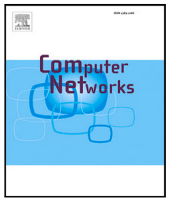
Computer Networks 218 (2022) 109407

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



PhishNot: A Cloud-Based Machine-Learning Approach to Phishing URL Detection

Mohammed M. Alani [a,b,*], Hissam Tawfik [c,d]

[a] Computer Science Department, Toronto Metropolitan University, Toronto, Canada
[b] School of IT Administration and Security, Seneca College of Applied Arts and Technology, Toronto, Canada
[c] College of Engineering, University of Sharjah, Sharjah, United Arab Emirates
[d] School of Built Environment, Engineering, and Computing, Leeds Beckett University, Leeds, United Kingdom

ARTICLE INFO

Keywords:
Phishing
Machine learning
Security
Attack
Cloud
Url

ABSTRACT

Phishing is constantly growing to be one of the most adopted tools for conducting cyber-attacks. Recent statistics indicated that 97% of users could not recognize a sophisticated phishing email. With over 1.5 million new phishing websites being created every month, legacy black lists and rule-based filters can no longer mitigate the increasing risks and sophistication level of phishing. Phishing can deploy various malicious payloads that compromise the network's security. In this context, machine learning can play a crucial role in adapting the capabilities of computer networks to recognize current and evolving phishing patterns. In this paper, we present PhishNot, a phishing URL detection system based on machine learning. Hence, our work uses a primarily "learning from data" driven approach, validated with a representative scenario and dataset. The input features were reduced to 14 to assure the system's practical applicability. Experiments showed that Random Forest presented the best performance with a very high accuracy of 97.5%. Furthermore, the design of our system also lends itself to being more adoptable in practice through a combination of high phishing detection rate and high speed (an average of $11.5\mu s$ per URL) when deployed on the cloud.

1. Introduction

Phishing is a social engineering attack that exploits the weakness in system processes caused by system users [1]. An attacker can send a phishing Uniform Resource Locator (URL) such that when the user clicks on that link, it takes the user to a phishing website. Phishing URLs are delivered in various ways, including emails, text messages, or on other suspicious websites, with email being the primary phishing medium. The phishing website might have a URL that resembles a legitimate link, such as a social media website, banking website, or an email website, and the webpage on the phishing URL would resemble a legitimate service webpage. It would typically ask the user to log in. At this stage, once the users type their login credentials, they are stolen, and the users are usually redirected to the original login page. In other phishing attacks, clicking on a link could download malware or spyware, install backdoors, or steal session information.

Fig. 1 shows the growth in phishing websites from Q1 2017 to Q1 2021. This rapid growth, as shown in the figure, of about three times in the last two years of this 5-year period indicates malicious actors' reliance on phishing as one of the most successful attack vectors. This "spike-like" increase in Q2/Q3 of 2020 might be linked to the massive increase in online living and working due to Covid-19, a trend that is likely to continue.

One of the main challenges is the network perimeter can be protected with state-of-the-art firewalls and intrusion detection systems but could still suffer from phishing. Phishing penetrates these protected network borders through encrypted web traffic or via emails. Once the user clicks this phishing URL, malicious activity proceeds to infect the target's device with malware or perform other harmful actions. Hence, protecting users from phishing is an integral part of securing the network.

With the increasing reliance on technology, phishing has become more wide-ranging, intense, and sophisticated. Spear phishing attacks have increased in number and improved in quality. In a spear-phishing attack, the attacker gathers information about a specific user or a small group of users and creates highly-crafted spoofed emails, usually impersonating well-known companies, trusted relationships, or contexts [2]. Another type of phishing is called Vishing, which is Voice-phishing. In vishing, the attack vector is a phone call instead of an email.

Lack of user awareness contributes heavily to the success rates of phishing. According to [3], only 52% of users raise an alarm upon

* Corresponding author at: School of IT Administration and Security, Seneca College of Applied Arts and Technology, Toronto, Canada.
E-mail address: m@alani.me (M.M. Alani).

https://doi.org/10.1016/j.comnet.2022.109407
Received 7 January 2022; Received in revised form 16 June 2022; Accepted 3 October 2022
Available online 13 October 2022

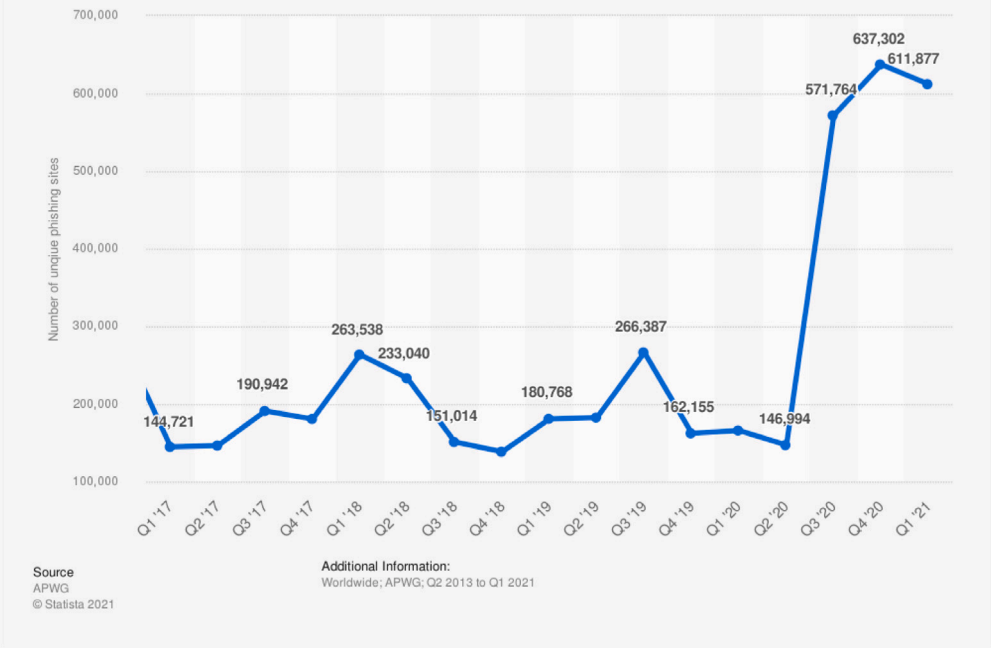


Fig. 1. Number of phishing websites from 2017 to Q1 2021.

receiving a suspected phishing email within 5 min. This behavior indicates weak user awareness about phishing and its potentially harmful impact. It became increasingly challenging when many organizations moved to work from home due to the Covid-19 pandemic. Phishing has become the most widely used attack vector to deliver malicious payloads to targets. According to the 2022 Verizon Data Breach Report, 82% of data breaches involved a human element [4].

Within a networking context, phishing is a commonly used technique at the "delivery" stage within the cyber kill-chain [5]. After reconnaissance and weaponization, the malicious actor intends to deliver the malicious payload to the target in the least suspicious way possible. When successful, phishing jeopardizes the network's security by enabling malicious actors to implant malware and Trojan horses or establish covert connections back to the command-and-control center. This action could create a strong foothold for the attacker to move vertically or horizontally in the network. While firewalls, intrusion detection systems, or other network security appliances can help defend the network border, they do not protect the network from phishing.

With malicious actors developing new techniques, static rule-based detection approaches do not provide sufficient protection against phishing. Hence, the machine learning paradigm presents itself as a possible method to build phishing detection systems that are inherently capable of adaptively protecting networks from current and evolving techniques of phishing attacks and the repercussions they can bring to the networks.

