DETECTION OF DEFACEMENT URLS USING MACHINE LEARNING

A MINI PROJECT REPORT

submitted by

Anandha Krishnan S **LIDK21CS074**

Aiswarya M **LIDK21CS069**

Arun Babu **LIDK21CS079**

to

the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the degree

of

Bachelor of Technology

in

Computer Science & Engineering

Department of Computer Science & Engineering

Government Engineering College Idukki

685603

May 2024

DECLARATION

We undersigned hereby declare that the mini project report **Detection of defacement URLs using Machine Learning** submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala, is a bonafide work done by us under supervision of Prof. Philumon Joseph. This submission represents our ideas in our own words and where ideas or words of others have been included, We have adequately and accurately cited and referenced the original sources. We also declare that We have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Anandha Krishnan S

Aiswarya M

Arun Babu

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Government Engineering College Idukki

Idukki

685603

CERTIFICATE

This is to certify that the report entitled **Detection of defacement URLs using Machine Learning** submitted by **Anandha Krishnan S, Aiswarya M, Arun Babu** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering is a bonafide record of the mini project work carried out by him/her under my/our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Prof. Philumon Joseph
(Project Guide and Coordinator)
Assistant professor
Dept. of CSE
GEC Idukki

Prof. Anish Abraham
(Project Coordinator)
Assosiate professor
Dept. of CSE
GEC Idukki

Dr. Unnikrishnan K
Associate Professor
Head of the Department
Dept. of CSE
GEC Idukki

ACKNOWLEDGMENT

We wish to record my indebtedness and thankfulness to all who helped me prepare this Project Report titled **Detection of defacement URLs using Machine Learning** and present it in a satisfactory way.

We are especially thankful to our guide and coordinator Prof. Philumon Joseph Assistant professor in the Department of Computer Science & Engineering for giving me valuable suggestions and critical inputs in the preparation of this report. We are thankful to my project coordinator Prof. Anish Abraham And Dr. Liji P I Associate professor in the Department of Computer Science & Engineering supporting us till successfully completing the project. We are also thankful to Dr. Unnikrishnan K, Head of the Department of Computer Science & Engineering for encouragement.

My friends in my class have always been helpful and I am grateful to them for patiently helping me with this project.

Anandha Krishnan S

Aiswarya M

Arun Babu

B. Tech. (Computer Science & Engineering)

Department of Computer Science & Engineering

Government Engineering College Idukki

ABSTRACT

In today's digital landscape, the internet plays a crucial role in communication, commerce, and information dissemination, but it also harbors malicious activities like website defacement. Detecting and countering these attacks is vital for maintaining the integrity of online information sources. To address this challenge, we propose a novel machine learning approach for automatically identifying defaced URLs, aiming to surpass the limitations of manual or rule-based detection methods. By leveraging ML algorithms and contextual information, our model aims to distinguish between legitimate and defaced URLs with high precision and recall. This initiative contributes to advancing cybersecurity by developing proactive defenses against website defacement, ultimately safeguarding the integrity of online information resources. Through empirical evaluation and analysis, we aim to demonstrate the effectiveness of our approach, paving the way for future innovations in cyber threat detection and mitigation.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The DETECTION OF DEFACEMENT URLS project, developed as a Mini Project under the B. Tech. program of APJ Abdul Kalam Technological University, Our mini-project focuses on leveraging machine learning techniques to automatically detect defaced URLs. By harnessing the power of ML algorithms, we aim to develop an efficient and accurate solution to identify and mitigate defacement attacks, thereby bolstering cybersecurity measures and preserving the credibility of online information sources.

1.1 GENERAL BACKGROUND

Website defacement, where attackers alter the appearance or functionality of websites, poses a significant threat to online entities. Traditional detection methods are often inefficient. Machine learning (ML) offers a solution by automating the process. By analyzing features of URLs, ML models can distinguish between legitimate and defaced URLs, enhancing cybersecurity measures and protecting online credibility.

1.2 OBJECTIVES OF THE PROJECT

This project aims to develop a machine learning-driven system for detection of defacement URLs. By assembling a diverse dataset of both legitimate and defaced URLs and employing feature engineering techniques, we will train and optimize machine learning models to accurately classify URLs in real-time. Through rigorous evaluation and validation, we will ensure the robustness and generalization of the models. The ultimate goal is to deploy a scalable solution that seamlessly

integrates with existing cybersecurity infrastructure, providing continuous monitoring and maintenance to adapt to evolving threats and enhance overall cybersecurity measures.

1.3 SCOPE OF THE PROJECT

This project will focus on the development and implementation of a machine learning-based system specifically tailored for the detection of defacement URLs. The scope encompasses data collection, feature engineering, model selection and training, as well as deployment and integration into existing cybersecurity frameworks. Evaluation and validation will be conducted to ensure the effectiveness and reliability of the solution. However, the project will not delve into other aspects of cybersecurity such as intrusion detection or malware analysis. Additionally, while the system will be designed to handle real-time classification of URLs, extensive scalability testing beyond the scope of typical operational loads will not be performed.

1.4 SCHEME OF THE PROJECT

The project will begin with the development of a user-friendly web interface enabling users to input URLs for defacement detection. This interface will be backed by a robust backend system consisting of a URL processing module to sanitize inputs and a detection system employing machine learning algorithms for classification. The machine learning pipeline will involve data collection, feature engineering, model training, and evaluation to select the best-performing model. Upon completion, the detection system will be deployed on a web server, undergo integration and user acceptance testing, and be continuously monitored for performance and usage. Regular updates will ensure the system remains effective against evolving threats. Through this approach, the project aims to deliver a scalable and reliable solution for detecting defacement URLs, contributing to enhanced cybersecurity measures.

Chapter 2

MOTIVATION

In today's digital world, website defacement remains a persistent threat to online security and credibility. Manual detection methods, while reliable, are often time-consuming and impractical for handling the vast volume of online content. Our project is motivated by the need to enhance cybersecurity measures by providing a semi-automated solution for detecting defacement URLs.

By incorporating machine learning techniques into the detection process, our system aims to augment the capabilities of human analysts rather than replacing them entirely. By assisting analysts in identifying potential instances of defacement, our project seeks to streamline the detection process, enabling more efficient use of resources and faster response times to emerging threats.

