You Got a Typo: Detecting Typo-Squatting domains using Machine Learning Approach

ADITI VENKATESH, School of Computer Science, University of Guelph

SHREYA BHOJE, School of Computer Science, University of Guelph

Typo-squatting is an unethical practice in the realm of cybersecurity that capitalizes on human oversight and inexpensive domain registrations to divert genuine traffic from top-level domains. This method is commonly employed for phishing attacks, redirecting users to competitors' sites, or disseminating inappropriate or harmful content, posing an ongoing concern for businesses. Instances have been documented where threat actors utilize typo-squatted domains for the distribution of malicious software. Despite multiple awareness initiatives designed to educate individuals on recognizing potentially harmful URLs, the number of victims continues to rise. Given that human error represents the inherent vulnerability exploited by these malicious actors, it is imperative to close this gap by enhancing user awareness, thereby averting potential phishing threats. Typo-squatting calls In this report we have implemented a browser extension with capability to detect a potentially typo squatting domain leveraging a machine learning model which allows the user to verify the legitimacy of the URL the user has types in the browser.

CCS CONCEPTS • **Computing methodologies → Machine learning • Human-centered computing → Usability**

Additional Keywords and Phrases: Typo-squatting, Browser extension, Real-time analysis, Support Vector Machine, Flask

1 INTRODUCTION

People, Process and Technology are three pillars of cybersecurity. With conjunction of these components the cyber threats can be reduced or at least mitigated effectively. Typo-squatting targets the people part from its deceitful way of persuasion. With more than 700 billion registered domains as of September 2023, there is a huge pool of domain names to spoof and to monetize on for the adversaries [1]. Typo-squatting is one of the relevant methods used by the spoofers. To make typo-squatted domain, adversaries first need to find a popular domain name. This can be a trending domain name or if the attacker wants to target any specific target, then they can choose any popular domain name relevant to that sector. The fake domains are created considering common misspellings of the top-level domains or by incorporating methods such adding characters at the beginning or end of the legitimate domain name while maintaining its resemblance with the legitimate domain name [2].

Typo-squatting, as indicated by a study [3], does not pose a substantial threat in terms of malware infections. The research findings highlighted that the incidence rate of malware on typo sites was lower when compared to legitimate sites within the Alexa top 1,000,000 site list. The impact of phishing attacks and typo-squatting on both users and businesses is substantial, encompassing a range of detrimental effects. Phishing attacks, characterized by increasingly sophisticated tactics, pose a significant threat to users who may inadvertently share sensitive information. The consequences of falling victim to such attacks include identity theft and financial loss. On the other hand, typo-squatting sites, which impersonate legitimate domains, have the potential to deceive users into divulging confidential details to malicious actors. Beyond the immediate losses incurred, these fraudulent activities tarnish the reputation of the genuine businesses, eroding customer trust. Additionally, businesses face financial setbacks, encompassing both lost sales and legal fees, as they combat the repercussions of fraudulent activities conducted by typo-squatters and successful phishers. The cumulative effect is a multifaceted challenge for users and businesses alike, emphasizing the critical need for robust cybersecurity measures to mitigate these pervasive threats [4]. It is crucial to design and build solutions against typo-squatting keeping in mind the balance between security and usability. Focusing on the ease of use in solution is essential as this attack exploits how easily users misspell the domain names and. This research contributes significantly to the cybersecurity field by presenting an innovative solution to address typo-squatting threats. Through the integration of machine learning, real-time analysis, and user-centric design, our Chrome browser extension serves as a proactive measure to protect users from the risks associated with deceptive domains, effectively closing the vulnerability gap that threat actors exploit.

2 RELATED WORK AND BACKGROUND

Over the years, there has been numerous works done on developing methods on detecting typo-squatting domains. The work generally falls under three categories: anatomy of typo-squatting domain, generation of typo-squatting domains and typo-squatting detection tools.

2.1 Anatomy of Typo-squatting domain:

With subtle variations in the legitimate domain names, multiple combinations of typo-squatting domains can be created. Many studies have focused on what can be different ways to detect such domain using different algorithms and models. The most common instances of typing errors often occur with a one-character distance, referred to as Damerau-Levenshtein (DL) distance one, from the accurate spelling [5]. As mentioned in the study by Wang *et al.* [6], there are five widely used generation models are: Missing-dot typos, Character-omission typos, Character-permutation typos, Character-replacement typos and Character-insertion typos.

