advertisements and phishing websites in various ways. Prakash et al. [5] used the traditional approach of blacklisting malicious ads-related sites using lexical feature analysis that gives out the list consisting of only known malicious URLs. Although it is a powerful feature indication, it takes several hours for a malicious URL to ban, and its implementation is infeasible. He et al. [6] and Ma et al. [7] later proposed URL analysis based on lexical aspects and statistical properties, respectively. These feature groups focused on URL’s string properties. The major limitation of the above systems was that they were ineffective in identifying similar pattern URLs and not trained on the distributed systems. Later, Antonakakis et al. [8] and Liu et al. [9] took new innovative Host-based, content and popularity-based features into consideration. This approach utilise old DNS system information as a seed having the issue of Web page scalability.

Based on the ML approaches, several fraudulent advertisement URL detection systems were created using logistic regression (LR), Nave Bayes, decision trees (DT), and ensembles. Garera et al. [10] research is based on LR, with a few variables incorporated in the extraction to categorise harmful URLs and have the possibility of spoofing the similar tokens. They use 18 features, including Google Web page quality and rank, achieving the detection accuracy of 97.3% over 2500 URLs. Fette et al. [11] classify phishing emails using statistical approaches from ML. The classifiers examine the structure and content of the email and the implementation of the work is not optimal. Later, the technique was improved by incorporating text classification algorithms to examine email content. Abu-Nimeh et al. [12] compare multiple classifiers on a database of phishing emails, using the frequency of the corpus’s top 43 keywords as attributes. Numerous tools are also available that can detect malicious URLs, such as URL Void, UnMask Parasites, Norton Safe Web, Google Safe Browsing Diagnostic, SiteAdvisor, VirusTotal, and Browser Defender. However, their effectiveness is limited due to their signature-based nature.

Systematic research was carried out by Carlini and Wagner [13] on the adversarial attack on DL models. This study produced adversarial examples by addressing an optimization problem to minimize perceptual distortion while inducing misclassification. Chen et al. [14] conducted an investigation of the robustness of deep NNs when they are exposed to small simulated perturbations. The adversarial
examples generated in this study were used to target the convolutional neural network (CNN)-recurrent neural network (RNN)-based picture captioning model. In this context, the adversarial robustness of ML models was also investigated. Gressel et al. [15] adopt one DL model to compare the effectiveness of their approach to various ML models. Using authentic adversarial phishing sites that seem just like their unaffected counterparts, he targets six tabular ML models with an average success rate of 13.05%. In a similar study, Sabir et al. [16] comprehensively analysed the robustness of ML-based phishing URL models by considering traditional ML models. To hide a set of nodes from community detection, Li et al. [17] generated an adversarial network by attacking surrogate models based on graphs of neural networks (GNNs).

We give a comparison that highlights existing research and our study on Adv-ML in terms of detection using various ML and DL applications in Table I. To the best of our knowledge, previous research has not taken into account such a wide range of datasets and feature groups in the URL feature extraction method. Previous research has only focused on the malicious URL detection framework, not on adversarial attacks on ensemble classifiers. Therefore, we aim to test the robustness of our ensembles using the exploratory attack scenario.

III. BACKGROUND

Here, we lay out the background information of datasets used in our experiment in Section III-A, statistical properties in Section III-B, and pre-processing task in Section III-C. We will also talk about classification techniques in Section III-D and the setting of an adversarial attack in Section III-E.

A. Datasets

The experiment setup for advertisement URLs from 12 distinct datasets include 3980870 URLs. We considered two kinds of URLs from these datasets, i.e., benign and malicious. Furthermore, the malicious URL dataset includes four distinct sub-categories: spam, defacement, malware, and phishing. We also examined all the URLs using the VirusTotal [18] tool to confirm their authenticity. Each benign URL is labelled with ‘0’ and malicious as ‘1’ in the datasets. Further information on the datasets is given below.

1) Benign Dataset: The benign URLs were gathered from available sources on the Internet. Table II gives an overview of the six benign URL datasets. In Alexa, [19] traffic rating is used to rate the URLs, which is calculated by a combination of the browsing behaviour of online users, the total unique visitors, and the number of site traffic. CrowdFlower [20] is a large-enhanced dataset of categorised websites; contributors visited supplied links and chose a primary and sub-category for URLs. DMOZ [21] is a big open directory that is communally managed and categorises Web material. The information is presented in a sophisticated XML format. Benign set URL [22] and Non-malicious URL [23] are the open-source datasets available on Kaggle. ISCX-URL-2016 [24] is produced by
the research and development unit Canadian Institute for Cybersecurity [25]. If we talk about the format of URLs, Alexa and CrowdFlower datasets have sub-domain and domain in the URL, DMOZ and Benign set URLs (Benign) have protocol and hostname, and Non-malicious_url and ISCX-URL-2016 (Benign) have all URL components.

2) Malicious Dataset: We consider four kinds of malicious URL datasets from various open sources on the Internet: spam, defacement, phishing, and malware. Spam URLs cause serious harm to the computer and are generally sent with the intent of advertisement. Defacement URLs are the changed visual aspects and some contents of the website. Phishing URLs tempt user to visit fake website and try to steal the information. Malware URLs take user to malicious website that installs malware on the devise in order to corrupt or theft. Table III contains the summary of the six malicious URL datasets. As an open community website, Phishtank [26] is a free service for sharing phishing URLs, and users can send suspicious URLs to Phishtank for verification. Cisco Talos Intelligence Group updates data on this website on an hourly basis. Phishstrom [27] is a malicious dataset with features built and used for evaluation in the paper [28]. Phishing Site URL [29], Malicious data URL [23], and Malicious set URI [22] are the open-source datasets available on the Web. ISCX-URL-2016 (malicious) [24] dataset of malicious URLs contains four different sub-categories of malicious URLs. All of the malicious datasets in this section have complete URL components.

The six benign and six malicious URL datasets are combined to create six balanced datasets which contain an equal number of malicious and benign URLs with a split size of half of the benign data. The combined information from the merged datasets is also displayed in Table IV.

B. Statistical Characteristics of the Datasets

For better understanding, we perform a statistical study of our data based on lexical properties. In this analysis, we discovered that the length of malicious ad URLs has a broader range of dispersion than benign URLs. The majority of benign URLs are between 25 and 50 characters long, with an average length of 44.28 characters. The average number of special characters in a benign URL is 8.64, with a path length of 17.54 characters. Fig. 1 shows the frequency distribution of benign URL lengths.

The majority of malicious ad URLs are between 25 and 105 characters long, with an average length of 63.14 characters. Malicious URLs have 13.98 special characters on average and a path length of 42.60 characters. We also discovered that less than 2% of malicious ad URLs use IP as a domain name. Fig. 2 presents the frequency distribution of malicious URLs.