Real-time data analysis is crucial in this endeavor, as it allows analysts to monitor and assess incoming URLs promptly. By leveraging machine learning algorithms to sift through large datasets and identify patterns indicative of defacement, our system empowers analysts to make informed decisions and take timely action to mitigate risks.

Chapter 3

LITERATURE SURVEY

This literature survey provides an overview of selected works relevant to the Detection of defacement URL project.

1. **The Ghost in the Browser**

N. Provos et al [9] introduced an overview of the current state of malware on the web. The evaluation is based on Internet-wide measurements conducted over a period of twelve months starting March 2006. The results reveal several attack strategies for turning web pages into malware infection vectors. They identified four different aspects of content control responsible for enabling browser exploitation: advertising, third-party widgets, user-contributed content, and web server security. Through analysis and examples, the paper shows how each of these categories can be used to exploit web browsers.

2. **All your iFrames Points to Us**

The fact that malicious URLs that initiate drive-by downloads are spread far and wide raises concerns regarding the safety of browsing the Web. However, to date, little is known about the specifics of this increasingly common malware distribution technique. P. Mavrommatis et al [10] attempt to fill in the gaps about this growing phenomenon by providing a comprehensive look at the problem from several perspectives. This study uses a large-scale data collection infrastructure that continuously detects and monitors the behavior of websites that perpetrate drive-by downloads. In-depth analysis of over 66 million URLs (spanning a 10-month period) reveals the scope of the problem.

3. **PhoneyC: A Virtual Client Honeypot**

J Nazario et al [11] published this paper in 2009. This paper discusses a virtual honeypot, namely PhoneyC, which is implemented to study the nature of malicious attacks by making itself vulnerable to attack. These systems are instrumented to discover what happened and how PhoneyC mimics the behavior of a user-driven network client application such as a web browser and can be exploited by an attacker's content.

4. **The Nocebo effect on the web: An analysis of fake antivirus distribution**

The paper proposed by M. A. Rajab et al [12] in the year 2010. The paper embeds the idea of emerging malware. Fake AV attacks attempt to convince users that their computer systems are infected and offer a free download to scan for malware. Fake AVs pretend to scan computers and claim to find infected files—files which may not even exist or be compatible with the computer's OS. Users are forced to register the Fake Antivirus program for a fee to make the fake warnings disappear.

5. **EVILSEED: A Guided Approach For Finding Malicious Web page**

EVILSEED focuses its searches "near" known malicious pages. Invernizzi and Benvenuti [15] focused their searches "near" known malicious pages. EVILSEED implements different techniques to extract from a page features that characterize its malicious nature. Pages with similar values for such features are also likely to be malicious. Then, by using the features extracted from an evil seed and by leveraging existing search engines, EVILSEED guides its search to the neighborhood around known malicious pages.

Chapter 4

SYSTEM DESIGN

The method of translating specifications into a representation of software is known as system design. A specification is an engineering image of something that would be constructed. Design provides us with software representations that can be evaluated for consistency. If product production as a whole ”blended the perspectives of advertising, architecture, and production in to the single entity,” single approaches to the products development,” then the design is most act of taking up the advertising information and creates the design for the product that to be manufactured. Most commonly used approaches for computer system architecture are object oriented analyzes and design methods. In objective oriented research or architecture, the UML has become the de facto standard. It’s commonly used for modelling computing systems, and it’s becoming more popular for non-software systems and organizations. As a result, systems architecture is the practice of defining and implementing systems to meet the user’s specific specifications.

Chapter 5

ARCHITECTURE DIAGRAM

A system architecture, also known as systems design, is the mathematical models that says describes the systems configuration, behavior, and some aspects. The systematic meaning and represents of a system arranged in the manner that needs for the facilitates that thinking about the systematic mechanisms is otherwise called as the architecture description. Device elements, publicly observable properties of certain components, and interactions between them can all be used in the system architecture. It will offer a blueprint for obtaining goods and developing processes that can work together to execute the overall structure.

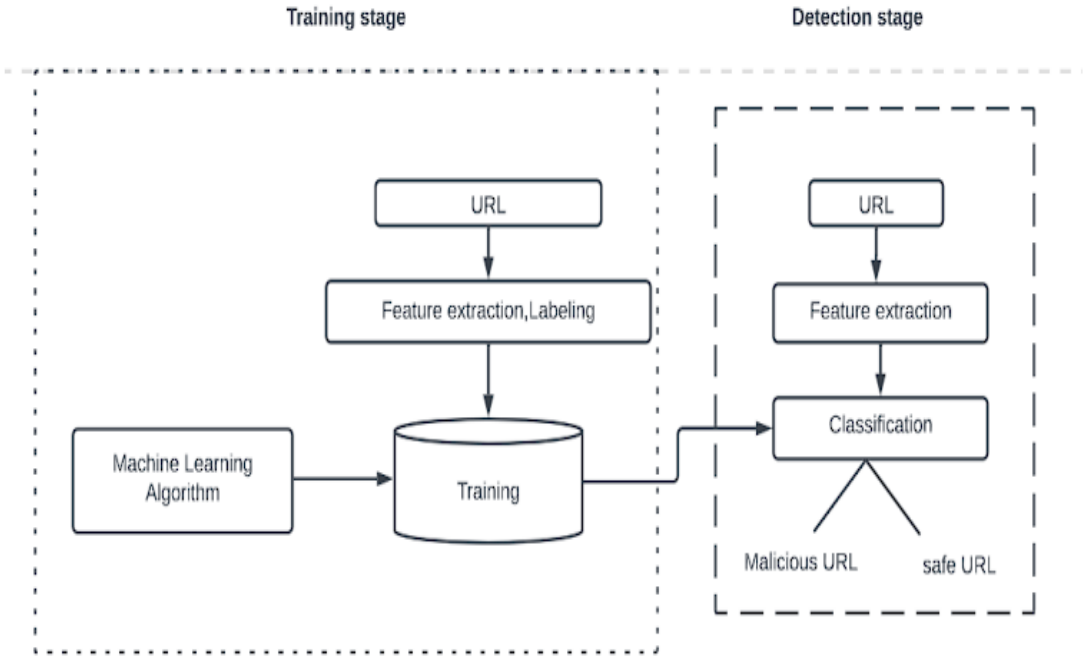


Figure 5.1: Architecture Diagram

Chapter 6

IMPLEMENTATION

The implementation of a system to detect defacement URLs using machine learning begins with the collection and preprocessing of a diverse dataset containing both legitimate and defaced URLs, followed by the extraction of relevant features such as domain names and query parameters. Feature engineering techniques are then applied to transform the raw URL data into informative features suitable for training machine learning models. Next, appropriate algorithms, such as decision trees or neural networks, are selected for classification, and the models are trained and evaluated using standard metrics like accuracy and precision. Once trained, the models are deployed into a production environment, where they process incoming URLs in real-time. Monitoring and maintenance mechanisms are implemented to track system performance, update models with new data, and ensure continuous adaptation to evolving threats. Through this implementation process, the system aims to provide an effective and scalable solution for detecting defacement URLs, thereby enhancing cybersecurity measures and safeguarding online information integrity.