Along with this, there are features of typo-squatting domains that must be considered to detect them. Such as domain name length, domain name popularity, effect of the top-level domain, etc. [7]. In this paper, we have chosen eight such features: Bi-gram extraction, Length of Domain, Number of Subdomains, Keyboard Proximity, Character Frequency, Levenshtein Distance, Hyphen Count, Vowel Count and Consonant Count, to be used to create vectorized data to feed the machine learning model.

2.2 Generation of typo-squatting domains

There have been many tools developed to generate potentially generated typo-squatting domains as it difficult to get the dataset of malicious typo-squatting domains publically. A study developed a tool Typo-writer that uses different algorithms as mentioned in above section to generate list of typo-squatting domain names [8]. In this paper, we have used a open source tool DNS Twist that performs multiple permutations and combinations to produce list of potentially typo-squatting domain names that we have used a classification feature to train the machine learning model [9].

2.3 Typo-squatting detection tools:

A study developed browser extension to detect typo-squatting domains using the list frequently visited websites as the legitimate in the extension's repository. The repository updates based on user feedback [10]. Typo-squatting detection tools have been developed to detect the attack against the vulnerabilities against the Domain Name Service (DNS) using machine learning approach [11]. Using machine learning gave flexibility and enhanced the accuracy of the detection model. As our motivation is to focus on usable security, we have built a browser extension with the capability to detect the typo-squatting domain using machine learning approach.

3 METHODOLOGY

The methodology, particularly designed for Google Chrome browser extension, leverages advanced technologies such as machine learning and real-time URL analysis to effectively identify and counteract typo-squatting threats. The effectiveness of the chrome browser extension is evaluated through user feedback of its typo-squatting detection capabilities. We have collected legitimate domains, top 50 domains from Similarweb.com (good) [12] and potentially typo-squatted domains DNS Twist (bad) [9] for those top 50 domains.

3.1 Requirements

To implement this idea the backend (Python/Flask) requires libraries such as pandas, scikit-learn [13], Flask, and joblib, along with a trained Support Vector Machine (SVM) which are effective for binary classification tasks and are known for their ability to handle high-dimensional feature spaces. The chosen features, both bi-grams and additional URL-based features, create a feature matrix that captures various aspects of URLs. SVMs can learn complex decision boundaries in such feature spaces, making them suitable for this task model. The frontend (JavaScript) involves Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) files for the user interface and leverages the fetch API for communication with the Flask API.

3.2 Algorithm

As mentioned in Fig. 1 the detailed algorithm for Backend code is given below:

1. **Data Loading:**
   - Read a CSV file ('Dataset.csv') into a Pandas DataFrame (df).

2. **Feature Engineering:**
    1. Use CountVectorizer [14] to convert character n-grams (bi-grams in this case) of the 'url' column into a sparse matrix (bi_grams).
    2. Create additional features based on the URL:
        - 'length_of_domain': Length of the URL.
        - 'number_of_subdomains': Number of subdomains in the URL.
        - 'keyboard_proximity': Sum of Levenshtein distances between each character in the URL and the letter 'a'.
        - 'character_frequency': Sum of ASCII values of characters in the URL.
        - 'levenshtein_distance': Levenshtein distance between the URL and the word 'legitimate' [15].
        - 'hyphen_count': Count of hyphens in the URL.
        - 'vowel_count': Count of vowels in the URL.
        - 'consonant_count': Count of consonants in the URL.
    3. Combine the bi-gram matrix and additional features into a single feature matrix X.
    4. Define the target variable y as the 'class' column.

