

Table of contents

S. No	Title	Page No.
1.	Introduction	2-3
	1.1 Importance	3-4
	1.2 Overview	4
	1.3 Objectives and goals	4-5
	1.4 Key components	6-7
2.	Background	7
	2.1 Phishing Link and Detection	7
	2.2 Malicious File and Detection	7-8
	2.3 Vulnerable Directory in URL and Detection	8
	2.4 Importance and Functionality	9-10
	2.5 Mechanism	10-13
3.	Training and Undertaken	13
	3.1 Case Diagram	13-14
	3.2 Methodology	14-16
	3.3 Hardware, language and software requirements	16-18
	3.4 Technology used	18-20
	3.5 Project Specification	20-22
4.	Code Snippets	23
	4.1 Source Code File Structure	23-24
	4.2 Codes	24-58
5.	Result and Testing	59
	5.1 Screenshot	59-62
	5.2 Conclusion	63
	5.3 Significance and impact	63-65
	5.4 Future scope	65-66
6.	References	66

1. Introduction

In an era where the digital frontier expands exponentially, so too do the threats that inhabit it. Cyber Blade is not merely a website; it is a digital bastion, forged with the mission to fortify users against the perils of the online world. At its core, Cyber Blade is a comprehensive cybersecurity platform engineered to empower users with the tools and knowledge necessary to navigate the digital landscape safely and securely.

In this project, our primary aim is to provide users with a robust defense mechanism against a variety of cyber threats. From phishing attempts to malware-infected files, Cyber Blade stands as a sentinel, tirelessly scanning and analyzing the digital terrain to identify and neutralize potential risks.

Our project encompasses three key components:

- **Phishing Link Detection:** Phishing remains one of the most common and insidious forms of cyber-attacks, preying on unsuspecting users through deceptive links. Cyber Blade employs advanced algorithms to scrutinize URLs, identifying and flagging suspicious links that may lead to phishing websites.
- **Malicious File Analysis:** With the proliferation of malware across the internet, users face the constant threat of downloading infected files. Cyber Blade provides users with the ability to upload files for analysis, leveraging state-of-the-art techniques to identify and isolate malicious content, thereby safeguarding users from potential harm.
- **Hidden Directory Vulnerability Scanning:** Hidden directories represent a potential entry point for cyber attackers seeking to exploit vulnerabilities in web applications. Cyber Blade conducts comprehensive scans of URLs, uncovering hidden directories and alerting users to potential security risks.

By combining these three pillars of cybersecurity, Cyber Blade offers users a comprehensive solution for protecting their digital assets and personal information. Our project is not merely about detection; it is about empowerment, equipping users

with the knowledge and resources needed to defend themselves against the ever-evolving threats of the digital age.

In the subsequent sections, we delve deeper into the functionality and methodology behind Cyber Blade, illustrating how our platform serves as a stalwart guardian in the ongoing battle for online security.

1.1. Importance

In an era defined by digital interconnectedness, the importance of robust cybersecurity measures cannot be overstated. Cyber threats, ranging from phishing scams to malware infections, pose significant risks to individuals, businesses, and organizations alike. In this landscape, Cyber Blade emerges as a critical tool in the ongoing battle against cyber threats, offering a host of features designed to enhance online security and protect users from potential harm.

- **Protection Against Phishing Attacks:** Phishing attacks continue to be a prevalent threat in the digital realm, often targeting unsuspecting users through deceptive emails, messages, or websites. Cyber Blade's advanced phishing link detection capabilities empower users to identify and avoid suspicious links, thus mitigating the risk of falling victim to phishing scams and identity theft.
- **Defense Against Malware:** Malware, including viruses, worms, and ransomware, can wreak havoc on computers and networks, compromising sensitive data and disrupting operations. Cyber Blade's malicious file analysis feature enables users to upload files for thorough examination, allowing for the prompt detection and removal of potentially harmful content before it can inflict damage.
- **Mitigation of Hidden Directory Vulnerabilities:** Hidden directories represent a potential entry point for cyber attackers seeking to exploit vulnerabilities in web applications. By conducting comprehensive scans of URLs, Cyber Blade helps users identify and address hidden directory vulnerabilities, thereby reducing the likelihood of unauthorized access and data breaches.

- **Empowerment Through Knowledge:** Beyond its technical capabilities, Cyber Blade serves as a beacon of empowerment, equipping users with the knowledge and resources needed to navigate the digital landscape safely and confidently. By providing real-time insights and actionable recommendations, Cyber Blade enables users to make informed decisions about their online activities, ultimately fostering a culture of cyber resilience and vigilance.

1.2. Project Overview


Cyber Blade stands as your vigilant protector in the digital realm, equipped with cutting edge tools to defend you against evolving threats. Our platform offers robust scanning and analysis capabilities to safeguard you from phishing attempts, malicious files, and hidden directory vulnerabilities.


With Cyber Blade, you benefit from advanced features such as:


- **Phishing Link Detection:** Real-time identification of fraudulent URLs and phishing attempts, ensuring the safety of your sensitive information.
- **Malicious File Analysis:** Comprehensive examination of suspicious files to detect malware and shield your devices from harm.
- **Hidden Directory Vulnerability Scanning:** Through meticulous brute-force techniques, Cyber Blade uncovers potential weaknesses in web applications or servers, fortifying your online assets against unauthorized access.


1.3. Objectives and Goals


Enhanced Cybersecurity Protection: The primary objective of Cyber Blade is to provide users with advanced cybersecurity protection against a wide range of digital threats, including phishing attempts, malware, and hidden directory vulnerabilities.


 **Real-time Phishing Link Detection:** Implement a robust system for real-time identification and blocking of fraudulent URLs and phishing attempts to prevent users from falling victim to phishing attacks and safeguard their sensitive information.


 **Comprehensive Malicious File Analysis:** Develop sophisticated algorithms and tools for thorough examination and detection of malicious files, ensuring that users' devices remain secure from malware and other harmful software.

 **Hidden Directory Vulnerability Scanning:** Utilize meticulous brute-force techniques to scan web applications and servers for potential hidden directory vulnerabilities, thereby fortifying users' online assets against unauthorized access and data breaches.

 **User-Friendly Platform:** Design an intuitive and user-friendly interface for Cyber Blade to ensure ease of use for both novice and experienced users, enabling them to effectively utilize its cybersecurity features without requiring specialized technical knowledge.

 **Continuous Improvement and Updates:** Commit to ongoing research and development to stay ahead of emerging cybersecurity threats, regularly updating Cyber Blade with the latest detection algorithms and security enhancements to maintain its effectiveness in safeguarding users' digital assets.

 **Scalability and Compatibility:** Ensure that Cyber Blade is scalable to accommodate the growing needs of users and compatible with various devices and operating systems, allowing individuals and organizations to deploy it across their digital infrastructure seamlessly.

 **Education and Awareness:** Provide educational resources and awareness campaigns to educate users about common cybersecurity risks and best practices for staying safe online, empowering them to make informed decisions and actively contribute to their digital security.

1.3. Key Components

1. Phishing Link Detection Module :

- Utilizes machine learning and real-time URL analysis.
- Identifies and blocks phishing attempts.
- Protects users from accessing malicious websites.

2. Malicious File Analysis Engine :

- Employs signature-based scanning and heuristic analysis.
- Utilizes sandboxing for in-depth file examination.
- Identifies and neutralizes malware threats.

3. Hidden Directory Vulnerability Scanner :

- Uses brute-force techniques and vulnerability scanning tools.
- Identifies hidden directories and potential weaknesses.
- Mitigates security risks and unauthorized access attempts.

4. User Interface and Dashboard :

- Offers an intuitive and user-friendly interface.
- Provides access to features, scan results, and settings.
- Presents real-time threat alerts and customizable options.

5. Reporting and Logging System :

- Tracks cybersecurity events and user interactions.
- Generates comprehensive logs and reports.
- Facilitates forensic analysis and compliance reporting.

6. Integration with SIEM Systems :

- Integrates with Virus Total API and python libraries like flask, concurrent_futur, requests etc.
- Enables insert of URL and Files.
- Enhances overall security posture and threat detection capabilities.

2. Background

2.1. Phishing Link and Detection

A phishing link is a type of malicious URL that is designed to deceive individuals into providing sensitive information such as usernames, passwords, credit card numbers, or other personal information. These links often mimic legitimate websites or services, tricking users into believing they are accessing a trusted source when, in fact, they are interacting with a fraudulent one. Phishing links are commonly distributed via email, text messages, social media, or other online communication channels.

Phishing link detectors are tools or algorithms designed to identify and flag potential phishing links to protect users from falling victim to online scams. These detectors analyze various characteristics of URLs, such as domain names, subdomains, SSL certificates, and other indicators of legitimacy or maliciousness.

2.2. Malicious File and Detection:

A malicious file is any software or document intentionally designed to harm computer systems or steal sensitive information. Examples include viruses, trojans, worms, ransomware, spyware, and adware. Malicious file detection involves identifying and neutralizing these threats using various techniques.

Signature-based detection scans files for known malware signatures or patterns, while heuristic analysis identifies suspicious behavior based on file characteristics. Behavioral analysis observes file actions in a controlled environment to detect malicious behavior, and machine learning algorithms analyze file attributes to identify anomalies. Sandboxing runs files in isolated environments to monitor their behavior safely.

By utilizing these detection methods, organizations can safeguard their systems against malware threats, preventing data breaches, system damage, and financial loss. Regular updates to antivirus software, security patches, and employee education on safe browsing habits are essential for maintaining effective malware detection and protection.

2.3. Vulnerable Directory in URL and Detection

A vulnerable directory in a URL refers to a directory within a web application or website that contains files or resources susceptible to security vulnerabilities. These vulnerabilities could potentially be exploited by attackers to gain unauthorized access, execute malicious code, or steal sensitive information.

Detection of vulnerable directories involves scanning web applications or websites for known directory traversal, file inclusion, or other common vulnerabilities. This can be done manually by security professionals using tools like directory brute-forcing, or automatically through vulnerability scanning tools that analyze web server responses for signs of vulnerable directories.

Some common vulnerable directories include those with default or weak permissions, outdated software versions, or misconfigured access controls. Detecting and securing vulnerable directories is crucial for maintaining the security of web applications and preventing unauthorized access or data breaches. Regular security assessments and proactive measures such as patching software vulnerabilities and implementing access controls are essential for mitigating the risk of exploitation.