6.1 MODULES:

1. **DATA PRE-PROCESSING**

   We load the metadata into this pre-processing step, then apply the metadata to the data and replace the transformed data with metadata. The data would then be carried on, with the unnecessary data in the list being removed and

the data being divided into train and test data. To break the data into train and test, we'll need to import train test split from scikit-learn. This will assist the pre-processed data in splitting the data into train and test based into the weights defined on the platform code. The test and train are divided by 0.2 and 0.8, or 20 and 80 percent, respectively.

2. **FEATURE EXTRACTION**

   Feature extraction is a dimensionality reduction method that reduces a large collection of raw data into smaller categories for processing. The vast number of variables in these large data sets necessitates a lot of computational power to process. Feature extraction refers to methods for selecting and/or combining variables into features into order reduces the quantity of the data that needs to be processed while still correctly and fully describing the existing environment.

3. **MODEL CREATION**

   We create data into two models:

   • Training model

   • Testing model

   The test and train are divided by 0.2 and 0.8, which equals 20% and 80%, respectively. For the training portion, we use machine learning algorithm to assess the model's accuracy, svm and logistic regression algorithms are used.

4. **PREDICTION**

   This module is built on the user interface. Bootstrap is used to build a web page. Enter the URL for the web page. We now obtain data from the customer

in order to compare the dataset values. Finally, it can determine whether the user is malicious or not.

6.2 IMPLEMENTATION DETAILS

1. HARDWARE REQUIREMENTS:

   Physical computing tools, also known as hardware, are the most basic set of specifications specified by the operating system and the software application. In these cases of operating systems, the hardware specification list is the often followed by the hardware configuration process. The below are the minimum hardware requirements:

   - PROCESSOR: INTEL / RYZEN.
   - RAM: 8 GB.
   - PROCESSOR: 2.4 GHZ.
   - MAIN MEMORY: 8GB RAM
   - PROCESSING SPEED: 600 MHZ
   - HARD DISK DRIVE: 1TB
   - KEYBOARD:104 KEYS

2. SOFTWARE REQUIREMENTS:

   Computer specifications are concerned with specifying the tools and prerequisites that must be implemented on a device in order for an application to work. These prerequisites must be installed before the app can be installed.

   - FRONT END: PYTHON
   - DATASET: CSV
   - IDE: ANACONDA / VS CODE
   - OPERATING SYSTEM: WINDOWS 10.

Chapter 7

RESULTS AND CONCLUSIONS

The work on show is still in its early stages. The aim of this paper is to provide a short overview of our approach. The extraction of lexical features may be used to detect malicious URLs, according to one theory. We used the Classifying approach based on the TF - IDF word association to complete the basic investigation. The features extracted from URL bigrams can be supported, and term frequency and inverse term frequency can provide the simplest classification setting. The main task, however, is to identify using the proposed features, and we have completed the preprocessing stage. The work presented here is an early effort in malicious URL detection; in a future work, we will cover the post-processing of the Feature set and include the classifying coefficients that are used as separating parameters.

```
# Computing the accuracy, f1_score, recall, precision of the model performance
acc_train_forest = metrics.accuracy_score(y_train, y_train_forest)
acc_test_forest = metrics.accuracy_score(y_test, y_test_forest)
print("Random Forest: Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random Forest: Accuracy on test Data: {:.3f}".format(acc_test_forest))
print()

f1_score_train_forest = metrics.f1_score(y_train, y_train_forest, pos_label=pos_label)
f1_score_test_forest = metrics.f1_score(y_test, y_test_forest, pos_label=pos_label)
print("Random Forest: f1_score on training Data: {:.3f}".format(f1_score_train_forest))
print("Random Forest: f1_score on test Data: {:.3f}".format(f1_score_test_forest))
print()

recall_score_train_forest = metrics.recall_score(y_train, y_train_forest, pos_label=pos_label)
recall_score_test_forest = metrics.recall_score(y_test, y_test_forest, pos_label=pos_label)
print("Random Forest: Recall on training Data: {:.3f}".format(recall_score_train_forest))
print("Random Forest: Recall on test Data: {:.3f}".format(recall_score_test_forest))
print()

precision_score_train_forest = metrics.precision_score(y_train, y_train_forest, pos_label=pos_label)
precision_score_test_forest = metrics.precision_score(y_test, y_test_forest, pos_label=pos_label)
print("Random Forest: precision on training Data: {:.3f}".format(precision_score_train_forest))
print("Random Forest: precision on test Data: {:.3f}".format(precision_score_test_forest))

# Computing the classification report of the model
print(metrics.classification_report(y_test, y_test_forest, target_names=['benign', 'defacement']))

# Visualizing an individual tree from the Random Forest
individual_tree = forest.estimators_[0]  # Selecting the first tree
plt.figure(figsize=(20, 10))
tree.plot_tree(individual_tree, filled=True, feature_names=X.columns, class_names=['benign', 'defacement'])
plt.show()
```