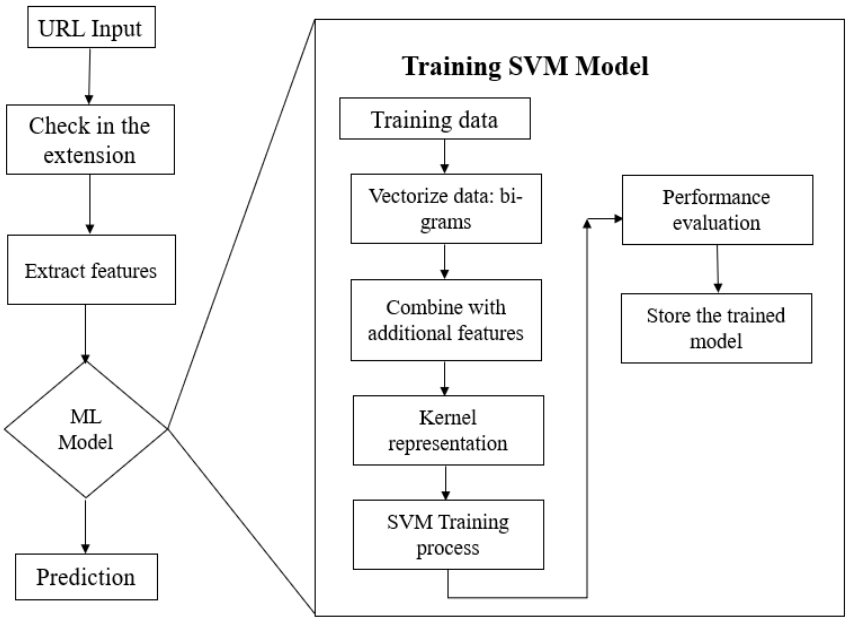

Fig. 1: Comprehensive URL Analysis Workflow

3. **Train-Test Split:**
    - Split the data into training and testing sets using train_test_split.
4. **SVM Model Training:**
    - Create an SVM model with a linear kernel using SVC.
    - Fit the model on the training data.
5. **Model Evaluation:**
    - Make predictions on the test set.
    - Get the predicted values, accuracy, classification report, and confusion matrix.
6. **Model and Vectorizer Saving:**
    - Save the trained SVM model and the CountVectorizer vocabulary to joblib files.
7. **Flask API [16]:**
    - Load the saved SVM model and CountVectorizer vocabulary in a Flask application.
    - Define a function predict_svm to make predictions on a given URL using the trained SVM model.
    - Create a Flask route /predict that accepts POST requests with a JSON payload containing a URL.

- Extract the URL from the JSON payload, make a prediction using predict_svm, and return the result as JSON.

8. **Flask Application Run:**
   - Run the Flask application on port 5000 in debug mode.

The integration of JavaScript within a web browser to perform URL checking using a backend server. The function checkURL() from frontend retrieves the user-entered URL from the input field with the id 'urlInput'. It checks if a URL is provided, and if not, it updates the content of element with the id 'result' to prompt the user to enter a URL. The function handles the response from the server. If the prediction from the server is "bad," it alerts the user that the URL may be a typosquat. Otherwise, it logs the prediction and alerts the user that the URL appears to be legitimate.

4 RESULTS

The output represents the predictions made by the Support Vector Machine (SVM) model on the test set. Each prediction corresponds to an entry in the test set is either good or bad. Following the predictions, the accuracy of the SVM model on the test set is calculated and printed, providing a quantitative measure of the model's overall correctness. Additionally, the classification report and confusion matrix are displayed. The classification report includes precision, recall, and F1-score for each class, offering insights into the model's performance on individual classes. The confusion matrix further details the count of true positive, true negative, false positive, and false negative instances, aiding in the assessment of the model's predictive accuracy. Fig 2 is the detailed output when we ran the dataset from our code.

```
Model Accuracy (SVM): 0.9622454751131222
Classification Report (SVM):
              precision    recall  f1-score   support

         bad       0.96      0.95      0.96      3068
        good       0.96      0.97      0.97      4004

    accuracy                           0.96      7072
   macro avg       0.96      0.96      0.96      7072
weighted avg       0.96      0.96      0.96      7072

Confusion Matrix (SVM):
[[2911  157]
 [ 110 3894]]
```

Fig. 2: Performance Summary Metrics

The Python script "Backend.py" is executed in a command prompt, initiating a Flask web application. The server is running in debug mode, as indicated by "Debug mode: on." The application is accessible locally at http://127.0.0.1:5000. The server restarts with a watchdog (windowsapi), and the debugger is active, with a provided PIN for debugging purposes. This information suggests that the Flask application is successfully running in a development environment, ready for local testing and debugging.

The provided Fig. 3 is from a Flask web application where HTTP requests are being processed for URL prediction. Here's a summary:

1. **Request 1:**
   - Method: OPTIONS
   - Path: /predict
   - Status Code: 200
   - Details: An OPTIONS request is made to the /predict endpoint, and the server responds with a status code of 200. The subsequent POST request includes a JSON payload with a URL parameter ('url': 'openai.com'). The server correctly

extracts the URL (‘openai.com’) and processes the prediction. The response indicates that the prediction for ‘openai.com’ is ‘good,’ and the status code for the POST request is 200.