2.4. Importance and Functionality

Malicious URL and file scanners, along with vulnerable directory finding, play critical roles in cybersecurity by addressing different aspects of threat detection and mitigation:

1. Malicious URL Scanner:

- **Importance:** Malicious URLs pose significant risks, including phishing attacks, malware distribution, and credential theft. Malicious URL scanners help identify and block access to harmful websites, protecting users from falling victim to cyber threats.
- **Functionality:** These scanners analyze URLs in real-time or through databases to determine if they lead to malicious content. They often utilize threat intelligence feeds, machine learning algorithms, and behavior analysis to detect suspicious patterns and indicators of compromise.

2. File Scanner:

- **Importance:** Malicious files, such as malware and ransomware, can compromise system integrity, steal sensitive data, and disrupt operations. File scanners examine files for signs of malicious behavior, preventing them from infecting systems and networks.
- **Functionality:** File scanners use signature-based detection, heuristic analysis, sandboxing, and machine learning to identify malicious files. They scan file attributes, code patterns, and behavioral characteristics to assess the risk level and determine appropriate actions, such as quarantine or deletion.

3. Vulnerable Directory Finder:

- **Importance:** Vulnerable directories on web servers can expose sensitive information, facilitate unauthorized access, and serve as entry points for cyberattacks. Vulnerable directory finders help identify and remediate

these security weaknesses, reducing the attack surface and enhancing overall resilience.

- **Functionality:** These tools scan web servers for directories with improper permissions, misconfigurations, or known vulnerabilities. They assess directory structures, HTTP responses, and file permissions to identify potential security risks. By identifying and fixing vulnerable directories, organizations can prevent unauthorized access and data breaches.

2.5. Mechanism

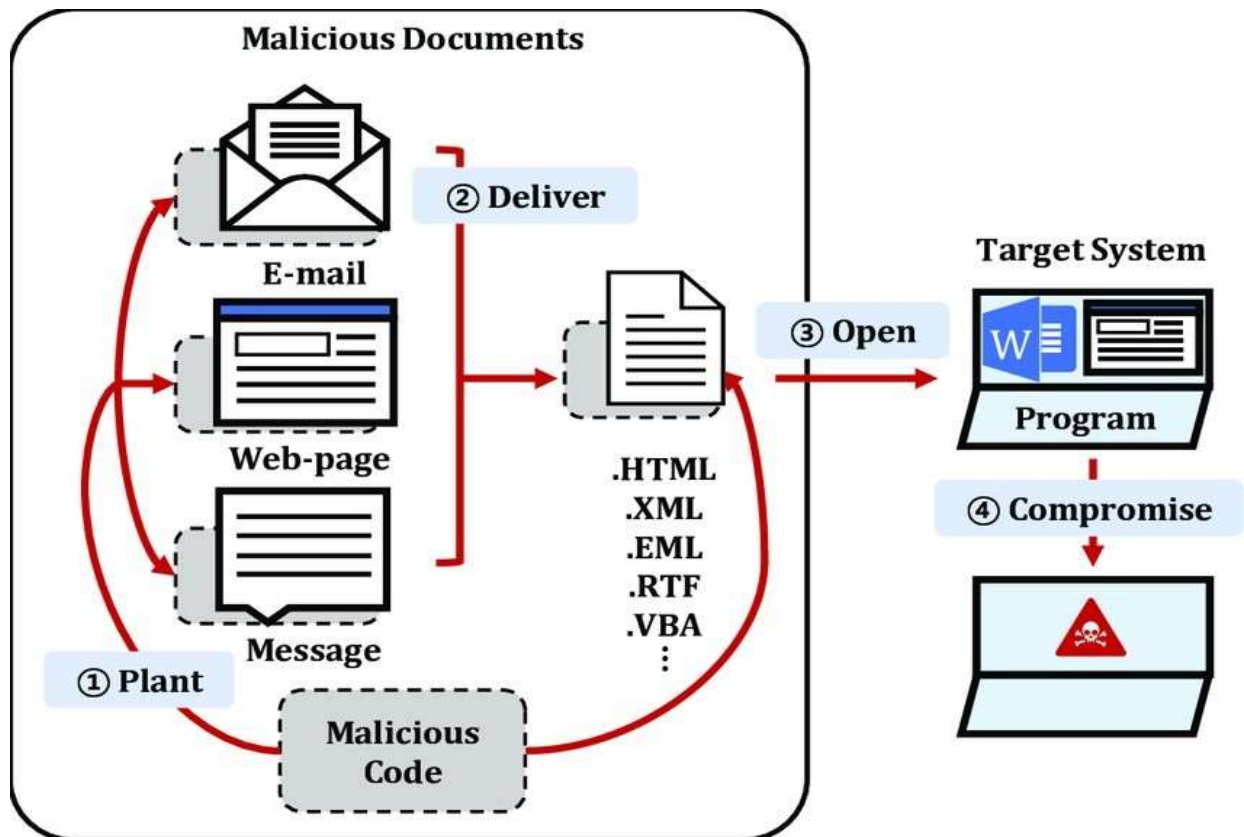
1. Phishing Link:

- Attackers leverage social engineering tactics to craft convincing emails or messages that appear to originate from reputable sources, such as banks, social media platforms, or online retailers.
- These emails often contain urgent or enticing language, prompting users to take immediate action, such as clicking on a link or downloading an attachment.
- When users click on the phishing link, they are redirected to a fake website that closely resembles the legitimate site. This fake site is often hosted on a server controlled by the attackers.
- In the background, as users enter their credentials or sensitive information into the fake website, the information is captured and sent to the attacker's server without the user's knowledge.
- The user may not be aware that their information has been compromised until they notice unauthorized activity on their accounts.



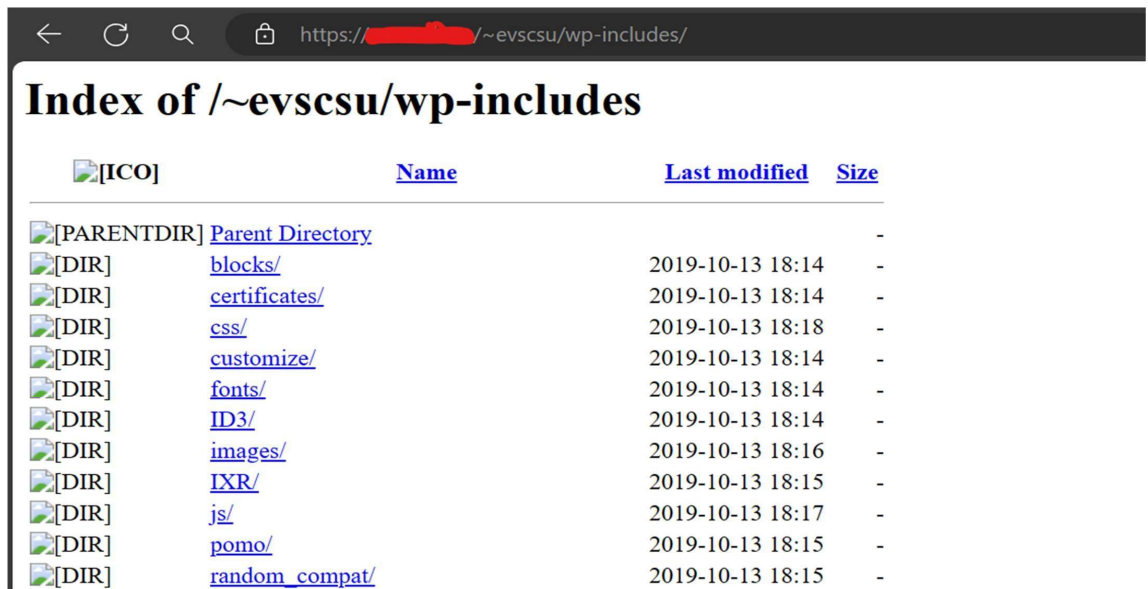
2. Malicious File














- Attackers create malicious files, such as executables or documents, that contain hidden code designed to exploit vulnerabilities in software or operating systems.
- When a user opens the malicious file, it triggers the execution of the hidden code in the background without the user's knowledge.
- The hidden code may exploit known vulnerabilities in the software or operating system to gain unauthorized access to the user's device, install malware, or perform other malicious actions.
- The user may not be aware that their device has been compromised until they notice unusual behavior, such as sluggish performance, unauthorized access to files, or unusual network activity.



3. Hidden Directory Vulnerability

- Web servers or applications may contain hidden directories or files that are intended to be restricted from public access for security reasons.
- However, due to misconfigurations or errors in the server settings, these hidden directories or files may be inadvertently exposed and accessible to anyone who knows the URL.
- Attackers can exploit these hidden directory vulnerabilities by directly accessing sensitive directories or leveraging directory traversal techniques to navigate to restricted areas of a website.
- In the background, attackers may use automated tools to scan websites for hidden directories or files and exploit any vulnerabilities they find to gain unauthorized access to sensitive information or perform other malicious actions.



	<u>Name</u>	<u>Last modified</u>	<u>Size</u>
	Parent Directory		-
	blocks/	2019-10-13 18:14	-
	certificates/	2019-10-13 18:14	-
	css/	2019-10-13 18:18	-
	customize/	2019-10-13 18:14	-
	fonts/	2019-10-13 18:14	-
	ID3/	2019-10-13 18:14	-
	images/	2019-10-13 18:16	-
	IXR/	2019-10-13 18:15	-
	js/	2019-10-13 18:17	-
	pomo/	2019-10-13 18:15	-
	random_compat/	2019-10-13 18:15	-

3. Training and Undertaken

3.1. Case Diagram

A use case diagram is a graphical representation of the interactions between actors (users or external systems) and a system under consideration. It illustrates the functional requirements of the system by showing the various ways users can interact with it to achieve specific goals. Here's how a use case diagram typically looks:

1. Actors:

- Actors represent the roles that interact with the system. They can be users, external systems, or any other entity that interacts with the system.
- Actors are depicted as stick figures or labeled blocks outside the system boundary.

2. Use Cases:

- Use cases represent the functionalities or actions that the system provides to its users.

- Each use case describes a specific interaction between an actor and the system to achieve a particular goal.
- Use cases are depicted as ovals inside the system boundary, with labels describing the action or functionality they represent.

3. Relationships:

- Relationships between actors and use cases show how actors interact with the system to accomplish tasks.
- Associations are depicted as lines connecting actors to use cases, indicating that the actor participates in or initiates the use case.