Figure 7.1: Random Forest

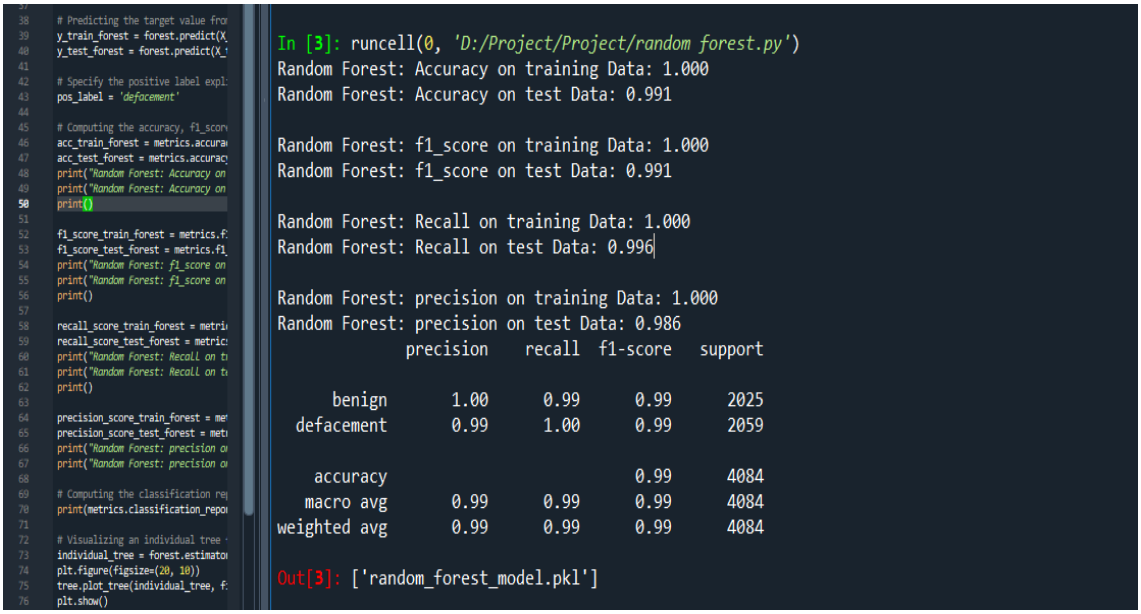


Figure 7.2: Finding Accuracy

The Random Forest model demonstrates exceptional performance across various evaluation metrics on both the training and test datasets, showcasing its robust predictive power and generalization capabilities. With an accuracy of 100% on the training data and 99.1% on the test data, the model exhibits outstanding classification accuracy. Moreover, achieving an F1 score of 100% on the training data and 99.1% on the test data, the model maintains a high balance between precision and recall.

Although the model displays perfect recall on the training set (100%), there's a slight decrease to 99.6% on the test set, still indicating excellent performance. Furthermore, the model maintains exceptional precision levels of 100% on the training data and 98.6% on the test data, demonstrating its ability to minimize false positives effectively.

The classification report provides deeper insights into the model's performance per class, revealing consistently high precision, recall, and F1-score for both benign and defacement classes. Overall, the Random Forest model emerges as an exceedingly effective tool for the classification task, delivering reliable predictions with an exceptional balance between precision and recall across different datasets.

```
# Example usage:
url = input("Enter the url: ")
features = test_it(url)
#print("Features")
#print(features)


# test
xnew = [features]
ynew = forest.predict(xnew)
print (ynew)
```

```
Enter the url: Https://www.google.com
['benign']
```

Figure 7.3: Detecting malicious url

The URL undergoes a preprocessing stage, wherein extraneous components are removed, followed by tokenization using pretrained data. This tokenization step enables a comparative analysis, facilitating the identification of malicious URLs. The entire codebase has undergone meticulous debugging and execution within VS Code environment to ensure reliability. All relevant files are stored with the .py extension. Once deployed on a website, users gain access to a tool for evaluating the security of provided links. Additionally, the codebase seamlessly integrates with Visual Studio Code (VS Code), enabling the transformation of its output into a fully functional website.

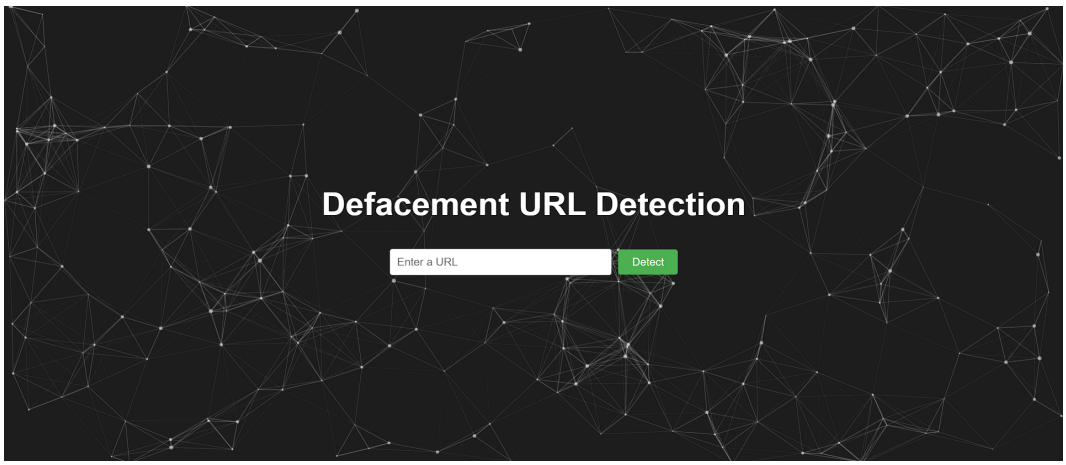


Figure 7.4: Webpage for user to check

7.1 CONCLUSION

In the realm of machine learning (ML), numerous cybersecurity applications rely on identifying malicious URLs, representing a promising avenue for advancement. Our study meticulously explored Malicious Detection through the lens of ML methodologies. We provided a systematic breakdown of Malicious detection from an ML perspective, followed by in-depth analysis. Our investigation centered on advancements in identifying malicious URLs, particularly focusing on innovative feature representations and the development of new learning algorithms for effectively categorizing malicious URL instances.

Through our comprehensive review of existing literature, we dissected the landscape of malignant URL detection, while also addressing the prerequisites and hurdles associated with crafting ML-driven solutions for cybersecurity applications. Furthermore, we underscored both peripheral issues and critical challenges that warrant further scrutiny. Despite considerable strides in research, the automated detection of spam URLs using ML remains a formidable challenge. Future endeavors will likely prioritize refining feature extraction and representation learning techniques, alongside the development of more efficient ML algorithms for predictive modeling, with a keen focus on managing concept drifts and domain adaptation.

Ultimately, devising astute methodologies for safeguarding sensitive information and integrating user feedback within a closed-loop system stands as an imperative task. Our study also underscored pertinent concerns within the application landscape and emphasized unresolved issues necessitating continued investigation, particularly in the context of ML-driven cybersecurity solutions.

