MINOR PROJECT

FINAL SEMESTER REPORT

on

CSRF Vulnerability Detection

Submitted By:

Name	Roll No	Branch
Aastha Virdi	A25305221019	CSE 6th Semester
Paranjay Goel	A25305221031	CSE 6th Semester
Yuvraj Singh	A25305221033	CSE 6th Semester
Tushar Negi	A25305221074	CSE 6th Semester

Under the guidance of

Dr. Shubhani Aggarwal

Associate Professor

Amity School of Engineering and Technology

School of Computer Science

Amity School of Engineering and Technology

Amity University, Mohali

2021-25

Approved By

(Dr. Shubhani Aggarwal)
Project Mentor

(Dr. Harvinder Singh)
Project Coordinator

ABSTRACT

In this era of digitization, web applications have become indispensable tools, handling daily needs ranging from finance to entertainment, and much more. However, the existence of cyber threats, particularly Cross-Site Request Forgery (CSRF) attacks, poses a significant risk to the security of these web applications, such as Internshala and Amizone. This attack exploits the authentication of users to initiate unauthorized actions, endangering sensitive information and the integrity of the end user's system. From the literature, we observed that the existing methods to mitigate this vulnerability often fall short in effectiveness and practicality. Hence, this report introduces a novel framework designed to detect and prevent this vulnerability effectively. The proposed approach leverages source code parsing to verify the presence of CSRF tokens, crucial for safeguarding against CSRF attacks. By intercepting and analyzing request headers containing cookies, including session IDs and tokens, the proposed method identifies and validates CSRF tokens embedded within the web pages. We have implemented this approach through an interpreter and token detector, which scans HTML code for token-like expressions, distinguishing genuine CSRF tokens from other data. In the end, we can say that this research contributes to enhancing the security of web applications against CSRF attacks and ensures the confidentiality and integrity of end-user data.

Keywords: Cross-Site Request Forgery (CSRF), CSRF tokens, web cookies, JavaScript, security, privacy.

TABLE OF CONTENTS

Contents

1 Preface **5**
1.1 Background 5

2 Introduction **5**

3 Literature Review **6**

4 Objectives **6**

5 Design **6**

6 Implementation **7**
6.1 Pseudo code 9
6.2 Output Screen 11
6.2.1 Submission 11
6.2.2 Interception and Crawler Interface 11
6.2.3 Generation of Code 12

LIST OF FIGURES

List of Figures

1 Level 0 DFD for the proposed method 7
2 Process Visualiation 8
3 Process Pseudo code 10
4 User Window 11
5 JavaScript for Interception and Crawling 12
6 Python Script for generation of CSRF Token 13
7 JavaScript for Interception and Crawling 13

1 Preface

In the ever-evolving landscape of digital interconnections, the Internet stands as a testament to humanity's ingenuity and innovation. Its transformative power has reshaped our lives, rendering once arduous tasks effortless and democratizing access to information and services. Central to this digital revolution are web applications, the backbone of our online interactions, facilitating everything from financial transactions to social networking with unparalleled convenience.

Yet, as our reliance on web applications deepens, so too does the threat of cyber threats loom larger. Among these threats, Cross-Site Request Forgery (CSRF) emerges as a formidable adversary, exploiting user trust and system vulnerabilities to execute unauthorized actions and compromise sensitive data. In an era defined by the relentless pursuit of connectivity and efficiency, the imperative to safeguard against CSRF attacks has never been more urgent.

1.1 Background

The evolution of the Internet throughout the 20th and 21st centuries has heralded an era of unprecedented connectivity and digital innovation. From its humble beginnings as a research project to its current status as a ubiquitous global network, the Internet has transformed the way we communicate, conduct business and access information. Central to this transformation are web applications, and dynamic software programs accessible via web browsers, which have become integral to modern life. These applications power a vast array of online services, from social media platforms and e-commerce websites to banking portals and productivity tools. Web applications have become indispensable tools in everyday life, powering diverse services ranging from social media platforms to online banking systems. However, alongside their utility comes the omnipresent threat of cyberattacks. Among these threats, Cross-Site Request Forgery (CSRF) stands out as a significant concern. The prevalence and severity of CSRF at-attacks underscore the critical need for robust mitigation strategies to protect web applications and their users. While various techniques and technologies have been developed to address CSRF vulnerabilities, gaps remain in their effectiveness and scalability.[1] Thus, there is a pressing need for innovative approaches that can detect and prevent CSRF attacks comprehensively and efficiently. By proposing a simple yet robust solution, this research contributes to enhancing the security of web applications against CSRF attacks, ensuring the confidentiality and integrity of user data in an increasingly interconnected digital landscape.