4. System Boundary:

- The system boundary defines the scope of the system under consideration. It encapsulates all the use cases and actors relevant to the system.
- Use cases and actors are placed inside the system boundary to indicate their association with the system.

5. Include and Extend Relationships:

- Include relationships indicate that one use case includes the functionality of another use case. It represents a common behavior shared by multiple use cases.
- Extend relationships represent optional or conditional behavior that extends the functionality of a base use case under certain conditions.

3.2. Methodology

- 1. Requirement Analysis:** Understand the objectives and scope of Cyber Blade. Identify the specific cybersecurity challenges it aims to address, such as threat detection, vulnerability assessment, or incident response.

- 2. Research and Planning:** Conduct research on current cyber threats, attack vectors, and defensive strategies. Plan the project's approach, including the selection of appropriate technologies, tools, and methodologies.
- 3. Design:** Design the architecture and components of Cyber Blade. Define the data flow, interfaces, and integration points. Consider factors such as scalability, performance, and security.
- 4. Development:** Implement the Cyber Blade solution according to the design specifications. Develop modules for threat detection, malware analysis, intrusion detection, or any other required functionalities. Ensure adherence to coding standards and cybersecurity best practices.
- 5. Testing:** Conduct rigorous testing of the Cyber Blade system to validate its functionality, reliability, and security. Perform unit tests, integration tests, and system tests. Use penetration testing and ethical hacking techniques to identify and address vulnerabilities.
- 6. Deployment:** Deploy Cyber Blade in a production environment or testing environment. Configure the necessary infrastructure, such as servers, databases, and network connections. Monitor performance and troubleshoot any deployment issues.
- 7. Training and Documentation:** Provide training to users and administrators on how to use Cyber Blade effectively. Prepare user manuals, technical documentation, and training materials. Ensure that users understand the system's capabilities and limitations.

- 8. Maintenance and Updates:** Establish procedures for ongoing maintenance, updates, and support. Monitor the system for new cyber threats and vulnerabilities. Apply patches and updates promptly to mitigate risks.
- 9. Continuous Improvement:** Continuously evaluate and enhance Cyber Blade to adapt to evolving cyber threats and security challenges. Gather feedback from users and stakeholders to identify areas for improvement. Incorporate new technologies and methodologies as needed.

3.3. Hardware, Language and Software Requirements

Hardware Requirements:

- 1. Server:** A computer or server capable of hosting the Flask web application. This can be a physical machine or a cloud-based virtual server.
- 2. Storage:** Sufficient storage space to store the application code, dependencies, and any uploaded files.
- 3. Network:** Stable internet connection to serve web requests and communicate with external services such as VirusTotal API.

Software Requirements:

- 1. Operating System:** Compatible with major operating systems like Windows, macOS, or Linux. Linux-based systems are commonly used for web hosting.
- 2. Python:** Version 3.x installed on the server to run the Flask web application.
- 3. Flask Framework:** Installed using pip, a Python package manager, to create the web application.

4. **Requests Library:** Python library for making HTTP requests to interact with external APIs like VirusTotal.
5. **Concurrent Futures:** Python library for parallelizing tasks, used for scanning directories concurrently.
6. **Web Server:** Optional but recommended. Popular choices include Nginx or Apache to serve the Flask application.

Language Requirements:

1. **Python:** Primary programming language for developing the web application logic, including URL scanning, file scanning, and vulnerable directory finding.
2. **HTML/CSS/JavaScript:** Frontend languages for designing the user interface and adding interactivity to the web pages. Bootstrap and jQuery are used for styling and DOM manipulation.

Other Requirements:

1. **VirusTotal API Key:** Required for accessing the VirusTotal API services for URL and file scanning functionality.
2. **Wordlist:** A text file containing a list of directory names to be used for vulnerable directory finding. It should be included in the project directory.
3. **Development Environment:** An integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Sublime Text for writing and debugging the code.
4. **Version Control:** Optional but recommended. Git for version control to track changes and collaborate with others on the project.

5. **Deployment Platform:** A platform for deploying the Flask application, such as Heroku, AWS, Google Cloud Platform, or a self-hosted server.
6. **Auto-py-to-exe:** Utilized for converting Python scripts into standalone executable files (EXE). This tool simplifies the process of packaging Python applications for distribution on Windows platforms, ensuring ease of installation and usage by end-users.

3.4. Technology Used

1. Flask Framework (Python):

Flask is a lightweight and flexible micro web framework for Python. It provides the tools and libraries needed to build web applications quickly and efficiently.

```
from flask import Flask, render_template, request, jsonify, redirect, url_for
```

2. HTML/CSS/JavaScript:

HTML, CSS, and JavaScript are essential technologies for building the front-end of web applications, providing structure, styling, and interactivity.

```
<!DOCTYPE html>  
<html lang="en">  
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Cyber Blade</title>
<!-- External CSS and JavaScript libraries -->
</head>
<body>
<!-- HTML content -->
</body>
</html>
```

3. Bootstrap Framework:

Bootstrap is a popular front-end framework for building responsive and mobile-first websites. It offers pre-designed components and layouts to streamline UI development.

```
<!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

4. jQuery Library:

jQuery is a fast, small, and feature-rich JavaScript library. It simplifies client-side scripting and enhances interactivity in web applications.

```
<!-- jQuery -->
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
```

6. Requests Library (Python):

The Requests library is used to send HTTP requests to external APIs. It simplifies the process of making HTTP calls and handling responses.

```
import requests
```

7. Concurrent Futures Module (Python):

The Concurrent Futures module provides a high-level interface for asynchronously executing callable. It is used for parallelizing tasks and improving performance.

```
import concurrent.futures
```

3.5. Project Specification:

1. URL Scanner:

- **Functionality:** Checks the maliciousness of a given URL using the VirusTotal API.
- **Code Example:**

```
def check_url_malicious(url, api_key):  
    endpoint = 'https://www.virustotal.com/vtapi/v2/url/report'params =  
    {'apikey': api_key, 'resource': url}  
    response = requests.get(endpoint, params=params)  
    result = response.json()  
    # Process the result and return appropriate response
```

2. File Scanner:

- **Functionality:** Scans uploaded files for malware using the VirusTotal API.
- **Code Example:**

```
def scan_file_with_virustotal(file):  
    url = 'https://www.virustotal.com/vtapi/v2/file/scan'  
    files = {'file': file}  
    params = {'apikey': VIRUSTOTAL_API_KEY}  
    response = requests.post(url, files=files, params=params)  
    result = response.json()  
  
    # Process the result and return appropriate response
```

3. Vulnerable Directory Finder:

- **Functionality:** Searches for vulnerable directories on a given website.
- **Code Example:**

```
def dirb(method, url, num_threads=50):  
    with  
        concurrent.futures.ThreadPoolExecutor(max_workers=num_threads) as  
            executor:  
        # Submit tasks for parallel execution  
        futures = [executor.submit(check_directory, method, url, directory) for  
                    directory in directories]  
        for future in concurrent.futures.as_completed(futures):  
            # Process the results asynchronously
```

4. User Interface:

Provides a user-friendly interface for interacting with the application.

Uses HTML/CSS/JavaScript along with Flask templates for rendering dynamic content.

```
<!-- Example: HTML form for submitting a URL -->
<form id="urlForm" action="/check_url" method="post">
  <label for="url">Enter URL:</label>
  <input type="text" id="url" name="url" class="form-control" required>
  <input type="submit" value="Check" class="btn btn-primary">
</form>
```

5. Real-Time Progress Monitoring:

Displays real-time progress updates during directory scanning.

Utilizes AJAX to fetch progress data from the server asynchronously.

```
// Fetch progress messages every 2 seconds
var intervalId = setInterval(fetchProgress, 2000);
```

6. Error Handling:

Implements robust error handling mechanisms to handle unforeseen issues gracefully and provide informative error messages to the user.

```
except requests.RequestException as e:
    # Handle request-related errors and provide feedback to the user
```

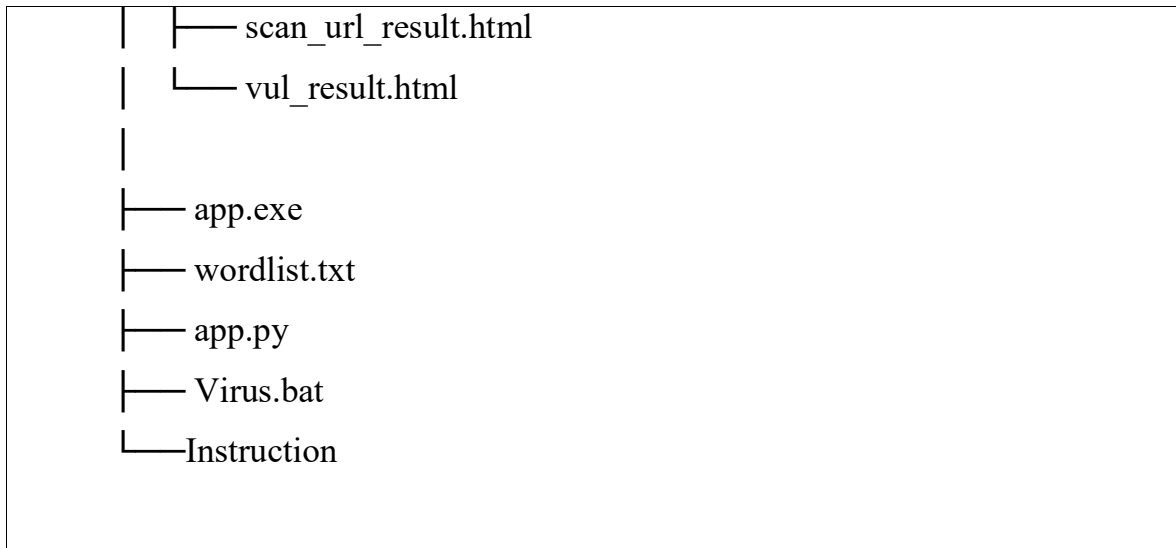
4. Code Snippets

4.1. Source Code File Structure

- **static/:** Contains static files such as CSS stylesheets, JavaScript files, and images used in the web application.
- **templates/:** Holds HTML templates used by Flask to render dynamic content.
- **index.html:** Template for the main page of the application.
- **scan_file_result.html:** Template for displaying results of file scanning.
- **scan_url_result.html:** Template for displaying results of URL scanning.
- **vul_result.html:** Template for displaying results of vulnerable directory scanning.
- **wordlist.txt:** Text file containing a list of common directory names used for directory scanning.
- **app.py:** Main Python file containing the Flask application code and server-side logic.
- **Instruction:** Documentation file providing information about the project, its setup, and usage instructions.

CYBER_BLADE_PROJECT/

