



Malicious QR Code Detection Using Machine Learning

¹ Kaushik Goswami, ² Subhagata Sardar, ³ Sreeja Saha, ⁴ Hrishita Ray

¹ Faculty, ^{2,3,4} Student

^{1,2,3,4} Postgraduate and Research Department of Computer Science

^{1,2,3,4} St. Xavier's College (Autonomous), Kolkata, India

Abstract: QR codes have become a widespread means for conveying URLs and other information in a quick, scannable format. However, this convenience has also created ways for cyber threats, where attackers embed malicious URLs that lead to phishing sites, malware downloads or other harmful content. This project presents a Malicious QR detection Module that uses machine learning to assess the safety of the URLs embedded within the QR codes. By extracting and analysing embedded URLs, the system classifies QR codes as either malicious or non-malicious based on learned patterns associated with cyber threats. The URL is normalized; features are extracted and vectorized such that a trained model can detect indicators of compromise and can enhance user security by warning against unsafe links before engagement.

Keywords: QR Code Security, Malicious URL Detection, Machine Learning, XGBoost, Real-time QR Scanning, Feature Extraction, TF-IDF Vectorization, URL Classification.

I. INTRODUCTION

QR codes have become a standard interface for quick access to websites, applications, and services in sectors ranging from retail and banking to healthcare and transportation. However, this convenience has also made them a target for exploitation by cybercriminals. Malicious QR codes can embed URLs that direct unsuspecting users to phishing websites, initiate malware downloads, or collect personal information without consent. These threats are often difficult to detect with the naked eye or traditional QR scanners, which lack the intelligence to assess the safety of the encoded content. As QR code usage continues to grow, so does the urgency for robust security mechanisms to detect and prevent such attacks. This project addresses that need by introducing a Malicious QR Detection Module powered by machine learning, capable of analysing and classifying URLs embedded in QR Codes based on threat potential, thereby improving user safety and mitigating digital risks in real-world applications. Leveraging both handcrafted features and TF-IDF-based textual analysis, our system accurately classifies QR-encoded URLs as benign, malicious, or invalid.

II. METHODOLOGY

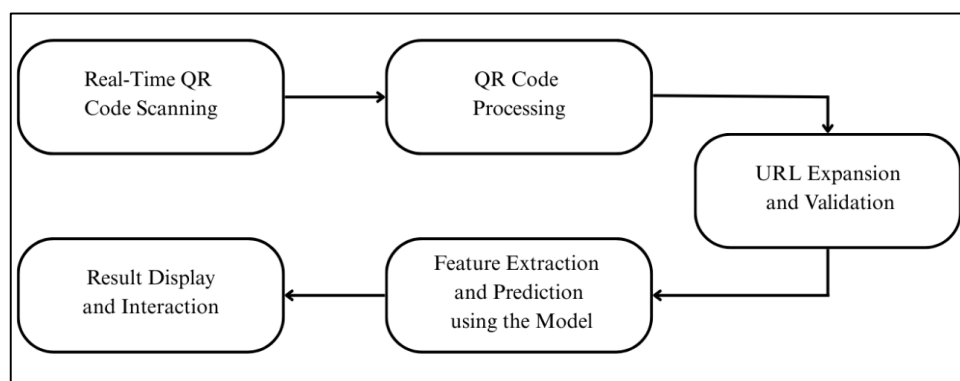


Figure 1: Methodology

2.1 Real-Time QR Code Scanning:

To extend the practical utility of the trained phishing URL detection model, we implemented a real-time QR code scanning module, which automatically extracts and classifies URLs embedded within QR codes. This module integrates computer vision and machine learning to detect malicious URLs at the point of interaction.

2.2 QR Code Processing Workflow:

The QR-Code Scanner is used for real-time video capture and QR code decoding. A video stream is being initiated from the system's default webcam. Each frame is processed in real-time to detect QR codes.

In order to enhance detection robustness, the script attempts QR code decoding on both the original grayscale image and its inverted binary counterpart, increasing the chance of successful reads under poor lighting or low contrast.