2 Introduction

Cross-Site Request Forgery (CSRF) attacks represent a persistent and formidable threat to the security of web applications in today's digital ecosystem. At its core, a CSRF attack exploits the trust between a user's browser and a web application to execute unauthorized actions on behalf of the user, often leading to financial loss, data breaches, and reputational damage.[2][1] In this research paper, we delve into the intricacies of CSRF attacks, exploring their underlying mechanisms and implications for web security. Furthermore, we investigate in the research paper, a range of prevention strategies designed to mitigate the risk of CSRF attacks effectively. By examining the efficacy of techniques such as synchronizer tokens, same-site cookies, Referer header checks, double submit cookies, and Content Security Policy, this paper aims to provide a comprehensive framework for defending against CSRF attacks and safeguarding the integrity of web applications against malicious exploitation.

3 Literature Review

The literature review is a big component of this report as it provides a comprehensive overview of the existing research relevant to CSRF vulnerabilities. The authors of [1] shed light on the nature and meaning of CSRF attacks, including its relevancy in today's scenario. Being one of the OWASP's top 10 security threats, CSRF poses a huge threat to online users. The authors have thus suggested many methods of checking for the issue, such as anti-CSRF tokens, verifying referer headers, and limiting the life of authentication cookies, all of which have their pros and cons, which include risks such as session hijacking attacks and unavailability of a unified procedure or system for implementation. They have also talked about the types and methods of CSRF attacks, which have provided information on various methods of mitigating the same for our own methodology.

Following this, the authors of [3] have implemented a system titled *Browser-Enforced Authenticity Protection (BEAP)*, that works as a Firefox browser extension and checks whether a request reflects any change in the sensitive authentication token, or any change in the user's intention, stripping the malicious requests, if it finds any. This method, thus, puts the control in the browser's hand, which might lead to increased load on the browser.

Further, the authors of [2] provide a design for an automatic algorithm for the direct filtering of HTML requests to check for cross-site origins. The method works on decreasing the attack surface of CSRF attacks in general but is still in development to avoid false positives.

From the literature review, we observe that the use of source code parsing has yet to be implemented in a simple and effective manner, and it can even be used by the common day-to-day user. Therefore, we design a method to parse source code for the requests that show signs of CSRF attack due to tampering of CSRF tokens present in HTML requests.

4 Objectives

The main objectives of this report can be described as follows:-

- To propose a method for checking if CSRF tokens are present in a web application or not.
- To ensure there is no tampering of data from a secure user's web request.

5 Design

The project is based on the implementation of the CSRF token generation after the request is thoroughly scanned via the crawler.

Figure 1 shows the level 0 DFD of the proposed methodology, showing the exchange of commodities between the user, server, and the detector itself.

Should the request be already secured by CSRF Token, then the request should be untouched, only in the absence of the token should the project run. Firstly, the request will be sent to the authentication layer, where it will be intercepted by the Authentication Layer. Having been seized by the interceptor application, its entire source code will be scanned in search of the presence of CSRF tokens. Should their presence not be found, then the following layer, the Application Layer, will generate a string of random characters that will work as the

CSRF Token. This string will be simultaneously sent to two locations: once to the commandeered source code to be appended and the other to the Server via a parallel delivery line. Having been appended, the new request will be sent on its way to the final layer, the Validation Layer, where the sent request will be entered into the server. It will now be compared to the token sent in parallel andwill pass if right.

It is an asymmetric algorithm for encrypting the data, which consists of the public key given to everyone and the private key given to the recipients only for decryption. It is one of the safest techniques as if a public key is compromised, even then it cannot decrypt the cipher as the private key is still unknown. The concept of key is derived from the fact that factorization of large integers is not an easy task.

