Multi-Layered Phishing Detection Algorithm as a Browser Extension

Introduction

Phishing attacks are among the most pervasive cyber threats today. Attackers deceive users into revealing sensitive information such as passwords, credit card numbers, or personal identification by impersonating legitimate entities through fraudulent websites or emails. The sophistication of these attacks makes it challenging for users to differentiate between authentic and malicious sources, leading to severe consequences like financial loss, identity theft, and data breaches.

As phishing tactics evolve, there is an urgent need for robust, multi-layered detection systems that can protect users in real-time. This exercise challenges you to develop a comprehensive phishing detection algorithm implemented as a browser extension, leveraging multiple detection methods to safeguard users from phishing attempts.

Objectives

- **Design and implement** a multi-layered phishing detection algorithm using both static and dynamic analysis methods.
- **Develop** a Chromium-based browser extension that efficiently analyzes websites and provides immediate feedback to users about potential phishing threats.
- **Ensure** the solution is resource-efficient, with minimal impact on memory and CPU usage, and delivers fast detection times.
- **Evaluate** the performance of your algorithm based on accuracy, response time, and resource consumption.

Background on Phishing Detection Techniques

Understanding existing phishing detection techniques and their limitations is crucial for developing an effective solution. Below are common methods:

1. **URL-based Analysis**: Inspects the structure of the URL (e.g., domain name, subdomains, length). It's lightweight and quick but can be bypassed by attackers using legitimate-looking URLs or slightly altered domains.
2. **Content-based Analysis**: Examines the HTML, Document Object Model (DOM), and content of a webpage. It's more accurate but computationally expensive and can be circumvented by dynamically loading content after detection scripts have run.
3. **Blacklist-based Detection**: Compares URLs against a precompiled list of known phishing sites. Effective for known threats but fails to detect new or rapidly changing phishing URLs.
4. **Heuristic-based Detection**: Uses predefined rules (e.g., suspicious keywords, uncommon top-level domains) to identify phishing. However, it has a high false-positive rate since legitimate URLs may match heuristic patterns.
5. **Machine Learning-based Detection**: Employs classifiers to learn from URL features and detect phishing attempts. While it can generalize better than rule-based approaches, it requires significant computational resources and large datasets for training, making it less suitable for real-time detection.

The Importance of Endpoint Detection

Detecting phishing URLs directly on the endpoint (the user's device) has become crucial as attackers find ways to bypass network-level and server-side defenses. Implementing detection within the browser offers several advantages:

- **Real-time Detection**: Phishing attempts can be blocked as users interact with potentially harmful websites.
- **Protection Against Obfuscated Attacks**: Endpoint detection can observe dynamic behaviors, such as delayed content loading, commonly used to evade server-side detection.
- **User-Specific Context**: It can consider the user's behavior and browser interactions, allowing for more tailored detection strategies.

Task Description

You are required to create a phishing detection solution that integrates at least two detection methods into a Chromium-based browser extension. Your solution should combine the strengths of different techniques to improve detection rates and reduce false positives while maintaining minimal resource consumption and fast response times.

Detection Methods to Implement

1. **Static URL Analysis**: Analyze URLs based on predefined patterns and features such as length, character entropy, or suspicious keywords.
2. **Static Content Analysis**: Extract and examine elements from the HTML/DOM, such as embedded scripts, suspicious form actions, or mismatched URLs, to identify signs of phishing.
3. **Dynamic Behavior Analysis**: Monitor real-time events like page redirects, dynamic content modifications, and external script loads to detect behaviors typical of phishing attacks. This is particularly useful against sophisticated phishing techniques that reveal malicious behavior only after the page is loaded.

Note: You must implement at least two of the above methods in your solution.

Constraints

- **No Signature-based Solutions**: You are not allowed to use signature-based solutions (e.g., blacklists of known malicious URLs) or any services that rely on external resources. All implementation and execution must be performed solely on the endpoint.

Implementation Requirements

- **Develop** a deployable and tested Chromium-based browser extension.
- **Ensure** your extension efficiently analyzes websites and provides immediate feedback to users about potential phishing threats.
- **Optimize** your extension for minimal impact on system resources (memory and CPU usage) and quick detection times.

Evaluation Criteria

Your browser extension will be evaluated based on the following criteria:

1. **Coverage Test (35%)**: A higher coverage indicates your solution detects a larger number of phishing cases.
2. **Error Test (35%)**: This test emphasizes precision, where minimizing false positives is crucial.
3. **Response Time Test (10%)** : Measured as the average time it takes for your algorithm to detect phishing after the DOM is fully loaded across all tested sites. Faster response times indicate better real-world performance.
4. **Memory Footprint Test (10%)**: Assesses how efficiently your extension uses system memory during operation, ensuring it doesn't adversely affect the user's browsing experience.

