

# **vulnerability scanner With AI**

Faculty of Computers and Information Technology

**Supervised By :**

Dr. Mohammed Zidan

Eng: Toka Ashraf

# vulnerability scanner With AI

*Discover the weaknesses of a given system*

## Project By

- ❖ Eslam Mohammed Moawed Elabd
- ❖ Abdelrahman Ail Abdelhafeez abdelbary
- ❖ Nourhan Mohammed Shaban Mohammed
- ❖ Alaa Elsayed Saber Eissa
- ❖ Walaa Ahmed Korani Mohamed
- ❖ Mohamed yasser hussein abdelhamid
- ❖ Walaa Mostafa Ahmed abdelaziz
- ❖ Mohamed Ahmed Ali

## Content Table:

The purpose of the Project :	7
Motivation:	7
Objectives	7
Remediation Path:	8
Chapter1 :Introduction	9
Introduction:	10
1.1 Motivation:	11
1.2 Motivation	12
How it work :	12
The scanner performs scanning in four steps:	12
1.3 Background :	13
Chapter 2: Related Work	14
2.1 Existing Vulnerability Scanners:	15
2.2 Research Papers and Frameworks:	16
2.3 Identified Gaps:	16
2.4 Integration with Modern Technologies:	17
Chapter3 :Analysis & Design	18
3.1Analysis of VulnerabilityScanner	19
The primary functions of the Tool:	19
Scope:	20
Impact:	20
Requirements :	21
Software Requirements:	21

Hardware Requirements:	24
Minimum System Requirements:	24
Recommended System Requirements:	24
the project built used:	25
2.2Design of Vulnerability Scanner	25
Design System:	25
Scanners Methodology	26
1-Network Scan Methodology :	26
2-Web Scan Methodology :	26
Network Scan Diagram:	27
Web Scan Diagram:	28
System Actors:	29
Use Case Vulnerability Scanner:	30
Use Case Scenario 1: Network Scan by user	31
Use Case Scenario2: Web Scan by a Freelance Web Developer	32
Vulnerability Scanner Sequence Diagram :	33
Web Scan Sequence Diagram:	34
Network Scan Sequence Diagram:	35
ERD Diagram For Vulnerability Scanner:	37
Vulnerability Scanner Schema:	38
UML Diagram For Vulnerability Scanner :	40
AntiVirus Diagram:	43
Chapter 4: AI(ANTI VIRUS)	44
introduction:	45
AI-Based Malware Detection:	45
the Main Topics in the Model :	45

ANTiVirus Process:	46
Explanation of Algorithm Selection:	47
conclusion:	49
Chapter 5: Implementation	50
Web Scanner Components :	51
1)Parser Class Documentation:	51
HtmlParser Class Documentation	53
XmlParser Class Documentation	55
HTTP Client:	57
HTTP Client Configuration:	58
HTTP Client:	59
Crawler Class Documentation:	60
PathTraversalScanner Class Documentation:	63
SQLi Class Documentation	65
Network Scanner Components :	68
Port Scanner:	69
Network Vulnerability Scanner :	73
NetworkScanner Class:	73
NetWork Scan linked with front:	76
AI Implementation :	78

# **vulnerability scanner With AI**

## The purpose of the Project :

Simply , the purpose is to discover and identify web and network vulnerabilities using modern technological developments and a simple and friendly user interface , through which the user can scan aURL or IP Address and Scan any file using our ANtiVirus developed by AI.

### Motivation:

1. Need for proactive security: Recognizes the necessity of proactive security measures in the current digital environment to combat the prevalent threat of data breaches.
2. Developing a robust vulnerability scanner: Aims to create a comprehensive Network and Web Applications Vulnerability Scanner.
3. Identifying security gaps: Focuses on detecting and addressing vulnerabilities in web and networking systems to enhance security posture.
4. User-friendly solution: Aims to deliver a user-friendly tool that organizations can easily integrate into their security protocols

### Objectives

The primary objectives of the vulnerability scanner are to enable organizations to detect, assess, and address potential security weaknesses in their IT system, network, or application. By targeting TOP 10 vulnerabilities in web applications, the scanner aims to detect attacks that lead to data breaches.

Moreover, the project extends its scope to network scanning, network scanners tailored to detect vulnerabilities in various network infrastructure components like routers and switches. These scanners analyze network security configurations, identifying potential misconfigurations and

vulnerabilities that attackers could exploit to gain unauthorized access or disrupt network operations

### Remediation Path:

Upon identifying vulnerabilities through scanning, organizations can follow a remediation path. This may involve patching vulnerabilities, closing risky ports, fixing misconfigurations, and implementing security measures on devices. The overarching aim is to enhance overall system security and reduce the risk of security incidents.



# Chapter1 :Introduction

## Introduction:

In today's digital world, websites, networks, and mobile apps are central to our online interactions. They're essential for shopping, collaboration, and information access, transforming how we connect and operate.