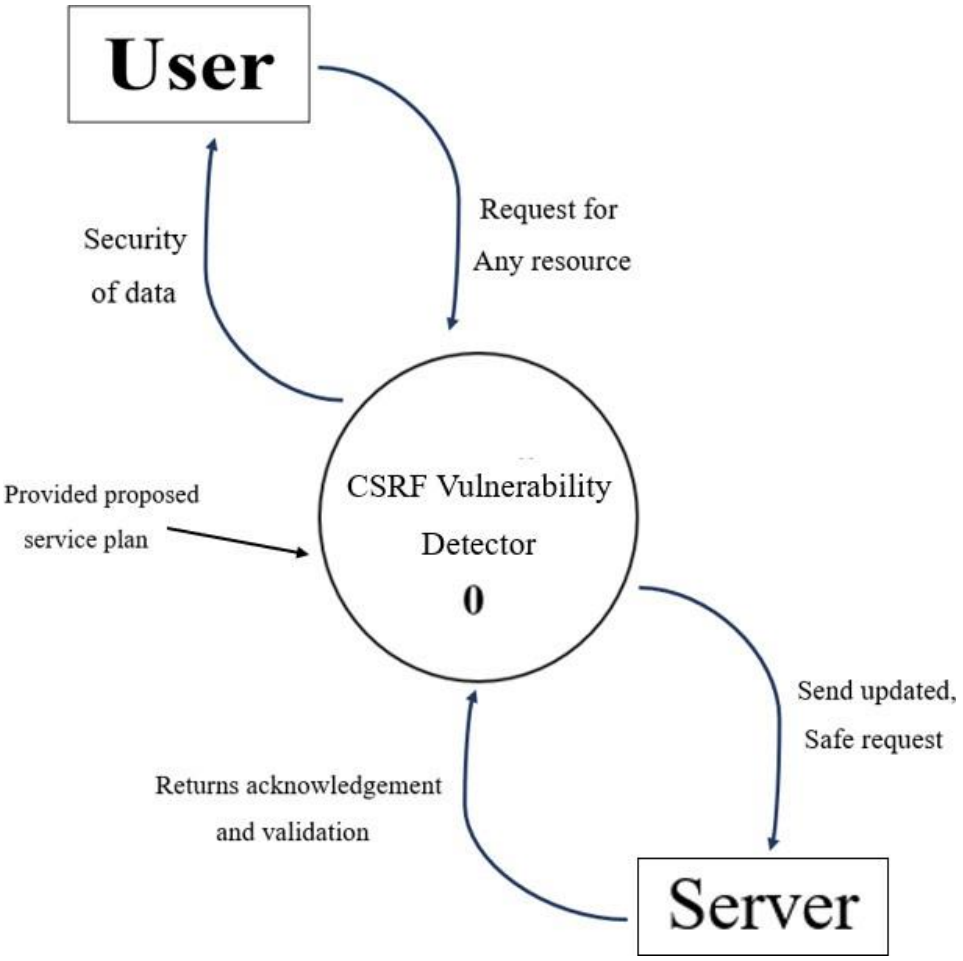

Figure 1: Level 0 DFD for the proposed method

6 Implementation

1. **Request Generation:**
   The first step, as shown in Figure 2, is the generation of an HTML request which will be generated by the user end or the attacker end. This request will contain headers, the method of request, cookies, and other headers as well.

2. **Interception:**

The request generated from Step 1 will be intercepted by initially the website and in the future our application. This application, engineered with JavaScript included, will have the combined task of intercepting, as well as scanning the contents of the request for the presence of CSRF Tokens