```
|
|
|— static/
|   |
|   |— logo.png
|   |— image2.jpg
|   |— phish.png
|   |— malicious.png
|
|— templates/
|   |— index.html
|   |— scan_file_result.html
```



Name	Date modified	Type	Size
static	4/26/2024 2:47 AM	File folder	
templates	4/25/2024 4:30 AM	File folder	
app	4/26/2024 2:14 AM	Application	14,046 KB
app	4/26/2024 2:11 AM	Python Source File	7 KB
Instructions---	4/26/2024 2:47 AM	Text Document	1 KB
virus	4/24/2024 2:19 PM	Windows Batch File	1 KB
wordlist	4/23/2024 11:31 PM	Text Document	3,250 KB

4.2. Codes

- **App.py (This is main code of the project that will work in backend of website)**

```
from flask import Flask, render_template, request, jsonify, redirect, url_for
import requests
import concurrent.futures
import os
```



```

app = Flask(__name__, template_folder='./templates',
static_folder='./static')

# Global variables to store progress messages and stop process flag
progress_messages = []

stop_process = False

# Define VirusTotal API key
VIRUSTOTAL_API_KEY =
'658b2d99745d227804de1ce8e498e7e6d8e5c7325167d71c37315ca7d1c
6263a'

# URL Scanner functionality
def check_url_malicious(url, api_key):

    endpoint = 'https://www.virustotal.com/vtapi/v2/url/report'

    params = {'apikey': api_key, 'resource': url}

    response = requests.get(endpoint, params=params)

    result = response.json()

    if response.status_code == 200:

        if result['response_code'] == 1:

            detections = result['scans']

            detected_vendors = []

```

```

        for vendor_name, detection in detections.items():

            if detection['detected']:

                detected_vendors.append((vendor_name, detection['result']))

            return render_template('scan_url_result.html',
detected_vendors=detected_vendors)

        else:

            return "Error: " + result['verbose_msg']

    else:

        return "Failed to retrieve results. Status code: " +
str(response.status_code)

# File Scanner functionality

def scan_file_with_virustotal(file):

    url = 'https://www.virustotal.com/vtapi/v2/file/scan'

    files = {'file': file}

    params = {'apikey': VIRUSTOTAL_API_KEY}

    response = requests.post(url, files=files, params=params)

    result = response.json()

    return result

def get_file_scan_report(resource):

    url = 'https://www.virustotal.com/vtapi/v2/file/report'

    params = {'apikey': VIRUSTOTAL_API_KEY, 'resource': resource}

```

```

response = requests.get(url, params=params)

result = response.json()

return result

# Vulnerable Directory Finder functionality
def check_directory(method, url, directory):

    global stop_process

    if stop_process:

        return

    if not url.endswith('/'):

        url += '/'

    target_url = url + directory

    headers = {

        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.3'

    }

    try:

        response = requests.request(method, target_url, headers=headers,
timeout=5) # Timeout after 5 seconds

        if response.status_code == 200:

```

```

        progress_messages.append({
            'method': method,
            'message': f"Directory found: <a href='{target_url}' style='color: green;' target='_blank'>{target_url}</a>",
            'status': 'success',
            'url': target_url,
            'response_code': response.status_code
        })
    else:
        progress_messages.append({
            'method': method,
            'message': f"Checking directory: {target_url} - Response code: {response.status_code}",
            'status': 'info',
            'url': target_url,
            'response_code': response.status_code
        })
except requests.RequestException as e:
    progress_messages.append({
        'method': method,
        'message': f"Failed to check directory {target_url}: {e}",
        'status': 'danger',
        'url': target_url,

```

```

        'response_code': 'N/A'

    })

def dirb(method, url, num_threads=50):

    global stop_process

    # Get the directory of the current Python script
    script_dir = os.path.dirname(__file__)

    # Define the path to the wordlist file
    wordlist_path = os.path.join(script_dir, 'wordlist.txt')

    with open(wordlist_path, 'r') as f:

        directories = [line.strip() for line in f]

    with
    concurrent.futures.ThreadPoolExecutor(max_workers=num_threads) as
    executor:

        futures = [executor.submit(check_directory, method, url, directory) for
        directory in directories]

        for future in concurrent.futures.as_completed(futures):

            pass # Wait for all tasks to complete

            if stop_process:

                break

```

```

# Routes for different functionalities

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

@app.route('/check_url', methods=['POST'])
def check_url():
    url = request.form['url']

    result = check_url_malicious(url, VIRUSTOTAL_API_KEY)

    return result

@app.route('/scan_file', methods=['POST'])
def scan_file():
    file = request.files['file']

    if file:
        scan_result = scan_file_with_virustotal(file)

        if scan_result.get('response_code') == 1:
            resource = scan_result.get('resource')
            report_result = get_file_scan_report(resource)

            if report_result.get('positives', 0) > 0:
                detections = report_result.get('scans', {})

                detected_vendors = [(vendor, data.get('result', '')) for vendor,
data in detections.items() if data.get('detected')]

```

```

        return render_template('scan_file_result.html',
detected_vendors=detected_vendors)

    else:

        return "The file is clean."

    else:

        return "Failed to scan the file."

else:

    return "No file uploaded"

@app.route('/vul_scanner', methods=['POST'])
def vul_scanner():

    global progress_messages, stop_process

    url = request.form['url']

    if not url.endswith('/'):

        url += '/'

    num_threads = int(request.form['threads'])

    progress_messages = [] # Reset progress messages

    stop_process = False

    method = request.form['method'] # Get the selected HTTP method

    dirb(method, url, num_threads)

    return redirect(url_for('vul_result')) # Redirect to the result page after
processing

```

```

@app.route('/progress')
def get_progress():
    global progress_messages
    return jsonify(progress_messages)

@app.route('/stop')
def stop():
    global stop_process
    stop_process = True
    return redirect(url_for('vul_result')) # Redirect to the result page after
stopping the process

@app.route('/vul_result')
def vul_result():
    return render_template('vul_result.html', progress=progress_messages)

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

```


- **HTML code {index.html} : This is main code of project the will working on frontend of a website**

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Cyber Blade</title>
  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min
.css">
  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css">

  <style>
    body {
      font-family: ui-sans-serif;
      background: linear-gradient(to right, #12c2e9, #c471ed, #f64f59);
      background-image: url('/static/image2.jpg');
      background-size: cover;
      background-repeat: no-repeat;
```

```
background-attachment: fixed;

margin: 0;

padding: 20px;

color: #121481; /* Text color */
}

.container {

background-color: #f2f8ec36;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 85px rgb(0 0 0);

margin-top: 20px;

}

.nav-link {

color: #121481; /* Nav link color */

font-weight: bold;

animation: pulse 3s infinite alternate;

}

.nav-tabs .nav-item.show .nav-link, .nav-tabs .nav-link.active {

color: #121481; /* Active tab text color */

background-color: #8abfec; /* Active tab background color */

border-color: transparent;

border-radius: 8px;
```

```

}

.nav-tabs .nav-link {

    border: 1px solid transparent;

}

.nav-tabs .nav-link:hover {

    background-color: #807dd8c9; /* Hovered tab background color */

    color: #121481; /* Hovered tab text color */

    border-color: transparent;

    border-radius: 8px;

}

.nav-tabs {

    background-color: aliceblue;

    border-radius: 8px;

    width: fit-content;

}

.tab-pane {

    padding: 20px;

    border-radius: 8px;

    box-shadow: -7px 0px 20px 16px rgba(0, 0, 0, 0.3);

    background: linear-gradient(to right, #12c2e9c5, #c471ed73,
#f64f597d);

}

```

```
.form-control {  
    background-color: #ffeae3; /* Form input background color */  
    color: #121481; /* Form input text color */  
    border-color: #111277; /* Form input border color */  
}  
  
input#file {  
    padding: 3.5px;  
}  
  
input.btn.btn-primary {  
    background: #121481; /* Primary button background color */  
    color: #fff; /* Primary button text color */  
    border-color: #ffb1b1; /* Primary button border color */  
    margin: 10px 0px 0px 0px;  
}  
  
.btn-danger {  
    background-color: #ff8484; /* Danger button background color */  
    border-color: #ff8484; /* Danger button border color */  
}  
  
.btn-danger:hover {  
    background-color: #ff5454; /* Hovered danger button background  
color */  
    border-color: #ff5454; /* Hovered danger button border color */  
}
```

```

}

.table {

    background-color: #ffeae3; /* Table background color */

    color: #121481; /* Table text color */

}

.h1, h1 {

    font-size: 2.5rem;

    color: #111277; /* Heading color */

}

img {

    vertical-align: middle;

    border-style: none;

    width: 25%;

    animation: pulse 3s infinite alternate;

}

label {

    display: inline-block;

    margin-bottom: .5rem;

    font-weight: 600;

}

@keyframes pulse {

    0% {

```

```

        transform: scale(1);

    }

    100% {

        transform: scale(1.05);

    }

}

</style>
</head>
<body>

    <div class="container">

        <center>

        <!-- Nav tabs -->

        <ul class="nav nav-tabs" role="tablist">

            <li class="nav-item">

                <a class="nav-link active" data-toggle="tab"
href="#urlScanner"><i class="fas fa-link"></i> URL Scanner</a>

            </li>

            <li class="nav-item">

                <a class="nav-link" data-toggle="tab" href="#securityScanner"><i
class="fas fa-file-alt"></i> File Scanner</a>

```

```

</li>

<li class="nav-item">

    <a class="nav-link" data-toggle="tab" href="#dirFinder"><i
class="fas fa-search"></i> Vulnerable Directory Finder</a>

</li>

</ul>

</center>

<!-- Tab panes -->

<div class="tab-content">

    <!-- URL Scanner Tab -->

    <div id="urlScanner" class="container tab-pane active"><br>

        <h1><i class="fas fa-link"></i> URL Scanner</h1>

        <form id="urlForm" action="/check_url" method="post">

            <label for="url">Enter URL:</label>

            <input type="text" id="url" name="url" class="form-control"
required>

            <input type="submit" value="Check" class="btn btn-primary">

        </form>

        <div id="result"></div>

    </div>

    <!-- Security Scanner Tab -->

    <div id="securityScanner" class="container tab-pane fade"><br>