Conventional techniques to detect phishing rely on old assumptions, static or not easily adaptable approaches that cannot catch up with the fast evolving nature of technological development and phishing methods. In this "data" and "cloud" era, machine learning paradigms present a natural opportunity to design, implement, deploy, and evolve better phishing detection techniques.

1.1. Contributions

This paper presents a machine-learning-based phishing detection system that extracts features from the URL with external features from other sources about that URL. This research focuses on producing a high-accuracy, implementable, efficient, and easily-accessible phishing URL detection system.

The following points summarize this research's contributions.

1. Build a high accuracy machine-learning-based phishing detection system that relies on the URL only, without the need to read the target webpage, to reduce the threats to the network's attack surface. This approach provides better network protection when compared to other approaches that require accessing the phishing webpage to detect the phishing attack.
2. Utilize a minimal number of critical features through recursive feature elimination (RFE) in the feature selection stage. This method not only improves efficiency by reducing the number of features fed into the machine learning classifier, but also reduces the number of features captured and extracted at the data acquisition phase.
3. Deploy the machine-learning-based phishing detection system with high detection accuracy on the cloud as an API that can be utilized to create browser plugins, email client plugins, or any other deployment architecture. This deployment enables easy access to the service by various networks and provides higher availability compared to locally-hosted solutions.

In addition to the above, our research produced a smaller version of the dataset with only 14 features that future research can use in phishing detection.

1.2. Paper layout

The following section will discuss related works in phishing detection using both machine-learning and non-machine-learning solutions. In Section 3, the proposed system is explained with an overview of how the detection process works. Section 4 describes the dataset, collected data, and features used in the training and testing of the proposed classifier. The detailed steps of the experiments with their results are presented in Section 5. The results are discussed and compared to previous works in Section 6. The last section provides the conclusions along with directions for future research.

2. Related works

Phishing has been a problem for a long time and a subject of study in many research publications. In this section, we will discuss examples of recent and relevant research in phishing detection.

2.1. Classical phishing detection

In 2017, Sonowal and Kuppusamy presented a multilayer phishing detection system named PhiDMA [6]. PhiDMA provides a model to detect phishing by incorporating five layers: Auto upgrade whitelist layer, URL features layer, Lexical signature layer, String matching layer, and Accessibility Score comparison layer. They built a prototype implementation of the proposed PhiDMA model. The testing results showed that the model could detect phishing sites with an accuracy of 92.72%.

Rao and Pais presented, in 2019, an application named Jail-Phish, that relies on search engine-based techniques to detect phishing sites [7]. The focus of the proposed system was on Phishing Sites Hosted on Compromised Servers (PSHCS) and the detection of newly registered legitimate sites. Jail-Phish compares the suspicious site and matched domain in the search results to calculate the similarity score between them. Testing results showed Jail-Phish to achieve an accuracy of 98.61%, a true positive rate of 97.77%, and a false positive rate of less than 0.64%.

2.2. Machine-learning based phishing detection

In 2018, Chin et al. presented a phishing detection system relying on Software Defined Networks (SDN) and Deep Packet Inspection (DPI) named Phishlimiter [8]. Phishlimiter starts with DPI and then leverages it with SDN to identify phishing activities through e-mail and web-based communications. The proposed DPI approach consists of two components, phishing signature classification and real-time DPI. The proposed system performs well in the SDN environment but is not suited for applications in end-user environments that are not reliant on SDNs.

Wei et al. presented, in 2019, a lightweight deep learning algorithm to detect the malicious URLs and enable a real-time and energy-saving phishing detection sensor [9]. The proposed deep learning classifier achieved an accuracy of 86.630%. The focus of the study was to create a low-power phishing detector for sensors and Internet of Things (IoT) devices. Although the proposed system acquitted itself well as a lightweight solution, it has a relatively lower and less practicable accuracy performance.

In 2019, Sahingoz et al. presented a machine-learning-based phishing URL detector based on Natural Language Processing (NLP) [10]. The proposed system used seven classification algorithms and natural language processing (NLP) features. Testing showed the proposed system could achieve an accuracy of 97.98% using an RF classifier, with a relatively high false-positive rate of 3%. These results were achieved using 27 features.

Also, in 2019, Chiew et al. presented another machine-learning phishing URL detector named HEFS [11]. The proposed work suggests a feature reduction from 48 to 10 using a Cumulative Distribution Function gradient (CDF-g) algorithm. Testing showed that the proposed model achieved 94.6% accuracy with 48 features, while accuracy dropped to 94.60% when the features were reduced to 10. They performed another testing phase with a second dataset, where the proposed system achieved 94.27% accuracy with 30 features.

Jain and Gupta published, in 2019, a paper proposing a machine-learning phishing URL detection based on analysis of the HTML code of the web page [12]. The proposed approach extracts 12 features from the HTML code of the page linked by the URL and feeds that information into a classifier. It achieved 98.4% accuracy on the logistic regression classifier. However, this method is considered high-risk because it reads the actual page contents before deciding whether it is a phishing or benign link. This process can be particularly dangerous if the phishing page contains malware or is used for drive-by attacks.

Abu Tair et al. proposed, in 2019, a case-based reasoning Phishing detection system (CBR-PDS) that relies on previous cases to detect phishing attacks [13]. CBR-PDS aims to improve the detection accuracy and the reliability of the results by identifying a set of discriminative features and discarding irrelevant features. CBR-PDS relies on a two-stage hybrid procedure using Information gain and Genetic algorithms. The reduction of the data dimensionality results in an improved accuracy rate and a reduced processing time. Testing shows that CBR-PDS can achieve an accuracy of 95%. However, the system requires high processing power and high memory.

Zhu et al. proposed, in 2020, Decision Tree and Optimal Features based Artificial Neural Network (DTOF-ANN) to target proper feature selection to help the ANN classifier perform better [14]. The proposed system starts with improving the traditional k-medoids clustering algorithm with an incremental selection of initial centers to remove duplicate points from the public datasets. Then, an optimal feature selection algorithm based on the newly defined feature evaluation index, decision tree, and local search method prunes out the negative and useless features. Finally, an optimal structure of the neural network classifier is constructed through finely-tuned parameters and trained by the selected optimal features. Testing results demonstrated that DTOF-ANN could achieve an accuracy of 97.80%.

In 2021, Mourtaji et al. introduced a hybrid Rule-Based Solution for phishing URL detection using Convolutional Neural Networks(CNN) [15]. The proposed system extracts 37 features from seven different methods, including the blacklisted method, lexical and host method, content method, identity method, identity similarity method, visual similarity method, and behavioral method. When tested, the proposed system achieved 97.945% with the CNN model and 93.216% for the MLP model.

Gandotra and Gupta presented, in 2021, a study on the role of feature selection methods in detecting phishing webpages efficiently and effectively [16]. Their comparative analysis of machine learning algorithms was carried out based on their performance without and with feature selection. The experiments demonstrate that employing a feature selection method along with machine learning algorithms improves the build time of classification models for phishing detection without compromising their accuracy.

In 2021, Wazirali et al. proposed another SDN-based phishing detection technique based on clustering and CNN [17]. The proposed work uses Recursive Feature Elimination (RFE) with a Support Vector Machine (SVM) algorithm for feature selection. The SDN transfers the URLs phishing detection process out of the user's hardware to the controller layer, continuously trains on new data, and then sends its outcomes to the SDN-Switches. RFE-SVM and CNN increase the accuracy of phishing detection. The experimental results showed 99.5% phishing detection accuracy. The proposed work consumed about 500MB of memory, which could heavily overload SDN devices where the classifier operates. Another shortfall of the proposed system is that it uses online training, which can make the classifier model susceptible to adversarial ML attacks.