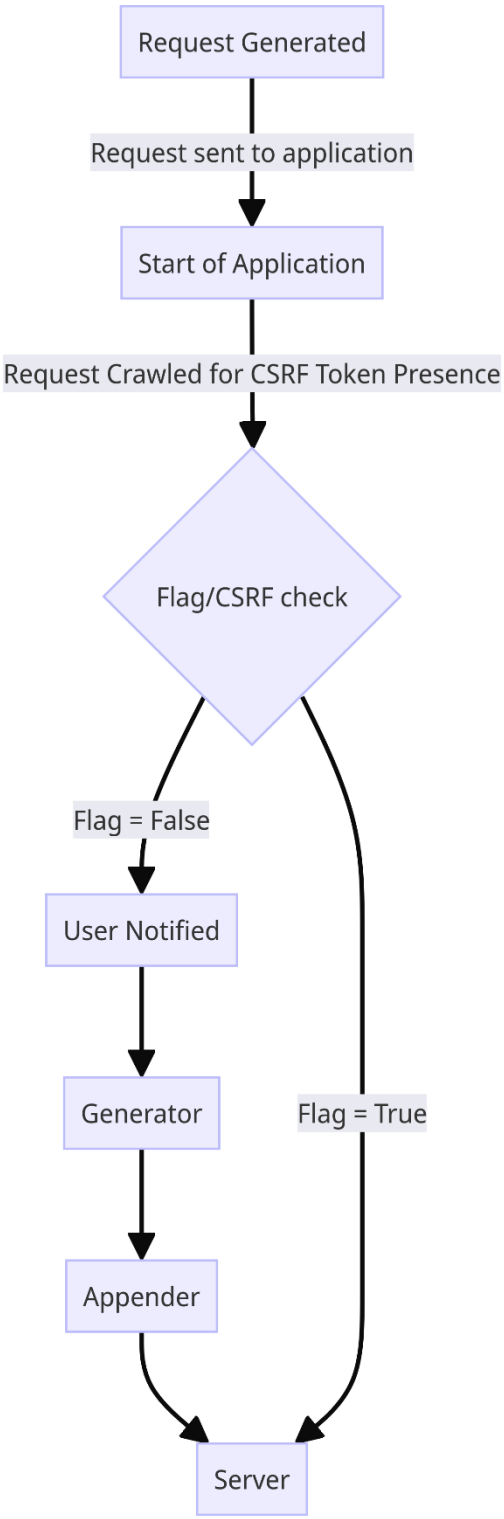

Figure 2: Process Visualiation

- **Proxy:** The application will act as a proxy, making the request pass through itself, hence being intercepted.
- **Request Transfer:** After interception, the request will be traversed by the detection code.

3. **Detector:**
   The request will then be sent to the detector, where it will check whether the CSRF tokens arepresent or not. The detector will scan the request and if the tokens are not present then the flag will be set to false, and this information will be forwarded to the prevention system.

4. **Prevention System:**
   The prevention system will be Flask Application, a web framework for Python that can handle web requests and responses. Upon reading the flag as false, this will generate a CSRF token that will be in text format, and this text file will be reverted to the application.

5. **Server:**
   Once the application appends the received text file with the request, the new secure request will be forwarded to the Server for due processing, while the same Token will have been sent to the server parallelly as well.

6 Implementation

6.1 Pseudo code

The proposed methodology involves the use of 5 distinct working parts or services, which workin harmony and case-based turns to properly implement the software. These 5 parts are namedas: -

- Interceptor - aims to ’catch’ the incoming request from the user side, and pass it into the detector.
- Crawler - aims to parse the request for any signs of CSRF tokens.
- Generator - aims to generate a hexadecimal CSRF token for the request, should a token not be found.
- Appender - aims to attach the hexadecimal token to the request, and send it back to its designated path towards the server.
- Verifier - aims to check the final incoming request from the user at the server itself, verifying the source and token to fulfill the request.

Each method does the specified task involving steps as specified in Figure 3, which contains the workflow of each part.

These 5 mechanisms work according to the request, whether the token is present or not, and do the needful in the case where a token is not found, meaning the data like email ID and its password could easily be stolen from the request. The addition of the token, after verification, confirms the source, which means that the server verifies that the request was indeed not tampered with.