However, the widespread use of web applications and networks also brings security risks, such as data theft and tampering. To address these threats, our project aims to develop a robust defense system using a Web Application and Network Vulnerability Scanner.

**So, what does the Vulnerability Scanner mean?:** is a tool designed to identify security weaknesses in websites, networks, and applications. It scans for potential vulnerabilities, such as outdated software, configuration issues, and weak passwords, that hackers could exploit. By detecting these vulnerabilities, the scanner helps organizations fix them before they can be used in an attack, enhancing overall security.

This scanner is meticulously designed to assess, identify, and address a spectrum of security vulnerabilities that commonly afflict web applications. It goes beyond the conventional boundaries to scrutinize and fortify against threats such as SQL Injection (SQLi), Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF).

A Network Scanner, in particular, focuses on examining network infrastructure to identify potential security gaps. It scans network devices, servers, and endpoints for vulnerabilities like open ports, insecure configurations, and unpatched software. By identifying these issues, the Network Scanner helps ensure that all components of the network are secure and less susceptible to attacks.

## 1.1 Motivation:

Vulnerability refers to "the quality or state of being exposed to the possibility of being attacked or harmed, either physically or emotionally

A vulnerability scanner is a computer program designed to assess computers, networks or applications for known weaknesses.

These scanners are used to discover the weaknesses of a given system.

They are utilized in the identification and detection of vulnerabilities arising from mis-configurations or flawed programming within a network-based asset such as a firewall, router, web server, application server, etc.

scanner will focus on scanning web and networking systems to identify flaws in these systems and recommending best solutions for these flaws with a user friendly ui to communicate with it.

Once vulnerabilities have been identified through scanning and assessed, an organization can pursue a remediation path, such as patching vulnerabilities, closing risky ports, fixing misconfigurations, and even changing default passwords, such as on the internet of things (IoT) and other devices.

## 1.2 Motivation

The work of the vulnerability scanner is the same as that of other antivirus programs, in that the user database stores a description of a different type of vulnerabilities.

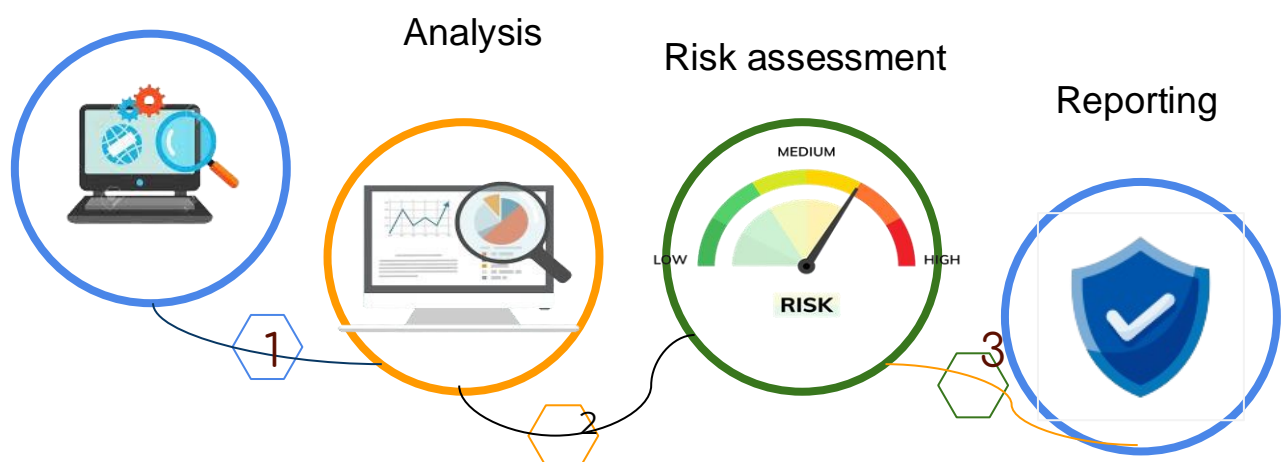
Vulnerability scanning software collects all this information from the network and then scans the ports of the network system, identifying any password violations and determining if any security fixes are missing.

### How it work :

<b>1: user Request to Scan URI OR IP (target) Address</b>
<b>2:the Tool will Scan the Target</b>
<b>3:The Result of scan will be shown</b>

The scanner performs scanning in four steps:

Vulnerability  
identify



## 1.3 Background :

Web application vulnerabilities can manifest in various forms, ranging from SQL Injection (SQLi) and Cross-Site Scripting (XSS) to Server-Side Request Forgery (SSRF) and Security Misconfigurations. These vulnerabilities pose significant risks, including unauthorized access to sensitive data, manipulation of application behavior, and potential breaches that can lead to severe consequences for individuals and organizations alike.