```

```

<h1><i class="fas fa-file-alt"></i> File Scanner</h1>

<form id="securityForm" action="/scan_file" method="post"
enctype="multipart/form-data">

    <label for="file">Upload File:</label>

    <input type="file" id="file" name="file" class="form-control"
required>

    <input type="submit" value="Scan" class="btn btn-primary">

</form>

<div id="result"></div>

</div>

<!-- Vulnerable Directory Finder Tab -->

<div id="dirFinder" class="container tab-pane fade"><br>

    <h1><i class="fas fa-search"></i> Vulnerable Directory
Finder</h1>

    <form id="dirForm" method="POST" action="/vul_scanner">

        <div class="form-group">

            <label for="url">URL:</label>

            <input type="text" class="form-control" id="url" name="url"
required>

        </div>

        <div class="form-group">

            <label for="threads">Number of Threads:</label>

```



```

        <input type="number" class="form-control" id="threads"
name="threads" value="10" required>

    </div>

    <select name="method">

        <option value="GET">GET</option>

        <option value="POST">POST</option>

        <!-- Add other HTTP methods as needed -->

    </select>

    <button style="background-color: #121481;" type="submit"
class="btn btn-primary">Start Finding</button>

    <button style="background-color: #da1b1b;" type="button"
class="btn btn-danger" id="stopBtn"><i class="fas fa-stop"></i>
Stop</button>

</form>

<hr>

<div id="progressTable" class="table-responsive">

    <table class="table table-bordered table-striped">

        <thead>

            <tr>

                <th>S.No</th>

                <th>Method</th>

                <th>Target URL</th>

                <th>Response Code</th>

```

```

        <th>Dir Found</th>

    </tr>

</thead>

<tbody id="progressBody">

    {% for i in range(progress|length) %}

        <tr class="{% if progress[i].status == 'success' %}table-
success{% elif progress[i].status == 'info' %}table-info{% else %}table-
danger{% endif %}">

            <td>{{ i + 1 }}</td>

            <td>{{ progress[i].method }}</td>

            <td><a href="{{ progress[i].url }}" target="_blank">{{
progress[i].url }}</a></td>

            <td>{{ progress[i].response_code }}</td>

            <td>{% if progress[i].status == 'success' %}✓{% else
%}✗{% endif %}</td>

        </tr>

    {% endfor %}

</tbody>

</table>

</div>

</div>

</div>

</div>

```

```

<!-- jQuery -->

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<!-- Bootstrap JS -->

<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js
"></script>

<script>

    $(document).ready(function() {

        // Function to fetch progress messages from server

        function fetchProgress() {

            $.get('/progress', function(data) {

                $('#progressTable tbody').empty();

                for (var i = 0; i < data.length; i++) {

                    var message = data[i];

                    var rowClass = message.status === 'success' ? 'table-
success' : (message.status === 'info' ? 'table-info' : 'table-danger');

                    var rowHtml = '<tr class="' + rowClass + '"><td>' + (i + 1) +
'</td><td>' + message.method + '</td><td><a href="' + message.url + '"
target="_blank">' + message.url + '</a></td><td>' +
message.response_code + '</td><td>' + (message.status === 'success' ?
'✓' : '✗') + '</td></tr>';

                    $('#progressTable tbody').append(rowHtml);

                }
            });
        }
    });

```

```

    });

}

// Fetch progress messages every 2 seconds
var intervalId = setInterval(fetchProgress, 2000);

// Stop button click handler
$('#stopBtn').click(function() {
    $.get('/stop', function() {
        clearInterval(intervalId);
    });
});

});

});

</script>
</script>
</body>
</html>

```

- **HTML code {scan_file_result.html} :** This is main code of project the will working on frontend of a website and show the result of scan files.

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>File Scan Result</title>
<style>
  body {
    font-family: ui-sans-serif;
    background: linear-gradient(to right, #12c2e9, #c471ed, #f64f59);
    background-image: url('/static/image2.jpg');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    margin: 0;
    padding: 20px;
    color: #121481; /* Text color */
  }
  .container {
    max-width: 800px;
    margin: 20px auto;
    background-color: #f2f8ec96;

    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 85px rgb(0 0 0);
    margin-top: 20px;
  }
  h2 {
    font-size: 2rem;
    color: #111277; /* Heading color */
    margin-top: 0;

```

```

border-bottom: 2px solid #111277;
padding-bottom: 10px;
}
table {
width: 100%;
border-collapse: collapse;
border: 1px solid #ddd;
margin-top: 20px;
box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
background: linear-gradient(to right, #12c2e9c5, #c471ed73,
#f64f597d);
}
th, td {
padding: 12px;
text-align: left;
border-bottom: 1px solid #ddd;
}
th {
background-color: #8abfec; /* Active tab background color */
color: #121481; /* Active tab text color */
border-color: transparent;
}
tr:hover {
background-color: #807dd8c9; /* Hovered tab background color */
color: #121481; /* Hovered tab text color */
border-color: transparent;
border-radius: 8px;
}
.detection-result {
font-weight: bold;

```

```

    }
    .malicious-logo {
        width: 20px;
        height: auto;
        vertical-align: middle;
        margin-right: 5px;
    }
    img.logo{
        width: 25%;
    }
</style>
</head>
<body>
    <div class="container">
        <center>
        <h2>File Scan Result</h2>
        <table>
            <thead>
                <tr>
                    <th>S.No.</th>
                    <th>Security Vendor</th>
                    <th>Detection Result</th>
                </tr>
            </thead>
            <tbody>
                {% for i in range(detected_vendors|length) %}
                <tr>
                    <td>{{ i + 1 }}</td>
                    <td>{{ detected_vendors[i][0] }}</td>

```

```

        <td class="detection-result">
            {% if detected_vendors[i][1] %}
                
                {{ detected_vendors[i][1] }}
            {% else %}
                No detection
            {% endif %}
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>
</div>
</body>
</html>

```

- **HTML code {scan_url_result.html} :** This is main code of project the will working on frontend of a website and show the result of Url Scan.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>URL Scan Result</title>
    <style>

```



```
body {
  font-family: ui-sans-serif;
  background: linear-gradient(to right, #12c2e9, #c471ed, #f64f59);
  background-image: url('/static/image2.jpg');
  background-size: cover;
  background-repeat: no-repeat;
  background-attachment: fixed;
  margin: 0;
  padding: 20px;
  color: #121481; /* Text color */
}

.container {
  max-width: 800px;
  margin: 20px auto;
  background-color: #f2f8ec96;

  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 0 85px rgb(0 0 0);
  margin-top: 20px;
}

h2 {
  font-size: 2rem;
  color: #111277; /* Heading color */
  margin-top: 0;
  border-bottom: 2px solid #111277;
  padding-bottom: 10px;
}

table {
  width: 100%;
```

```

border-collapse: collapse;
border: 1px solid #ddd;
margin-top: 20px;
box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
background: linear-gradient(to right, #12c2e9c5, #c471ed73,
#f64f597d);
}
th, td {
padding: 12px;
text-align: left;
border-bottom: 1px solid #ddd;
}
th {
background-color: #8abfec; /* Active tab background color */
color: #121481; /* Active tab text color */
border-color: transparent;
}
tr:hover {
background-color: #807dd8c9; /* Hovered tab background color */
color: #121481; /* Hovered tab text color */
border-color: transparent;
border-radius: 8px;
}
.detection-result {
font-weight: bold;
}
.malicious-logo {
width: 20px;
height: auto;
vertical-align: middle;

```

```

        margin-right: 5px;
    }

    .phishing-logo {
width: 20px;
height: auto;
vertical-align: middle;
margin-right: 5px;
}

    img.logo{
        width: 25%;
    }
</style>
</head>
<body>
    <div class="container">
        <center>
        <h2>URL Scan Result</h2>
        <div class="background">
            <table>
                <thead>
                    <tr>
                        <th>S.No.</th>
                        <th>Security Vendor</th>
                        <th>Positive Detection</th>
                    </tr>
                </thead>
                <tbody>
                    {% for i in range(detected_vendors|length) %}
                        <tr>

```

```

<td>{{ loop.index }}</td>
<td>{{ detected_vendors[i][0] }}</td>
<td class="positive-detection">
    {% if "malicious" in detected_vendors[i][1] %}
        
    {% elif "phishing" in detected_vendors[i][1] %}
        
    {% endif %}
    {{ detected_vendors[i][1] }}
</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>
</html>

```

- **HTML code {vul_result.html} :** This is man code of project the will working on frontend of a website and this show the result of vulnerable directory from URL.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

<title>Vulnerable Directory Finder Result</title>

<!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.m
in.css">

<!-- jQuery -->
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<!-- Bootstrap JS -->
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.j
s"></script>

<style>
  body {
    font-family: ui-sans-serif;
    background: linear-gradient(to right, #12c2e9, #c471ed, #f64f59);
    background-image: url('/static/image2.jpg');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    margin: 0;
    padding: 20px;
    color: #121481; /* Text color */
  }
  .container {
    max-width: 800px;
    margin: 20px auto;
    background-color: #f2f8ec96;

```

```

padding: 20px;
border-radius: 10px;
box-shadow: 0 0 85px rgb(0 0 0);
margin-top: 20px;
}
img.logo{
width: 25%;
}
h1 {
font-size: 2rem;
color: #111277; /* Heading color */
margin-top: 0;
border-bottom: 2px solid #111277;
padding-bottom: 10px;
}
table {
width: 100%;
border-collapse: collapse;
border: 1px solid #ddd;
margin-top: 20px;
box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
background: linear-gradient(to right, #12c2e9c5, #c471ed73,
#f64f597d);
}
th, td {
padding: 12px;
text-align: left;
border-bottom: 1px solid #ddd;
}
th {

```

```

        background-color: #8abfec; /* Active tab background color */
        color: #121481; /* Active tab text color */
        border-color: transparent;
    }
    tr:nth-child(even) {
        background-color: #f2f2f2;
    }
    .btn-primary {
        margin-top: 20px;
        background-color: #007bff;
        border-color: #007bff;
    }
    .btn-primary:hover {
        background-color: #0056b3;
        border-color: #0056b3;
    }
    .modal-dialog {
        max-width: 400px;
    }
    .modal-body {
        font-weight: bold;
    }
</style>
</head>
<body>
    <div class="container">
        <center>
        <h1>Vulnerable Directory Finder Result</h1>
        <div class="table-responsive">

```

