VerifyMyLink

A Tool for Safe and Verified Download Links

Cyber Security 2025 Assignment 1 Report

Submitted by:

Sarvesh Jhawar

160123737061

BE (Information Technology(IT-1))

Chaitanya Bharathi Institute of Technology (CBIT)

August 2025

Contents

Abstract

Users often download software from search results without verifying their authenticity. Many top results may lead to malicious sites that distribute malware or phishing content. This project, **VerifyMyLink**, solves this problem by providing a tool that verifies download links using the VirusTotal API and a simple frontend. The backend (Spring Boot) checks whether a given URL is malicious or harmless, and the frontend (HTML/CSS/JS) provides a clean interface for users to test any link. This report documents the design, implementation, results, and future improvements of VerifyMyLink.

Chapter 1

Introduction

The internet is filled with download sources, but many are unsafe. Clicking on fraudulent links can expose users to trojans, ransomware, or phishing attacks. Traditional antivirus software focuses on files *after* download. This project addresses the issue at the *link level* – before a user even downloads the file.

Objective:

- Build a tool that verifies URLs using trusted cybersecurity services.
- Provide users with instant feedback if a link is safe or malicious.
- Create a user-friendly web interface for accessibility.

Chapter 2

System Design

2.1 Architecture

The system is composed of:

1. **Frontend (index.html)** – A web page with input form, styled with CSS, using JavaScript to send requests.
2. **Backend (Spring Boot)** – REST controller that connects with VirusTotal API, analyzes URLs, and sends results.
3. **VirusTotal API** – Third-party service providing real-time analysis of URLs against multiple antivirus engines.

2.2 Workflow

1. User enters a URL in the frontend form.
2. Frontend sends a POST request to backend (‘/check‘).
3. Backend submits the URL to VirusTotal API.
4. VirusTotal returns analysis (harmless vs. malicious).
5. Backend sends formatted response back to frontend.
6. Frontend displays result in green (safe) or red (unsafe).

Chapter 3

Implementation

3.1 Backend (Spring Boot)

The backend controller uses `RestTemplate` to call VirusTotal API, process the response, and return results.

Listing 3.1: UrlCheckController.java

```
@RestController
@RequestMapping("/check")
@CrossOrigin(origins = "*")
public class UrlCheckController {
    private static final String API_KEY = "YOUR_API_KEY";
    private static final String API_URL = "https://www.virustotal.com/
       api/v3/urls";

    @PostMapping
    public String checkUrl(@RequestParam String url) {
        try {
            RestTemplate restTemplate = new RestTemplate();
            HttpHeaders headers = new HttpHeaders();
            headers.set("x-apikey", API_KEY);
            headers.setContentType(MediaType.
               APPLICATION_FORM_URLENCODED);

            HttpEntity<String> request = new HttpEntity<>("url=" + url,
                headers);
            ResponseEntity<String> response = restTemplate.
               postForEntity(API_URL, request, String.class);

            ObjectMapper mapper = new ObjectMapper();
            JsonNode root = mapper.readTree(response.getBody());
            String analysisId = root.path("data").path("id").asText();

```

```
23              String reportUrl = "https://www.virustotal.com/api/v3/
                    analyses/" + analysisId;
24              HttpEntity<Void> entity = new HttpEntity<>(headers);
25              ResponseEntity<String> reportResponse = restTemplate.
                    exchange(reportUrl, HttpMethod.GET, entity, String.class
                    );
26
27              JsonNode stats = mapper.readTree(reportResponse.getBody())
28                                      .path("data").path("attributes").
                                            path("stats");
29
30              int harmless = stats.path("harmless").asInt();
31              int malicious = stats.path("malicious").asInt();
32
33              return "   ␣Harmless:␣" + harmless + "␣|␣   ␣Malicious:␣" +
                     malicious;
34          } catch (Exception e) {
35              return "Error␣checking␣URL:␣" + e.getMessage();
36          }
37      }
38  }
```

3.2 Frontend (index.html)

The frontend provides a clean UI, with JavaScript making requests to the backend.

Listing 3.2: index.html

```
<main class="container">
    <h2>       URL Safety Checker</h2>
    <form id="urlForm" class="url-form">
        <input type="url" id="urlInput"
               placeholder="Enter␣a␣URL" required>
        <button type="submit">Check</button>
    </form>
    <div id="loader" class="loader"></div>
    <p id="result"></p>
</main>
<script>
const urlForm = document.getElementById("urlForm");
urlForm.addEventListener("submit", async (e) => {
    e.preventDefault();
    const url = document.getElementById("urlInput").value;
    const formData = new URLSearchParams();
    formData.append("url", url);

    try {
        const response = await fetch("http://localhost:8081/check", {
            method: "POST",
            headers: { "Content-Type": "application/x-www-form-
               urlencoded" },
            body: formData
        });
        const resultText = await response.text();
        document.getElementById("result").innerText = resultText;
    } catch (err) {
        document.getElementById("result").innerText = "   ␣Error:␣" +
           err.message;
    }
});
</script>
```

Chapter 4

Results

4.1 Testing with URLs

- Safe input: `https://www.google.com` Output: Harmless: 90 — Malicious: 0
- Unsafe input: `http://malicious-example.com` Output: Malicious detected

4.2 Output Screenshots

The following screenshots illustrate the working of the system:

