Malicious URL Detection Using Machine Learning

*Network Security Final Project Report

Jiahao Shen, Lingze Zeng, Yue Zhu, Yan Pang
Department of Electrical and Computer Engineering, Department of Computer Science
University of Virginia
Charlottesville, Virginia, USA
{ajy6qv, qvv4pv, hvs6uc, trv3vx}@virginia.edu

***Abstract*—A malicious URL is a web address that has been created for the purpose of causing harm or exploiting vulnerabilities in a user's computer or mobile device. These types of URLs are usually associated with phishing scams, malware, and other forms of cyber attacks. Nowadays, malicious URL is causing more and more damage to people's property and privacy. In the paper, the team extracted 23 features that have been well discussed together with 5 URL content-based features introduced by this work and formed 3 categories of training input with a combination of different features to train 4 distinct machining learning models. We evaluated all 12 training strategies by analyzing the accuracy of the models. Then, with the conclusion, the Multilayer Perceptron (MLP) achieved the highest accuracy of 96% when using all features, leading us to recommend selecting the MLP model with the feature extraction strategy for malicious URL detection.**

I. Introduction

Resources on the Internet are located by their Uniform Resource Locator (URL). The features and two fundamental parts of a URL are described in [4] by Sahoo et al. These are the protocol identifier, which determines the protocol to use, and the resource name, which identifies the IP address or domain name where the resource is located. Each URL has a distinct structure and format. Attackers frequently attempt to alter one or more URL structural elements in an effort to trick users into accessing and sharing their malicious URLs.

There are now two primary categories of approaches to solving the problem of identifying malicious URLs: via blacklisting, and machining-learning based approaches [4]. Malicious URLs can be precisely detected using a matching approach based on existing blacklists without requiring much computing resources. This strategy, however, is unable to identify new malicious URLs that do not match any specified component. While the other method of malicious URL detection uses machine learning algorithms to categorize URLs according to website behavior analysis, which enables generality and scalability in detecting malicious URLs with high dynamic.

In our project, we focus on extracting three categories of features from a website, then applying 3 sets of features to 4 machining learning techniques, performances on the test set of all the 12 fitted models will be compared and analyzed. The structure of the essay is as follows. The description of the raw dataset is covered in Section II. Section III presents the URL filtering and attribute extraction. The new features for the URL detection process are also thoroughly detailed in this section. The model training is available in Section IV. Section V covered the evaluation and analysis of the performances. Section VI devotes the conclusion.

II. Data Collection

Network privacy and security are significantly threatened by malicious websites. These URLs harbor unwanted content, such as spam, phishing schemes, drive-by downloads, and more, putting users at risk for scams involving financial loss, theft of personal information, and malware installation. Each year, such scams result in billions of dollars in damages. To address this issue, we need to first construct a comprehensive dataset containing numerous examples of malicious URLs as well as benign websites. This dataset will enable the exploration of finding the best model training strategy to identify and prevent these harmful URLs from infecting computer systems or spreading across the internet.

We began with a raw dataset found on Kaggle. This dataset contains a huge number of labeled websites. According to the dataset, four different labels are tagged: Benign, Phishing, Defacement, and Malware. But there are some challenges coming with this raw dataset that can not be neglected.

The first challenge is that this dataset was last updated 2 years ago, but the website owners would have to pay for web services or domain names annually or monthly, which means websites are not held permanently. We investigated some URLs randomly selected from the labeled URLs, the investigation showed that most (intuitively more than half, we will show this observation is right in the next section) websites are no longer or temporarily not available, such as `3XX redirection`, `4XX client errors`, and `5XX server errors`. Here we list some typical abnormal *HTTP Response Status-Code*:

- `301 MOVED PERMANENTLY`: This and all future requests should be directed to the given URI (Uniform Resource Identifier).
- `404 NOT FOUND`: The requested resource could not be found but may be available in the future. Subsequent requests by the client are permissible.
- `503 SERVICE UNAVAILABLE`: The server cannot handle the request (because it is overloaded or down for maintenance). Generally, this is a temporary state.