Some of the machine-learning systems mentioned earlier achieved relatively high accuracy. However, we argue that the proposed system needs to attain a combination of high accuracy, practical implementation, and adaptability to a wide range of phishing attacks, which the previous literature did not sufficiently address. This combination of objectives makes our proposed system a stronger candidate for realistic implementation and high accuracy.

Further information on previous works can be found in [18–21].

3. PhishNot

The proposed system was based on the following design goals:

- High accuracy: This was achieved by experimenting with several types of classifiers and choosing the one with the highest accuracy. In addition, feature selection contributed to maintaining high accuracy by removing redundant or minimally relevant features that can negatively impact ML's efficiency and prediction accuracy.

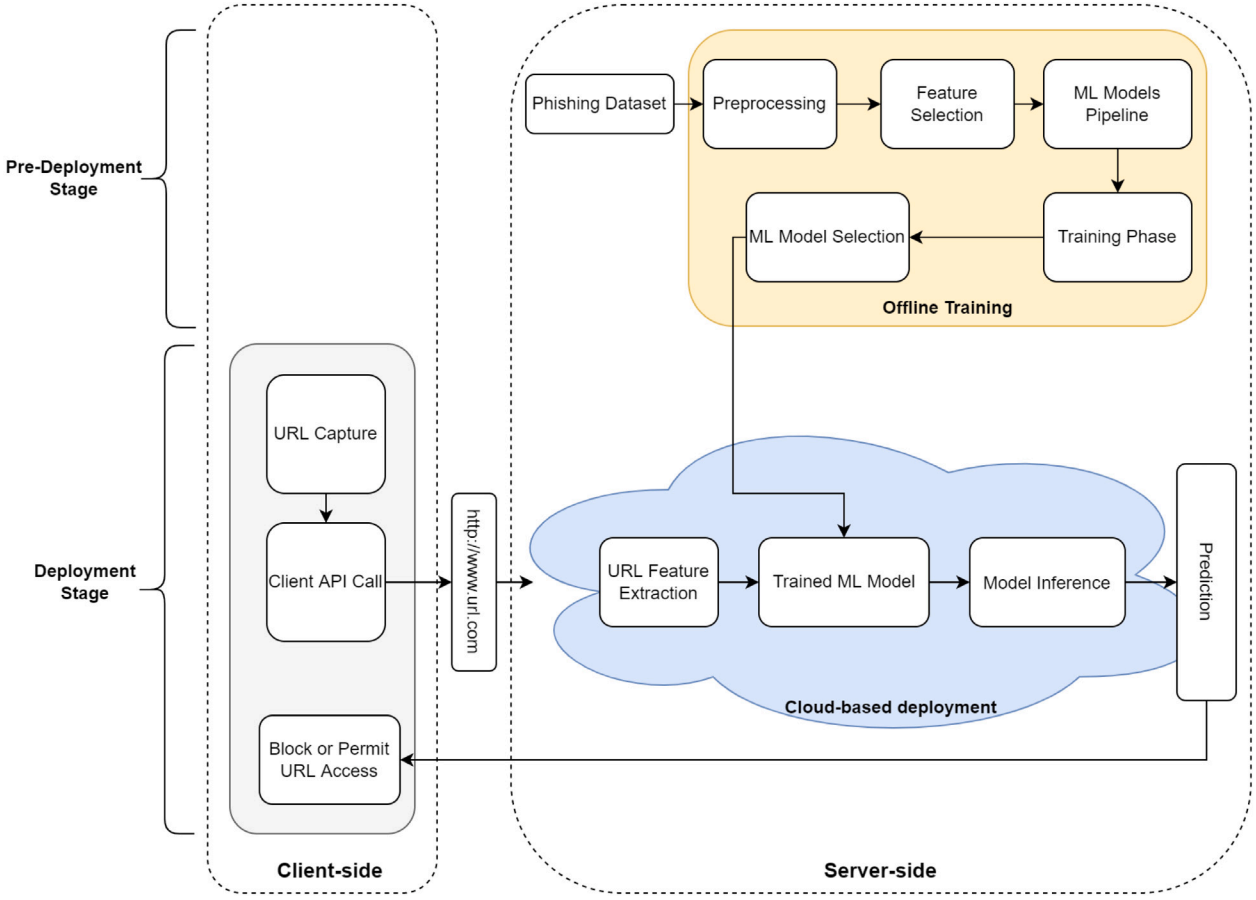


Fig. 2. PhishNot system overview.

- Implementability: The proposed system relies on a few features to simplify the feature extraction in a real-life deployment. In addition, the features selected are easy to acquire and extract. The feature selection process focused on feature importance, in that the features extracted at the deployment stage can feed directly into the system without preprocessing.
- Efficiency: The proposed system needs to achieve high efficiency by using a small number of features. This efficiency translates to lower data acquisition time and lower prediction time. In addition, the actual processing takes place on the cloud, and the system performance should be efficient even on low-processing power devices.
- Ease of Access: Being cloud-based, the detection system deploys as a web service with an easy-to-access Application Programming Interface (API). Thus, the system can be embedded as a plug-in in email clients, web browsers, or any other service where URLs are shared.

Fig. 2 shows an overview of the proposed system. The proposed system is split into two stages, a pre-deployment stage and a deployment stage. At the pre-deployment stage, the dataset undergoes a preprocessing stage that prepares data, addresses missing data, and establishes data balancing. The preprocessing output feeds into the feature reduction and selection process, where a relatively small number of features is selected. The resulting feature-reduced dataset is then used for training and testing of a pipeline of ML classifiers. After the initial testing phase, the best-performing model is further tested to ensure that it generalizes well beyond the training dataset. After systematic testing, the model is stored for later deployment into a cloud environment.

At the deployment stage, the client captures the URL and sends it through an API call to the cloud-based server. As the URL might contain one or more symbols, the URL is encoded using UTF-8 encoding and then sent in a simple API call with the URL as an HTTP parameter. The server receives the URL, extracts the features, and requests the external features from their sources. Once the extracted features are gathered, they are sent as an input to the trained ML classifier. The classifier then produces a prediction of "benign" or "phishing". The server returns this prediction to the client, where the client can accordingly decide to block or allow the URL access.

4. The dataset

The dataset chosen to train and test our models was the one introduced in [22]. The dataset was built by collecting data about well-known phishing URLs from PhishTank [23] and benign URLs from Alexa [24].

The original dataset included 111 features extracted for 88,646 URLs. Within these URL instances, 58,000 were labeled "benign", and 30,646 were labeled as "phishing". Of the 111 features collected, 96 were extracted from the URL itself, while 15 others were collected from external sources such as Google search indexing.

Features were extracted from the URL after dissecting it into four parts, as shown in Fig. 3. These features counted the occurrences of specific special characters, such as the number of "." within the domain name or the number of "/" within the directory part. In addition, some of these features were included for the whole URL. The dataset also included several external features from external sources, such as domain age and whether the domain has an indexed Alexa website ranking or not. Details of all extracted features can be found in [22].

Fig. 3. Anatomy of a URL.

5. Experiments and results

The experiment included three phases: a preprocessing phase, model training, and cloud deployment. Algorithm 1 shows the main steps of the conducted experiments.