Chapter 8

FUTURE DIRECTIONS

There are several potential areas of improvement and future work for this project, including:

8.1 INTRODUCTION

- Incorporating more advanced feature engineering techniques to capture additional information about URLs.
- Exploring other machine learning algorithms and comparing their performance with Random Forest.
- Integrating real-time data sources to continuously update and improve the model's performance.
- Enhancing the user interface and interactivity of the deployed system to provide a seamless user experience.
- Conducting more extensive testing on a larger and more diverse dataset to validate the model's robustness.

By undertaking this project, we contribute to the field of cybersecurity by developing an effective solution to detect phishing URLs. The Random Forest algorithm proves to be a powerful tool for this task, and the system has the potential to protect individuals and organizations from phishing attacks.

BIBLIOGRAPHY

[1] Luca Invernizzi and Stefano Benvenuti. Evilseed: A guided approach for finding malicious webpages. In *IEEE Security Symposium*, 2012.

[2] J. Nazario. Phoneyc: A virtual client honeypot. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[3] N. Provos and P. et al. Mavrommatis. All your iframes point to us. *USENIX*, 2008.

[4] N. Provos and D. et al. McNamee. The ghost in the browser: Analysis of web-based malware. *USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[5] M. A. Rajab and L. Ballard. The nocebo effect on the web: An analysis of fake anti-virus distribution. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2010.

REFERENCES

[1] Su, K.-W., et al. Suspicious URL filtering based on logistic regression with multiview analysis. In: 8th Asia Joint Conference on Information Security (Asia JCIS). IEEE (2013)

[2] Le, A., Markopoulou, A., Faloutsos, M. PhishDef: URL names say it all. In: Proceedings IEEE, INFOCOM. IEEE (2011)

[3] Breiman, L. Random forests. Mach. Learn. 45(1), 5–32 (2001)

[4] Thomas, K., et al. Design and evaluation of a real-time URL spam filtering service. In: Proceeding of the IEEE Symposium on Security and Privacy (SP) (2011)

[5] Ma, J., et al. Identifying suspicious URLs: an application of large-scale online learning. In: Proceedings of the 26th Annual International Conference on Machine Learning. ACM (2009)

[6] Nunan, A.E., et al. Automatic classification of cross-site scripting in web pages using document-based and URL-based features. In: IEEE Symposium on Computers and Communications (ISCC) (2012)

[7] Choi, H., Zhu, B.B., Lee, H. Detecting malicious web links and identifying their attack types. In: Proceedings WebApps (2011)

[8] Huang, D., Kai, X., Pei, J. Malicious URL detection by dynamically mining patterns without pre-defined elements. World Wide Web 17(6), 1375–1394 (2014)

[9] Provos, N. and McNamee, D. et al. The Ghost in the Browser: Analysis of Web-based Malware. USENIX Workshop on Hot Topics in Understanding Botnet (2007)

[10] Provos, N. and Mavrommatis, P. et al. All Your iFrames Point to Us. USENIX (2008)

[11] Nazario, J. PhoneyC: A Virtual Client Honeypot. USENIX Workshop on Large-Scale Exploits and Emergent Threats (2009)

[12] Rajab, M. A. and Ballard, L. The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution. USENIX Workshop on Large-Scale Exploits and Emergent Threats (2010)

[13] Cova, M. and Kruegel, C. and Vigna, G. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. International World Wide Web Conference (WWW) (2010)

[14] Lu, Long and Yegneswaran, Vinod and Porras, Phillip and Lee, Wenke. BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware Infections (2010)

[15] Invernizzi, Luca and Benvenuti, Stefano. Evilseed: A Guided Approach for Finding Malicious Webpages. IEEE Security Symposium (2012)