- `504 GATEWAY TIMEOUT`: The server was acting as a gateway or proxy and did not receive a timely response from the upstream server.

And this problem is not shown uniquely within this particular dataset but exists in every outdated open-source dataset. So we have to find a solution to filter the inactive first to get a reliable labeled URL dataset.

Another challenge is that we found that some websites in which the response is different from the original label provided by the dataset builder, such as changing from benign to defacement or adverse, or other changes that we cannot define properly. Due to the limitation of time and the potential huge workloads on rebuilding a reliable dataset ourselves, we assume there is only a little mismatch between the label and the real type of website it is, which would have a limited impact on the results of model training.

III. Attribute extraction

We developed a multi-crawler and implemented several Python scripts to parse the URLs and collect desired features that can sufficiently describe the URL, and at the same time, they can be interpreted mathematically by machine learning models. There are three categories of features [4] we mainly focused on: lexical features, host-based features, and content-based features. Next, we will talk about the challenges and implementations for each part in detail in this section.

A. MileStone I: Multi-thread crawler

We implemented two versions of the web crawler, both powered by Python *Resquest*. The first version is a single-thread crawler, which is also the test version where we deal with bugs and continuously discover unknown characteristics by making requests to the web server via label URLs from the original raw dataset. There came up several challenges when developing the crawler.

The first challenge was that there were a lot of simplified URLs in the dataset, they lack the declaration of protocol at the beginning, this problem will raise the `MissingSchema`: `Invalid URL` error of *Resquest*, but most of which can be correctly accessed by input the URL to browser manually, this is because the auto-completion has been supported by most modern browsers but not yet by the Python *Resquest* library, so we have to consider this kind of exceptions by adding ”http://” or ”https://” to the head of URL which caused the error.

The second challenge was that there were some corrupt websites in the dataset, abnormal *Response Status-Code*, which we have discussed in the Data Collection section, would return when trying to request to those servers, to keep the correctness of label in the raw dataset, neither redirection nor client/server error would be considered as active URL. To be short, we decided only to keep URLs that return `Response 200`.

Moreover, connectivity issues might occur and hang the crawler without raising any error when making requests to the server, so additional timeout and retry mechanisms have to be added.

By applying the single-thread crawler to go through the raw dataset, we found that most URLs were not currently active, which were 11709 active out of 26054 requested URLs, and in which only 300 phishing URLs were detected to be active, but the whole requesting process took 492 minutes for the single-thread crawler to finish crawling.

So in order to enable large-scale research on this topic, we extend the crawler to a multi-thread version with the same functionality. 1000 threads are scheduled to perform 40 URL-crawling each, which is the maximum number of threads supported by the testing environment. And this multi-thread crawler turned out to be able to go through 40000 URLs from the raw dataset in 52 minutes. The filtered dataset consists of 17919 active URLs, of which 15504 are benign, 1663 are defacement, 485 are phishing websites, and 267 are labeled as malware.

Here is a link to the final version of the filtered dataset we would use for the data processing procedures next, the raw HTML content has been kept in this file so that we need not access each URL one more time when performing feature extraction. And we decided not to include more data because the data volume has already come to 4.56 GB for the same reason, but we have shown our capability to grab more URLs within an acceptable time.

B. Milestone II: Lexical Features

A script was developed to parse the URLs. Figure 1 shows the features the script extracted from the processed URL from the last milestone. In the milestone, the URLs were parsed and extracted with different features. Compared with safe and normal URLs, malicious URLs always contain more hyphens and symbols. Therefore, it is an important procedure to learn about how many different symbols contain in an URL that is being analyzed. Moreover, short URLs are always URLs in disguise. Usually, the short link is a mask for another link. Thus, it is hard to know the true source of the link. Besides length and special symbols in URLs, some meaningful information contained also needs to be considered. For instance, words like ’account, secure, banking, login, and signin’ are commonly used in phishing URLs.