2.3 URL Expansion and Validation:

Upon successful QR code detection, the embedded URL is first passed through an unshortening function. Many malicious actors exploit URL shorteners to disguise harmful destinations. Known URL shortening services (e.g., *bit.ly*, *t.co*, *goo.gl*) are checked using domain extraction logic.

Once expanded, the URL undergoes a validation check to ensure syntactic correctness. This includes confirming the presence of a valid scheme (http or https) and a top-level domain (TLD). Invalid or unresolvable URLs are immediately flagged and rejected from further processing.

2.4 Feature Extraction and Prediction:

For all URLs not classified as inherently safe, the system extracts a set of engineered features identical to those used during model training. These include: URL structure-based attributes (e.g., length, symbol frequency, keyword presence), Shannon entropy measures for domain and path randomness, Executable file indicators and TF-IDF vectorization for capturing textual semantics.

This feature vector is then fed into the pre-trained XGBoost classifier, which returns a binary prediction: 1 for Malicious URL and 0 for Safe URL.

This ensures that even if a URL is syntactically valid and not in a known blacklist, it can still be accurately flagged as suspicious based on behavioural patterns and structural cues.

2.5 Result Display and Interaction:

If a valid prediction is made, the result is printed to the terminal, immediately informing the user whether the scanned QR code leads to a safe or malicious link. The scanning loop stops after one successful detection, or can be exited manually by pressing the 'q' key.

III. MODEL TRAINING

3.1 Model Selection: XGBoost

XGBoost (Extreme Gradient Boosting) follows a boosted tree-based ensemble learning approach that optimizes both speed and accuracy. Its architecture consists of key components that enhance its efficiency, reduce overfitting, and improve predictive performance. XGBoost follows a gradient boosting approach where new trees are created to minimize errors from previous ones. By following an iterative approach, XGBoost builds a strong ensemble model, where each tree refines the predictions made by the previous ones.

This results in a highly accurate and efficient classification model, especially for detecting complex patterns in data, such as distinguishing malicious and benign QR codes or URLs.

3.2 Data Collection and Labelling

The proposed system utilizes four distinct datasets comprising URLs classified into benign, malicious, and UPI-based categories. These include general benign URLs, general malicious URLs, benign UPI URLs and malicious UPI URLs.

Each record is labelled accordingly: 0 for benign, 1 for malicious, and 2 for syntactically invalid entries determined during preprocessing. The dataset is further augmented with heuristic safety rules based on trusted domains and specific UPI scheme validation.

3.3 URL Normalization and Validation

3.3.1 URL Cleaning:

URLs are cleaned and normalized before feature extraction to ensure consistency using the `urlparse` and `tlxextract` libraries. To ensure consistency and syntactic integrity, a normalization function is applied to all input URLs. This function performs the following steps:

Step 1: Use `urlparse` and `tlxextract` to parse the domain and suffix.

Step 2: Ensure the URL has a valid scheme (defaulting to HTTPS if absent).

Step 3: Reconstruct the domain from its extracted parts to maintain structural coherence.

Step 4: Label URLs as invalid (label 2) if the suffix (TLD) is missing or the parsing fails.

Step 5: Removing extra slashes (`https://example.com/` vs. `https://example.com`)

Step 6: Reformatting URLs to make them consistent.

3.3.2 Invalid URL Handling:

URLs that are not valid (due to incorrect structure or missing domain parts) are flagged as invalid and marked with label 2. This ensures that invalid URLs don't interfere with the classification process.

3.3.3 Handling Missing or Inconsistent Data:

Some URLs sometime miss crucial parts (like the domain or path). These cases can be either removed or imputed with default values. We flag invalid URLs, ensuring they're not used for training. That's filling the missing data.

In case of labelling invalid URLs, URLs are deemed invalid after the preprocessing step (for example, missing or malformed domain), label 2 (Invalid) is assigned to those URLs. This helps to train the model on distinguishing between malicious, benign, and invalid URLs.

3.4 Feature Engineering

The feature extraction process in this script is identical to what was used while training the model. This ensures that the input data (URLs) are transformed in the same way as during model training. If the features were extracted differently, the model would receive data in an unfamiliar format, leading to inaccurate predictions.