```

<table class="table table-bordered table-striped">
  <thead>
    <tr>
      <th>S.No</th>
      <th>Method</th>
      <th>Target URL</th>
      <th>Response Code</th>
      <th>Dir Found</th>
    </tr>
  </thead>
  <tbody>
    {% for i in range(progress|length) %}
      {% if progress[i].status == 'success' %}
        <tr>
          <td>{{ i + 1 }}</td>
          <td>{{ progress[i].method }}</td>
          <td><a href="{{ progress[i].url }}" target="_blank">{{
progress[i].url }}</a></td>
          <td>{{ progress[i].response_code }}</td>
          <td>✓</td>
        </tr>
      {% endif %}
    {% endfor %}
  </tbody>
</table>
</div>
</div>

<!-- Modal -->

```



```

<div class="modal fade" id="confirmModal" tabindex="-1" role="dialog"
aria-labelledby="confirmModalLabel" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title"
id="confirmModalLabel">Confirmation</h5>
        <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <!-- <div class="modal-body">
        Do you want to brute force vulnerable
directories?<br><strong>This action may have serious consequences.
Proceed with caution.</strong>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-
dismiss="modal">No</button>
        <button type="button" class="btn btn-primary"
id="startBruteForceBtn">Yes</button>
      </div> -->
    </div>
  </div>
</div>

<!-- Custom JavaScript -->
<!-- <script>
  $(document).ready(function() {

```

```
// Delay modal display after 10 seconds
setTimeout(function() {
    $('#confirmModal').modal('show');
}, 10000);
// Handle start button click
$('#startBruteForceBtn').click(function() {
    window.location.href = "/"; // Redirect to the homepage to start
the process
});
});
</script> -->
</body>
</html>
```

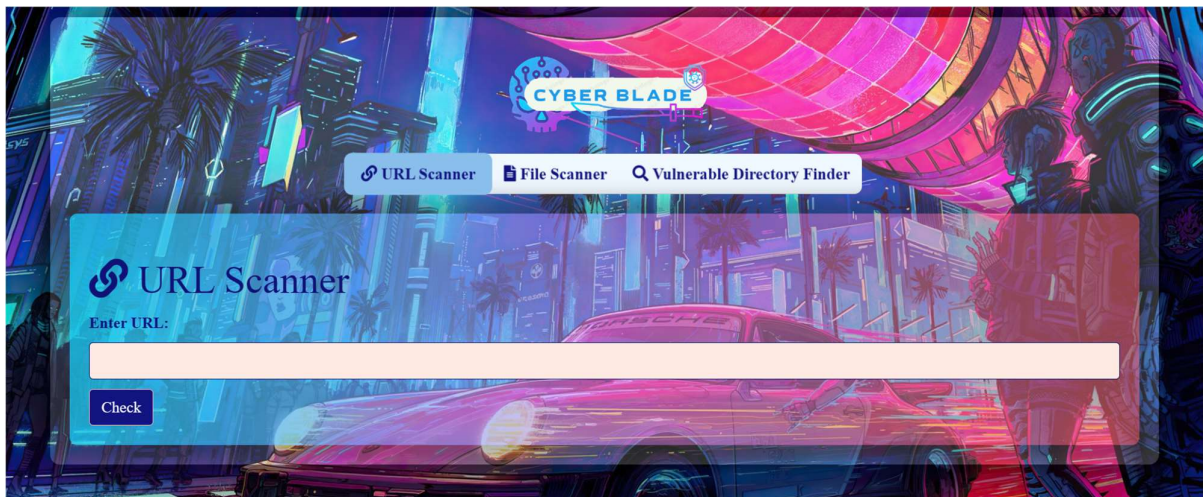
5. Result and Testing

5.1. Screenshots

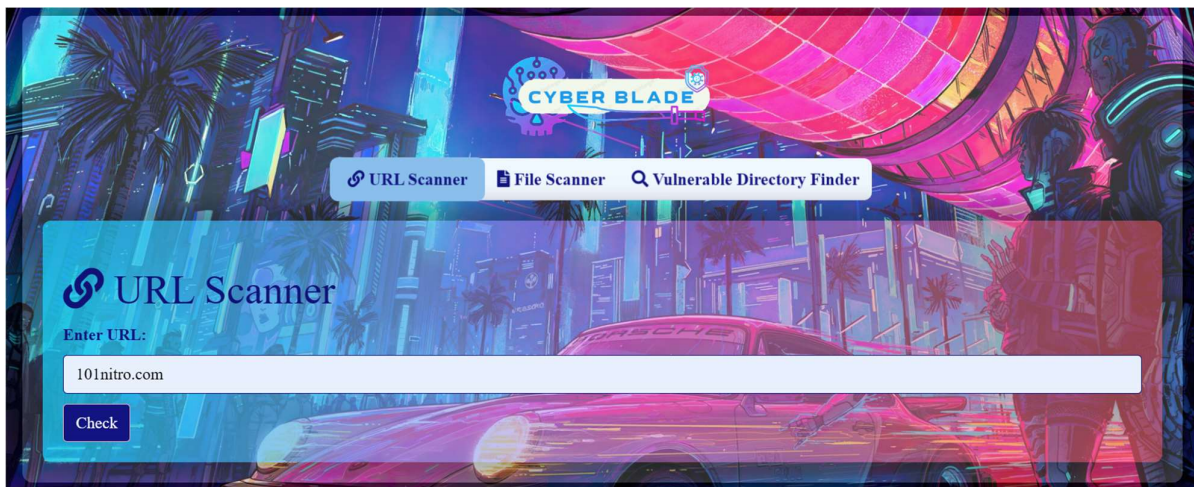
- Run the app.exe