C. Preprocessing of Datasets

Our prototype platform’s implementation accepts a specified shape of the input dataset. Therefore, for more precise outcomes, the dataset must be modified and reshaped to meet our reliance’s required input format. We scale the data using the interquartile range method during the preprocessing stage. All redundant and raw-valued cells were removed, and repeated hostname URLs were excluded. The URL datasets were then shuffled, and samples were taken from the datasets for further investigation.
D. Classification Techniques

The classification method is a supervised learning technique that uses training data to identify the category of new observations. Predictive models train from a given collection of observations and afterwards categorize subsequent observations into one of many classes. Boosting is an ensemble modelling approach that aims to construct a robust classifier from many weak classifiers by building a model by sequentially connecting weak models. First, a model is constructed using the training data. Then the second model is constructed to address the faults in the previous model. This process is repeated again and again until the entire training data set is adequately predicted.

1) Random Forest: A Random Forest is a large collection of decision trees that differ slightly from the others. Random Forest is based on the premise that while every tree may predict quite well, it almost certainly overfits some data. The important parameter to alter are n_estimators, max features, and pre-pruning settings such as max depth.

2) Gradient Boost: Gradient boosting is another ensemble approach that combines numerous decision trees to generate a more significant model. Unlike the Random Forest method, Gradient Boosting works by successively creating trees for every tree, attempting to fix the flaws of the preceding one.

Gradient Boost tree models depend on three primary parameters: n_estimators, number of trees, and learning rate that deduce the extent by which the new tree corrects the errors of the previous one.

3) AdaBoost: Another ensemble method in ML, AdaBoost, or Adaptive Boosting, is a boosting algorithm. During the data training period, it generates n decision trees. When the first decision tree/model is constructed, the incorrectly classified record in the first model is prioritised. Only these records are sent to the second model as input. The process is repeated until we specify how many base learners we want to create. With all boosting techniques, record repetition is permitted. The incorrectly classified record is used as input for the next model.

4) XGBoost: XGBoost enhances performance and speed aspects of Gradient Boost decision trees implementation. Models using basic, predictive functions will be favoured by the regularised objective. Here, the model is taught additively instead of the traditional optimization method for the tree ensemble model in Euclidean space. Due to its simultaneous and distributed processing, XGBoost is faster than other algorithms.

E. Adversarial Attack

Adv-ML technique aims to exploit models by developing harmful attacks using accessible model information [30]. The main reason for exploitation is to affect ML and fail it. Here, we will examine the adversarial attack on our models. It has been discovered that an intelligent malicious URL-generating system can generate URLs that can pass ML classifiers at a low rate. White-box, gray-box, and black-box attacks are the three main types of adversarial attacks.

- **White-Box Attack:** In a white-box attack, the attackers have complete access to the model’s architecture, parameters, and weights. L-BFGS [31] black-box constrained method to minimise the additive perturbation based on classification restrictions and Fast Gradient Sign Method (FGSM) [32] followed by the primary iterative method of adding perturbations iteratively. Other white-box adversarial attacks include JSMA, BIM & ILCM, ATNs, and DeepFool.

- **Black-Box Attack:** In black-box attacks, the attacker has no access to the system. The system’s parameters and weights cannot be obtained here. ZOO may directly approximate the gradients of the target network using zeroth order stochastic coordinate descent. Other examples of black-box attacks are One-pixel, UPSET, and ANGRI.

- **Grey-Box Attack:** The grey-box attacks use only training access to the target model to create adversarial samples. Counter-Forensic Attack (CF): Here, we shall formulate the universal adversarial paradigm for CF attacks in this text. An adversarial model is characterised by demonstrating the adversary’s goal, the system’s knowledge, and the capacity to corrupt the system through data manipulation [3], [33].

- **Attacker’s goal:** It describes the type of security violation and the type of error the attacker seeks. CF attacks are often either integrity violations or evasion attacks. The adversary’s goal determines the loss function that the adversary seeks to maximise.

- **Attacker’s knowledge:** This deals with if the attack has a limited amount of knowledge about the model or perfect knowledge (PK). The attacker is fully aware of the forensic algorithm in a PK scenario and aware of a few details or settings relevant to the forensic algorithms in a restricted knowledge scenario.

- **Attackers capability:** It refers to the adversary’s level of control over training and testing data. The attacker’s capability in the exploratory attack paradigm is restricted to modifications to test data, whereas modifications to training examples are still not permissible. The attack can interrupt the training process in a causative attack scenario, referred to as a poisoning attack.

This article considers the exploratory attack scenario by applying the ZOO attack. The ZOO attack is a version of the Carlini & Wagner (C&W) attack [13] that employs ADAM coordinate descent to perform numerical gradient estimates. As with any tree attack, there is the risk of misclassifying objects or data. It can also involve picture captioning, voice recognition misclassification, and data regression effects. Because a white-box attack relies on the model’s overall knowledge growth, this black-box attack has just an LK of the targeted model. It has an unknown training technique as well as data. It also includes unidentified output classes and model confidence.

Many research challenges are concerned with complex data creation processes that can provide types of measurements. In black-box models, these challenges are incorporated into ZOO. The ZOO Attack, also known as Derivative-Free Optimization, solves problems that do not have direct access to gradients.

IV. Methodology

In this section, we present our working mechanism with feature extraction procedure in Section IV-A, empirical study
in Section IV-B, and adversarial attack on our detection models in Section IV-C.

A. Feature Extraction

This phase of research aims to gather important information about the URL string. It entails a set of features with distinguishing characteristics used to specify different dataset categories. Since we wish to differentiate between malicious and benign advertisement URLs, we do this work using two key feature categories: lexical and Web-scraped features.

1) Lexical Features: We used the fundamental features from the URL string for easy demonstration and some advanced aspects for an unconventional approach in this set of features. These are further classified into two types: linguistic features and human-engineered features. Linguistic features comprise essential URL string qualities such as length-based features (length of the URL string, domain length, etc.), presence of an object (exe presence in the string, etc.), count-based features (alphabet, special character count, etc.), and TLDs. In this scenario, features such as the digits-to-alphabet ratio are also relevant because most malicious algorithmically created domains have considerably more digits than alphabets. Aside from the bag-of-words features, statistical features help significantly in gaining a quick understanding of the path, query, and URL length.

If the domain name has any credibility in terms of being a meaningful, easy-to-pronounce, and random string, as proven in Algorithm 1, Human-Engineered Features perform well enough in this classification task. We considered the U.K./U.S. English dictionary to state the domain name as a meaningful, pronounceable or random string in the identification process.