Key aspects of consistency in feature extraction consists of ensuring URLs are properly formatted and trusted domains are marked as safe, computing Shannon entropy for both domain and path, as done in training. Then it involves character counting that measures occurrences of . (dots), - (hyphens), @, ?, =, and & to detect suspicious patterns, checking for terms like "login", "secure", "bank", "verify", and "update", which are common in phishing attempts, flagging URLs containing .exe, .zip, .rar, and .apk as they might deliver malware and finally converting the URL text into numerical features using the same TF-IDF vectorizer trained on the dataset.

By keeping feature extraction identical, the script ensures that the model works optimally while scanning URLs extracted from QR codes.

A hybrid feature extraction strategy was adopted, combining statistical heuristics with textual vectorization:

3.4.1 Handcrafted features:

The features derived from the raw URL consists of URL length and character distribution (dots, hyphens, slashes, etc.), presence of suspicious keywords (e.g., login, secure, update), use of IP addresses in place of domain names, presence of suspicious file extensions (e.g., .exe, .zip, .apk) and Shannon entropy for both domain and path components to quantify randomness

These features aim to capture common obfuscation and deception tactics employed in phishing URLs.

Length of URL: The total number of characters in the URL.

Number of Dots (.): The number of dots in the URL can indicate subdomain complexity or suspicious patterns.

Number of Hyphens (-): Malicious URLs may have multiple hyphens to create fake-looking domains (e.g., login-secure).

Number of Special Characters: Counts of characters like: @ (commonly used in phishing); ? (used for query strings); = (used in query parameters); & (also for query parameters).

Subdomain Count: URLs with more subdomains could be suspicious, often used by malicious sites to mimic legitimate ones (e.g., login.secure.example.com).

Number of Digits: A higher count of digits may indicate a suspicious or fraudulent URL (e.g., online payment links often have numbers).

Number of Alphabetical Characters: Checks how many letters are in the URL.

Starts with https: URLs that use HTTPS are typically more trustworthy, as they indicate encryption.

Presence of an IP Address: Detects URLs containing IP addresses (\d+\.\d+\.\d+\.\d+). These URLs may not be fully valid or might be obfuscated.

Redirect Indicators: URLs containing "redirect" might indicate attempts to take the user to a different site than the one they expect.

Suspicious File Extensions: Checks if the URL contains file extensions like .exe, .zip, .apk, which are commonly associated with malware downloads.

Pattern Recognition using Regex based detection: The URL is checked for certain words commonly associated with phishing or scams: login, secure, bank, verify, update. These keywords suggest that the URL might be trying to impersonate a legitimate website and may lead to phishing attempts.

Entropy of Domain and Path: Entropy is a measure of randomness. A higher entropy in the domain or URL path can suggest the URL is randomly generated or obfuscated, which is a common trait in malicious URLs.

Path Complexity: The path of the URL (after the domain) is analyzed for complexity. Malicious URLs might use complex or seemingly random paths to evade detection.

3.4.2 TF-IDF Vectorization:

URLs are further processed using a TF-IDF vectorizer configured for character n-grams (2–3). This captures contextual patterns within the URL string that are often indicative of malicious intent (e.g., repeated substrings, encoded payloads).

The TF-IDF vectorizer is trained across the combined dataset and serialized using joblib for reusability. It extracts text patterns like character-level bigrams and trigrams i.e., extracts the most significant terms from the URL using 2-grams and 3-grams. The most frequent n-grams might indicate whether the URL is trying to impersonate known brands, using words like login, secure, pay. It helps capture things like: log.in, bank.verify, secure-login, etc. This vectorizer will later be used to convert URLs into numbers. For example, "https://www.login-secure.com/payments/verify.php?id=123" could have n-grams like ['ht', 'tp', 'ps', 's:', 'se', 'cu', 're', ...]

3.5 Model Architecture

The final feature vectors are composed of 22 handcrafted statistical features and Top 3 TF-IDF values (selected to balance dimensionality and information density). This feature set is used to train an XGBoost classifier configured for multi-class classification (objective='multi:softmax') with 3 output classes (benign, malicious, invalid).