The motivation behind developing a Web Application Vulnerability Scanner stems from the critical need to address these vulnerabilities proactively. Data breaches often result from unpatched vulnerabilities, making it imperative to identify and eliminate these security gaps. A dedicated scanner serves as a preemptive measure, allowing organizations to fortify their systems, protect sensitive data, and maintain a robust security posture.

Understanding the intricacies of vulnerability scanning requires delving into the motivations driving malicious attacks and the methods employed by cybersecurity professionals to thwart these threats. This background section aims to provide a comprehensive overview of the cybersecurity landscape, the significance of vulnerability assessment, and the role of a Web Application Vulnerability Scanner in mitigating risks and safeguarding digital assets. By exploring the context in which this project operates, we lay the groundwork for a deeper understanding of its purpose and potential impact.

## Chapter 2: Related Work

The Related Work chapter provides an overview of existing tools, frameworks, and research in the field of web application vulnerability scanning. Understanding the landscape of related work is crucial for identifying gaps, building on existing knowledge, and ensuring that the developed scanner adds value to the current state of the art.

## 2.1 Existing Vulnerability Scanners:

Several web application vulnerability scanners currently exist, each with its unique features and capabilities. Some noteworthy examples include:

1. OWASP ZAP (Zed Attack Proxy):
  - An open-source security tool for finding vulnerabilities in web applications during the development and testing phases.
2. Nessus:
  - A widely-used vulnerability scanner that can identify vulnerabilities, misconfigurations, and compliance issues in networks and web applications.
3. Acunetix:
  - A web vulnerability scanner designed to automatically identify security flaws in web applications, including SQL injection and cross-site scripting.
4. Burp Suite:
  - An integrated platform for security testing of web applications, offering features like crawling, scanning, and manual testing tools.
5. Nexpose:
  - A vulnerability management solution that helps organizations identify and remediate vulnerabilities across their IT infrastructure.

## 2.2 Research Papers and Frameworks:

Reviewing academic research and frameworks in the field can provide valuable insights into the latest advancements. Some notable works include:

1. OWASP TOP 10:
  - The OWASP Top 10 is a well-known and widely recognized document that highlights the top 10 most critical security risks to web applications. It serves as a guide for developers, security professionals, and organizations to understand and mitigate common vulnerabilities. The list is periodically updated to reflect the evolving threat landscape.
  - Link: [OWASP Top 10](#)
2. PortSwigger Research :
  - PortSwigger's research provides valuable insights into web application security, including advanced vulnerabilities and exploitation techniques. Exploring this research contributes to a deeper understanding of real-world threats and countermeasures.

## 2.3 Identified Gaps:

While existing tools and research contribute significantly to the field, there are potential gaps and opportunities for improvement. Some areas to explore in the development of the Web Application Vulnerability Scanner include:

1. Usability and User Interface Design:
  - Many existing tools may lack user-friendly interfaces. The scanner aims to address this gap by providing an intuitive and accessible user interface.
2. Customizable Scanning Policies:
  - The ability to configure custom scanning policies is identified as a specific feature gap that the scanner aims to fulfill.
3. Comprehensive Remediation Support:



- While vulnerability scanners often identify issues, the scanner aims to go beyond detection by offering comprehensive remediation support, including actionable recommendations.
4. Multi-User Accounts and Role-Based Access:
    - Enhancing security through multi-user accounts and role-based access control is a key focus, providing organizations with granular control over scanner usage.

## 2.4 Integration with Modern Technologies:

Understanding the integration landscape is crucial for ensuring the scanner aligns with modern technologies. Integration points may include:

1. API Integration:
  - Exploring the ability to integrate with other security tools or platforms through APIs for seamless workflow integration.
2. Cloud Compatibility:
  - Considering compatibility with cloud-based infrastructure and services, ensuring the scanner can adapt to evolving deployment models.
3. Containerization Support:
  - Assessing the potential for containerization support to enhance flexibility and scalability.

# Chapter3 :Analysis & Design

## 3.1 Analysis of Vulnerability Scanner

In this chapter, we embark on the journey of defining the requirements that will shape the architecture, features, and capabilities of the Web Application Vulnerability Scanner. Requirements serve as the guiding compass, providing a comprehensive understanding of what the scanner must achieve, and the Diagrams that express the relationships and how the program works.

### The primary functions of the Tool:

- **Proactively identify vulnerabilities:** Utilize automated scanning tools and manual testing techniques to identify common vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection (SQLi), and Cross-Site Request Forgery (CSRF).
  - **Address security gaps:** Provide recommendations and remediation strategies to effectively address identified vulnerabilities. This includes patching software, updating configurations, and implementing additional security measures to mitigate risks.
- Enhance security posture:** Strengthen the overall security of organizations by systematically identifying and addressing vulnerabilities in both web applications and network infrastructures. This reduces the likelihood of successful cyber attacks and minimizes the potential impact of security incidents.