Scoring Methodology

Results for each test will be collected and normalized individually.
For instance, a group can receive maximum points for the coverage test and minimum points for the error test.

- **Top 10%**: Will receive the maximum points for each test.
- **Next 75%**: Will receive 80% of the maximum points for each test.
- **Bottom 15%**: Will receive 60% of the maximum points for each test.

Minimal score in the evaluation section (i.e., bottom 15% in each test) is 60% out of 90.

Overall Grade Breakdown

- **Extension Performance (90% of total grade)**
    - Coverage Test: 35%
    - Error Test: 35%
    - Response Time Test: 10%
    - Memory Footprint Test: 10%
- **Report (10% of total grade)**

Report Requirements

Your report should be written in English and include the following sections in the specified order:

1. **Related Work**
    - Provide an overview of academic and non-academic methods and research in this domain.
    - Describe at least two techniques for each detection method.
    - Reflect on each technique: Does it address the problem effectively?
2. **Method**
    - Describe your proposed method, highlighting its structure and flow.
    - Explain the logic behind your design choices.
    - Detail the feature selection and parameter tuning processes.
3. **Results**
    - Present a performance analysis of your method.
    - Provide metrics such as Accuracy, True Positive Rate (TPR), False Positive Rate (FPR), and weighted F1 Score.
4. **Limitations and Future Work**
    - Discuss the limitations of your approach.
    - Identify scenarios where your method may not perform well.
    - Suggest alternative strategies or future research directions to address these limitations.

Submission Guidelines

- The assignment must be completed in groups of **two students only**.
- The submission should be a single archive file containing:
    - **Report**: In PDF format.
    - **Python Notebook Code**: In IPYNB format (not a Google Colab link).
    - **Archived Extension File**: Ready to deploy in ZIP or RAR format.

Important: The extension must be deployable.

Notes

- Remember that real-world applicability is key; your phishing detection algorithm should perform effectively in practical scenarios

Good Luck!

Appendix A: Technical Guide on Creating a Chromium-Based Browser Extension

Creating a browser extension for Chromium-based browsers (like Google Chrome, Microsoft Edge, or Brave) involves developing web technologies like HTML, CSS, and JavaScript. Below is a concise guide to help you get started.

1. Directory Structure

Create a new folder for your extension project. This folder will contain all the files related to your extension.

```
my-extension/
├── manifest.json
├── background.js
├── content_script.js
├── popup.html
└── icons/
    └── icon.png
```

2. The Manifest File (manifest.json)

The manifest.json file is the blueprint of your extension. It provides essential information and defines the resources your extension will use.

Basic manifest.json Template:

```
{
  "manifest_version": 3,
  "name": "Phishing Detection Extension",
  "version": "1.0",
  "description": "Detects phishing attempts using multi-layered analysis.",
  "permissions": ["tabs", "webNavigation", "scripting"],
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["content_script.js"]
    }
  ],
  "action": {
    "default_popup": "popup.html",
    "default_icon": "icons/icon.png"
  }}
```

Key Components:

- "manifest_version": Should be 3 for the latest version.
- "permissions": Permissions required by the extension (e.g., access to tabs, web navigation).
- "background": Background scripts that run continuously.
- "content_scripts": Scripts that run in the context of web pages.
- "action": Defines the extension's toolbar icon and popup.

3. Background Script (background.js)

The background script manages events and communicates between different parts of the extension.

Example background.js:

```
chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {
    if (request.action === "GetResultsFromContentScript") {
      // Do Something
    };

    return true;
  });
```

4. Content Script (content_script.js)

The content script analyzes the web page's content and performs phishing detection.

Example content_script.js:

```
function getClassification(){
  const isURL = isDetectedByURL() == 1;
  const isContent = isDetectedByContent() == 1;
  const isPhishing = isURL || isContent;
  return {
    isURL,
    isContent,
    details: isPhishing ? "This page may be a phishing attempt." : "This page seems safe."
  };
}

window.addEventListener('load', () => {

  // Wait for 3 seconds before executing GetClassification
  setTimeout(() => {
    GetClassification();
  }, 3000); });
```

5. Popup Page (popup.html)

The popup page provides a user interface when the extension's icon is clicked.