After feature extraction, a supervised machine learning algorithm, XGBoost is used to classify URLs based on the extracted features. XGBoost ensembles learning with gradient boosting. It handles high-dimensional, sparse, and structured data and automatically learns complex feature interactions.

Training Setup include dataset sources i.e, URLs labelled as safe or malicious from sources like OpenPhish, PhishTank, Kaggle and Labels:0 for Safe;1 for Malicious.

Training Process includes Feature vectorization along with TF-IDF transformation. Then addressing class imbalance using scale_pos_weight and tuning Hyperparameters (max_depth, learning_rate, n_estimators, subsample, colsample_bytree using RandomizedSearchCV). The training process also involves regularization, i.e., to avoid overfitting, regularization techniques like L1 (Lasso) and L2 (Ridge) are used. These techniques penalize overly complex models and reduce the impact of irrelevant features. These techniques penalize overly complex models and reduce the impact of irrelevant features.

In Hyperparameter Tuning, we optimize the model for accuracy, generalization, and real-time performance by using RandomizedSearchCV that provides efficient hyperparameter tuning by sampling random combinations.

- max_depth- Controls tree depth. Prevents overfitting by limiting complexity.
- learning_rate- Shrinks the impact of each tree (boosting step size). Lower = better generalization.
- n_estimators- Number of trees in the model. More trees = better fit, but costlier.
- subsample- Fraction of data used per tree. Helps prevent overfitting.

- `colsample_bytree`- Fraction of features used per tree. Boosts generalization.

The optimization target is the weighted F1 score, ensuring robustness across all classes despite class imbalance.

In the Tuning Process the dataset is split into train/test using `train_test_split` with stratification. Then a parameter grid with value ranges for each hyperparameter is defined and combinations from the grid are sampled randomly. After this comes evaluation using cross-validation and classification metrics. And finally selecting the best-performing combination. This prevents model overfitting, improves detection of rare malicious patterns.

3.6 Post Processing

3.6.1 Final Model Evaluation: Once the model is trained, it is evaluated using the test set. This ensures that the model generalizes well to unseen URLs.

3.6.2 Model Saving: The trained model is saved as a pickle file (`xgboost_url_model2.pkl`) so that it can be used for future predictions on new URLs without retraining.

3.6.3 Prediction on New URLs: After the model is trained and saved, new URLs can be passed to the model to classify them as malicious, benign, or invalid.

IV. ALGORITHM

Step-1: Start

Step-2: Load the trained XGBoost model

Step-3: Load the trained TF-IDF Vectorizer

Step-4: Define known URL shorteners

Step-5: Initialize webcam using OpenCV

Step-6: Start real-time frame capture loop

Step-7: Convert each captured frame to grayscale

Step-8: Attempt to decode QR codes from the grayscale image

Step-9: If QR code is not detected, invert the grayscale image and try again

Step-10: If QR code is detected then

 Extract and decode the embedded URL

 Print the scanned URL

 Check if the URL is shortened, if yes, expand it using HTTP redirection

 Print the final unshortened URL

Step-11: Validate the structure of the URL using `urlparse` and `tlldextract`

 If invalid then print an error and exit

Step-12: Extract features from URL

Step-13: Combine features and reshape for prediction

Step-14: Pass features to the trained XGBoost model and predict the class

If prediction=1 then

Print “Malicious URL detected! Proceed with caution.”

If prediction=0 then

Print “Safe URL.”

Step-15: Exit the scanning loop and close webcam window

Step-16: End

V. TESTING

Following the completion of model training and feature extraction, the system proceeds with the evaluation phase to assess its effectiveness on unseen data. This phase involves using a separate CSV file containing URLs for testing purposes.

The testing script first loads the pre-trained XGBoost classification model and the corresponding TF-IDF vectorizer, both of which were serialized during the training phase. These are essential to ensure consistency in the transformation of raw input URLs into the same feature space used during training.