Algorithm 1: High-level Experimental Steps

Input: Raw Dataset (88,646 instance, 111 features)
Output: Preprocessed Dataset (57,024 instances, 14 features), Trained Model, Experiment Results, Cloud-Deployment

load $RawDataset$
$PreprocessedDataset = Preprocessing(RawDataset)$
create $MLModelPipeline$
train $MLModelPipeline$ with $PreprocessedDataset$
test $MLModelPipeline \rightarrow InitialResults$
$ReducedDataset \leftarrow FeatureSelection(PreprocessedDataset)$
test $MLModelPipeline \rightarrow ExperimentResults$
select $BestMLModel$
store $BestMLModel$
deploy $BestMLModel$

5.1. Implementation environment

The hardware and software specifications of the implementation environment used in the preprocessing, training, testing, and storage of the trained model can be described in the following points:

- Desktop computer with AMD Ryzen 5 3600 CPU, 4.2 GHz, 16 GB RAM, NVidia GeForce GT710 GPU
- Python v3.8.5 [25].
- TensorFlow v2.3.0 [26].
- Keras v2.4.3 [27].
- Sci-Kit Learn v1.0.1 [28].

5.2. Performance measures

The standard four performance measures of a binary ML-based classifier are:

1. True Positive(TP): The number of test instances whose true value and predicted value is 1, divided by the number of test instances whose true value is 1.
2. True Negative (TN): The number of test instances whose true value and predicted value is 0, divided by the number of test instances whose true value is 0.
3. False Positive (FP): The number of test instances whose true value is 0 and predicted value is 1, divided by the number of test instances whose true value is 0.
4. False Negative (FN): The number of test instances whose true value is 1 and predicted value is 0, divided by the number of test instances whose true value is 1.

These four measures, when combined, generate the confusion matrix.

Our work uses the main four performance parameters:

1. Accuracy: This measures the ratio of correct predictions using the equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

2. Precision: This measures the ratio of the accuracy of positive predictions using the equation:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

3. Recall: This measures the ratio of positive instances that are correctly detected by the classifier using the equation:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

4. F_1 Score: This measures the harmonic mean of precision and recall using the equation:

$$F_1 Score = 2 * \frac{Recall * Precision}{Recall + Precision} \quad (4)$$

5.3. Dataset preprocessing

This stage is used to address the following aspects:

- Handling 'Invalid' features: Some features are deemed invalid, such as the feature qty_questionmark_domain that indicates the number of question marks in a domain name. According to the DNS RFC [29], domain names cannot contain a symbol, such as an underscore, comma, semicolon, or question mark. We found 16 invalid features in the dataset.
- Handling 'Redundant' Features: Some features can be easily mathematically deduced from other features, such as qty_hyphen_url feature that indicates the number of hyphens (-) symbols. Although this is a valid feature, four other features count the number of hyphens in the domain, folder, file, and parameter. Thus, this feature is redundant. Based on our domain knowledge and close examination of the features, 19 redundant features were found and removed.
- Handling instances with 'Missing values': Some instances were missing some values.

Based on the observations mentioned earlier, we created the preprocessing phase as shown in algorithm 2.

Algorithm 2: Data Preprocessing

Input: Raw Dataset with 111 features
Output: Balanced Dataset with no missing data and 77 features
$Array \leftarrow RawDataset$
Remove invalid features from $Array$
Remove redundant features from $Array$; **for** $instance \in Array$ **do**
  if $feature \in instance$ *is empty* **then**
    Remove $instance$
  end
end

The first two steps of preprocessing, as shown in algorithm 2, identified and removed invalid and redundant features. This process results in the reduction of the number of features to 77. Next, the last step removed all instances with missing values from the dataset. This process reduced the number of instances from 88,646 to 57,024. Table 1 shows the number of features and instances before and after preprocessing.

As shown in Table 1, the number of "benign" instances after preprocessing was 38,962, while the number of "phishing" instances was 18,062. Thus, the minority class is 31.64% of the dataset, which does not cause a severe imbalance problem.

Table 1
Number of features and instances before and after preprocessing.

Specification	Raw dataset	After preprocessing
Number of instances	88,646	57,024
Number of benign instances	58,000	38,962
Number of phishing instances	30,646	18,062
Number of features	111	77

Table 2
Initial results with 77 features.

Model	Accuracy	Precision	Recall	F_1 Score
RF	0.9715	0.9692	0.9683	0.9687
LR	0.9210	0.9248	0.9005	0.9108
DT	0.9508	0.9458	0.9461	0.9460
GNB	0.7979	0.8407	0.7220	0.7408
MLP	0.9122	0.9049	0.9001	0.9024

5.4. Initial model training

After preprocessing, a pipeline of five machine-learning classifiers was created. These five classifiers were:

- Random Forest (RF)
- Logistic Regression (LR)
- Decision Tree (DT)
- Gaussian Naive Bayes (GNB)
- Multi-Layer Perceptron (MLP)

The data was split into two parts; 75% used for training and 25% used for testing. This random split considers the balance of the two classes when creating the two subsets. For data splitting, we used the SciKit learn built-in function named train_test_split with shuffling to perform the task [28]. After the initial training and testing phase, the best-performing model was selected for further testing to assure that it generalizes well beyond the training subset of the dataset and that it does not suffer from overfitting. The selected model would then be stored and deployed on a cloud instance for performance testing.

Table 2 shows the performance measures of the initial testing of the five trained classifiers.

As shown in Table 2, the RF classifier outperforms all the other classifiers in terms of accuracy and F_1 Score. The second in terms of performance was the DT classifier.

5.5. Feature selection

This research aims to create a high-accuracy phishing URL detector based on a sub-dataset from the original dataset utilizing a smaller number of features while maintaining high accuracy. The selected features for the deployed model can be easily extracted during data acquisition. This direction rules out some statistical dimensionality reduction algorithms such as Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and Linear Discriminant Analysis (LDA). The reason is that these methods do not reduce the number of acquired features and would add preprocessing steps that could hinder the performance of real-life deployments.

The method we used to select the features in this research was RFE based on feature reduction using feature importance. The steps followed are shown in algorithm 3.

As shown in algorithm 3, a Random Forest classifier model was created and trained with 75% of the dataset instances and tested with the remaining 25%. After testing, feature importance was calculated for each feature. The feature with the lowest feature importance was then removed, another cycle of training and testing continued, and this step was repeated until it reached a minimum number of features that produces high prediction accuracy. This reduction method lowers the number of features that need to be used in prediction in live deployments and not just the dimensionality of the data input to the system. It enables more efficient data acquisition, training, testing, and lightweight real-life deployment.

Algorithm 3: Recursive Feature-Elimination Using Feature Importance

Input: Dataset with 77 features
Output: Dataset with 14 features

$Array \leftarrow Dataset$
$model = RandomForestClassifier$
$TargetFeatures = 14$
while $Features(Dataset) > TargetFeatures$ **do**
  train $model$ with $Array$
  $importance = FeatureImportance(model)$
  i = index of feature with lowest importance
  $Array.DeleteFeature(i)$
end
Store $Array \rightarrow Dataset$

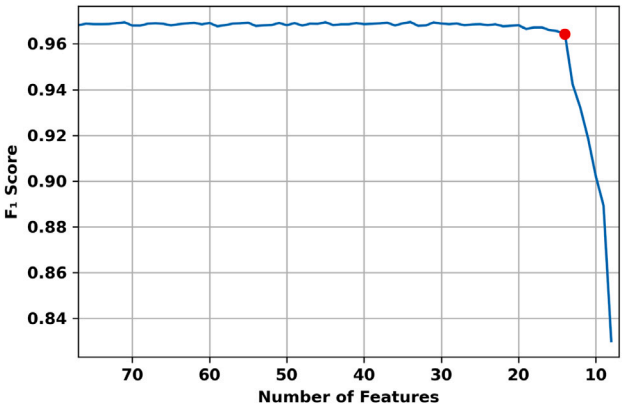


Fig. 4. Impact of feature reduction on F_1 score.