Algorithm 1 Check Domain Is English Language Word

\begin{algorithm}
\begin{algorithmic}
\State \textbf{INPUT:} domain_name
\State \textbf{OUTPUT:} Meaningful, English word, Pronounceable, Random string
\State break domain_name in words = words[]
\If {domain_name belongs to wordnet.synsets}
\State Return Meaningful
\Else
\If {words[]} is not NULL
\State Check for parts of speech
\If {words[]} has Noun, Pronoun
\State Return Meaningful
\ElseIf {words[]} has Verb, Adjective
\State Return Pronounceable
\Else
\State Return Random string
\EndIf
\EndIf
\EndIf
\end{algorithmic}
\end{algorithm}

We also check whether the domain name is present in the list of suspected lists. Table V gives a complete list and overview of lexical features.

2) Web-Scraped Features: In this feature group, we extract the Web information of the domain name and deal with the obfuscation issue. Some malicious URLs overlap lexical features with benign URLs. As a result, Web scraping of URLs will offer us the required solution. We want to identify ways to spoof a domain name in URL segmentation. Whence, with Python, the domain name is sent as a query request in the Google search engine, and the first 60 results are collected for the process of hit counts. If the exact domain name exists, it is recorded as a count. This approach is a little time-consuming but significantly influences the straightforward categorisation procedure. If a similar name is found in netloc, it is an excellent chance to be a clean domain name.

In Deep-Web Features, we create a list of 11 distinct types of typosquats of the searched domain name (see Table VI) and examine all registered domain names among them. We also created a fuzzer for this step, which attempts to find typosquat domain names for each searched domain name. The phishing domains can be identified using the above technique. We also computed the Lavenstein distance and Shannon entropy with the most comparable query request results and saved it as a feature using the same method. The Lavenstein distance between two words is the number of single-character modifications necessary to transform one term into another. In the Host-based Features, we included URL host-name attributes such as WHOIS information, IP address, location details, and country code. As these attributes have identity-related qualities, they are recorded in numerical vectors with unique identities. For malicious URLs, the time-to-live from domain registration is nearly instantaneous. Many of them employed botnets for hosting for themselves on machines spread across various countries. As a result, host-based characteristics play an essential role in detection. When the Web page is completely downloaded, Content-Based Features are gained. Our research extracts HTML document-level information of the Web page, i.e., total images, the total links, and different tags from the source code. If malicious URLs remain undetected by URL-based features, then these attributes will function effectively in detecting threats by analysing the Web page.

B. Empirical Study

This section describes our suggested supervised URL classification and unsupervised clustering approaches for detecting fraudulent advertisement URLs. We intend to assess if a given URL is benign or malicious by making binary classification problem. Consider the following collection of N URLs: \((u_1, y_1), \ldots, (u_N, y_N)\). Here, \(n = 1, \ldots, N\) specifies a URL, and URL label is denoted by \(y_u\), with \(y = 0\) denoting the benign URL and \(y = 1\) denoting the malicious URL. The first step is to represent the features \(u_n \rightarrow a_n\), where \(a_n\) is \(n\)-dimensional feature vector corresponding to the URL \(u_n\). The problem can be expressed as a function \(y_u = \text{sign}(L_f(a_n))\). We may reduce the overall number of errors in the classification process by...
minimising the loss function. We retrieved 89 features from each URL and processed them for the classification phase. A balanced dataset of both benign and malicious URLs is used to detect fraudulent URLs. We used 1000 URLs from each of our six balanced merged datasets to train our four classifiers with appropriate parameters. Depending on the simulation task, we used a split size of around 0.7 for training and 0.3 for testing. In contrast to prior research that only considered a restricted element of the feature categories, we created a unique set of these highly influential feature categories and attempted to combine a good combination. For supervised ML methods, we used Random Forest, Gradient Boost, AdaBoost, and XGBoost approaches. For each of our datasets, we examined the relative performance of all models. Then, to study the detection accuracy behaviour, we trained our models on mismatched datasets, training the complete one dataset and evaluating it on the other five. This approach tests the effectiveness of responding to unseen features, which means that during the training process, unnoticed features in different featured datasets, the model cannot react to the test data. While in the case of seen features, a particular feature is noticed, and while testing with a similar feature, the model can respond to it. This procedure is carried out on all datasets with all model selections. We took all of the necessary criteria for the categorisation stage, as discussed in the previous section.

The process of grouping data points, such as similar data points placed in the same group, and dissimilar from other data points in different groups is known as clustering. We employed the K-Means clustering algorithm of unsupervised ML to integrate the results in a visual comprehension by forming clusters of malicious and benign data [34]. The K-Means clustering easily adapts to new examples and scales to large