APPENDIX I: SCREENSHOTS

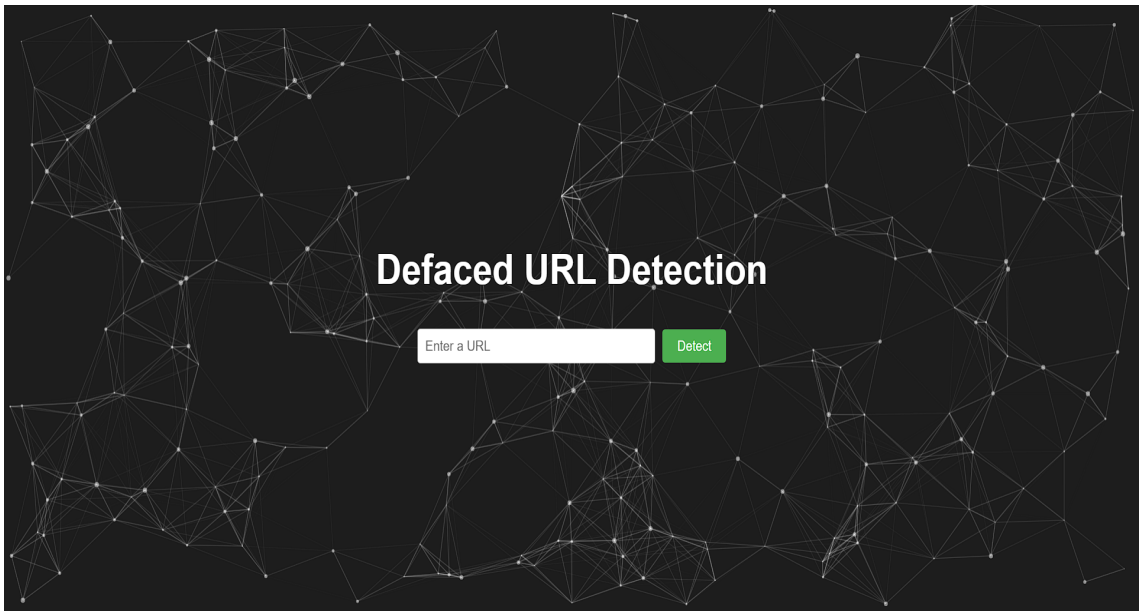


Figure 8.1: Landing Page

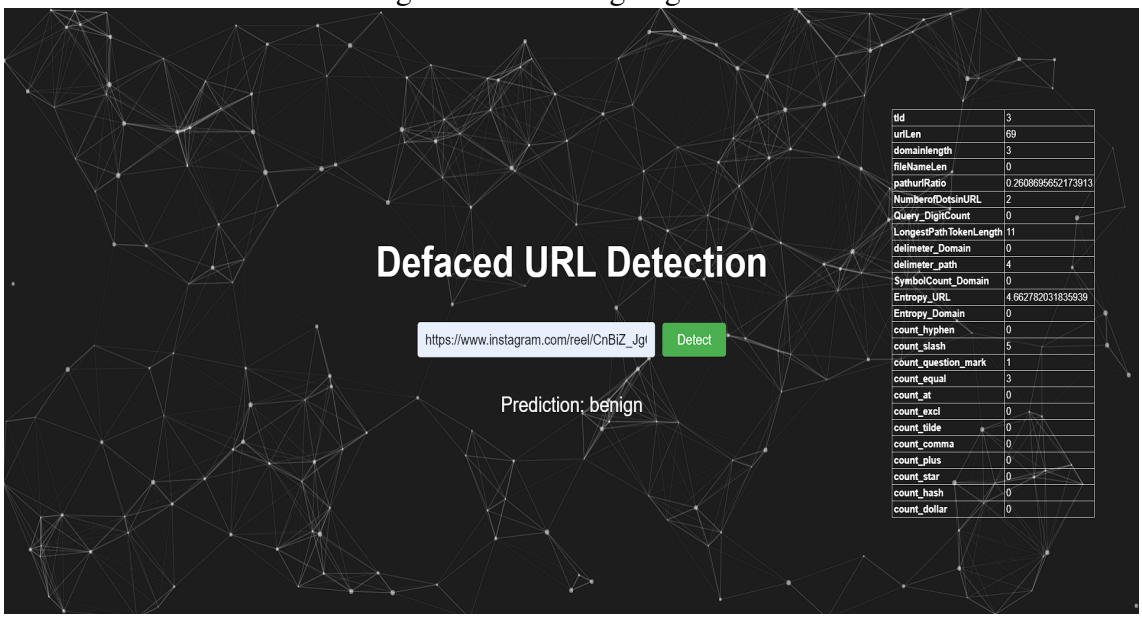


Figure 8.2: Prediction webpage

APPENDIX II: SAMPLE CODE

1. App.py

Here's the code with all the commented lines removed:

```
python
from flask import Flask, render_template, request, jsonify
from urllib.parse import urlparse, parse_qs
from sklearn.model_selection import train_test_split
import os
import pandas as pd
import string
import math
import pickle
from sklearn.ensemble import RandomForestClassifier

app = Flask(__name__, template_folder='templates')

def extract_tld(url):
    parsed = urlparse(url)
    domain = parsed.netloc.split('.')[-1]
    return len(domain)

def url_length(url):
    return len(url)

def domain_length(url):
    parsed = urlparse(url)
    domain = parsed.netloc.split('.')[0]
    return len(domain)
```

```
def filename_length(url):
    parsed = urlparse(url)
    return len(os.path.basename(parsed.path))

def path_url_ratio(url):
    parsed = urlparse(url)
    path_length = len(parsed.path)
    url_length = len(url)
    return path_length / url_length if url_length > 0 else 0

def num_dots_url(url):
    count1 = url.count('.')
    return count1

def count_digits_query_string(url):
    parsed_url = urlparse(url)
    query_string = parsed_url.query
    parsed_query = parse_qs(query_string)
    digit_count = sum(1 for value in parsed_query.values() for item in valu
    return digit_count

def longest_path_token_length(url):
    parsed = urlparse(url)
    path_tokens = parsed.path.split('/')
    return max(len(token) for token in path_tokens)

def count_delimiters_domain(url):
    parsed = urlparse(url)
    domain = parsed.netloc.split('.')[0]
```

```
    return sum(1 for char in domain if char in string.punctuation)

def count_delimiters_path(url):
    parsed = urlparse(url)
    return sum(1 for char in parsed.path if char in string.punctuation)

def symbol_count_domain(url):
    parsed = urlparse(url)
    domain = parsed.netloc.split('.')[0]
    return sum(1 for char in domain if char in string.punctuation)

def entropy(s):
    probabilities = [float(s.count(c)) / len(s) for c in set(s)]
    entropy = - sum(p * math.log(p) / math.log(2.0) for p in probabilities)
    return entropy

def url_entropy(url):
    if url.startswith("http://"):
        url = url[len("http://"):]
    elif url.startswith("https://"):
        url = url[len("https://"):]
    if url.startswith("www."):
        url = url[len("www."):]
    url = ''.join(e for e in url if e.isalnum())
    return entropy(url)

def entropy_domain(url):
    parsed = urlparse(url)
    domain = parsed.netloc.split('.')[0]
    length = len(domain)
```

```
    if length <= 1:
        return 0
    else:
        entropy = 0
        for char in string.ascii_lowercase:
            p_i = domain.count(char) / length
            if p_i > 0:
                entropy -= p_i * math.log2(p_i)
        return entropy

def count_hyphen(url):
    return url.count('-')

def count_slash(url):
    return url.count('/')

def count_question_mark(url):
    return url.count('?')

def count_equal(url):
    return url.count('=')

def count_at(url):
    return url.count('@')

def count_exclamation(url):
    return url.count('!')

def count_tilde(url):
    return url.count('~')
```

```
def count_comma(url):
    return url.count(',')

def count_plus(url):
    return url.count('+')

def count_star(url):
    return url.count('*')

def count_hash(url):
    return url.count('#')

def count_dollar(url):
    return url.count('$')

def test_it(url):
    features = []
    features.append(extract_tld(url))
    features.append(url_length(url))
    features.append(domain_length(url))
    features.append(filename_length(url))
    features.append(path_url_ratio(url))
    features.append(num_dots_url(url))
    features.append(count_digits_query_string(url))
    features.append(longest_path_token_length(url))
    features.append(count_delimiters_domain(url))
    features.append(count_delimiters_path(url))
    features.append(symbol_count_domain(url))
    features.append(url_entropy(url))
```

```
    features.append(entropy_domain(url))
    features.append(count_hyphen(url))
    features.append(count_slash(url))
    features.append(count_question_mark(url))
    features.append(count_equal(url))
    features.append(count_at(url))
    features.append(count_exclamation(url))
    features.append(count_tilde(url))
    features.append(count_comma(url))
    features.append(count_plus(url))
    features.append(count_star(url))
    features.append(count_hash(url))
    features.append(count_dollar(url))
    return features

@app.route('/')
def home():
    return render_template('index.html')

data = pd.read_csv('enhanced_feature_set.csv')

@app.route('/analyze', methods=['POST'])
def predict():
    model = RandomForestClassifier()
    X = data.drop(["class"], axis=1)
    y = data["class"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

    model.fit(X_train, y_train)
    if 'url' in request.form:
```

```
            url = request.form['url']
            print(f"Received URL: {url}")
            features = test_it(url)
            xnew = [features]
            ynew = model.predict(xnew)
            return jsonify({'features': features, 'prediction': ynew[0]})
        else:
            return jsonify({'error': 'URL key not found in form data'})

if __name__ == '__main__':
    app.run(debug=True)
```