To help our machine learning model better analyze the URLs, the script needs to extract the features introduced above. Besides only the four different labels of URLs, after processing with the script, features like the number of different symbols, length of URLs, and statistics of sensitive words are added to the dataset. From our expectation, these new features can help machine learning models to have better performance in recognizing malicious URLs.

C. Milestone III: Host-based Features

In addition to URL features extract from the last step, there is also a lot of important information contained in WHOIS records. Usually, a WHOIS record contains contact information associated with the person, group, company, registrant, or registrar. A script powered by Python *whois* library is implemented to query the WHOIS server directly and return

Feature	Description
NumDots	Number of character '.' in URL
NumDash	Number of the dash character '-'
AtSymbol	There exists a character '@' in URL
TildeSymbol	There exists a character '~' in URL
NumUnderscore	Number of the underscore character
NumPercent	Number of the character '%'
NumQueryComponents	Number of the query components
NumAmpersand	Number of the character '&'
NumHash	Number of the character '#'
NumNumericChars	Number of the numeric character
QueryLength	Length of the query
HostnameLength	Length of hostname
NumSensitiveWords	Number of sensitive words

Fig. 1. URL Lexical Base Features Might Contain Risks

parsed WHOIS data for a given domain. With this information, it is easier for our models to summarize the commonalities of the URLs.

In order to help machine learning models to understand and map into mathematical space, a Python script is created to translate WHOIS information. In the project, because the state name and registrar have a lot of duplicates, and they are the string type information, we decide to make a hash table for this information. However, the information from WHOIS has a server overlapping problem. For instance, the state name, New York, is interpreted as "New York" and "NY" from WHOIS. Therefore, to make a consistent and precise hash table, we have to process the data from WHOIS again to merge the same states with different spelling.

Moreover, instead of directly using data time information precise to second, we decided to use only the specific year the domain is created. Since we believe accurate to second contains too much detail for a machine model.

Feature	Describtion
registrar	Name of registrar
domain	Name of domain
create_date	Register date of the URL
state	State where the domain registered

Fig. 2. URL Host Base Features Might Contain Risks

D. Milestone IV: Content-based Features

As the entire webpage of each active URL has been downloaded by our crawler, we now only need to implement a script to extract features from HTML content in the file. Content-based features are considered as "heavy-weight" compared to URL-based features [4] but are expected to provide more information that might help to detect a malicious URL. Besides whether HTML content contains empty HTML and client-side JavaScript, we mainly focused on two categories of content-based features: the number of different hyperlinks and the number of certain JavaScript functions, as listed in Table 3.

The number of different hyperlinks is introduced by this work because we believe different hyperlinks are an important part that interactive websites consist of. Ankit Kumar Jain and B. B. Gupta have shown this importance by combining hyperlinks analysis with machining learning methods in [3]. But due to the workloads, we can not replicate the same approach to divide hyperlink features into 12 categories, we chose 5 wide categories to describe the URLs, they are listed as `href_XXX_cnt` in Table 3. Different hyperlinks are simply divided by their structures. For example, *href="#XXX"* represents a relative hyperlink, and *href="https://XXX"* represents an external one. The text mining part of this script is powered by Python *html.parser* library.

Feature	Description
emptyHTML	Check if the website responses an empty HTML
javascript	Check if the HTML source code contains a client-side JavaScript code
hrefs_empty_cnt	Number of empty URLs in attrs 'href' of the HTML source code
hrefs_jsFunc_cnt	Number of JS function called in attrs 'href' of the HTML source code
hrefs_tel_cnt	Number of telephone number in attrs 'href' of the HTML source code
hrefs_relative_cnt	Number of relative URLs in attrs 'href' of the HTML source code
hrefs_external_cnt	Number of external URLs in attrs 'href' of the HTML source code
eval_cnt	Number of JS function eval() in HTML source codes
escape_cnt	Number of JS function escape() in HTML source codes
exec_cnt	Number of JS function exec() in HTML source codes
search_cnt	Number of JS function search() HTML source codes

Fig. 3. Content-based features from HTML

The number of certain JavaScript functions is chosen because JavaScript functions are commonly used by hackers to encrypt malicious code or to execute unwanted routines without the client's permission [2]. [1] identify seven native JavaScript functions that are frequently used in Cross-site scripting and Web-based malware distribution. We selected a subset of 4 of these functions as listed at the bottom of Table 3. More functions may be added to model training in future works.