---

**TABLE V
LEXICAL FEATURE GROUP**

| Sr. No. | Feature Subgroup | Feature Name | Data Type | Feature Description |
|---------|------------------|--------------|-----------|---------------------|
| 1       | URL Length       | URLLength    | Integer   | Compute length of the URL, e.g., the URL length is 51 of https://www.example.com所以他tools/count-chars |
| 2       | CheckIPHostName | CheckIPHostName | Binary  | Check if IP address is used as hostname e.g., 192.168.0.1 |
| 3       | CheckXSD        | CheckXSD     | Binary   | To look the presence of .xsl in URL |
| 4       | DigitAlphabetRatio | DigitAlphabetRatio | Float  | To compute the ratio of number of digits to alphabets in URL |
| 5       | SpecialCharAlphabetRatio | SpecialCharAlphabetRatio | Float  | To compute the ratio of number of special characters to alphabets in URL |
| 6       | UppercaseLowercaseRatio | UppercaseLowercaseRatio | Float  | It computes ratio of uppercase characters to lowercase characters in URL |
| 7       | DomainURLRatio  | DomainURLRatio | Float   | To compute the ratio of domain length to URL length |
| 8       | NumericCharCount | NumericCharCount | Integer | Number of numeric character viz., 0, 1, 2, ..., 9 |
| 9       | EnglishLetterCount | EnglishLetterCount | Integer | Number of English letter viz., a, b, c, ..., z and A, B, C, ..., Z |
| 10      | SpecialCharCount | SpecialCharCount | Integer | Number of special character viz., !, $, %, etc. |
| 11      | DomainCount     | DomainCount  | Integer   | Number of "-" characters |
| 12      | SemiColonCount  | SemiColonCount | Integer   | Number of ";" characters |
| 13      | UnderScoreCount | UnderScoreCount | Integer   | Number of "_" characters |
| 14      | QuotMarkCount   | QuotMarkCount | Integer   | Number of "'" characters |
| 15      | HashCharCount   | HashCharCount | Integer   | Number of ";" characters |
| 16      | EqualCount      | EqualCount   | Integer   | Number of "=" characters |
| 17      | PercentCharCount | PercentCharCount | Integer | Number of ";" characters |
| 18      | AmpersandCount  | AmpersandCount | Integer   | Number of ";" characters |
| 19      | DashCharCount   | DashCharCount | Integer   | Number of ";" characters |
| 20      | DelimeterCount  | DelimeterCount | Integer   | Number of the delimeter characters in URL viz., (, ), {, }, [, ], etc. |
| 21      | NaNCharCount    | NaNCharCount | Integer   | Number of "NaN" characters |
| 22      | TitleCharCount  | TitleCharCount | Integer   | Number of "~" characters |
| 23      | DoubleSlashCount | DoubleSlashCount | Integer | Number of "//" characters in the link path |
| 24      | IsHashed        | IsHashed     | Binary    | To check if URL is hashed (If any hash function is used to convert URL) |
| 25      | TLD             | TLD          | String    | To look for top level domain of URL e.g., com, us, org |
| 26      | DistinctDigitalAlphabet | DistinctDigitalAlphabet | Float  | Distance between number to alphabet |
| 27      | HttpsInUrl      | HttpsInUrl   | Binary    | Check the presence of https in URL |
| 28      | FileExtension   | FileExtension | String  | To check the extension of the file in URL |
| 29      | TLDInSubdomain  | TLDInSubdomain | Binary  | Check whether subdomain have TLD or ccTLD as its part |
| 30      | TLDInPath       | TLDInPath    | Binary    | Check whether subdomain have TLD or ccTLD in link of URL |
| 31      | HostNameLength  | HostNameLength | Integer | Check for the disarrangement of https in URL e.g., ‘www.wordforhttps.com’ |
| 32      | PathLength      | PathLength   | Integer   | Path length of URL |
| 33      | QueryLength     | QueryLength  | Integer   | Length of the query in URL |
| 34      | DistWordBased   | DistWordBased | Binary   | Check if URL is contain anonymous words (personal, bin, abuse etc.) |
| 35      | URLWithoutwww   | URLWithoutwww | Binary  | To check if www is present in the URL |
| 36      | FTPUsed         | FTPUsed      | Binary    | Check if "ftp://" is there in the URL |
| 37      | ISPUsed         | ISPUsed      | Binary    | Check if "isp" is there in the URL |
| 38      | FileInURL       | FileInURL    | Binary    | Check if files is there in the URL |
| 39      | CSSUsed         | CSSUsed      | Binary    | Check if "css" is used in the URL |
| 40      | IDomainEnglishWord | IDomainEnglishWord | Binary  | Check if domain name is English language word as per US/UK dictionary |
| 41      | IDomainMeaningful | IDomainMeaningful | Binary  | To check if domain name is English language word and meaningful |
| 42      | IDomainPronounceable | IDomainPronounceable | Binary  | To check if domain name can be pronounced |
| 43      | IDomainPossess | IDomainPossess | Binary   | To compute the randomness of the URL string |
| 44      | Um switch       | Um switch    | Float     | To calculate uni-gram probability |
| 45      | Bigram          | Bigram       | Float     | To calculate bi-gram probability |
| 46      | TriGram         | TriGram      | Float     | To calculate tri-gram probability |
| 47      | SensitiveWordCount | SensitiveWordCount | Integer | Sensitive words in URL (i.e., "secure", "account", "webcam", "login" etc) |
| 48      | InSuspiciousList | InSuspiciousList | Binary  | To check if URL is present in suspicious list from malicious dataset |

---

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
TABLE VI
WEBS-SCRAPED FEATURE GROUP

| Sr. No. | Feature Subgroup | Feature Name                  | Data Type | Feature Description                                                                 |
|---------|------------------|-------------------------------|-----------|-------------------------------------------------------------------------------------|
| 1       | Deep-Web Features| LevenshteinDistance           | Float     | To calculate Levenshtein distance                                                  |
| 2       | Deep-Web Features| Entropy                       | Float     | To calculate shannon entropy of the URL                                            |
| 3       | Deep-Web Features| HyphenString                  | String    | To get hyphenated domain name of URL for typosquat                                 |
| 4       | Deep-Web Features| Homonglyph                    | String    | To get homonglyph string for typosquat                                             |
| 5       | Deep-Web Features| Vowel                         | String    | To get vowel swap string for typosquat                                             |
| 6       | Deep-Web Features| Bitquatting                   | String    | To get bitquatting string for typosquat                                             |
| 7       | Deep-Web Features| InsertionString               | String    | To get insertion string for typosquat                                              |
| 8       | Deep-Web Features| Omision                       | String    | To get omission string for typosquat                                               |
| 9       | Deep-Web Features| Repetition                    | String    | To get repetition string for typosquat                                              |
| 10      | Deep-Web Features| Replacement                   | String    | To get replaced string for typosquat                                               |
| 11      | Deep-Web Features| Subdomain                     | String    | To get subdomain string for typosquat                                              |
| 12      | Deep-Web Features| Transposition                 | String    | To get transposition string for typosquat                                           |
| 13      | Deep-Web Features| AdditionString                | String    | To get addition string for typosquat                                               |
| 14      | URL Segmentation | GoogleSearchFeature            | Integer   | Array of top 60 results of URL querying the Google search engine                    |
| 15      | URL Segmentation | IPAddress                     | Numeric   | To get the IP address of the host name                                             |
| 16      | URL Segmentation | ASNNumber                     | Numeric   | To get the ASN number of the URL                                                   |
| 17      | URL Segmentation | ASNCountryCode                | Numeric   | To get ASN country code                                                          |
| 18      | URL Segmentation | ASN_CIDR                      | Numeric   | To get the CIDR number of the host name                                            |
| 19      | URL Segmentation | ASNPostalCode                 | Numeric   | To get ASN postal code                                                             |
| 20      | URL Segmentation | ASNCreationDate               | Numeric   | To get the creation date of the domain name                                        |
| 21      | URL Segmentation | ASNUndatedDate                | Numeric   | To get the updation date of the domain name                                         |
| 22      | URL Segmentation | DomainAgeInDays               | Numeric   | The duration of a domain’s registration                                             |
| 23      | Content-Based Features | ImgCount                   | Integer   | To count the images in the webpage                                              |
| 24      | Content-Based Features | TotalInlinks                 | Integer   | To count links in the webpage                                                     |
| 25      | Content-Based Features | NumParameters               | Integer   | Number of parameters of the URL                                                  |
| 26      | Content-Based Features | NumSegments                  | Integer   | Number of segments of the URL                                                    |
| 27      | Content-Based Features | BodyTagCount                 | Integer   | Counting the body tag in webpage’s html source code                                |
| 28      | Content-Based Features | MetaTagCount                 | Integer   | Counting the meta tag in webpage’s html source code                                |
| 29      | Content-Based Features | DivTagCount                  | Integer   | Counting the div tag in webpage’s html source code                                 |
| 30      | Content-Based Features | FakeLinkInStatusBar          | Binary    | Check whether display of fake URL is command on MouseOver                           |
| 31      | Content-Based Features | RightClickDisabled           | Binary    | Check the presence of command to disable right click                              |
| 32      | Content-Based Features | PopUpWindow                  | Binary    | Check the presence of command to start popup window                                |
| 33      | Content-Based Features | CheckEmailAddress           | Binary    | Whether ‘@’ is present in HTML source code                                         |
| 34      | Content-Based Features | CheckFrameTag                | Binary    | Whether frame or iframe used in HTML source code                                   |
| 35      | Content-Based Features | TitleCheck                   | Binary    | In HTML source codes, see whether title tag is empty                               |
| 36      | Content-Based Features | SourceEvalCount              | Integer   | Count of eval () functions in HTML source code                                    |
| 37      | Content-Based Features | SourceEscapeCount            | Integer   | Count of escape () functions in HTML source code                                    |
| 38      | Content-Based Features | SourceExecCount              | Integer   | Count of exec () functions in HTML source code                                    |
| 39      | Content-Based Features | SourceSearchCount            | Integer   | Number of search () functions HTML source code                                     |
| 40      | Content-Based Features | ImageOnlyInHtml              | Binary    | Check whether only image are present in HTML source code                            |