Several other papers, as explained in Section 2, relied on feature importance to select the features with the highest importance. However, our research followed an alternative method. Instead of choosing the features with the highest feature importance, our proposed method relies on repetitive elimination of the feature with the lowest importance and re-training the model again. This successive elimination considers that the importance of one feature might be impacted by the existence (and therefore the elimination of) another feature. Hence, we re-train and re-calculate the importance after the elimination of each feature.

Our feature reduction phase was terminated at 14 features because further reduction would cause noticeable performance degradation, as validated by our experimentation results. Fig. 4 shows the change in average F_1 Score with feature reduction, where score rapidly drops below 14 features.

The 14 features selected at the end of the feature selection process are summarized in Table 3.

Table 4 shows the performance measures of the classifiers when trained and tested with the feature-reduced dataset.

The RF classifier still outperforms the other classifiers. Notably, GNB performance improved significantly, while RF, LR, DT, and MLP had marginal improvements.

Fig. 5 shows the confusion matrix plot of the RF classifier with 14 features. As shown in the figure, the FP rate is 2.18%, while the FN rate was 3.22%.

5.6. Validation with 10-fold cross-validation

To assure that the high accuracy measured earlier was not due to overfitting, we used 10-fold cross-validation on the reduced-feature

Table 3
Features resulting from feature-reduction phase.

Feature name	Feature description
qty_dot_domain	Number of "." in the domain
qty_vowels_domain	Number of English language vowels in the domain name
domain_length	Number of characters in the domain name
qty_dot_directory	Number of "." in the directory part of the URL
qty_slash_directory	Number of "/" in directory part of the URL
directory_length	Number of characters in the directory part of the URL
qty_dot_file	Number of "." in the filename part of the URL
file_length	Number of characters in the filename part of the URL
params_length	Number of characters in the URL's parameters
time_response	Domain lookup time response (in seconds)
asn_ip	Autonomous System Number to which the domain's IP address belongs
time_domain_activation	Number of days since domain activation
time_domain_expiration	Number of days remaining until domain expiry
ttl_hostname	Time-to-Live (TTL) when contacting the domain (in milliseconds)

Table 4
PhishNot testing results with 14 features.

Model	Accuracy	Precision	Recall	F_1 Score
RF	0.9748	0.9692	0.9730	0.9711
LR	0.9300	0.9255	0.9110	0.9177
DT	0.9600	0.9533	0.9545	0.9539
GNB	0.8757	0.8808	0.8261	0.8460
MLP	0.9387	0.9248	0.9361	0.9301

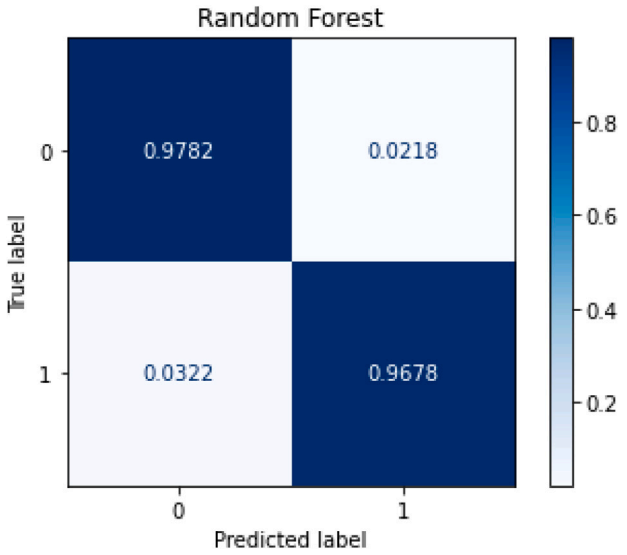


Fig. 5. Confusion matrix plot for RF classifier tested on the original dataset.

data, shown in algorithm 4. In a 10-fold cross-validation process, the data is split into ten subsets randomly. Then, the data go through ten cycles of training and testing. In each cycle, one subset of the ten is excluded from the training process and used for the testing process. This procedure repeats ten times until all ten subsets have been used for testing one time. Each cycle produces a classifier with specific performance parameters. If these parameters have high variance, the classifier suffers from over-fitting and is not generalizing properly within the dataset. If the variance is low, then the mean values of the performance parameters are reliable results.

Table 5 shows the 10-fold cross-validation results for the RF classifier. As the table details, there is minimal variance in the values of the performance measures for the 10 folds. In addition, the standard deviation was as little and 0.002 in accuracy while maintaining a mean

Algorithm 4: 10-Fold Cross-Validation

Input: Dataset with 14 features
Output: testing results for 10 runs; *results*

```
Array ← Dataset
Split Array into 10-folds randomly to produce fold(1to10)
model = RandomForestClassifier
for i = 1to10 do
    TestingData = fold(i)
    TrainingData = Null
    for j = 1to10 do
        if j <> i then
            TrainingData = TrainingData.Append(fold(j))
        end
    end
    train model(TrainData)
    test model(TestData) to produce results(i)
end
print results
```

Table 5
RF classifier 10-fold cross-validation results.

Fold	Accuracy	Precision	Recall	F_1 Score
1	0.975101	0.955100	0.965807	0.960424
2	0.974399	0.953577	0.966242	0.959868
3	0.978082	0.960677	0.970751	0.965688
4	0.976854	0.956660	0.973118	0.964819
5	0.975447	0.958243	0.965574	0.961894
6	0.973518	0.954595	0.962493	0.958528
7	0.974570	0.957119	0.962942	0.960022
8	0.980533	0.963227	0.975542	0.969345
9	0.971764	0.952302	0.958147	0.955216
10	0.976149	0.953553	0.969835	0.961625
Mean	0.975642	0.956505	0.967045	0.961743
Sdev	0.002328	0.003265	0.005017	0.003795

accuracy of 0.9756. Thus, the classifier generalizes well beyond its training dataset.

5.7. Validation using a second dataset

To further validate the generalization of our proposed model, we conducted additional training and testing using the dataset from [10]. This dataset contains 73,575 URLs (36,400 benign and 37,175 phishing).

We extracted the 14 features selected in our research and examined the dataset to ensure that it does not require any preprocessing before training and testing. The dataset was randomly split into a randomly selected 75% training subset and a 25% testing subset, similarly to our original dataset.

Fig. 6 shows the confusion matrix plot for the testing results with the second dataset. Our trained model achieved an accuracy of 0.9976, precision of 0.9963, recall of 0.9974, and F_1 Score of 0.9968. As the performance metrics and the figure illustrates, the overall performance was better than our original dataset and the original results from [10], where the accuracy was 0.98 with an FP rate of 3% and FN rate of 1%. These figures indicate that the selected features and the trained model can generalize well beyond our original dataset.

5.8. Cloud-based deployment

After the classifier testing and cross-validation, the model was saved using a Python library called joblib [30]. The next step created the deployment web service that would use the saved model to produce

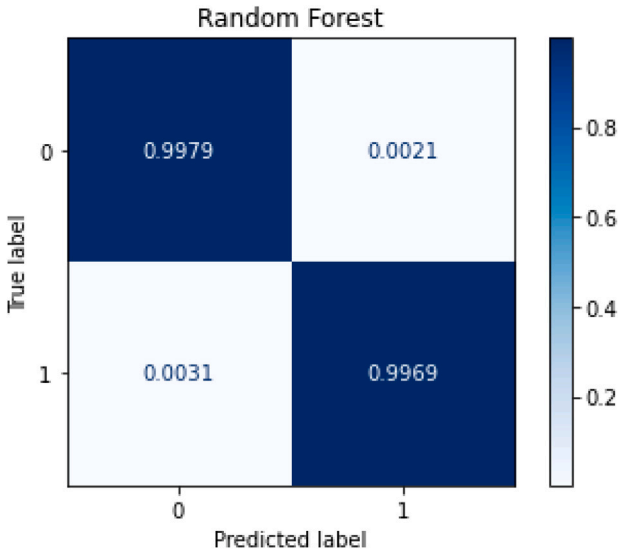


Fig. 6. Confusion matrix plot for the second dataset testing.

predictions. Fig. 7 shows an overview of the server-side operation after deployment.