6. Loading the Extension into the Browser

1. Open Chrome and navigate to chrome://extensions/.
2. Enable **Developer mode** by toggling the switch in the upper-right corner.
3. Click **Load unpacked** and select your extension's directory.

7. Testing the Extension

- Navigate to different websites to see if the extension detects any phishing attempts.
- Use test phishing sites (e.g., sites designed for phishing awareness training) to validate your detection methods.

8. Packaging the Extension

For submission, compress your extension folder (my-extension/) into a ZIP or RAR file, Ensure all necessary files are included and paths are correct.

Appendix B: Sample Report Outline

Below is a concise example report, submitted by one of the students in the previous year course, of how your report should be structured and what content to include in each section.

Please note: The task subject is changing throughout the years, so the content of the report is irrelevant.

Cyber Security and AI - Task 2

Ploni Almoni (999999999)
ploni.almoni@post.runi.ac.il,
Reichman University
Israel

ABSTRACT

This article focuses on developing effective and lightweight detection mechanisms to combat the significant cyber-security threat of ransomware. Traditional methods of recognizing malware have limitations in keeping up with the evolving landscape of ransomware, necessitating the use of advanced techniques like machine learning (ML). ML algorithms can analyze the intricate characteristics and patterns exhibited by ransomware, enabling accurate and efficient detection. While previous research has primarily concentrated on detection, there has been limited attention given to ransomware classification. In this study, a simple approach utilizing static analysis and N-gram representation is presented for conducting static analysis on a malicious database to classify ransomware. By employing ML algorithms and feature selection techniques, the study achieves accurate classification of ransomware samples. The article addresses challenges such as feature engineering and the need for flexible feature selection strategies to optimize classifier performance. Notably, the results demonstrate an accuracy of 0.993 and a recall of 0.961 on a compact model size of 1.2 MB. These findings underscore the significance of analyzing malware and ransomware for classification and detection purposes while emphasizing the necessity for further research and development in the field of ransomware classification.

ACM Reference Format:
Ploni Almoni (999999999). 2023. Cyber Security and AI - Task 2. In *Proceedings of (Cyber Security and AI).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Ransomware, a malicious software that encrypts files and demands a ransom for their release, continues to be a significant cybersecurity threat, causing substantial financial losses and operational disruptions for individuals and organizations. Detecting and recognizing ransomware in a timely and accurate manner is crucial for effective cybersecurity defense. Traditional approaches to malware recognition have limitations in keeping up with the ever-evolving landscape of ransomware, necessitating the development of advanced techniques that can provide robust and lightweight detection mechanisms.

Machine learning (ML) has emerged as a promising approach for malware detection and classification, including ransomware. By leveraging ML algorithms, it is possible to analyze the intricate characteristics and patterns exhibited by ransomware and differentiate it from benign software. ML models can learn from vast amounts of data and identify subtle indicators that distinguish ransomware behavior, enabling accurate and efficient detection.

To date, the majority of research on ransomware has focused on its detection, with limited attention given to classification. However, one known study has specifically addressed ransomware classification, researchers collected API sequences from samples belonging to seven ransomware families by executing the ransomware in a controlled sandbox environment. They then employed machine learning algorithms for classification, identifying the multilayer perception (MLP) classifier as the most effective. However, ransomware that uses dynamic environment fingerprinting techniques can evade detection through dynamic analysis. As a result, dynamic analysis methods are ineffective in detecting this type of ransomware.

Static analysis, an innovative approach to ransomware detection, involves examining the structure and characteristics of malware samples without executing them. This approach enables the identification of ransomware based on its static attributes, such as file properties, metadata, and code structure. Static analysis provides an effective means to detect ransomware even before it is executed, allowing for proactive measures to prevent its harmful effects.

In the context of static ransomware detection using ML, the primary objective is to develop automated systems that accurately classify malware samples as either ransomware or benign software. These systems analyze the static attributes of the samples, extract relevant features, and train ML models to make precise predictions. The use of ML algorithms enables the identification of complex patterns and behaviors exhibited by ransomware, facilitating accurate detection and classification.

However, there are challenges that need to be addressed in the development of lightweight static ransomware detection systems. One major challenge is the extraction of meaningful and discriminative features from malware samples that capture the essence of ransomware behavior. Feature engineering plays a critical role in ensuring that ML models can effectively differentiate between ransomware and benign software based on static attributes.

This study presents a simple approach for conducting static analysis on the malicious database by treating the code as a text file and representing it using N-gram. The analysis involves employing machine learning models to train a model capable of accurately identifying malicious in subsequent instances.

Cyber Security and AI, June 28, 2023, Reichman University, Israel