datasets. This method classifies the objects into $K$ groups of similarity, with the expected optimal value of $K$ equal to 2 for exact results using the elbow method. The Euclidean distance is used as a measurement to calculate that similarity. We cycle through the $K$ values from 1 to 10 to calculate the distortion for each $K$ value in the defined range. To get the optimal number of clusters, we must determine the value of $K$ at the elbow or the point where the distortion begins to drop linearly. However, we can also use other clustering algorithms like DBSCAN, but in DBSCAN the number of cluster forming need not be specified and can not handle high dimensional data efficiently.

C. ZOO - Adversarial Attack

This section will investigate our ensemble’s vulnerability to adversarial attacks. Unlike white-box attack approaches, which need the target network to be differentiable, in black-box ZOO attack, the gradients are estimated using zeroth order stochastic coordinate descent of the targeted network. Here, for this reason, we will select a loss function that is only dependent on the output of the targeted network to compute the gradient by using the finite difference method. We will use ZOO to solve the optimization problem. Here, for the input vector $a$, the loss function $L_f$ can be defined for output $O_f$ as

$$L_f(a, b) = \max \left\{ \max_{j \neq b} \log [O_f(a)]_j - \log [O_f(a)]_b, -\rho \right\},$$  

(1)

where $\rho \geq 0$ and $\log a \to -\infty$ whenever $a \to 0$. Because the log function is a strictly monotonic function, $\max = 0$ follows. As a result, $a$ has the greatest confidence score for targeted class label $b$. Strictly monotonic functions are those functions which, for $x > y$ satisfy $f(x) > f(y)$. Here, the dominant influence is decreased by the log operator and maintains the confidence order due to monotonicity. For untargeted attacks, if $a$ is classified as any different from the original label $b_0$, then an adversarial attack will be considered successful. For untargeted attack, the most probable predicted class after eliminating $b_0$ is

$$L_f(x) = \max \left\{ \log [O_f(a)]_b - \max_{j \neq b_0} \log [O_f(a)]_j, -\rho \right\},$$

(2)

where original class label for $a$ is denoted by $b_0$, and the most probable predicted class is represented by $\max_{j \neq b_0} \log [O_f(a)]_j$ other than $b_0$. The attack employs the optimization technique for the general $L_f$. If a function's
Algorithm 2 SCD - Stochastic Coordinate Descent

Initialise \( l \leftarrow 0; \)

\[ \text{WHILE} \ 1 \leq K \text{ DO} \]

Choose \( j(l) \) out of \( \{1, \ldots, m\} \) with same probability

Evaluate and update \( \eta^* \)

\[ \arg \min_{\eta_j} \ L_f (a + \eta e_j) \]

Updating \( a_j \leftarrow a_j + \eta^* \)

\[ l \leftarrow l + 1 \]

\[ \text{END WHILE} \]

Algorithm 3 Coordinate-Wise ADAM Zeroth Order SCD

\[ \text{Requirement:} \ \text{Set step size} \ h, \ \text{ADAM hyper-parameters} \ \alpha_1 = 0.9, \ \alpha_2 = 0.99, \ \epsilon = 10^{-8}, \ \text{and set ADAM states} \ N, \ \tau \in \mathbb{R}^P \]

Initialise \( N \leftarrow 0, \ \tau \leftarrow 0, \ U \leftarrow 0 \)

\[ \text{WHILE} \ \text{diverges} \ \text{DO} \]

Choose \( j(l) \) out of \( \{1, \ldots, m\} \) with same probability

Evaluate \( U_j \leftarrow U_j + 1 \) and approximate \( g_j \) using (3)

Evaluate \( \hat{N}_j \leftarrow \alpha_1 N_j + (1 - \alpha_1) g_j \)

Evaluate \( \tau_j \leftarrow \alpha_2 \tau_j + (1 - \alpha_2) g_j^2 \)

Evaluate \( \hat{N}_j = N_j / (1 - U_j), \ \check{\tau} = \tau_j / (1 - U_j) \)

\[ \eta^* = -h \frac{\hat{N}_j}{\sqrt{\check{\tau} + \epsilon}} \]

Update \( a_j \leftarrow a_j + \eta^* \)

\[ \text{END WHILE} \]

The symmetric derivative exists at point \( a \), it is symmetrically differentiable at that point. Therefore, the symmetric difference quotient is:

\[ \tilde{g}_j := \frac{\partial L_f (a)}{\partial a_j} \approx \frac{L_f (a + k e_j) - L_f (a - k e_j)}{2k}, \quad (3) \]

for gradient estimation. Here, we use \( k \) as a very small constant and standard basis vector is denoted by \( e_j \) where only the \( j \)th component have value \( 1 \) and all other components equal to \( 0 \). Although numerical precision is important, predicting the gradient precisely is typically not required for efficient adversarial attacks. In the coordinate descent approach, one variable is chosen at random at each iteration and changed by minimising the objective function along that coordinate. We can obtain the estimated optimum delta by estimating the gradient and Hessian for \( a_j \). Here, Hessian estimate is

\[ \hat{h}_j := \frac{\partial^2 L_f (a)}{\partial a_j^2} \approx \frac{L_f (a + k e_j) - 2L_f (a) + L_f (a - k e_j)}{k^2}, \quad (4) \]

The stochastic coordinate descent method in Algorithm 2 and the ADAM optimizer in Algorithm 3 increased the speed of the ZOO attack [35]. The ZOO attack is more dependable to utilise than frequently used gradient-based techniques.

V. RESULTS AND DISCUSSION

Here, we will summarise and discuss our study results by inspecting the effectiveness of several classifiers, clustering techniques, and the effects of adversarial attacks on ensembles. To assess our suggested system, we run a set of rigorous tests on a desktop computer. The system’s setup is Intel Core Intel Core i5-4300M 2.6GHz, Ubuntu 20.04 LTS with 4GB RAM, and Python version 3.8.10.