The input test data, supplied in CSV format, is subjected to the same validation and preprocessing pipeline as described previously. This includes URL normalization, safety rule enforcement (e.g., identifying trusted domains and UPI links), and rejection of syntactically invalid entries. Once pre-processed, the dataset is transformed into numerical feature representations using the pre-loaded vectorizer.

Predictions are then generated by the XGBoost model. Each URL in the test set is classified into one of three categories: benign, malicious, or invalid. If the test dataset includes ground truth labels, the system evaluates the performance using standard classification metrics:

1. Accuracy Score, indicating the overall correctness of predictions.
2. Precision, Recall, and F1-Score, provided per class through a detailed classification report to highlight the model's effectiveness in handling imbalanced data and differentiating between safe and malicious URLs.

In the absence of ground truth labels, the system omits the evaluation phase and outputs the predicted results directly to a new CSV file for further analysis or downstream use.

This testing process ensures that the model performs reliably in real-world scenarios, particularly when integrated with systems that scan and assess URLs extracted from sources such as QR codes.

Making Predictions: The model classifies each URL as safe or malicious, and the results are saved in a new CSV file.

Evaluating the Model: If the dataset contains actual labels, the script calculates- Accuracy score that measures overall correctness of predictions and the Classification report shows precision, recall, and F1-score.



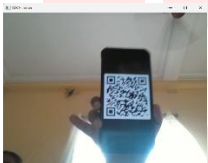

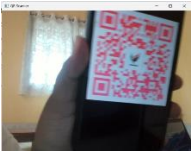

If no labels are available, the script only saves predictions without evaluation.

VI. RESULTS AND DISCUSSION

6.1 Results:

We used several QR codes of three types namely malicious, non-malicious or safe and those embedded with invalid URLs; to test the accuracy of our system. The following are a few examples:

Table 1: Results

QR Code Scanner	QR Code	URL Classification
		<pre>PS C:\Users\HP\OneDrive\Documents\subhagata> python scan.py Loaded trained model and initialized TF-IDF vectorizer. Scanning QR Code... Press 'q' to quit. Scanned URL: https://sxccal.edu/ Final URL: https://sxccal.edu/ True Safe URL.</pre> <p>Scanned URL: https://sxccal.edu/ Final URL: https://sxccal.edu/</p>
		<pre>PS C:\Users\HP\OneDrive\Documents\subhagata> python scan.py Loaded trained model and initialized TF-IDF vectorizer. Scanning QR Code... Press 'q' to quit. Scanned URL: https://tinyurl.com/29soc9qj Final URL: https://dramacool.bg/video-watch/i-live-alone-2013-episode-589-b True Malicious URL detected! Proceed with caution.</pre> <p>Scanned URL: https://tinyurl.com/29soc9qj Final URL: https://dramacool.bg/video-watch/i-live-alone-2013-episode-589-b</p>
		<pre>PS C:\Users\HP\OneDrive\Documents\subhagata> python scan.py Loaded trained model and initialized TF-IDF vectorizer. Scanning QR Code... Press 'q' to quit. Scanned URL: https://tinyurl.com/2d3b52sy Final URL: INCORRECT_URL False Incorrect or Invalid URL detected! Please check the QR code.</pre> <p>Scanned URL: https://tinyurl.com/2d3b52sy Final URL: https://www.placeography.org/index.php/Wigington_Pavilion%2C_Harriet_Island%2C_Saint_Paul%2C_Minnesota</p>

This real-time scanning system is particularly useful in environments where users frequently interact with printed QR codes—such as public transport, educational institutions, or events—without a reliable way to verify the safety of embedded links. The integration of machine learning in this context enables proactive phishing mitigation by providing instant feedback before the user engages with a potentially harmful destination.

Report Analysis: Overall Model Performance shows an accuracy of 89% (overall correct predictions), a macro average of 0.89 (average precision, recall, and F1-score across all classes) and a weighted average of 0.89 (considers class imbalance in calculation). The trusted domain approach ensures safe URLs are not misclassified, while regex-based checks and machine learning enhance general URL validation.