## Scope:

The vulnerability scanner project encompasses both web applications and network infrastructure and AntiVirus for any file , targeting a wide range of common vulnerabilities and attack vectors. Key areas of focus include:

- **Web application vulnerabilities:** Common vulnerabilities such as XSS, SQLi, CSRF, and others are targeted to identify potential security weaknesses in web applications.
- **Network infrastructure vulnerabilities:** Network devices such as routers, switches, firewalls, and servers are scanned for misconfigurations, outdated firmware, open ports, and other vulnerabilities that could be exploited by attackers.
- **AntiVirus developed by AI :** the Antivirus detects any vulnerabilities based on the file user uploaded .
- **Comprehensive coverage:** The project aims to provide comprehensive coverage of both web applications and network infrastructure and Antivirus to ensure that all potential security risks are identified and addressed.

## Impact:

The vulnerability scanner project has the potential to have a significant impact on enhancing overall security posture and mitigating the risk of cyber attacks. Some potential impacts include:

- **Reduced risk of data breaches:** Proactively identifying and addressing vulnerabilities helps reduce the likelihood of successful cyber attacks and minimizes the risk of data breaches.
- **Improved compliance:** Conducting regular vulnerability assessments and remediating identified vulnerabilities ensures compliance with regulatory requirements, helping organizations avoid potential fines or penalties.
- **Enhanced reputation:** Proactively securing sensitive data and protecting against cyber threats enhances an organization's reputation and builds trust with customers, partners, and stakeholders.

## Requirements :

In this pivotal chapter, we embark on the journey of defining the requirements that will shape the architecture, features, and capabilities of the Web Application Vulnerability Scanner. Requirements serve as the guiding compass, providing a comprehensive understanding of what the scanner must achieve. The key components of this chapter include : Hardware and Software Requirements .

## Software Requirements:

### 1- Functional Requirements

#### - **Vulnerability Scanning:**

The scanner should be able to identify common vulnerabilities in web applications, including but not limited to SQL injection, cross-site scripting (XSS), XML External Entity (XXE) Injection, Cross-Site Request Forgery (CSRF), and security misconfigurations.

Different scanning modes (e.g., passive scanning, active scanning) should be available to cater to various use cases.

- **Passive Scanning:** In passive scanning, it is determined whether a tool can enlist the vulnerabilities by considering the existing network.
- **Active Scanning:** In active scanning, it is determined whether the queries can be made to the network for the vulnerability.

#### - **User Authentication:**

- **Secure User Authentication:**  
implement secure user authentication to control access to the scanner's features and reports.
- **Password Security:**  
The scanner enforces strong password policies, including minimum length, complexity requirements, and regular password expiration.

- Session Management:

Implement secure session management to handle user sessions, including session creation, expiration, and secure session token storage.

- Users Accounts:

every user after login will have a unique token (JWT) and id .

- User Profile Management:

users have the ability to manage their profiles, including updating personal information.

- User Interface

- Navigation to Website:

the website have the user interface, allowing users to easily access different features and sections

- Website :

features of a user-friendly that provides a real-time overview of scanning progress and identified vulnerabilities. The home page has a lot of sections explaining more information about our tool.

- Responsiveness:

The user interface is responsive, adapting to different screen sizes and devices for a seamless user experience.

### **-Action Plans & Vulnerability Management:**

A vulnerability scan can identify numerous vulnerabilities. To manage them effectively, sort vulnerabilities by risk level (high, medium, low, critical) and assign tasks to the security team for quick remediation.

- **Scanning Reports & Overall Risk Score:**

- After scanning the Ui that highlights the risk scores (high, medium,low, Critical) for all vulnerabilities.
- A centralized user interface will provide you valuable insights.

- After each scanner, a Report will be available containing more details about this scanner that the user can download

#### **- Customizable Scanning Policies:**

- Policy Configuration:  
Provide a user-friendly interface for configuring custom scanning policies.
- Policy Enforcement:  
Ensure the scanner adheres to the configured scanning policies during each scanning session.

## **2- non-Functional Requirements:**

#### **- Performance Efficiency:**

- Response Time:  
The scanner provides timely responses to user interactions, ensuring a seamless and efficient experience.

#### **- Scalability:**

Conduct tests to assess the system's ability to handle a growing number of users, targets, and scanning sessions.

#### **- Authentication and Authorization:**

- Secure Authentication:  
Employ industry-standard protocols for user authentication to ensure the confidentiality of user credentials.

#### **- Reliability & Availability :**

- Uptime Requirements:  
Define acceptable levels of system downtime for maintenance and upgrades.

- the website and server operate 24 hours a day and can handle a large number of users

#### **- Encryption:**

All communication, especially sensitive data, should be encrypted using secure protocols (e.g., HTTPS).

#### **-Browser Compatibility:**

Validate that the scanner's user interface functions consistently across popular web browsers.