A. Experimental Results

We pick 1000 URLs from each balanced dataset with an equal number of malicious and benign advertisement URLs for the experiment. The experiment is performed on each of our four classifiers with 5 and 10 folds. We also consider the runtime complexity, which grows as the number of folds increases. It is a problem throughout the training stage but not during testing. Each of the six datasets took roughly 150 minutes to process. However, the model’s remarkable detection accuracy is worth it in the end. We used Python’s Scikit-learn package and the GridSearchCV library function to cycle over predefined hyperparameters, selecting 1, 100, 200, 500, 1000, and 1500 \( n \_\text{estimators} \) to fit on our training set. We got more precise results as the number of trees increased. The sampled datasets were examined using the Random Forest model, and we noticed a gain in detection accuracy and precision as we moved to AdaBoost, Gradient Boost, and XGBoost, as well as a drop in FPR and FNR. We discovered that the ten folds of the XGBoost classifier had the best detection accuracy. We created the evaluation matrix using the following formulas.

FPR: The FPR is calculated as the proportion of malicious data that is mistakenly recognised as benign.

\[ \text{FNR} = \frac{FP}{FP + TN}, \quad \text{FNR} = \frac{FN}{TP + FN} \]

Accuracy: Accuracy is calculated as the ratio of accurately predicted examples to total examples.

\[ \text{Precision} = \frac{TP}{TP + FP}, \quad \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \]

1) Detection Results: Here, we discuss our ensemble’s detection accuracy and precision through the confusion matrix. To assess the generalising capability and uniform functioning of our classifiers, we considered matched and mismatched datasets for performance evaluation. This approach of considering mismatched datasets test the model’s ability to adapt properly to new, previously unseen data drawn from the same distribution as the one used to create the model.

- Matched Case: In matched case, the classifiers are trained and tested on the same datasets. Table VII shows the performance of several classifiers as well as the gain in performance as folds increase. This supports the usage of various new features, mainly human-engineered feature group, and also the benefits of URL segmentation. The results suggest that the detectors are improving and behaving consistently; however, there is a minor discrepancy in the FPRs of very few datasets. Dataset 5 exhibits an increase in FPR from 0.0045 to 0.0083 as folds increase which is not similar to the results of other
TABLE VII
MATCH CASE DETECTION EXPERIMENTAL RESULTS

| Dataset   | Folds | Accuracy | Precision | FPR   | FNR   |
|-----------|-------|----------|-----------|-------|-------|
| Dataset 1 | 5     | 98.72    | 98.98     | 0.0104| 0.0152|
|           | 10    | 99.13    | 99.42     | 0.0059| 0.0119|
| Dataset 2 | 5     | 98.78    | 98.77     | 0.0120| 0.0124|
|           | 10    | 99.07    | 99.37     | 0.0062| 0.0125|
| Dataset 3 | 5     | 99.06    | 99.37     | 0.0063| 0.0129|
|           | 10    | 99.13    | 98.86     | 0.0116| 0.0077|
| Dataset 4 | 5     | 98.42    | 98.95     | 0.0107| 0.0208|
|           | 10    | 98.91    | 98.93     | 0.0112| 0.0107|
| Dataset 5 | 5     | 98.84    | 98.86     | 0.0115| 0.0114|
|           | 10    | 99.18    | 98.92     | 0.0112| 0.0057|
| Dataset 6 | 5     | 98.58    | 98.27     | 0.0175| 0.0116|
|           | 10    | 99.07    | 99.38     | 0.0061| 0.0023|

- **Random Forest**

- **Adaboost**

- **Gradient Boost**

- **XGBoost**

- **Mismatched Case:** The classifiers are trained on one full dataset and evaluated on the remaining other five datasets in the mismatched case. Table VIII compares the performance of our classifiers on mismatched datasets. Relatively similar findings are obtained for detection accuracy, precision, FPR, and FNR, as in the matched case, are achieved. Among all ML algorithms, XGBoost is shown to be the high performer in the classification, with superior accuracy and the lowest FNR. With the same hyperparameters as in the matched case, the accuracy of XGBoost ranges between 98.82% and 99.21%. The results show that XGBoost achieves FPR and FNR as low as 0.0060 and 0.0045, respectively. In the multi-class datasets, the performance of the models signifies inferiority from the matched case. Overall, the results demonstrated the classifier’s uniformly high performance while working with a diverse collection of datasets. The results are much better in the mismatched case from the previous studies and investigations.

2) **Clustering Results:** In this section, we distinguish two types of malicious and non-malicious URLs and integrate the result in a visual comprehension with the formation of clusters. Here, we took a random sample of 1000 URLs from the test datasets, balanced the distribution of the two classes, malicious and benign, and extracted the feature vectors of these URLs from one of the layers. After recombining the outputs of the ratio of special characters to URL length and URL character length, we picked the feature vectors and produced another vector. We use the elbow technique to determine the number of clusters that will form to construct based on the retrieved characteristics. The number of clusters to be created, $K$, is varied in the elbow technique. When we examine the graph, we can observe that it rapidly changes at $K = 2$, resulting in an elbow formation. The graph begins to move practically parallel to the X-axis at this point. This point’s $K$ value corresponds to the ideal $K$ value or the optimal number of clusters, as shown in Figure 3(a). By considering 2 clusters, as obtained from elbow method and with error less than 1.7e6, we achieve the best outcomes. We find a clear distinction between the two groups, where red represents benign and blue represents malicious URL.

3) **Adversarial Attack Results:** In Table IX, we compute the attack success rate (ASR) of the ZOO attack on our classifiers. Improved attacks against the targeted ML classifiers may be carried out via ZOO, eliminating the requirement for training substitute models and avoiding the loss of attack transferability. To evaluate the susceptibility of our models, we first trained our classifiers with clean data, and then employed the adversarial attack in the testing phase (referred to as exploratory attacks). Mainly, in this study, we want to
determine if the original robustness assessments pass our binarization test. For this, we considered 50 and 100 confidence of adversarial examples with 0.25 as the step size for numerical estimation of derivatives with a maximum number of 100 iterations in each simulation. We considered 10 coordinate updates to run in parallel and 20 times to adjust the constant with binary search for this attack. Notably, the ASR is higher in the Random Forest and least in XGBoost classifiers with a confidence level of 50 and 100. We got an ASR of 96.67% in Random Forest algorithm, signifying it as a weak classifier.