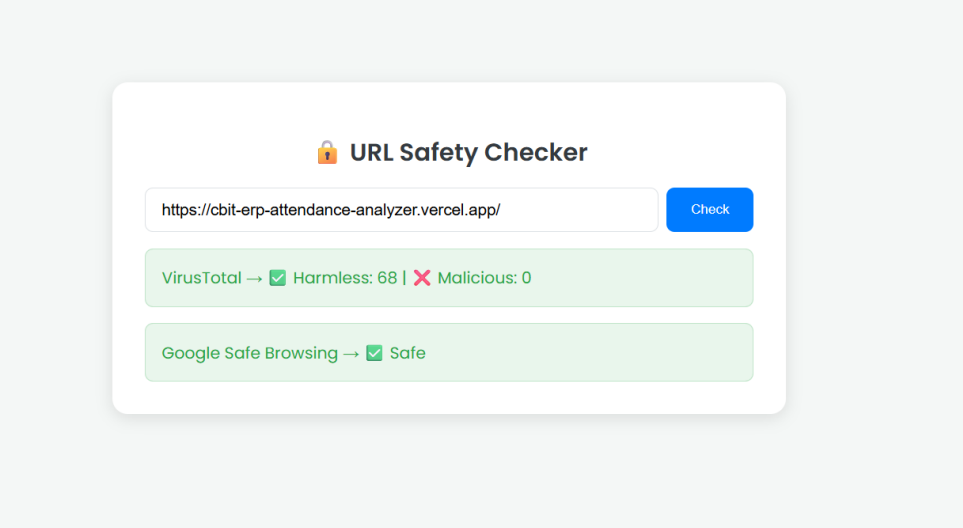

Figure 4.1: Checking a safe URL .

4.3 User Interface

The frontend clearly shows safe results in green and unsafe results in red, providing an intuitive experience.

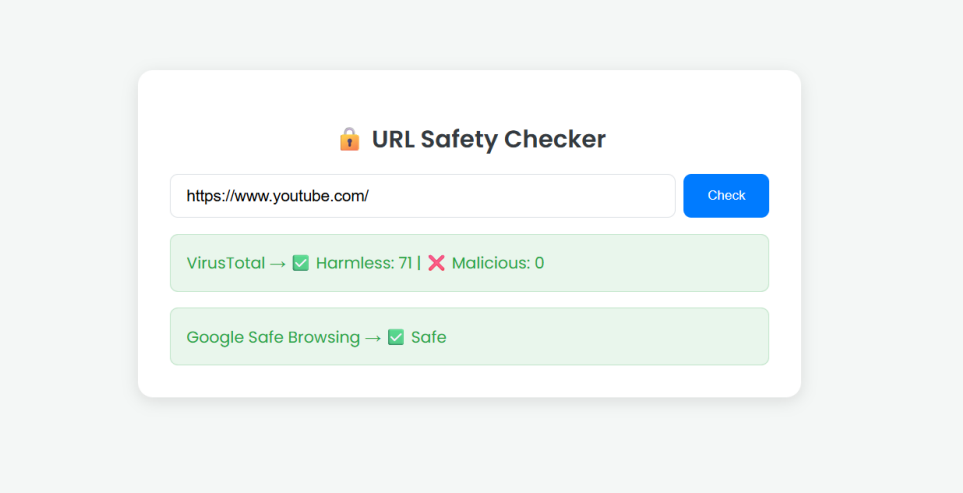

Figure 4.2: Checking a Safe URL.

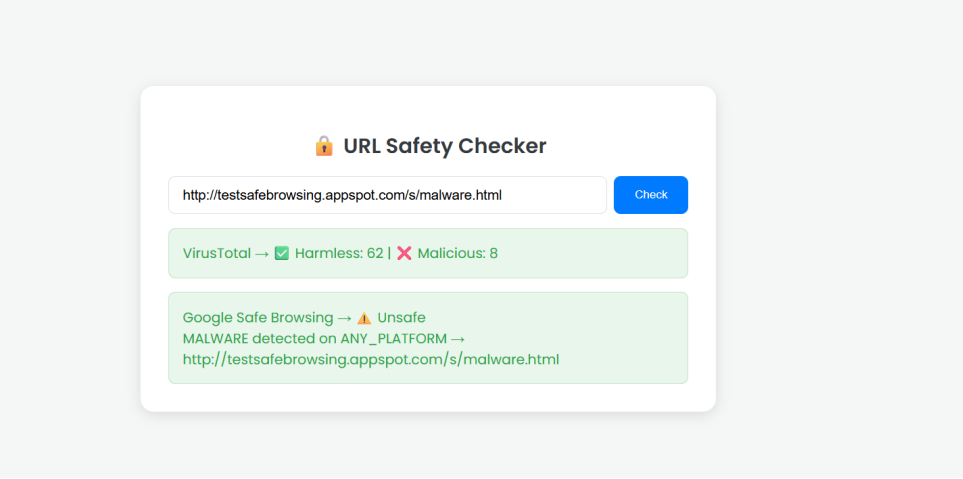

Figure 4.3: cChecking a malicious website.

Chapter 5

Advantages and Limitations

5.1 Advantages

- Detects unsafe links before download.
- Simple interface for non-technical users.
- Uses VirusTotal's global reputation database.

5.2 Limitations

- Depends on third-party API (VirusTotal).
- API key limits the number of free requests.
- Some very new malicious links may not be detected.

Chapter 6

Conclusion and Future Work

Conclusion: VerifyMyLink successfully verifies URLs, protects users from malicious downloads, and offers a clean interface for everyday use.

Future Enhancements:

- Build a Chrome/Firefox extension.
- Use AI models for phishing detection.
- Add a database of trusted vendors.
- Deploy the tool publicly for wider access.

Chapter 7

References

- VirusTotal API – `https://www.virustotal.com/`
- OWASP Security Guidelines – `https://owasp.org/`
- Google Safe Browsing – `https://safebrowsing.google.com/`