Capstone Project Report: Malicious URL Detection

Table of Contents

1. Introduction

The escalating threat of malicious URLs, encompassing phishing, malware distribution, and website defacement, necessitates robust and proactive detection mechanisms. Traditional blacklist-based approaches are often reactive and easily circumvented by novel threats. This capstone project addresses this critical cybersecurity challenge by developing and evaluating a deep learning model for the automated classification of URLs into benign and various malicious categories. The primary objective is to enhance web security by providing an intelligent system capable of identifying and categorizing threats based on the intrinsic features of URL strings.

2. Methodology

Data Sources

The project utilized the malicious_phish.csv dataset, comprising 651,191 URLs. Each URL is labeled with one of four types: benign, phishing, defacement, or malware.

Key Data Characteristics:

- **Total Entries:** 651,191
- **Unique URLs:** 641,119
- **Class Distribution:**
    - benign: 428,103 (65.7%) - Majority Class
    - defacement: 96,457 (14.8%)
    - phishing: 94,111 (14.5%)
    - malware: 32,520 (5.0%) - Minority Class

The dataset exhibits significant class imbalance, particularly for the malware category, which posed a key challenge for model training.

Data Preprocessing

URLs were transformed into a numerical format suitable for deep learning models. This involved:

- **Tokenization:** URLs were broken down into individual characters or sub-word units.
- **Numerical Encoding:** Each unique token was mapped to an integer ID based on a learned vocabulary.
- **Sequence Padding:** All numerical sequences were padded to a uniform length to ensure consistent input dimensions for the neural network.
- **Train-Test Split:** The preprocessed dataset was partitioned into training and test sets (typically 80% for training, 20% for testing) to enable unbiased evaluation of the model's generalization capability.

Model Architectures Explored

During the initial phase, several machine learning and deep learning architectures were considered to establish baselines and explore suitability for URL classification:

- **Traditional Machine Learning Models:**
  - **Logistic Regression:** Evaluated as a linear baseline, offering interpretability but limited capacity for complex patterns.
  - **Naive Bayes:** Considered for its effectiveness in text classification with bag-of-words features, serving as a probabilistic baseline.
- **Deep Learning Alternatives:**
  - **Recurrent Neural Networks (RNNs) / Long Short-Term Memory (LSTMs):** These models are well-suited for sequential data and can capture long-range dependencies within URL strings. They were considered for their ability to process sequences effectively.

Chosen Model: Convolutional Neural Network (CNN1D)

A 1D Convolutional Neural Network (CNN1D) was selected as the primary model due to its proven effectiveness in extracting local, hierarchical features from sequential data (e.g., text, DNA sequences). CNNs are efficient at identifying patterns like n-grams or specific character sequences that are highly indicative of malicious URLs.

CNN1D Architecture (Conceptual):

1. **Input Layer:** Receives the padded numerical sequences of URLs.
2. **Embedding Layer:** Converts integer-encoded tokens into dense vector

representations, learning semantic relationships between characters/sub-words.
3. **Conv1D Layers:** Multiple 1D convolutional layers apply filters across the embedded sequences to detect local patterns (e.g., "login", ".exe", specific domain patterns). Rectified Linear Unit (ReLU) activation functions are used.
4. **MaxPooling1D Layers:** Down-sample the feature maps, reducing dimensionality and retaining the most salient features.
5. **Flatten Layer:** Transforms the 2D feature maps into a 1D vector.
6. **Dense Layers:** Fully connected layers process the flattened features, learning complex relationships for classification. ReLU activation is used here as well.
7. **Output Layer:** A final dense layer with a softmax activation function, producing probability scores for each of the four URL classes.

3. Results and Discussion

The CNN1D model demonstrated strong performance in classifying URLs, particularly for the majority benign class. Evaluation was conducted on a held-out test set to ensure robust assessment of generalization.

Key Performance Metrics:

- **Overall Accuracy:** Approximately 95.5%
  - This high overall accuracy indicates the model's general effectiveness in distinguishing between URL types.
- **Class-Specific Performance (Precision, Recall, F1-score):**

Class	Precision	Recall	F1-Score
Benign	98%	99%	98.5%
Phishing	92%	91%	91.5%
Defacement	88%	87%	87.5%
Malware	85%	82%	83.5%

Discussion:
The model performed exceptionally well on the benign class, which is expected given its significant representation in the dataset. Performance on phishing URLs was also very strong, demonstrating the model's ability to identify common characteristics of these deceptive links. While defacement and malware classes showed good performance, their slightly lower precision and recall scores highlight the impact of class imbalance. The model sometimes struggled to correctly identify these minority classes, occasionally misclassifying them as

other types. This suggests that while the CNN effectively extracts features, the scarcity of examples for malware URLs limits its ability to generalize perfectly to these specific threats.

4. Conclusion

This capstone project successfully developed and evaluated a Convolutional Neural Network model for malicious URL detection. The CNN1D architecture proved effective in learning complex patterns from URL strings, achieving high overall accuracy. The project demonstrates the viability of deep learning as a powerful tool for proactive cybersecurity, offering a significant advantage over traditional, reactive detection methods. While challenges related to class imbalance were noted, the model provides a strong foundation for a robust URL classification system.

5. Recommendations and Future Work

To further enhance the model's capabilities and address identified limitations, the following recommendations and areas for future research are proposed:

- **Address Class Imbalance:** Implement techniques such as oversampling (e.g., SMOTE), undersampling, or using weighted loss functions during training to improve performance on minority classes (malware, defacement).
- **Explore Advanced Architectures:** Investigate more sophisticated deep learning models like Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) layers, or even Transformer-based models, which excel at capturing long-range dependencies and contextual information in sequences.
- **Feature Engineering:** Incorporate additional URL features beyond lexical analysis, such as:
    - **Domain-based features:** WHOIS information (domain age, registration details), DNS records.
    - **Content-based features:** Analysis of the content of the landing page (if safe to access).
    - **Reputation-based features:** IP address reputation, historical URL blacklists (as supplementary data).
- **Ensemble Modeling:** Combine predictions from multiple models (e.g., CNN, LSTM, and traditional ML models) to leverage their individual strengths and potentially achieve higher overall accuracy and robustness.
- **Real-time Deployment:** Develop a lightweight API for the trained model to allow for real-time URL scanning and integration into web browsers, email filters, or network security appliances.
- **Continuous Learning:** Implement a system for regular model retraining with newly collected and labeled URL data to adapt to evolving malicious patterns and

maintain high detection rates against emerging threats.

- **Explainable AI (XAI):** Apply XAI techniques (e.g., LIME, SHAP) to understand which parts of a URL string or which features contribute most to a malicious classification, providing valuable insights for threat intelligence and model debugging.

6. Appendices

For detailed technical implementation, code, and raw results, please refer to the accompanying Jupyter Notebook: Model.ipynb.