| Dataset | Random Forest | AdaBoost |
|---------|--------------|----------|
| Trained | Tested       | Accuracy | Precision | FPR | FNR | Trained | Tested       | Accuracy | Precision | FPR | FNR |
| Dataset 1 | Dataset 1 | 98.69 | 98.77 | 0.0121 | 0.0143 | Dataset 1 | 98.70 | 98.89 | 0.0131 | 0.0129 |
| Dataset 2 | Dataset 2 | 98.81 | 98.99 | 0.0063 | 0.0143 | Dataset 3 | 98.93 | 98.81 | 0.0140 | 0.0079 |
| Dataset 3 | Dataset 4 | 98.92 | 98.69 | 0.0129 | 0.0093 | Dataset 4 | 98.84 | 98.81 | 0.0140 | 0.0096 |
| Dataset 4 | Dataset 5 | 98.65 | 98.79 | 0.0119 | 0.0154 | Dataset 5 | 98.99 | 98.85 | 0.0142 | 0.0068 |
| Dataset 5 | Dataset 6 | 98.78 | 98.95 | 0.0103 | 0.0141 | Dataset 6 | 98.77 | 98.89 | 0.0131 | 0.0117 |
| Dataset 6 | Dataset 7 | 98.70 | 98.88 | 0.0111 | 0.0150 | Dataset 7 | 98.86 | 99.06 | 0.0114 | 0.0117 |
| Dataset 7 | Dataset 2 | 98.70 | 98.75 | 0.0123 | 0.0137 | Dataset 8 | 98.92 | 99.12 | 0.0103 | 0.0105 |
| Dataset 8 | Dataset 2 | 98.56 | 98.16 | 0.0181 | 0.0106 | Dataset 9 | 98.96 | 99.29 | 0.0084 | 0.0122 |
| Dataset 9 | Dataset 2 | 98.68 | 98.41 | 0.0157 | 0.0104 | Dataset 10 | 98.87 | 99.25 | 0.0088 | 0.0135 |
| Dataset 10 | Dataset 3 | 98.83 | 98.59 | 0.0138 | 0.0094 | Dataset 11 | 98.88 | 99.20 | 0.0082 | 0.0121 |
| Dataset 11 | Dataset 2 | 98.66 | 98.81 | 0.0113 | 0.0147 | Dataset 12 | 98.68 | 99.00 | 0.0103 | 0.0181 |
| Dataset 12 | Dataset 4 | 98.39 | 98.04 | 0.0194 | 0.0128 | Dataset 13 | 98.59 | 98.99 | 0.0103 | 0.0181 |
| Dataset 13 | Dataset 3 | 98.61 | 98.37 | 0.0161 | 0.0117 | Dataset 14 | 98.70 | 99.19 | 0.0082 | 0.0177 |
| Dataset 14 | Dataset 3 | 98.77 | 98.59 | 0.0140 | 0.0106 | Dataset 15 | 98.95 | 98.71 | 0.0131 | 0.0081 |
| Dataset 15 | Dataset 4 | 98.53 | 98.64 | 0.0166 | 0.0131 | Dataset 16 | 98.64 | 99.11 | 0.0090 | 0.0180 |
| Dataset 16 | Dataset 4 | 98.43 | 98.58 | 0.0174 | 0.0143 | Dataset 17 | 98.59 | 99.31 | 0.0070 | 0.0211 |
| Dataset 17 | Dataset 4 | 98.59 | 98.58 | 0.0139 | 0.0130 | Dataset 18 | 98.89 | 99.14 | 0.0085 | 0.0141 |
| Dataset 18 | Dataset 4 | 98.71 | 98.98 | 0.0165 | 0.0104 | Dataset 19 | 98.80 | 99.37 | 0.0064 | 0.0174 |
| Dataset 19 | Dataset 4 | 99.04 | 98.84 | 0.0113 | 0.0079 | Dataset 20 | 98.04 | 99.23 | 0.0078 | 0.0114 |
| Dataset 20 | Dataset 5 | 98.67 | 98.60 | 0.0137 | 0.0128 | Dataset 21 | 98.70 | 98.44 | 0.0158 | 0.0103 |
| Dataset 21 | Dataset 5 | 98.70 | 98.56 | 0.0142 | 0.0118 | Dataset 22 | 98.90 | 98.64 | 0.0135 | 0.0087 |
| Dataset 22 | Dataset 5 | 98.49 | 98.13 | 0.0184 | 0.0118 | Dataset 23 | 98.66 | 98.39 | 0.0164 | 0.0101 |
| Dataset 23 | Dataset 5 | 98.63 | 98.27 | 0.0170 | 0.0104 | Dataset 24 | 98.78 | 98.59 | 0.0145 | 0.0113 |
| Dataset 24 | Dataset 5 | 98.73 | 98.33 | 0.0164 | 0.0090 | Dataset 25 | 98.79 | 98.79 | 0.0123 | 0.0118 |
| Dataset 25 | Dataset 5 | 98.38 | 98.05 | 0.0229 | 0.0105 | Dataset 26 | 98.52 | 97.99 | 0.0293 | 0.0094 |
| Dataset 26 | Dataset 5 | 98.34 | 98.22 | 0.0209 | 0.0129 | Dataset 27 | 98.71 | 98.20 | 0.0182 | 0.0075 |
| Dataset 27 | Dataset 5 | 98.80 | 98.05 | 0.0228 | 0.0121 | Dataset 28 | 98.45 | 98.12 | 0.0191 | 0.0119 |
| Dataset 28 | Dataset 5 | 98.07 | 98.55 | 0.0170 | 0.0212 | Dataset 29 | 98.68 | 98.19 | 0.0184 | 0.0081 |
| Dataset 29 | Dataset 5 | 98.16 | 98.39 | 0.0190 | 0.0179 | Dataset 30 | 98.58 | 98.39 | 0.0164 | 0.0121 |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
against the adversarial attack. Similarly, XGBoost was less affected by these adversarial perturbations and had an ASR of nearly 93.16%. Although there is a little disruption in the outcomes of XGBoost for the second dataset but general trend shows the vulnerability propensity of the ML model against the adversarial perturbation. The technical investigations on our model are the impact of adversarial training. The running time attack on detectors only takes close to 1.2 minutes in this study and is inexpensive, since for the attacker, just the misclassification mistake is relevant and in ML this time is appropriate. For sure, considering a few examples, such as 500, we can run the attack in 13.3 seconds that for sure considering 500 also, from our perspective, enough to fool a detector or misclassification error.

### B. Discussion

The combination of lexical and Web-scrapped features we collected is a good foundation for our analysis. Various ML techniques used for categorisation give us light and profound output. We found that the XGBoost classifier with a maximum depth of ten yields the best outcomes in the classification process. When compared to other classifiers, XGBoost has the highest detection accuracy and precision, followed by Gradient Boost, AdaBoost, and Random Forest in comparison to previous research works. If the number of folds in the simulation work increases, the FPR and FNR drop. The model grows more sophisticated and overfits the data as the maximum depth of the tree is increased. The same thing occurs with the other classifiers. Our findings and recommended techniques improve the accuracy of prediction. The accuracy of detection we discovered is substantially greater than prior techniques improve the accuracy of prediction. The accuracy of prediction with the other classifiers. Our findings and recommended techniques improve the accuracy of prediction.