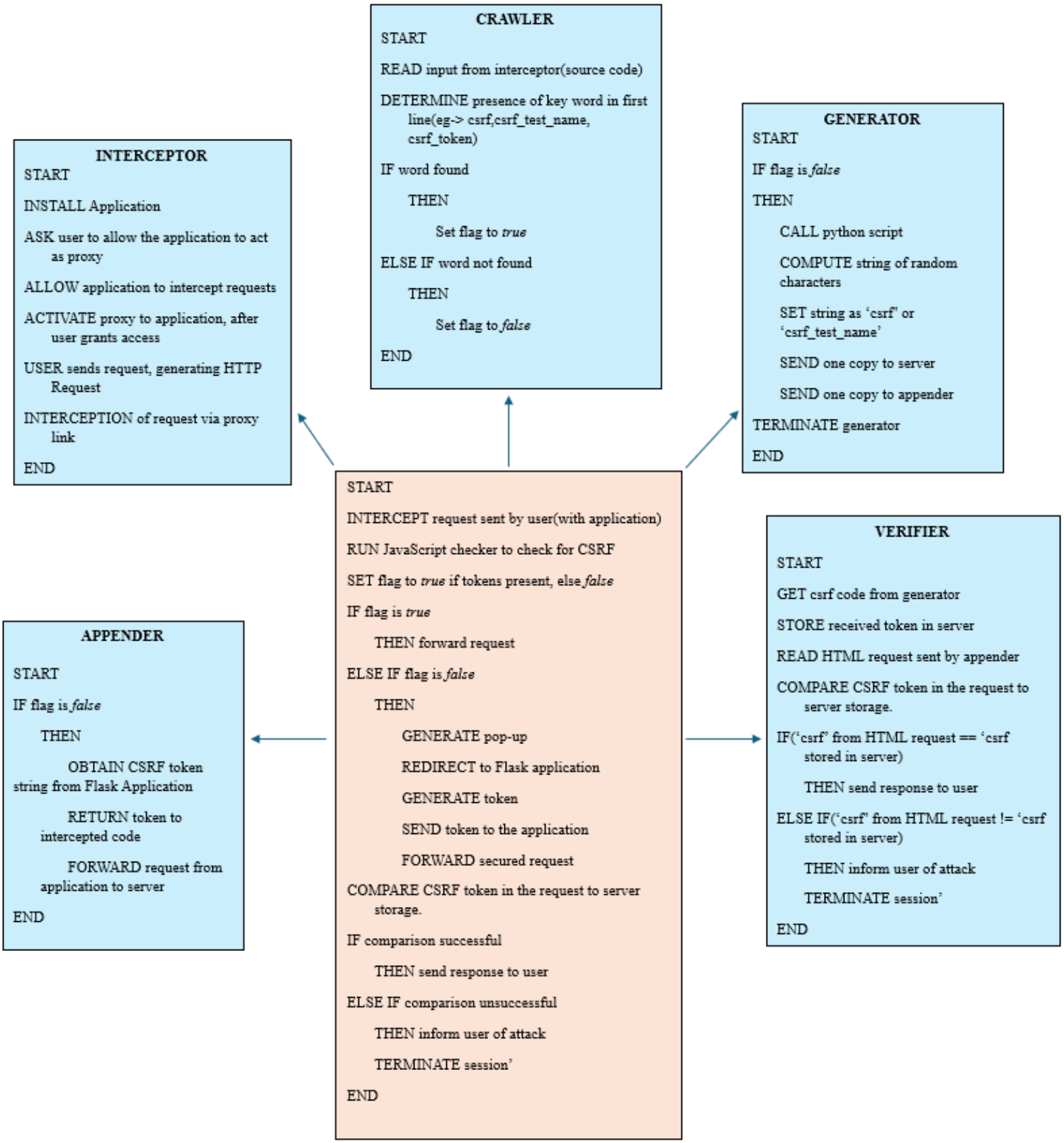

Figure 3: Process Pseudo code

6.2 Output Screen

This section defines the outputs of the various components related to the CSRF detection and prevention system.

6.2.1 Submission

This section describes the image of the output of the interceptor as seen by the user when they click “Sign In”. The pop-up alerts the user of the lack of CSRF Tokens, as found by the application code below.