APPENDIX III: REVIEW SLIDES

CSD 334

Mini Project

Detection of defacement URL's

Team Members

Anandha Krishnan S – LIDK21CS074
Arun Babu – LIDK21CS079
Aiswarya M – LIDK21CS069

Project Guide

Prof. Philumon Joseph
Assistant Professor
Dept. Computer Science
Govt.Engineering College Idukki

2

Introduction

Protecting websites from unauthorized modifications is crucial for cybersecurity. Machine learning offers a promising solution for automated detection of defacement URLs.

- Defacement URLs are web addresses that direct users to websites or pages that have been tampered
- Importance of detecting defacement URLs
- Role of machine learning in addressing this challenge

Problem Statement

The challenge lies in accurately identifying unauthorized modifications to website content amidst a large volume of web pages and evolving attack techniques.

- Detecting defacement URLs
- Challenges associated with the detection process

Motivation and Application

Detecting defacement URLs is essential for preserving the integrity and security of websites.

- Potential consequences of leaving defacement URLs undetected
- Real-world applications in various industries (e.g, e-commerce, banking, government)

Literature Survey

- [1] Defacement url :URL Features-Based Phishing Detection System Using Machine Learning
- Authors: Jain A.K,Gupta B B
- Proposed a URL-based anti-phishing machine learning method.

They have taken 14 features of the URL to detect the website as a malicious or legitimate to test the efficiency of their method.

DISADVANTAGE:

- **Limited Evaluation**: The paper might have a narrow evaluation scope.
- **Scalability and Efficiency**: The proposed system may not be enough to handle large-scale phishing detection tasks in real-time, which could limit its practical utility.

Literature Survey

- [2] Detecting Defaced Web Pages with Visual Similarity Assessment Based on Earth Mover's Distance (EMD)
- Authors: Anthony Y Fu,Liu Wenyin,Xiaotic Deng
- A virtual honey pot that is used to study the nature of malicious URLs that cybercriminals use to steal information.

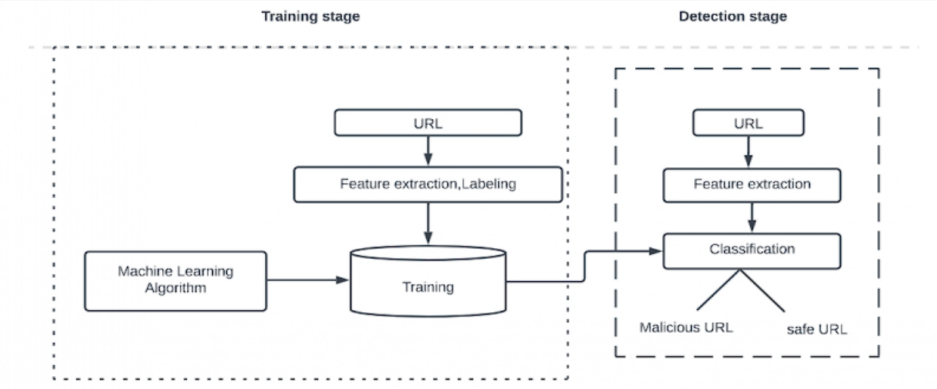
DISADVANTAGES:

- **Dependency on Visual Features**: The author may overlook non-visual indicators of phishing, such as URL structure, content semantics, or behavioral patterns.

Implementation Details

- Hardware: Standard server or cloud computing resources
- Software: Programming languages (e.g., Python), libraries/frameworks (e.g., scikit-learn, Flask), web scraping tools

Architecture diagram



9

Architecture diagram

- **Dataset** (features 25)

tld,urlLen,domainlength,fileNameLen,pathurlRatio,NumberofDotsinURL,Query_DigitCount,LongestPathTokenLength,delimeter_Domain,delimeter_path,SymbolCount_Domain,Entropy_URL,Entropy_Domain,count_hyphen,count_slash,count_question_mark,count_equal,count_at,count_excl,count_tilde,count_comma,count_plus,count_star,count_hash,
count_dollar.

10

Architecture diagram

- **Data preprocessing**

Preprocess the testing data similarly to the training data including cleaning and standardizing the URLs.

- **Feature Extraction**

Feature extraction condenses raw data into meaningful representations, aiding machine learning algorithms in making accurate predictions or classifications.

11

Architecture diagram

- **Model training**

The Random Forest model was trained on a dataset containing features and a target variable. It leverages an ensemble of decision trees, each trained on random subsets of the data and features, to make predictions with improved accuracy and generalization.

12

Architecture diagram

- **Testing random forest**

Apply the trained random forest model to the testing dataset to classify URLs as defacement or benign.

- **Model Evaluation**

Assess the random forest model's performance on the testing dataset using identical metrics as those applied to the decision tree model.

13

Packages

- **sklearn**

In python, sklearn is a machine learning package which include a lot of ML algorithms. Here, we are using some of its modules like train_test_split, Decision Tree Classifier, Logistic Regression and accuracy score.

- **Flask**

Python web framework for building web applications with simplicity and efficiency.

- **NumPy**

It is a numeric python module which provides fast maths functions for calculations.

- **Pandas**

Used to read and write different files. Data manipulation can be done easily with data frames.

- **Matplotlib**

Data visualization is a useful way to help with identify the patterns from given dataset.