### VI. Conclusion and Future Work

In this paper, we developed a framework for identifying fraudulent advertisement URLs using innovative feature extraction, and we created a detection system with the highest detection accuracy results. We investigated novel feature extraction using an unconventional combination of six feature classes, including various advanced features. Most of the existing approaches were based on only the traditional features set like a bag of words, which were insufficient to make the detection system sufficiently efficient. We provide a comprehensive analysis by examining the performance of supervised and unsupervised ML techniques. The comprehensive study of our detectors, considering matched and mismatched diverse datasets analyse the generalising capability of our model. Our classifiers optimized the detection accuracy, which we can see in the clustering visualisation. We also looked into the vulnerability of our detectors, e.g., Random Forest, AdaBoost, Gradient Boost, and XGBoost, against the ZOO adversarial attack.

Although the Web-scrapped features extraction process is time-consuming due to the ample features, we will decrease it with additional resources in our future efforts. Moreover, throughout the training phase, we will explore alternative backdoor and poisoning adversarial approaches. We would prefer to conduct a thorough investigation of possible defence mechanisms for strengthening detector security in the presence of causative attacks.

### References

[1] J. Hong, “The state of phishing attacks,” Commun. ACM, vol. 55, no. 1, pp. 74–81, 2012.
[2] D. Sahoo, C. Liu, and S. C. Hoi, “Malicious URL detection using machine learning: A survey,” 2017, arXiv:1701.07179.
[3] E. Nowroozi, A. Dehghantanha, R. M. Parizi, and K.-K. R. Choo, “A survey of machine learning techniques in adversarial image forensics,” Comput. Security, vol. 100, Jan. 2021, Art. no. 102092.
[4] “GitHub code,” Accessed: Apr. 7, 2022. [Online]. Available: https://github.com/ehsannowroozi/Sec_Classifying_URL_Detection
[5] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, “PhishNet: Predictive blacklisting to detect phishing attacks,” in Proc. IEEE INFOCOM, 2010, pp. 1–5.
[6] Y. He, Z. Zhong, S. Krasser, and Y. Tang, “Mining DNS for malicious domain registrations,” in Proc. 6th Int. Conf. Collaborative Comput. Netw., Appl. Worksharing (CollaborateCom), 2010, pp. 1–6.
[7] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond blacklists: Learning to detect malicious Web sites from suspicious URLs,” in Proc. 15th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., 2009, pp. 1243–1254.
[8] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, “Building a dynamic reputation system for DNS,” in Proc. 19th USENIX Security Symp. (USENIX Security), 2010, p. 18.
[9] W. Liu, X. Deng, G. Huang, and A. Y. Fu, “An antiphishing strategy based on visual similarity assessment,” IEEE Internet Comput., vol. 10, no. 2, pp. 58–65, Mar/Apr. 2006.
[10] S. Garera, N. Provos, M. Chew, and A. D. Rubin, “A framework for detection and measurement of phishing attacks,” in Proc. ACM Workshop Recurring Malcode, 2007, pp. 1–8.
[11] I. Fette, N. Sadeh, and A. Tomasic, “Learning to detect phishing emails,” in Proc. 16th Int. Conf. World Wide Web, 2007, pp. 649–656.

[12] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, “A comparison of machine learning techniques for phishing detection,” in Proc. Anti-Phishing Work. Groups 2nd Annua. eCrime Researchers Summit, 2007, pp. 60–69.

[13] N. Carlini and D. Wagner, “Audio adversarial examples: Targeted attacks on speech-to-text,” in Proc. IEEE Security Privacy Workshops (SPW), 2018, pp. 1–7.

[14] H. Chen, H. Zhang, P.-Y. Chen, J. Yi, and C.-J. Hsieh, “Attacking visual language grounding with adversarial examples: A case study on neural image captioning,” 2017, arXiv:1712.02051.

[15] G. Gressel, N. Hegde, A. Sreekumar, and M. Darling, “Feature importance guided attack: A model agnostic adversarial attack,” 2021, arXiv:2106.14815.

[16] B. Sabir, M. A. Babar, R. Gaire, and A. Abuadbba, “Reliability and robustness analysis of machine learning based phishing URL detectors,” IEEE Trans. Dependable Secure Comput., early access, Nov. 3, 2022, doi: 10.1109/TDSC.2022.3218043.

[17] J. Li, H. Zhang, Z. Han, Y. Rong, H. Cheng, and J. Huang, “Adversarial attack on community detection by hiding individuals,” in Proc. Web Conf., 2020, pp. 917–927.

[18] “VirusTotal.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.virustotal.com/gui/home/url

[19] “Alexa Dataset.” Accessed: Aug. 24, 2021. [Online]. Available: https://www.kaggle.com/datasets/chedcheed/top1m

[20] “Crowdflower Dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://data.world/crowdflower/url-categorization

[21] “DMOZ Dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.kaggle.com/datasets/shawnoni/url-classification-dmoz

[22] “Benign and malicious set URL.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.kaggle.com/kaggle/datasets/siddarthkumar25/malicious-and-benign-urls

[23] “Benign and malicious data URL dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.kaggle.com/antonij453/urldataset#data.csv

[24] “ISCX-URL-2016 dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.unh.ca/cic/datasets/url-2016.html

[25] M. S. I. Manun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, “Detecting malicious URLs using lexical analysis,” in Proc. Int. Conf. Netw. System Security, 2016, pp. 467–482.

[26] “Phishtank dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.phishtank.com/developer_info.php

[27] “Phishstorm dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://research.aalto.fi/en/datasets/phishstorm-phishing-legitimate-url-dataset

[28] S. Marchal, J. François, R. State, and T. Engel, “PhishStorm: Detecting phishing with streaming analytics,” IEEE Trans. Netw. Service Manag., vol. 11, no. 4, pp. 458–471, Dec. 2014.

[29] “Phishing site URL dataset.” Accessed: Mar. 24, 2022. [Online]. Available: https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls

[30] E. Nowroozi, Y. Mekdad, M. H. Berenjestanaki, M. Conti, and A. El Fergougui, “Demystifying the transferability of adversarial attacks in computer networks,” IEEE Trans. Netw. Service Manag., vol. 19, no. 3, pp. 3387–3400, Sep. 2022.

[31] C. Szegedy et al., “Intriguing properties of neural networks,” 2013, arXiv:1312.6199.

[32] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 2014, arXiv:1412.6572.

[33] E. Nowroozi, M. Barni, and B. Tondi, “Machine learning techniques for image forensics in adversarial setting,” Ph.D. dissertation, Dept. Inf. Eng. Math., Univ. Siena, Siena, Italy, 2020.

[34] Q. Wang, L. Li, B. Jiang, Z. Lu, J. Liu, and S. Jian, “Malicious domain detection based on K-means and SMOTE,” in Proc. Int. Conf. Comput. Sci., 2020, pp. 468–481.

[35] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, “ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models,” in Proc. 10th ACM Workshop Arif. Intell. Security, 2017, pp. 15–26.