As shown in Fig. 7, The process starts with the client sending an HTTP request to the running web service. The format of the request is http://serveraddress/?url=URL-to-inspect, where URL-to-inspect is the URL that the system must inspect to flag as "phishing" or "benign". All URLs passed to the server must be encoded using URI encoding [31]. Once the HTTP request arrives at the web service, the URL is extracted from the request and parsed. During the parsing process, the URL is split into the domain name, folder, file name, and additional parameters. After removing the protocol name (HTTP:// or HTTPS://), the first occurrence of a "/" marks the end of the domain name, while the last occurrence of a "/" marks the start of the file name (if any), and all the text in between (if any) is considered the folder name, including any "/". Then, the parsed URL is passed to two feature extraction units. The first unit extracts internal features (i.e., features to be extracted from the URL itself, such as qty_dot_domain). The second unit contacts other services to collect the required data, including the following features:

1. time_response: This feature is obtained by calculating the time needed for the domain lookup time.
2. asn_ip: This feature is obtained by querying a whois database using a package named cymruwhois [32], which queries whois servers for information about the domain name, including the ASN.
3. time_domain_activation: This value represents the number of days since the domain was first registered. In our work, this information is also obtained using a library called python-whois [33].
4. time_domain_expiration: This value represents the number of days left until the domain registration expires. It is also obtained using the pythonwhois library.
5. ttl_hostname: The TTL measure is obtained by sending a PING request to the domain name and waiting for a response to obtain the TTL from the response message.

Once all the features are obtained, the data is orderly fed into the trained RF classifier. The trained ML classifier then provides its prediction in the output form of either "phishing" or "benign". The client then fetches this response to be used to block phishing URLs.

If one of the external parameters could not be obtained, such as a website not responding to PING, or a missing domain age, the missing feature would be assigned a value of −1.

For testing purposes, the classifier was deployed on an Amazon Web Services (AWS) instance in the cloud. Figs. 8 and 9 show a sample cloud server response to phishing URL and a benign URL, respectively.

As shown in Figs. 8 and 9, the server adopts a simple RESTful call method where the URL to be inspected is passed as a parameter to the server using the HTTP protocol. Then, the response is sent as simple text "phishing" or "benign". This response can be passed to an email client plugin, a web browser plugin, or even a mobile text message plugin to prevent the user from clicking on phishing URLs. The average request processing time on the cloud was 11.5 μs per URL.

6. Discussion

The majority of the previous research discussed in Section 2 only focused on accuracy. Our research focuses on proposing a phishing detection system capable of achieving high accuracy while maintaining practical applicability and delivering high efficiency and ease of access. Achieving these goals delivers a system highly suitable for practical implementation, with the elasticity and availability advantages of cloud deployment.

This section contains three subsections: impact of feature engineering, real-world implementation considerations, and a comparative analysis of results with previous research.

6.1. Impact of feature Engineering

Creating an implementable model was one of the main aims of this research. This goal is that data acquisition be easy and realizable and requires minimal preprocessing to extract features in real-life deployment. This requirement is one of the main reasons behind choosing successive feature elimination based on feature importance. In addition to reducing the number of features to improve the efficiency, accuracy, and generalizability of the ML classification algorithms, the features require no preprocessing before being fed into the classifier, which is another important practical consideration.

Fig. 10 shows the change in performance measures of the RF classifier with the full dataset of 77 features and the reduced version of 14 features. The figure clearly shows the improvement in the ML classification performance when the features were reduced. This improvement was expected when reducing the number of irrelevant features that might otherwise limit the classifier's training and generalization performance. In particular, and crucially, the "Recall" performance reflects the ML's ability to correctly identify the cases of "Phishing". This ability is significantly enhanced after feature reduction and is reflected by improvement in the overall F1 and accuracy scores.

6.2. Real-world implementation considerations

The proposed system reduced the number of required features from 111 to 14 in a manner that did not only boost accuracy, but also boosted implementability. The 14 features selected in the proposed system are easy to acquire and extract, as explained in Section 5. Nine of the selected features can be extracted from the URL itself without the need for external sources. The remaining five features are easily obtained by accessing publicly available Whois databases or by a simple ping command. The successful deployment of the proposed system on AWS cloud infrastructure demonstrates its potential practical implementability.

Fig. 11 shows the feature importance of the 14 selected features.

It is clear from Fig. 11 that four features contribute to the classification process significantly more than the other nine. These features are directory_length, time_domain_activation, qty_slash_directory, and file_length. The feature time_domain_activation, which is the number of days since the domain was registered, is a strong indicator of phishing domains, as most domains used in phishing are registered recently and do not stay registered for a long time. In most cases, these domain

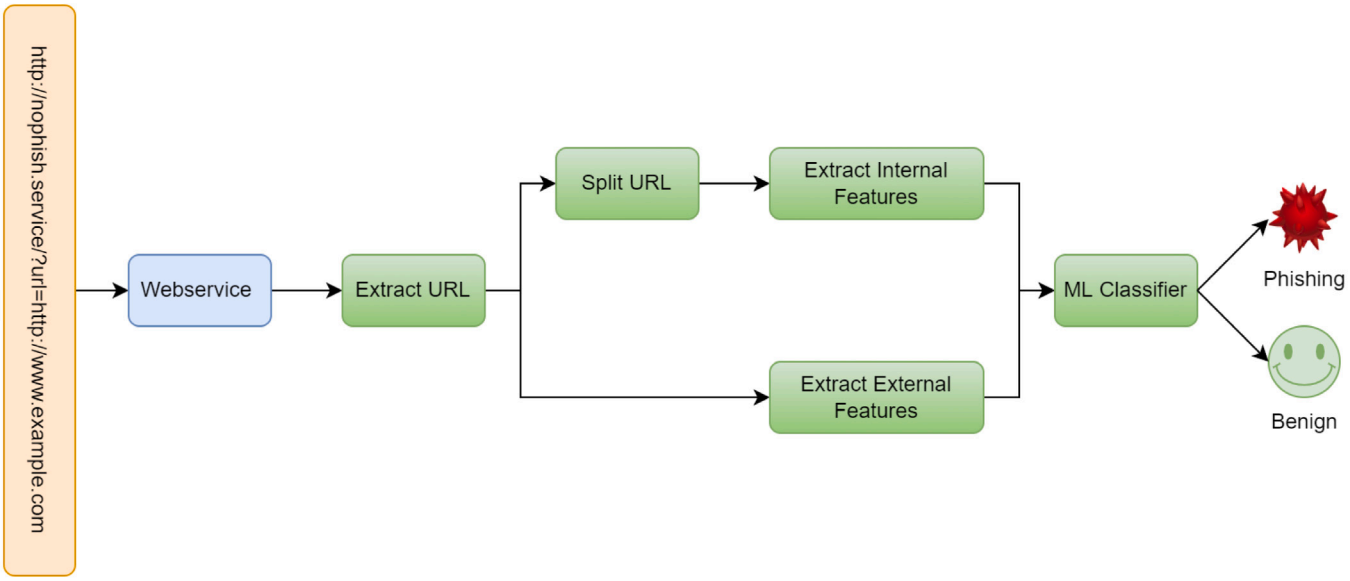


Fig. 7. Overview of the server-side operation.