#### **- Usability:**

- User Training:  
Specify any training requirements for users to effectively utilize the scanner.
- Accessibility:  
Ensure that the scanner's interface complies with accessibility standards for all types of users .(any user can use it ).

## Hardware Requirements:

### • Minimum System Requirements:

1. Processor: Dual-core processor (or equivalent)
2. Random Access Memory (RAM): 4 gigabytes '
3. Storage: 20 gigabytes of free space
4. Network Interface: 1 Gigabit Ethernet

### • Recommended System Requirements:

1. Processor: Quad-core processor (or equivalent)
2. Random Access Memory (RAM): 8 gigabytes or higher
3. Storage: 50 gigabytes or more of free space
4. Network Interface: 1 Gigabit Ethernet or higher



The Technologies used:

- **BackEnd:**

**Node.JS , TypeScript , Socket.IO , JWT**

- **FrontEnd:**

**React.js , CSS**

- **Database:**

**postgreSql**

- **AI:**

**Python**

## 2.2Design of Vulnerability Scanner

### Design System:

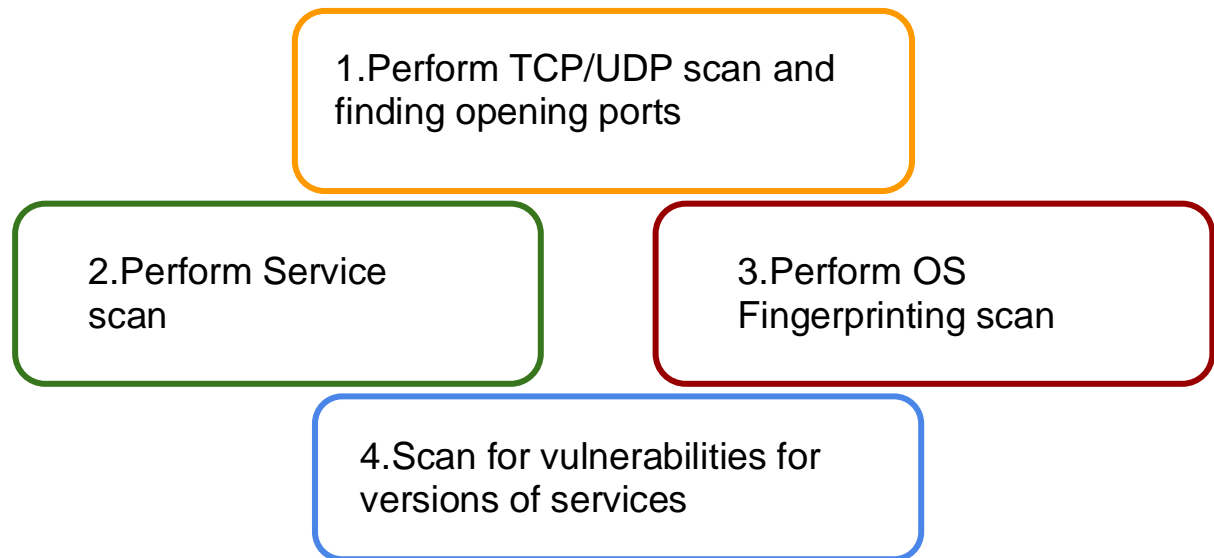
Software design is the core of the software engineering process, applicable across various paradigms and domains. It is the initial phase in creating any engineered product. The main objective is to model an entity for subsequent development stages. After identifying and analyzing requirements, system design is the first step in the essential activities of design, code, and test.

The essence of design is "Quality." It is where quality is established in the software development lifecycle. Design provides representations to evaluate software quality and translates customer perspectives into a realized product. It lays the foundation for all subsequent engineering steps. Without strong design, the system may be unstable, difficult to test, and its quality uncertain until the end stages.

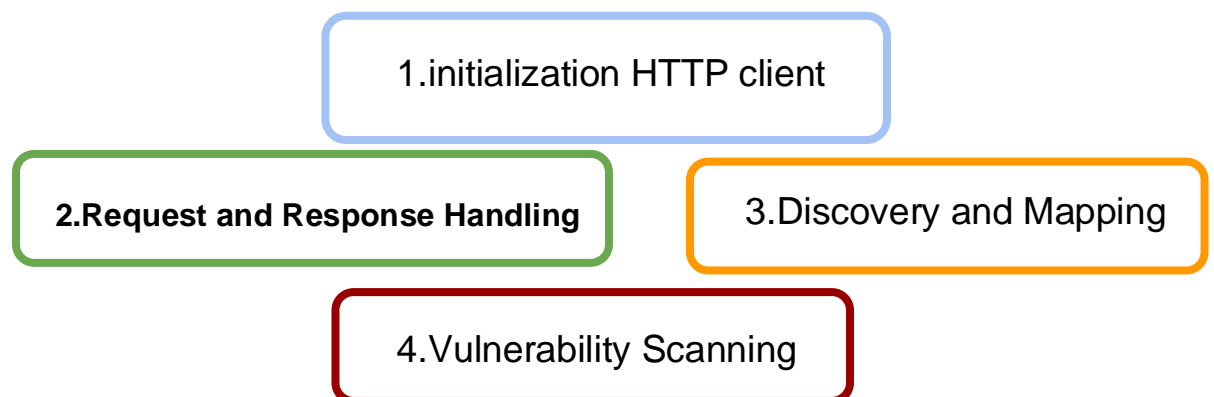
The design process continually refines data structures, program structures, and procedural details. It involves four key activities: architectural design, data structure design, interface design, and procedural design. These activities shape the foundation for a successful software product or system