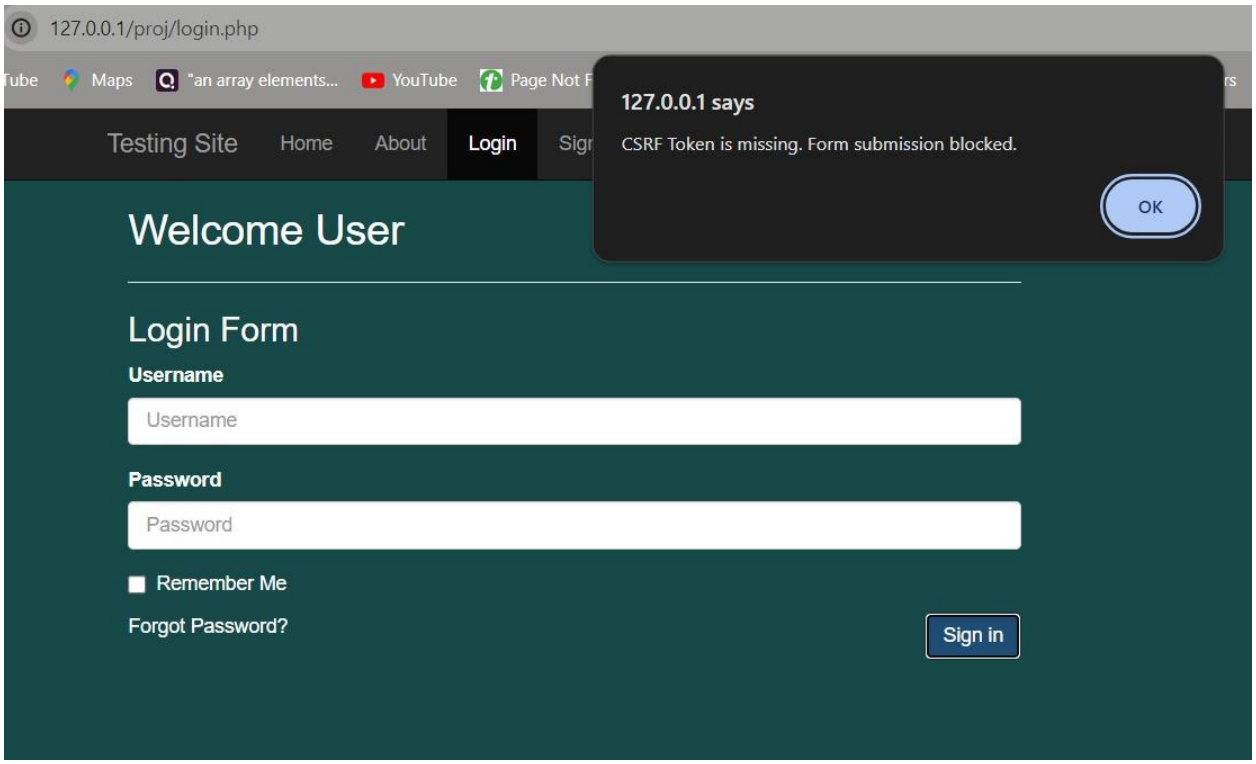

Figure 4: User Window

6.2.2 Interception and Crawler Interface

This section describes the image of the JavaScript code which currently acts as the interceptor (Figure 5), a task later implemented by our application. Additionally, the code checks for the presence of CSRF Tokens, and redirects to Flask should they not be present.

```php
<?php include_once 'partials/footer.php';?>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    window.onload = function() {
        var csrf_token = '<?php echo isset($csrf_token)? $csrf_token : '';?>';
        if (csrf_token === '' || document.getElementById('loginForm').querySelector('input[name="csrf_token"]').length == 0) {
            // If CSRF token is not present or empty, prevent form submission
            document.getElementById('loginForm').addEventListener('submit', function(event) {
                event.preventDefault();
                alert('CSRF Token is missing. Form submission blocked.');
                // Redirect to Flask app to fetch CSRF token
                window.location.href = 'http://127.0.0.1:5000/get_csrf_token';
            });
        } else {
            // If CSRF token is present, append it to the form
            document.getElementById('loginForm').querySelector('input[name="csrf_token"]').value = csrf_token;
        }
    };
</script>
```