Fig. 8. A sample of phishing URL sent to the cloud server.

Fig. 9. A sample of benign URL sent to the cloud server.

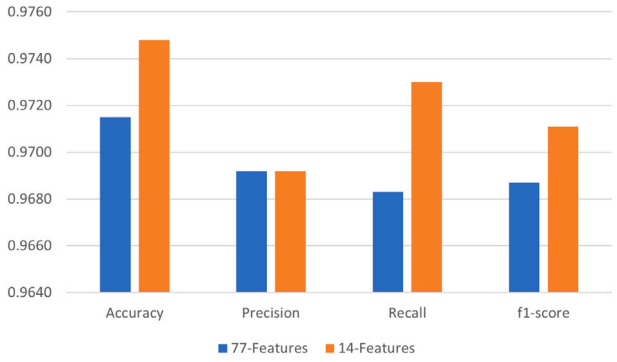


Fig. 10. RF classifier performance with 77 and 14 features.

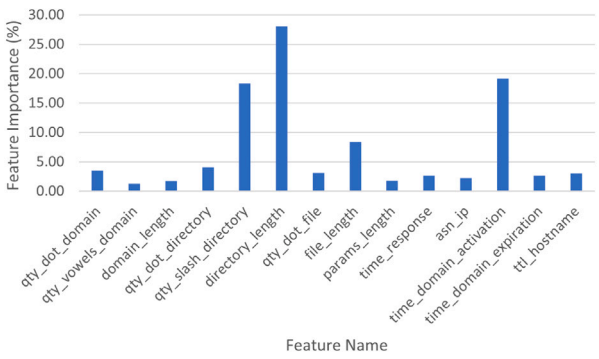


Fig. 11. Features importance.

names get blocked, taken over by law enforcement, or blacklisted, so attackers regularly register new domains to use in phishing. The remaining three features usually have high values in phishing URLs. This includes the number of "\\" in the directory part of the URL (which indicates the use of many sub-directories, the length of the file name, and the length of the directory part.

Deploying the proposed system on the cloud gives it the advantages of high scalability, redundancy, and high availability. Cloud infrastructure is highly scalable, and resources are dynamically created whenever needed. These features enable the proposed system to

Table 6
Comparison of performance measures of PhishNot with other research.

Reference	Dataset	Language	Features	Instances	Balanced	Classifier	Accuracy	FP	FN
[10]	[10]	English	27	73,575	✓	RF	0.980	0.030	0.010
[11]	[11]	English	48	10,000	✓	RF	0.9617	–	–
			10			RF	0.9460	–	–
[12]	[12]	English	12	2544	✓	RF	0.9737	0.0315	0.0197
						LR	0.9842	0.0161	0.0152
[15]	[15]	English	37	40,000	×	CNN	0.9794	–	–
						MLP	0.9321	–	–
PhishNot	[22]	English	14	57,024	✓	RF	0.9748	0.0322	0.0218
	[10]	English	14	73,575	✓		0.9976	0.002	0.003

provide the service to a large user base. In terms of redundancy, the cloud infrastructure eliminates the single point of failure in a single-server architecture. The cloud also provides the basis for easier future updates in that new datasets are available. A newer trained ML classification model can be deployed centrally into the cloud infrastructure, eliminating the need for distributed updates.

6.3. Comparative analysis of results

As mentioned earlier in Section 2, phishing detection using machine learning has been the subject of many research articles. However, the system we proposed has contributions on four fronts:

- Implementability: The number and type of the selected features make them easy to acquire and extract in real-life deployments.
- Accuracy: Our proposed system has improved accuracy, benefiting from systematic preprocessing and feature selection.
- Efficiency: The proposed system is considered highly efficient due to the low number of features selected and the cloud deployment.
- Scalability: As the proposed system is deployed and tested on the cloud, the scalability of the cloud is extended to the system and makes it very well placed to serve a huge number of users.

Table 6 shows performance comparison with other state-of-the-art research.

In terms of the ML classification performance, as shown in Table 6, the proposed system outperforms the systems presented in the combination of high accuracy and a low number of features. Other research that achieved better accuracy had a noticeably higher number of features, making them difficult to implement, and hence less practical.

When compared to [10], using the same dataset, our proposed system achieved even higher accuracy and lower FP and FN measures. These figures indicate that the chosen 14 features are effective and widely applicable.

There were challenges in utilizing datasets from other related works to test our proposed system for two reasons. Some previous works had created their own datasets that were not publicly available, while other related studies made their datasets available but used different data representations and extracted different features. Hence, it was not possible to find the raw data used, and we could not extract the features that fit our proposed system to create an ideal comparison.

Compared to [12], we found that PhishNot achieved higher accuracy with fewer features. Upon closer examination, this research relies on features from the web page hosted on the URL rather than examining the URL itself. Therefore, we argue that this approach has a high risk as phishing pages can contain malware that is automatically downloaded upon access. In addition, many malicious actors use phishing URLs to deliver drive-by attacks. Hence, our proposed system is more secure and does not download the contents of the web page for inspection. It strikes a very good overall balance between accuracy and the number of features used.

On the downside, our proposed system relies on extracting five features out of the selected 14 from external sources. This process means that there is a possibility, albeit low, that these services might not be available or might cause delay. However, this possibility is deemed an acceptable risk given the valuable contribution of these features to the decision process.

7. Conclusions and future work

In this paper, we presented PhishNot, a cloud-based machine-learning-based phishing URL detector. The detector relies on features extracted from the URL itself, in addition to external features, such as the domain age, to infer whether the received URL is a phishing or benign URL. The dataset used in training the classifier was reduced to 14 from 111 using successive feature elimination based on feature importance. The proposed ML classifier was tested and recorded an accuracy of 0.9748, with an FP rate of 2.18%, and an FN rate of 3.22%. The trained model was deployed on the AWS cloud as a web service and tested successfully. The combination of high-performing ML classification, low number of features, and cloud basis position our system favorably in terms of accuracy of phishing detection, efficiency, scalability, and implementability.

Future directions in this research can focus on exploring the potential of our system as a web service in a fog or edge environment. Within a smart city context, the proposed work could be expanded to provide better efficiency using edge and fog computing concepts. The improvement in efficiency can be noticeable as these techniques rely on high-performing networks, such as 5G and beyond.

CRediT authorship contribution statement

Mohammed M. Alani: Conceptualization, Methodology, Software, Formal Analysis, Writing – Original Draft. **Hissam Tawfik:** Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] M. Khonji, Y. Iraqi, A. Jones, Phishing detection: a literature survey, IEEE Commun. Surv. Tutor. 15 (4) (2013) 2091–2121.
[2] D.D. Caputo, S.L. Pfleeger, J.D. Freeman, M.E. Johnson, Going spear phishing: Exploring embedded training and awareness, IEEE Secur. Priv. 12 (1) (2013) 28–38.
[3] Time to report phishing email 2020 | statista, Statista (2021) [Online; accessed 24. Dec. 2021], https://www.statista.com/statistics/1256790/time-to-report-phishing-email.