IV. Model Training

There are three different training strategies employed for this model by using different combinations of datasets from the last section. The first strategy involves training the model using features extracted from URLs. The second training strategy utilizes features extracted from the WHOIS information and HTML source code of webpages. The third strategy combines both URL features and features from WHOIS information and the HTML source code. Four different machine learning models will be tested using these three distinct training strategies to identify the most effective strategy and model.

For processing and selecting the training data, several common techniques are employed, including mapping character data to numerical representations and scaling data with large ranges to a reasonable interval using data scaling methods before proceeding with training. It is essential to ensure the proper handling of data and the application of appropriate techniques to achieve optimal results.

A. K-means

The k-means algorithm is a widely used unsupervised machine learning method for clustering data points based on their similarity. The objective of the k-means algorithm is to partition the data into 'k' distinct clusters, where each data point belongs to the cluster with the nearest mean, serving as the cluster's centroid.

In the context of the given task, the k-means model is employed to identify patterns and groupings within the data. This is achieved by iteratively updating cluster centroids

and reassigning data points to their closest centroids until convergence is reached or a predefined number of iterations have been performed. By applying the k-means algorithm, insights can be gained into the underlying structure of the data, facilitating further analysis and decision-making.

B. Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during the training phase. By combining the predictions of these individual trees, the model can provide more accurate and robust results compared to a single decision tree. The model creates diversity among the decision trees by training each tree on a random subset of the dataset with replacement, also known as bootstrapping. This process helps to minimize overfitting, a common issue in single decision trees. Random Forest offers the ability to estimate feature importance, providing insights into which features (URL, HTML source code, or both) are the most influential in determining the classification of a website as malicious or non-malicious. To classify a new website, the Random Forest model takes a majority vote of the individual decision trees. Each tree casts its prediction, and the final output is determined by the class with the most votes. This approach enhances the model's overall accuracy and stability. the Random Forest model is well-suited for this classification task, as it leverages the power of multiple decision trees to deliver improved accuracy, robustness, and insights into feature importance. By utilizing this model, the task of identifying malicious websites based on URL and HTML source code features can be effectively achieved.

C. Multilayer Perceptron

MLP is a type of feedforward artificial neural network consisting of multiple layers of nodes, with each layer fully connected to the next one. MLP can learn complex, non-linear patterns in the data, making it a powerful choice for classification tasks. Each node in the hidden layers of an MLP uses a non-linear activation function, such as the rectifier or hyperbolic tangent functions. These activation functions enable the model to capture non-linear relationships between the input features and the target variable, which can be crucial for accurately classifying malicious and non-malicious websites. MLP employs the backpropagation algorithm to calculate the gradient of the loss function with respect to each weight by using the chain rule. This gradient is then used by an optimization algorithm, such as stochastic gradient descent, to update the weights and minimize the loss function, resulting in better model performance. MLP offers flexibility in terms of the number of hidden layers and nodes in each layer, allowing the model's complexity to be adjusted to match the complexity of the underlying data. Additionally, MLP can scale well with large datasets, making it suitable for tasks with a significant amount of data.

D. Support Vector Machine

SVM is a powerful supervised learning algorithm that seeks to find the optimal hyperplane that best separates the data points into different classes. It does so by maximizing the margin between the closest data points (support vectors) and the separating hyperplane, resulting in robust classification boundaries. SVM can efficiently handle non-linearly separable data by utilizing the kernel trick, which transforms the data into a higher-dimensional space where it becomes linearly separable. This enables the SVM model to capture complex relationships between the input features and the target variable, making it suitable for classifying malicious and non-malicious websites. SVM includes a regularization parameter, C, which controls the trade-off between maximizing the margin and minimizing the classification error. This allows for better control over the model's complexity and helps prevent overfitting, ensuring the model generalizes well to new data. SVM models are known for their robustness and stability, as they focus on the most critical data points (support vectors) when constructing the decision boundary. This makes the SVM model less sensitive to noise and outliers in the data, leading to more accurate and reliable predictions.