2. **Request 2:**
   - Method: OPTIONS
   - Path: /predict
   - Status Code: 200
   - Details: Another OPTIONS request is made to the /predict endpoint, receiving a 200 status code. The subsequent POST request includes a JSON payload with a URL parameter (‘url’: ‘quorao.com’). The server correctly extracts the URL (‘quorao.com’) and processes the prediction. The details of the prediction result are not provided in this snippet, but the status code for the POST request is 200.

```
(base) C:\Users\MCTI Student\Documents\Project topics>python Backend.py
 * Serving Flask app 'Backend'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 803-236-270
127.0.0.1 - - [04/Dec/2023 19:06:01] "OPTIONS /predict HTTP/1.1" 200 -
{'url': 'openai.com'}
openai.com
good
127.0.0.1 - - [04/Dec/2023 19:06:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [04/Dec/2023 19:06:33] "OPTIONS /predict HTTP/1.1" 200 -
{'url': 'quorao.com'}
quorao.com
bad
127.0.0.1 - - [04/Dec/2023 19:06:33] "POST /predict HTTP/1.1" 200 -
```

Fig. 3: Flask Web Application Startup Log

Fig. 4 depicts the result of typo-squatting URL detection, specifically highlighting URL is classified as “legitimate” by the machine learning model.

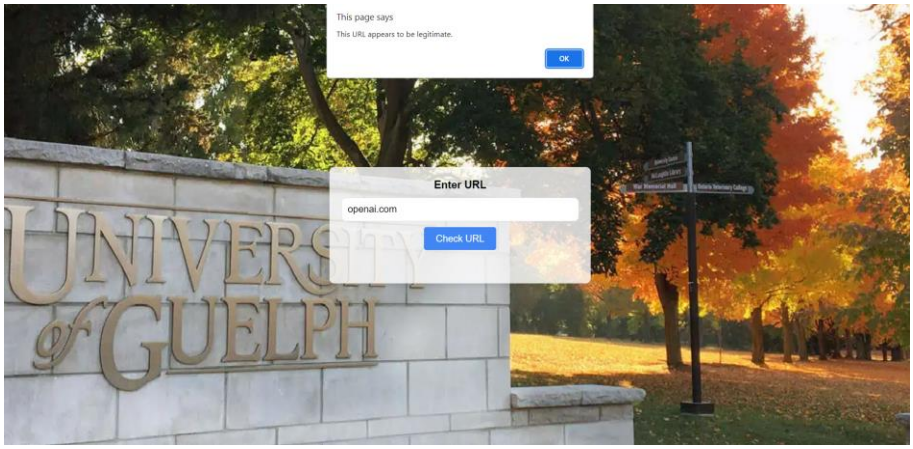

Fig. 4: Legitimate URL Detection

Fig. 5 visually represents the output of typo-squatting URL detection, focusing on URL classified as “typosquat.”

Fig. 6 depicts the output of legitimate URL in the context of typo-squatting detection within a browser extension. It visually represents instances where the integrated machine learning model correctly classifies URLs as "legitimate," ensuring accurate identification and providing a visual summary of safe URLs during web browsing.

Fig. 7 illustrates the output of typo-squatting URL detection within a browser extension context. It visually represents the results of URL classified as "typosquat" by the machine learning model integrated into the extension. The figure includes how the extension successfully identifies and alerts users to potential typo-squatting threats during web browsing.