14

Model Evaluation

We assessed three models—Logistic Regression, Decision Trees, and Random Forest—by executing the following steps for each :

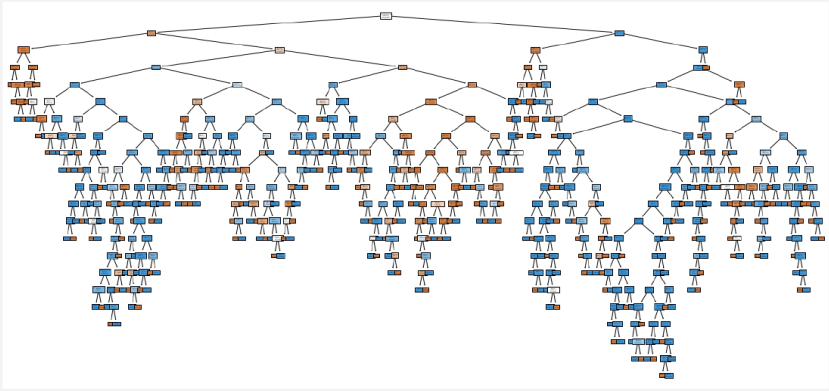
1. Data extraction from a CSV file.
2. Data preparation through Train-Test splitting.
3. Model execution.
4. Evaluation of model performance using metrics such as Accuracy, Confusion Matrix, Precision, Recall, and F1 Score.

15

Logistic Regression

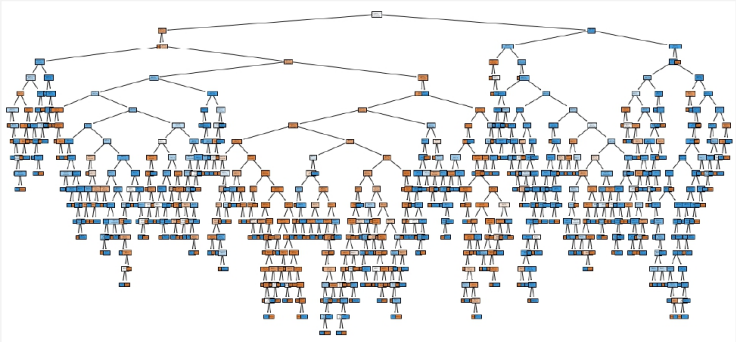
16

Decision Tree



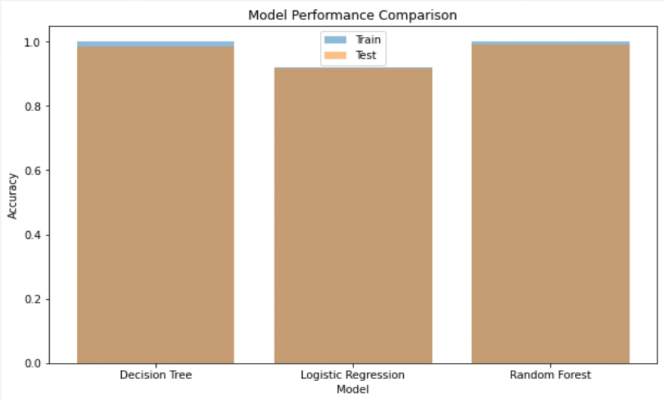
17

Random Forest



18

Comparison

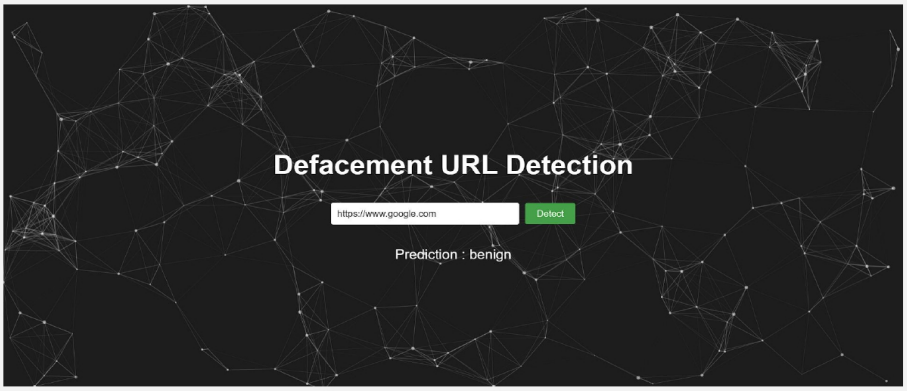


19

Screenshots

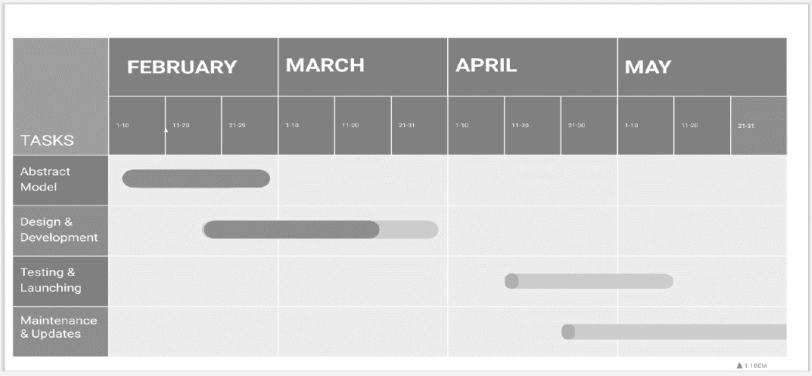
20

Screenshots



21

Time Line Chart



22

Conclusions

- Our proposed architecture presents a practical and effective solution for detecting defacement URLs using machine learning.
- Evaluation results demonstrate the efficacy of our system in accurately identifying unauthorized modifications.
- Moving forward, we suggest further exploration of advanced machine learning algorithms and real-time monitoring capabilities

References

- [1] Luca Invernizzi and Stefano Benvenuti. Evilseed: A guided approach for find- ing malicious webpages. In IEEE Security Symposium, 2012.
- [2] J. Nazario. Phoneyc: A virtual client honeypot. In USENIX Workshop on Large-Scale Exploits and Emergent Threats, 2009.
- [3] N. Provos and P. et al. Mavrommatis. All your iframes point to us. USENIX, 2008.
- [4] N. Provos and D. et al. McNamee. The ghost in the browser: Analysis of web- based malware. USENIX Workshop on Hot Topics in Understanding Botnet, 2007.
- [5] M. A. Rajab and L. Ballard. The nocebo effect on the web: An analysis of fake anti-virus distribution. In USENIX Workshop on Large-Scale Exploits and Emergent Threats, 2010. 16