# Scanners Methodology

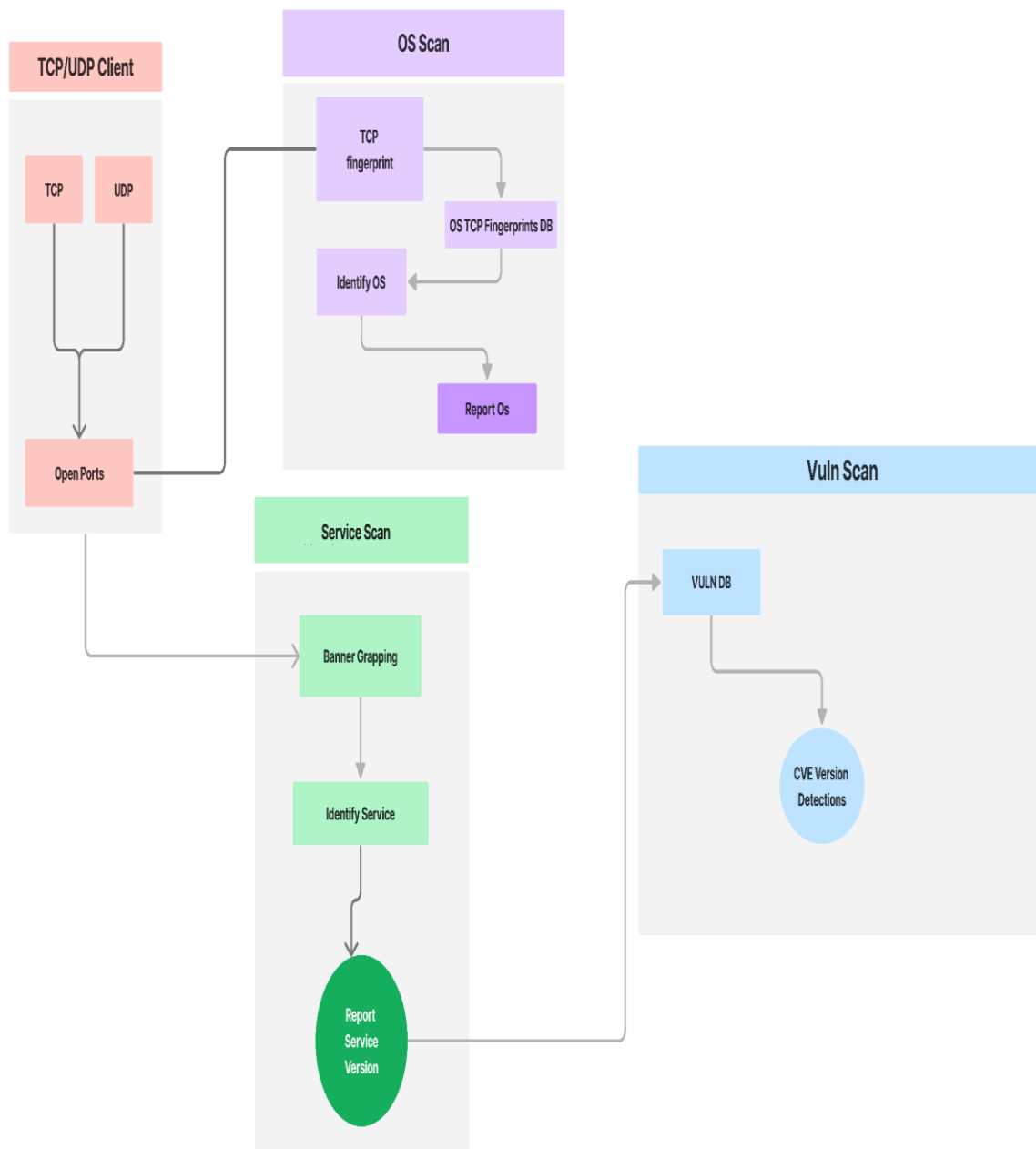
## 1-Network Scan Methodology :