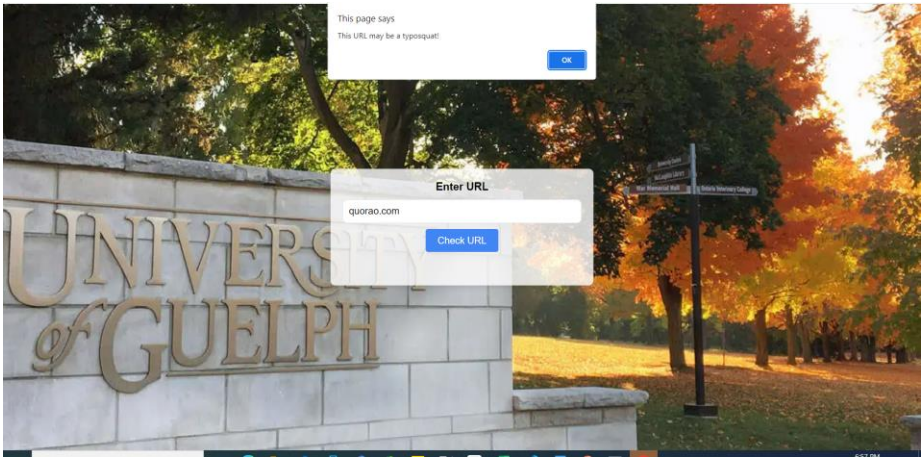

Fig 5: Typo-squatting URL Detection

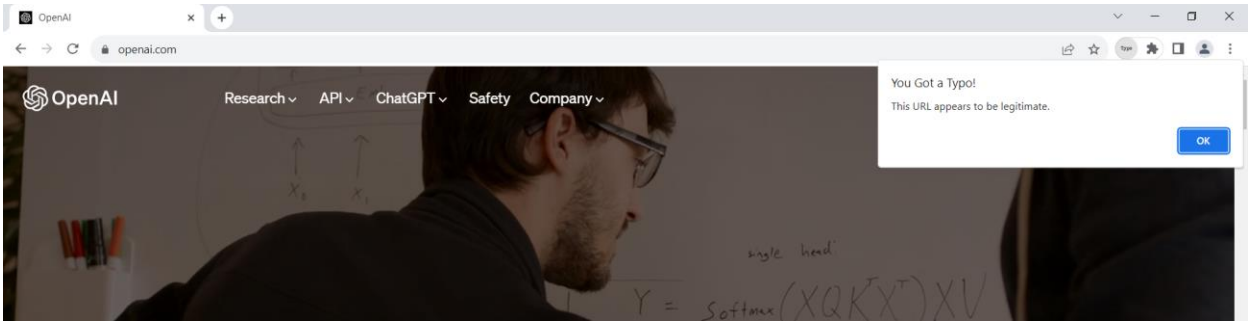

Fig 6: Legitimate URL Detection Results in Browser Extension

Fig 7: Typo-squatting Detection Results in Browser Extension

5 FUTURE WORKS

The browser extension's usability can further be enhanced by implementing multimodal analysis that will detect the typo-squatting domain using URL as well by analyzing the web page content. To make the detection accurate, dynamic update can be implemented by integrating the typo generating tool with the extension. This solution can be implemented on organizational level and integrated with the security products such Intrusion Detection System (IDS), Web application filtering system and Firewall as active threat intelligence collection source. In summary, this future check involves the integration of advanced machine learning capabilities, a dynamic model, and collaboration with existing security infrastructure to create a robust and adaptive defense mechanism against typo-squatting attacks at an organizational level. The emphasis on continuous updates and efficient tools reflects a commitment to staying ahead of evolving cybersecurity threats.

REFERENCES

[1] Domain Name Stat “WHOIS Database Download” https://domainnamestat.com/whois-database-download

[2] Bolster AI “Typosquatting” https://bolster.ai/glossary/typosquatting#How_Typosquatting_Works

[3] J. Szurdi, B. Kocso, G. Cseh, M. Felegyhazi, and C. Kanich, “The Long “Taile” of Typosquatting Domain Names,” in Proceedings of the 23rd USENIX Security Symposium, 2014.Chelsea Finn. 2018. Learning to Learn with Gradients. PhD Thesis, EECS Department, University of Berkeley.

[4] Medium “Typosquatting: Unveiling the Hidden Dangers Lurking in Your Typos” https://medium.com/@troof.io/typosquatting-unveiling-the-hidden-dangers-lurking-in-your-typos-34d2c4a4aad

[5] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, Chris Kanich “The Long “Taile” of Typosquatting Domain Names” in the Proceedings of the 23rd USENIX Security Symposium

[6] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, Brad Daniels “Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting” in USENIX SRUTI, 2006

[7] Jeffrey Spaulding, Shambhu Upadhyaya, Aziz Mohaisen “You’ve Been Tricked! A User Study of the Effectiveness of Typosquatting Techniques” in

[8] Ishtiyaque Ahmad, Md Anwar Parvez, Anindya Iqbal “TypoWriter: A Tool to Prevent Typosquatting” in IEEE 43rd Annual Computer Software and Applications Conference, 2019.

[9] DNS Twist. https://dnstwist.it

[10] Guanchen Chen, Matthew F. Johnson, Pavan R. Marupally, Naveen K. Singireddy, Xin Yin, Vamsi Paruchuri “Combating Typo-Squatting for Safer Browsing” in International Conference on Advanced Information Networking and Applications Workshops, 2009

[11] Abdallah Moubayed, MohammadNoor Injadat, Abdallah Shami, and Hanan Lutfiyya “DNS Typo-squatting Domain Detection: A Data Analytics & Machine Learning Based Approach” in IEEE Global Communications Conference, 2018

[12] Similarweb https://www.similarweb.com

[13] Scikit-learn Documentation. https://scikit-learn.org/stable/documentation.html

[14] Levenshtein Distance. https://en.wikipedia.org/wiki/Levenshtein_distance

[15] CountVectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[16] Flask Documentation. https://flask.palletsprojects.com/en/2.0.x/