[4] Key findings from the 2022 verizon data breach investigations report (DBIR) underscore the role of the human element in data breaches | proofpoint US, 2022, [Online; accessed 3. Jun. 2022], https://www.proofpoint.com/us/blog/email-and-cloud-threats/key-findings-2022-verizon-data-breach-investigations-report-dbir.
[5] T. Yadav, A.M. Rao, Technical aspects of cyber kill chain, in: International Symposium on Security in Computing and Communication, Springer, 2015, pp. 438–452.
[6] G. Sonowal, K. Kuppusamy, PhiDMA–A phishing detection model with multi-filter approach, J. King Saud Univ. Comput. Inf. Sci. 32 (1) (2020) 99–112.
[7] R.S. Rao, A.R. Pais, Jail-Phish: An improved search engine based phishing detection system, Comput. Secur. 83 (2019) 246–267.
[8] T. Chin, K. Xiong, C. Hu, Phishlimiter: A phishing detection and mitigation approach using software-defined networking, IEEE Access 6 (2018) 42516–42531.
[9] B. Wei, R.A. Hamad, L. Yang, X. He, H. Wang, B. Gao, W.L. Woo, A deep-learning-driven light-weight phishing detection sensor, Sensors 19 (19) (2019) 4258.
[10] O.K. Sahingoz, E. Buber, O. Demir, B. Diri, Machine learning based phishing detection from URLs, Expert Syst. Appl. 117 (2019) 345–357.
[11] K.L. Chiew, C.L. Tan, K. Wong, K.S. Yong, W.K. Tiong, A new hybrid ensemble feature selection framework for machine learning-based phishing detection system, Inform. Sci. 484 (2019) 153–166.
[12] A.K. Jain, B.B. Gupta, A machine learning based approach for phishing detection using hyperlinks information, J. Ambient Intell. Humaniz. Comput. 10 (5) (2019) 2015–2028.
[13] H. Abutair, A. Belghith, S. AlAhmadi, CBR-PDS: a case-based reasoning phishing detection system, J. Ambient Intell. Humaniz. Comput. 10 (7) (2019) 2593–2606.
[14] E. Zhu, Y. Ju, Z. Chen, F. Liu, X. Fang, DTOF-ANN: An artificial neural network phishing detection model based on decision tree and optimal features, Appl. Soft Comput. 95 (2020) 106505.
[15] Y. Mourtaji, M. Bouhorma, D. Alghazzawi, G. Aldabbagh, A. Alghamdi, Hybrid rule-based solution for phishing URL detection using convolutional neural network, Wirel. Commun. Mob. Comput. 2021 (2021).
[16] E. Gandotra, D. Gupta, An efficient approach for phishing detection using machine learning, in: Multimedia Security, Springer, 2021, pp. 239–253.
[17] R. Wazirali, R. Ahmad, A.A.-K. Abu-Ein, Sustaining accurate detection of phishing URLs using SDN and feature selection approaches, Comput. Netw. 201 (2021) 108591.
[18] A. El Aassal, S. Baki, A. Das, R.M. Verma, An in-depth benchmarking and evaluation of phishing detection research for security needs, IEEE Access 8 (2020) 22170–22192.
[19] Z. Dou, I. Khalil, A. Khreishah, A. Al-Fuqaha, M. Guizani, Systematization of knowledge (sok): A systematic review of software-based web phishing detection, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2797–2819.
[20] A. Khormali, J. Park, H. Alasmary, A. Anwar, M. Saad, D. Mohaisen, Domain name system security and privacy: A contemporary survey, Comput. Netw. 185 (2021) 107699.
[21] A.A. Zuraiq, M. Alkasassbeh, Phishing detection approaches, in: 2019 2nd International Conference on New Trends in Computing Sciences (ICTCS), IEEE, 2019, pp. 1–6.
[22] G. Vrbančič, I. Fister Jr., V. Podgorelec, Datasets for phishing websites detection, Data Brief 33 (2020) 106438.
[23] PhishTank | join the fight against phishing, 2021, [Online; accessed 24. May 2021]. URL http://phishtank.org.
[24] citizenlab, Test-lists, 2021, [Online; accessed 24. May 2021]. URL https://github.com/citizenlab/test-lists.
[25] Welcome to python.org, 2021, [Online; accessed 29. Apr. 2021]. URL https://www.python.org.
[26] TensorFlow, 2021, [Online; accessed 29. Apr. 2021]. URL https://www.tensorflow.org.
[27] K. Team, Keras: the python deep learning API, 2021, [Online; accessed 29. Apr. 2021]. URL https://keras.io.
[28] Scikit-learn: machine learning in python – scikit-learn 1.0.1 documentation, 2022, [Online; accessed 2. Jan 2022]. URL https://scikit-learn.org/stable.
[29] Rfc1035, 2021, [Online; accessed 23. Dec. 2021]. URL https://datatracker.ietf.org/doc/html/rfc1035.
[30] Joblib: running python functions as pipeline jobs – joblib 1.2.0.dev0 documentation, 2021, [Online; accessed 25. Dec. 2021]. URL https://joblib.readthedocs.io/en/latest.
[31] HTML URL encoding reference, 2022, [Online; accessed 21. May 2022]. URL https://www.w3schools.com/tags/ref_urlencode.asp.
[32] Welcome to cymruwhois's documentation! – cymruwhois v1.0 documentation, 2021, [Online; accessed 25. Dec. 2021]. URL https://pythonhosted.org/cymruwhois.
[33] Pythonwhois, 2021, [Online; accessed 25. Dec. 2021]. URL https://pypi.org/project/pythonwhois.

Mohammed M. Alani holds a Ph.D. in Computer Engineering with specialization in network security. He has worked as a professor, and a cybersecurity expert in many countries around the world. His experience includes serving as VP of Academic Affairs in the United Arab Emirates, network and security consultancies in the Middle-East, and Cybersecurity Program Manger in Toronto Canada. He currently works as a Cybersecurity Professor at Seneca College, Toronto, Canada.

He has authored 4 books, and edited one in different areas of networking and cybersecurity along with many research papers published in highly ranked journals and conferences. He currently serves in editorial board of Journal of Reliable and Intelligent Environments. He has also guest edited many special issues in highly ranked journals.

He also holds many industrial certifications such as Security+, Cybersecurity Analyst+ (CySA+), PenTest+, CompTIA Advanced Security Practitioner+ (CASP+), Server+, Cisco Certified Network Associate, CCAI, Microsoft Azure Data Science Associate, and Microsoft AI Fundamentals. His current interests include applications of ML in cybersecurity, and ML security.

Hissam Tawfik is a Professor of Artificial Intelligence. He holds a Ph.D. in Computer Engineering from the University of Manchester (then UMIST) and has a research track record of more than 150 refereed publications in reputable international journals and conference proceedings in Applied Artificial Intelligence, Biologically Inspired Systems, and Data Science.

Hissam is currently working as a professor of Artificial Intelligence at the College of Engineering in Sharjah University, Sharjah, UAE. Hissam also leads the Computer Science and Artificial Intelligence research at the School of Built Environment, Engineering and Computing. Before joining Leeds Beckett University in 2015, Hissam worked for University of Salford and Liverpool Hope University.

Hissam is an editor of the International Journal of Future Generation Computer Systems (Elsevier), the International Journal of Neural Computing and Applications (Springer), and International Journal of Reliable Intelligent Environments (Springer). He is a visiting Professor at the University of Seville (Spain), and is a chair of the International Conference Series on Developments in eSystems Engineering (DESE).

Hissam is currently guest editing the journal special issue on "Special Issue on Expert Decision Making for Data Analytics with Applications"; International Journal of Applied Soft Computing (Elsevier), and the journal special issue on "Robotics, Sensors and Industry 4.0"; Journal of