## 2-Web Scan Methodology :



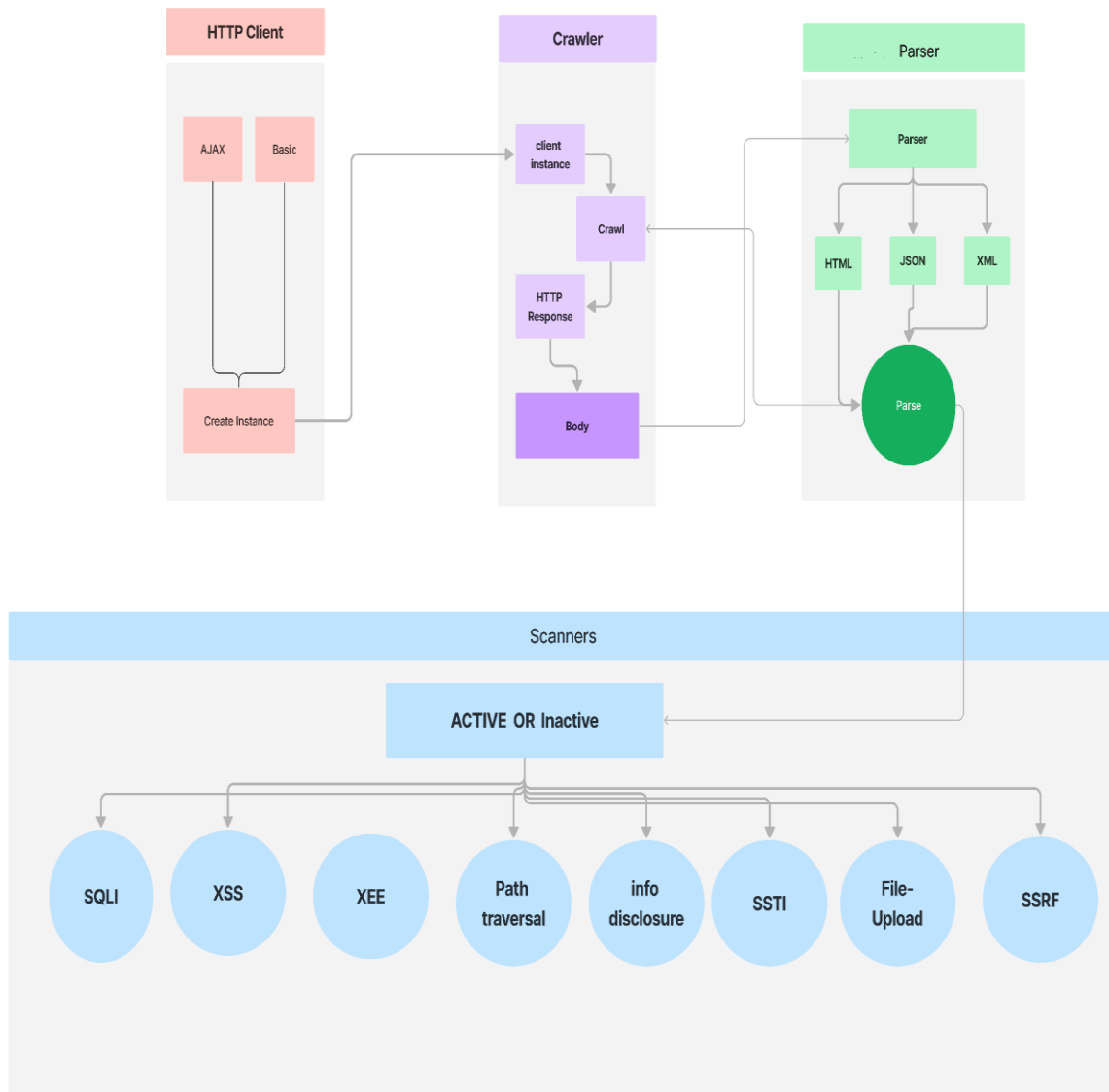
## Network Scan Diagram:



### Link:

<https://www.figma.com/board/Q21KqkEMxRktsWkiXzpXWr/unixity-scanner-network?node-id=509-17&t=0PGzRH1q0N1d0pWK-0>

## Web Scan Diagram:



### **Link:**

<https://www.figma.com/board/Q663BWEhgoJHXYl4qygGiM/unixity-scanner-web?node-id=0-1&t=2wKhWRafPO1jJ2q0-0>

## System Actors:

- **Actor 1 : New user:**

This encompasses the registration process for users who are accessing the system for the first time. Upon successful registration, users gain login credentials, granting them access to the scanner service.

- **Actor 2 : Existing users**

For existing users, it simplifies the process to log in. Once logged in, users can seamlessly utilize the scanner services.