Figure 5: JavaScript for Interception and Crawling

6.2.3 Generation of Code

This section describes the Python script (Figure 6) that generates the CSRF token should they not be present. The output of the generator code (Figure 7) will be a token sent back in text format.

```python
from flask import Flask, jsonify
import os
import secrets

app = Flask(__name__)

@app.route('/get_csrf_token')
def get_csrf_token():
    # Read the result from the file
    result_file_path = 'result.txt'
    csrf_token_file_path = 'csrf.txt'

    with open(result_file_path, 'r') as file:
        result = file.read().strip()

    # If result is false, generate and save CSRF token
    if result == 'false':
        csrf_token = generate_csrf_token()
        with open(csrf_token_file_path, 'w') as token_file:
            token_file.write(csrf_token)
    else:
        csrf_token = None

    # Return the CSRF token in JSON format
    return jsonify({'csrf_token': csrf_token}), 200

# Route for generating CSRF token
def generate_csrf_token():
    # Generate CSRF token logic here
    csrf_token = secrets.token_hex(16)
    return csrf_token

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

Figure 6: Python Script for generation of CSRF Token

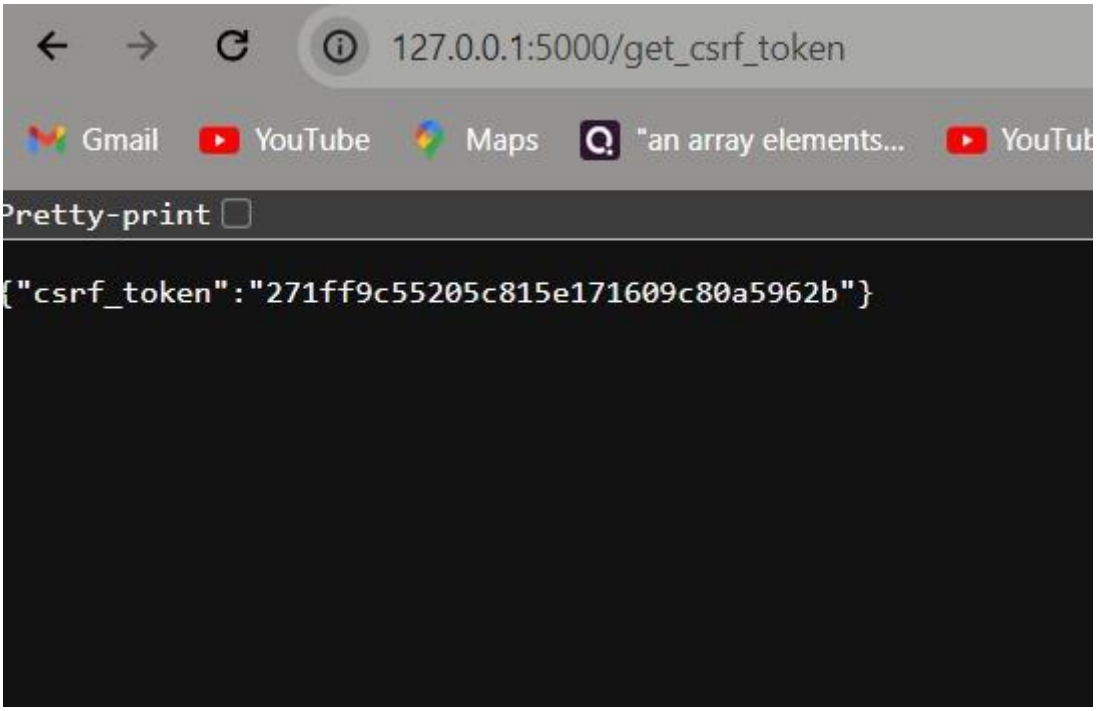

Figure 7: JavaScript for Interception and Crawling

References

[1] R. D. Kombade and B. Meshram, "Csrf vulnerabilities and defensive techniques," *International Journal of Computer Network and Information Security*, vol. 4, no. 1, p. 31, 2012.

[2] P. De Ryck, L. Desmet, W. Joosen, and F. Piessens, "Automatic and precise client-side protection against csrf attacks," in *Computer Security–ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings 16*. Springer, 2011, pp. 100–116.

[3] Z. Mao, N. Li, and I. Molloy, "Defeating cross-site request forgery attacks with browser-enforced authenticity protection," in *Financial Cryptography and Data Security: 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers 13*. Springer, 2009, pp. 238–255.