6.2 Achievements:

The Malicious QR Code Detection system has achieved several key milestones in the development and deployment of secure, intelligent QR scanning technology:

1. A machine learning model was developed and trained to classify QR-embedded URLs as malicious or non-malicious, demonstrating high classification accuracy.
2. A hybrid detection framework was designed, integrating URL feature extraction with supervised learning algorithms to enable real-time analysis of QR code contents.
3. An advanced feature extraction pipeline was implemented, utilizing over 25 handcrafted indicators such as URL length, suspicious character patterns, redirection behaviour, Shannon entropy, and domain reputation metrics.
4. Heuristic-based filters were incorporated to enhance phishing detection using keyword matching for terms like “login”, “secure”, “bank”, and “verify”—common markers in deceptive URLs.
5. Real-world testing was conducted to evaluate the system's effectiveness in dynamic online environments. Results demonstrated strong classification performance with low false positive rates, validating the system's practical reliability and robustness.

6.3 Applications:

The Malicious QR Code Detection module is applicable across a wide array of industries and use cases, including:

1. Financial Services: Detecting and blocking QR codes used for fraudulent transactions or phishing.
2. Retail & E-commerce: Verifying the legitimacy of QR-based promotions, offers, and payment portals.
3. Hospitality and Public Services: Identifying fake QR codes used in menus, check-in systems, and service kiosks.
4. Digital Marketing: Ensuring campaign integrity by validating QR links used in advertisements and promotions.
5. General Consumers: Empowering users to verify QR codes encountered in public places, printed materials, or online platforms.
6. Cybersecurity Agencies: Assisting in the identification and tracking of QR-based attack vectors, including links to the dark web or illegal marketplaces.

VII. CONCLUSION

The use of the XGBoost algorithm provides a reliable and scalable solution for identifying harmful patterns commonly associated with phishing and malware. The system was successfully integrated into a real-time scanning interface and can be deployed through a browser extension, allowing users to assess QR codes instantly and securely.

The accuracy of QR code detection can be limited by several real-world factors including differences in QR code structure, poor lighting, and image distortion during scanning. Real-time performance is also dependent on available computing power, which may pose a challenge when deploying the system at scale.

When it comes to identifying malicious QR codes, things get trickier. Advanced phishing URLs that closely mimic legitimate domains can be hard to classify accurately. Attackers also use techniques like domain generation algorithms (DGA) and URL shorteners to hide the true intent of a link, making it even harder to catch malicious behavior. Real-time URL checks that rely on external security databases can sometimes introduce delays, impacting the speed of threat detection.

Future improvements include optimizing detection models for higher accuracy, integrating deep learning techniques for more robust analysis, and expanding the system to support a wider range of QR code formats. A mobile application can be developed for on-the-go scanning, and blockchain technology could be explored for added security in QR-based transactions. Enhancing the threat intelligence module with real-time updates will further strengthen malicious QR detection.

VIII. REFERENCES

- [1] S. Rafsanjani, N. B. Kamaruddin, H. M. Rusli, and M. Dabbagh, "QsecR: Secure QR Code Scanner According to a Novel Malicious URL Detection Framework," *IEEE Access*, vol. 11, pp. 98765–98780, 2023, doi: 10.1109/ACCESS.2023.3301256.
- [2] K. Kalpana, V. Maheshwaram, and K. Umarani, "Secure QR Code Scanner to Detect Malicious URL using Machine Learning," *Proc. IEEE International Conference on Computing and Digital Systems*, 2023.
- [3] H. Almousa, S. Alsuhaibany, and A. Alabdulatif, "QR Shield: A Dual Machine Learning Approach Towards Securing QR Codes," *International Journal of Computing and Digital Systems*, vol. 12, no. 4, pp. 456–472, 2023.
- [4] Pawar, C. Fatnani, R. Sonavane, R. Waghmare, and S. Saoji, "Detection of QR Code-based Cyberattacks using a Lightweight Deep Learning Model," *Engineering, Technology & Applied Science Research*, vol. 13, no. 5, pp. 2381–2387, 2023.
- [5] M. A. Rahman and S. Hossain, "Enhancing Security in QR Code Technology Using AI: Exploration and Implementation," *International Journal of Information Security*, vol. 14, no. 2, pp. 89–102, 2024.