```
C:\Users\akash\Downloads\Sti x + v
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
```

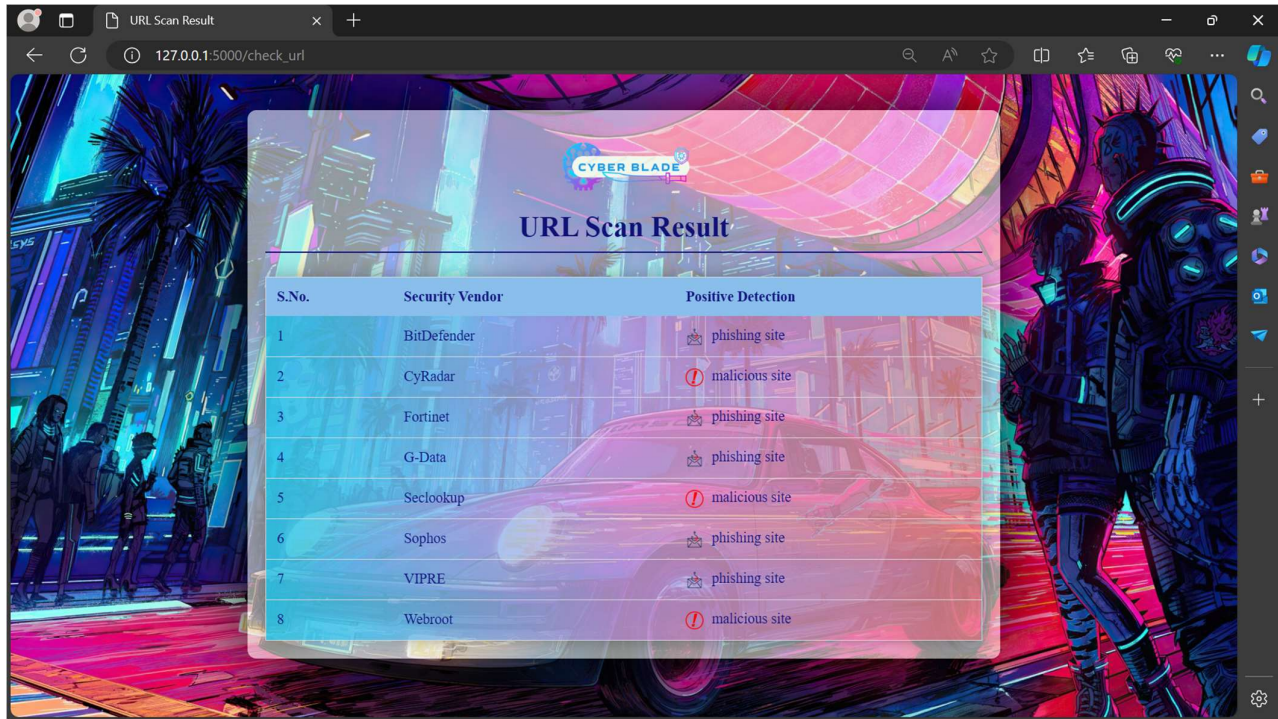
- Click on the <http://127.0.0.1:5000>



1. Now , we can Scan the URL

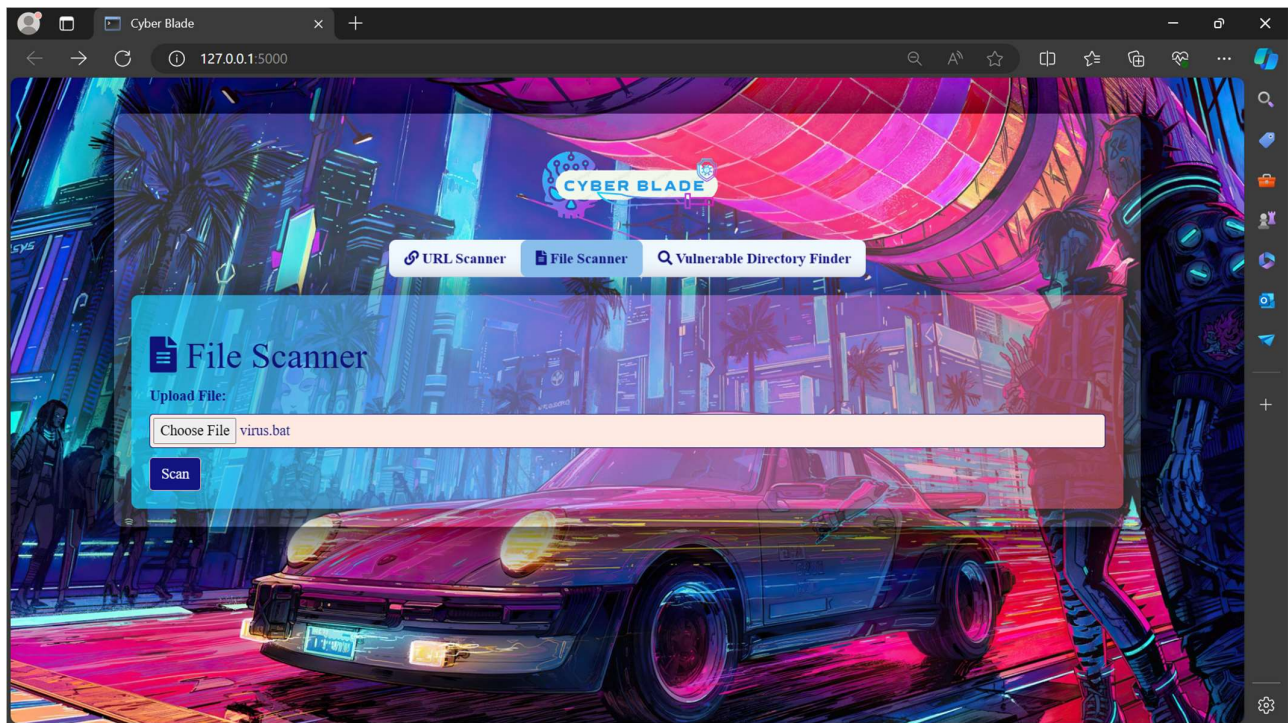


Result:

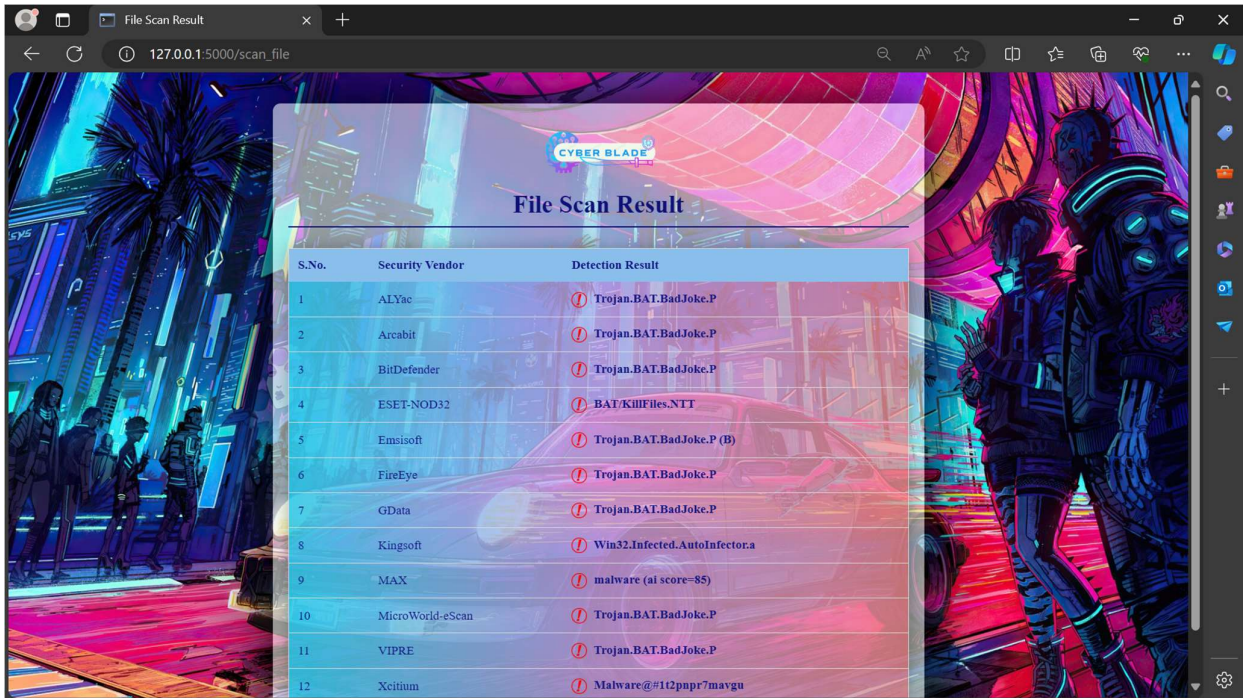


S.No.	Security Vendor	Positive Detection
1	BitDefender	phishing site
2	CyRadar	malicious site
3	Fortinet	phishing site
4	G-Data	phishing site
5	Seclookup	malicious site
6	Sophos	phishing site
7	VIPRE	phishing site
8	Webroot	malicious site

2. Now we scan a Malicious File. I already upload virus file.

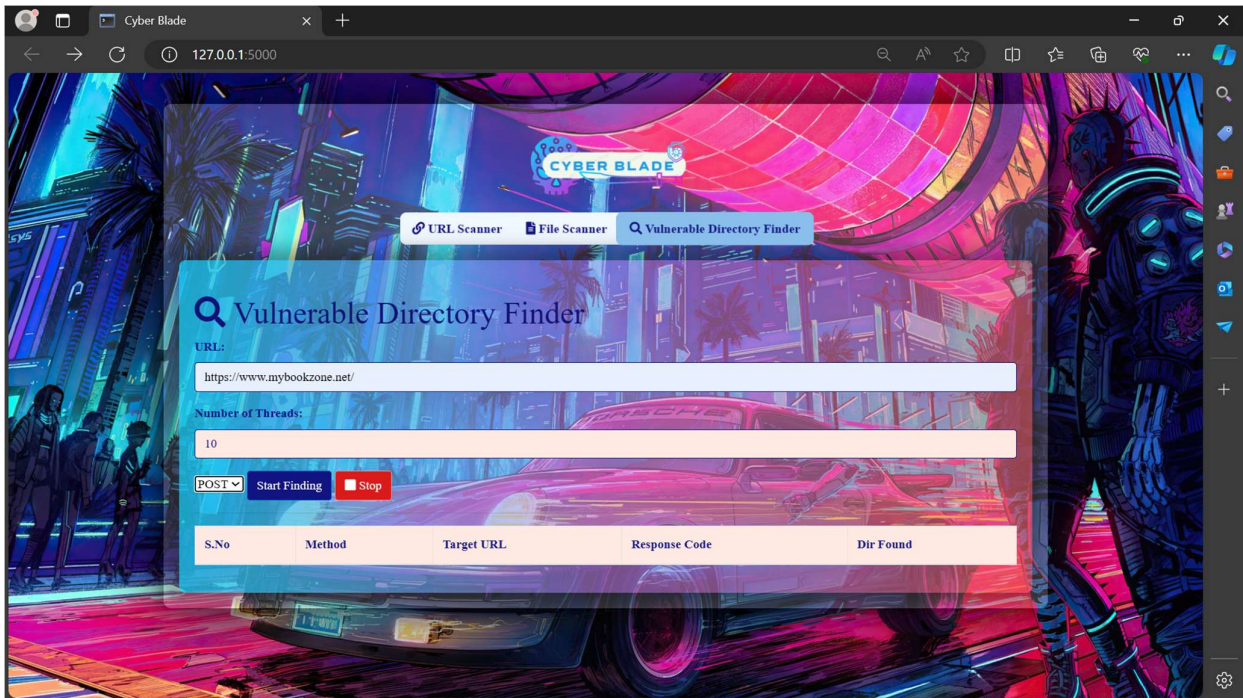


Result:



S.No.	Security Vendor	Detection Result
1	ALYac	Trojan.BAT.BadJoke.P
2	Arcabit	Trojan.BAT.BadJoke.P
3	BitDefender	Trojan.BAT.BadJoke.P
4	ESET-NOD32	BAT/KillFiles.NIT
5	Emsisoft	Trojan.BAT.BadJoke.P (B)
6	FireEye	Trojan.BAT.BadJoke.P
7	GData	Trojan.BAT.BadJoke.P
8	Kingsoft	Win32/Infected.AutoInfector.a
9	MAX	malware (ai score=85)
10	MicroWorld-eScan	Trojan.BAT.BadJoke.P
11	VIPRE	Trojan.BAT.BadJoke.P
12	Xcitium	Malware@#1t2pnr7mavgu

3. Now, we find hidden or vulnerable directory from URL. I already input the website.



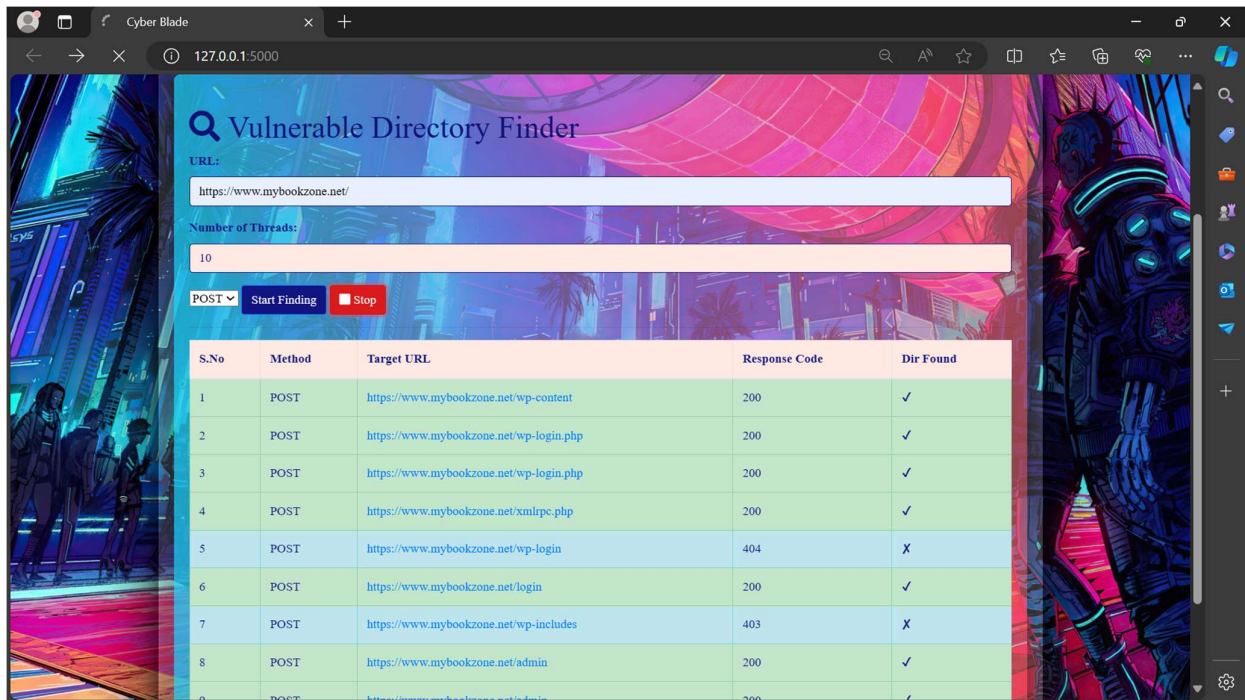
Vulnerable Directory Finder

URL:

Number of Threads:

S.No	Method	Target URL	Response Code	Dir Found
------	--------	------------	---------------	-----------

Result:



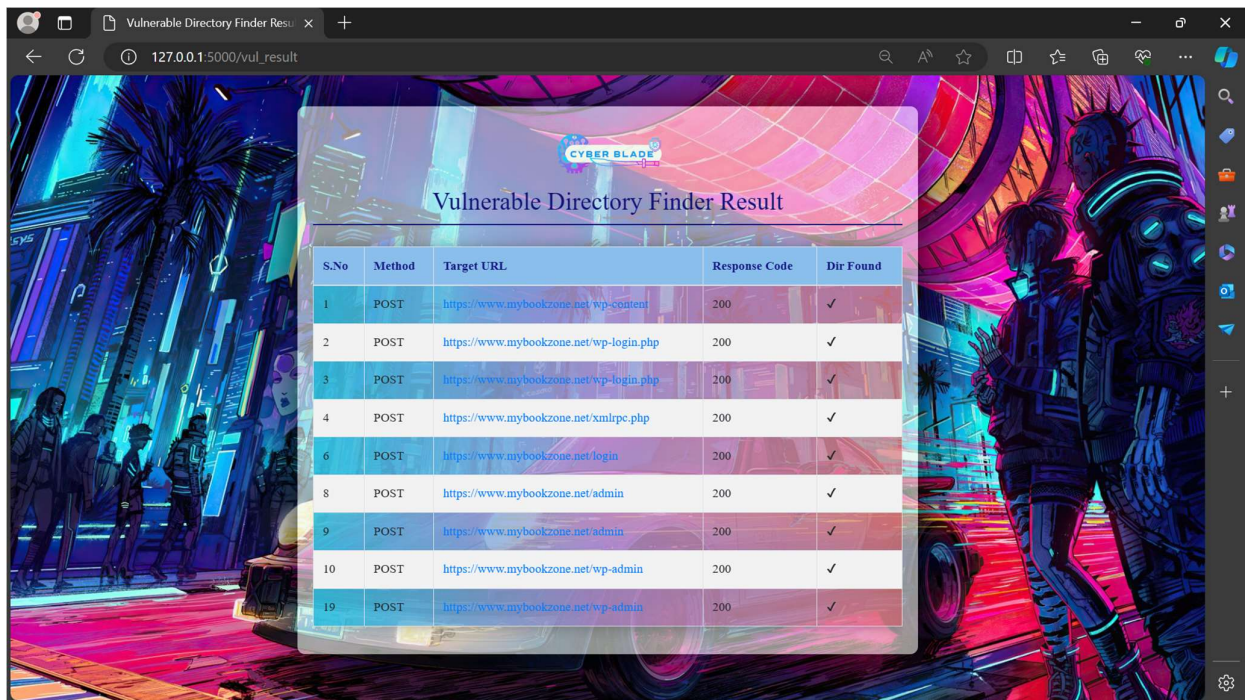
Vulnerable Directory Finder

URL:

Number of Threads:

POST

S.No	Method	Target URL	Response Code	Dir Found
1	POST	https://www.mybookzone.net/wp-content	200	✓
2	POST	https://www.mybookzone.net/wp-login.php	200	✓
3	POST	https://www.mybookzone.net/wp-login.php	200	✓
4	POST	https://www.mybookzone.net/xmlrpc.php	200	✓
5	POST	https://www.mybookzone.net/wp-login	404	✗
6	POST	https://www.mybookzone.net/login	200	✓
7	POST	https://www.mybookzone.net/wp-includes	403	✗
8	POST	https://www.mybookzone.net/admin	200	✓



Vulnerable Directory Finder Result

S.No	Method	Target URL	Response Code	Dir Found
1	POST	https://www.mybookzone.net/wp-content	200	✓
2	POST	https://www.mybookzone.net/wp-login.php	200	✓
3	POST	https://www.mybookzone.net/wp-login.php	200	✓
4	POST	https://www.mybookzone.net/xmlrpc.php	200	✓
6	POST	https://www.mybookzone.net/login	200	✓
8	POST	https://www.mybookzone.net/admin	200	✓
9	POST	https://www.mybookzone.net/admin	200	✓
10	POST	https://www.mybookzone.net/wp-admin	200	✓
19	POST	https://www.mybookzone.net/wp-admin	200	✓

5.2. Conclusion

In conclusion, the Cyber Blade website represents a pivotal component within your project, offering advanced cybersecurity capabilities to users and organizations. Through integration with VirusTotal's API, Flask, Requests, and other Python libraries, Cyber Blade empowers users to detect, analyze, and mitigate cyber threats effectively. The website serves as a central hub for accessing powerful cybersecurity tools, enhancing threat awareness, and streamlining security operations

Looking ahead, the future scope of the Cyber Blade website is promising, with opportunities for expansion and innovation in various areas. These include advanced threat intelligence integration, machine learning and AI enhancements, automation and orchestration capabilities, and integration with cloud security services. Additionally, there is potential for enhanced user experience, compliance and governance features, global expansion, and ecosystem development through strategic partnerships.

Overall, the Cyber Blade website holds significant potential to evolve into a leading platform for cybersecurity innovation, empowering users to defend against evolving cyber threats and safeguard their digital assets effectively. By embracing these opportunities for future development and enhancement, Cyber Blade can continue to deliver tangible value to users and organizations in the ever-changing landscape of cybersecurity.

5.3. Significance and Impact

- a) **Enhanced Cybersecurity Awareness:** By providing a user-friendly interface through the Cyber Blade website, individuals and organizations gain access to powerful cybersecurity tools and resources. This fosters greater awareness of

cyber threats and empowers users to take proactive measures to protect their digital assets.

- b) Improved Threat Detection and Mitigation:** Through integration with the VirusTotal API and other cybersecurity tools, the Cyber Blade website enables users to scan files and URLs for potential threats. This facilitates early detection and mitigation of malware, phishing attempts, and other cyber attacks, thereby reducing the risk of data breaches and financial losses.
- c) Streamlined Security Operations:** The website's functionality allows users to efficiently manage their cybersecurity operations from a centralized platform. They can easily submit files or URLs for scanning, access detailed analysis reports, and take appropriate actions based on the findings. This streamlines security processes and enhances overall operational efficiency.