V. RESULT

In this section, the result of trained models will be discussed. In the process of the project, we used three different datasets to train and test the models.

Features	URL	WHOIS + HTML	All
K-means	0.63	0.92	0.92
Random Forest	1.0	1.0	0.97
MLP	0.99	0.99	0.96
SVM	0.93	0.96	0.94

Fig. 4. Comparison of Machine Learning Algorithm Accuracies with Different Feature Combinations

In the experiment, We explored three different training strategies and four distinct machine learning models. According to the data in the table, the Multilayer Perceptron (MLP) achieved the highest accuracy (0.96) when using all features, leading us to select the MLP model.

The reasons for not choosing the other three models are as follows: First, the accuracy of K-means is relatively low across all feature combinations. This may be due to the fact that K-means is a clustering-based algorithm and might not be suitable for classification tasks. Additionally, K-means is sensitive to feature scaling and initial cluster centers, which may require extensive tuning to obtain optimal results. Second, although Random Forest performed exceptionally well with URL, WHOIS and HTML features, its accuracy was slightly lower than MLP when using all features. Random Forest may have overfitted the training data, resulting in a somewhat inferior performance on the test data. However, in some cases, I might consider choosing Random Forest based on other evaluation metrics (such as F-1 score, recall, or precision). Finally, the accuracy of SVM was lower than that of MLP. While SVM may offer superior performance in certain situations, the experimental results indicate that its performance on this task was inferior to that of MLP. Possible

reasons include an unsuitable choice of the kernel function, insufficient hyperparameter tuning, or training data not being suitable for SVM.

In this experiment, we chose to use accuracy as the evaluation metric. Accuracy is the most intuitive and easily understood metric in classification tasks, measuring the proportion of correctly classified instances to all instances, thus providing a simple overview of overall performance. When a class distribution is relatively balanced, accuracy is typically an effective evaluation metric, as it can accurately reflect the model's performance across all categories. If the task has no specific focus (e.g., concern about false alarms or misses), then using accuracy as an evaluation metric may be appropriate. Conversely, if concerned with specific types of errors (such as false alarms or misses), then F-1 score, recall, and precision might be more suitable evaluation metrics.

VI. Conclusion

In the project, we extracted 23 features that have been well discussed together with 5 URL content-based features. In the experiment, We explored 3 different training strategies and 4 distinct machine learning models. We evaluated all 12 training strategies by analyzing the testing result, the Multilayer Perceptron (MLP) achieved the highest accuracy of 96% when using all features, leading us to recommend the feature extraction strategy with the MLP model for malicious URL detection.

VII. Team Member Roles

Jiahao Shen: Involved in all projects and work actively, mainly in data analysis and data processing. Follow up on all project steps. Analyzed and summarized all project data.

Yue Zhu: Involved in all projects and work actively, mainly working on data collection, attribute extraction, and data analysis. Followed up on all project steps.

Lingze Zeng: Involved in all projects and work actively, mainly in data collection, attribute extraction, and data analysis. Followed up on all project projects.

Yan Pang: Involved in all projects and work actively, mainly in model training and data analysis. Followed up on all project projects.

Github link.

References

[1] Hyunsang Choi, Bin B. Zhu, and Heejo Lee. Detecting malicious web links and identifying their attack types. In *Proceedings of the 2nd USENIX Conference on Web Application Development*, WebApps'11, page 11, USA, 2011. USENIX Association.

[2] Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. Malicious web content detection by machine learning. *expert systems with applications*, 37(1):55–60, 2010.

[3] Ankit Kumar Jain and Brij B Gupta. A machine learning based approach for phishing detection using hyperlinks information. *Journal of Ambient Intelligence and Humanized Computing*, 10:2015–2028, 2019.

[4] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*, 2017.