- d) Empowerment of Users:** By offering access to advanced cybersecurity tools and resources, the Cyber Blade website empowers users to take control of their digital security. They can make informed decisions, implement best practices, and proactively defend against evolving cyber threats, thereby reducing their reliance on external cybersecurity providers.
- e) Business Value and Reputation:** For businesses offering Cyber Blade as a service or product, the website serves as a key touchpoint for customers. A well-designed and functional website enhances the perceived value of the offering and contributes to a positive brand reputation. It can attract more users, drive customer loyalty, and ultimately increase revenue and market share.
- f) Compliance and Regulatory Alignment:** For organizations operating in regulated industries, such as finance or healthcare, the Cyber Blade website can assist in meeting compliance requirements. By providing tools for malware

detection, data protection, and security incident response, the website supports organizations in aligning with industry regulations and standards.

5.4. Future Scope

The future scope of the Cyber Blade website within your project holds immense potential for expansion and innovation in the field of cybersecurity. Here are some avenues for future development and enhancement:

- **Advanced Threat Intelligence Integration:** Continuously integrate with additional threat intelligence sources beyond VirusTotal to enhance the breadth and depth of threat detection capabilities. Partner with leading cybersecurity firms and research organizations to access cutting-edge threat intelligence feeds and expand the scope of Cyber Blade's threat detection capabilities.
- **Machine Learning and AI Integration:** Incorporate machine learning and artificial intelligence algorithms into Cyber Blade's scanning and analysis processes to enable more accurate and proactive threat detection. Develop algorithms that can identify emerging threats, analyze behavior patterns, and adapt Cyber Blade's defenses in real-time to counter evolving cyber threats effectively.
- **Automation and Orchestration:** Implement automation and orchestration capabilities within Cyber Blade to automate routine cybersecurity tasks such as threat detection, incident response, and remediation. Integrate with security orchestration, automation, and response (SOAR) platforms to streamline security operations and enhance overall efficiency and effectiveness.
- **Enhanced User Experience and Interactivity:** Continuously improve the user experience of the Cyber Blade website by incorporating user feedback, conducting usability testing, and implementing design enhancements. Develop interactive features such as customizable dashboards, real-time threat alerts, and educational resources to engage users and empower them to make informed cybersecurity decisions.

- **Integration with Cloud Security Services:** Extend Cyber Blade's capabilities to include integration with cloud security services and platforms, allowing users to protect their cloud infrastructure and applications from cyber threats. Integrate with leading cloud security providers to offer features such as cloud workload protection, container security, and server less security.
- **Expanded Compliance and Governance Features:** Enhance Cyber Blade's compliance and governance features to help organizations meet regulatory requirements and industry standards more effectively. Develop features such as compliance reporting templates, audit trails, and policy management tools to support organizations in achieving and maintaining compliance with applicable regulations and standards.
- **Global Expansion and Localization:** Explore opportunities for global expansion of Cyber Blade's reach by localizing the website and its content to cater to diverse linguistic and cultural preferences. Partner with regional cybersecurity experts and organizations to adapt Cyber Blade to local cybersecurity challenges and regulatory environments in different geographic regions.
- **Partnerships and Ecosystem Development:** Forge strategic partnerships with technology vendors, cybersecurity firms, industry associations, and government agencies to expand Cyber Blade's ecosystem and enhance its value proposition. Collaborate with ecosystem partners to integrate complementary cybersecurity solutions, share threat intelligence, and leverage synergies to deliver comprehensive cybersecurity solutions to users.

5. References

- [VirusTotal API v3 Overview](#)
- [The Request Context — Flask Documentation \(2.2.x\) \(palletsprojects.com\)](#)
- [Requests: HTTP for Humans™ — Requests 2.31.0 documentation \(pythonrequests.org\)](#)