- **Actor 3 : Admin:**

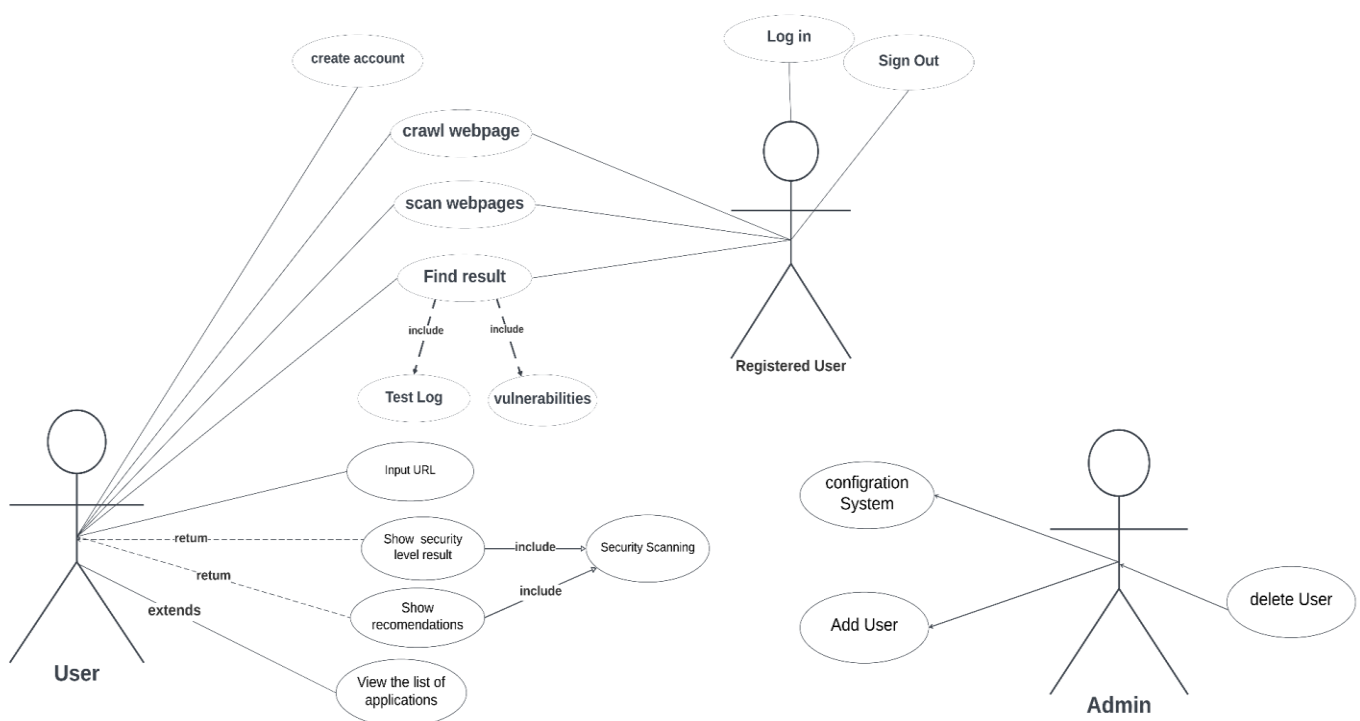
This is dedicated to administrative functions, where the admin holds the responsibility of overseeing the database and addressing user-related issues. The admin possesses full privileges, enabling them to access any user's profile and make necessary modifications.

Additionally, the admin has the authority to delete and add user accounts when required.

## Use Case Vulnerability Scanner:

A use case for a vulnerability scanner provides a detailed representation of how the scanner system functions when interacting with its users or administrators. It presents specific scenarios to illustrate how the vulnerability scanner responds to user actions, offering a comprehensive understanding of its capabilities from an end-user perspective. By outlining various interactions, such as initiating scans, managing scan results, and administering user accounts, the use case helps stakeholders visualize how the vulnerability scanner fulfills its role in identifying and addressing security vulnerabilities within web applications and network infrastructure.

- the diagram link: [vulnerability scanning use case: Lucidchart](#)
- Diagram Picture:



# Use Case Scenario 1: Network Scan by user

## Actors

- user
- Vulnerability Scanner Website

## Scenario Steps:

1	Login and Access	The Small Business Owner logs into the vulnerability scanner website using their credentials.
2	Select Network Scan	The User accesses the network scanning feature of the Vulnerability Scanner System. They input the IP address(es) of the network device(s) they want to scan.
3	Scan Execution	The Vulnerability Scanner System begins scanning the specified IP address(es). It systematically checks for vulnerabilities across the network devices.
4	Analysis and Report Generation	As the scan progresses, the Vulnerability Scanner System analyzes the network devices for vulnerabilities. It categorizes the vulnerabilities based on severity and generates a detailed report.
5	Display Results	Once the scan is complete, the Vulnerability Scanner System presents the results to the User. The results include a list of vulnerabilities found, along with their risk levels (e.g., low, medium, high).



# Use Case Scenario2: Web Scan by a Freelance Web Developer

## Actors

- Freelance Web Developer
- Vulnerability Scanner System

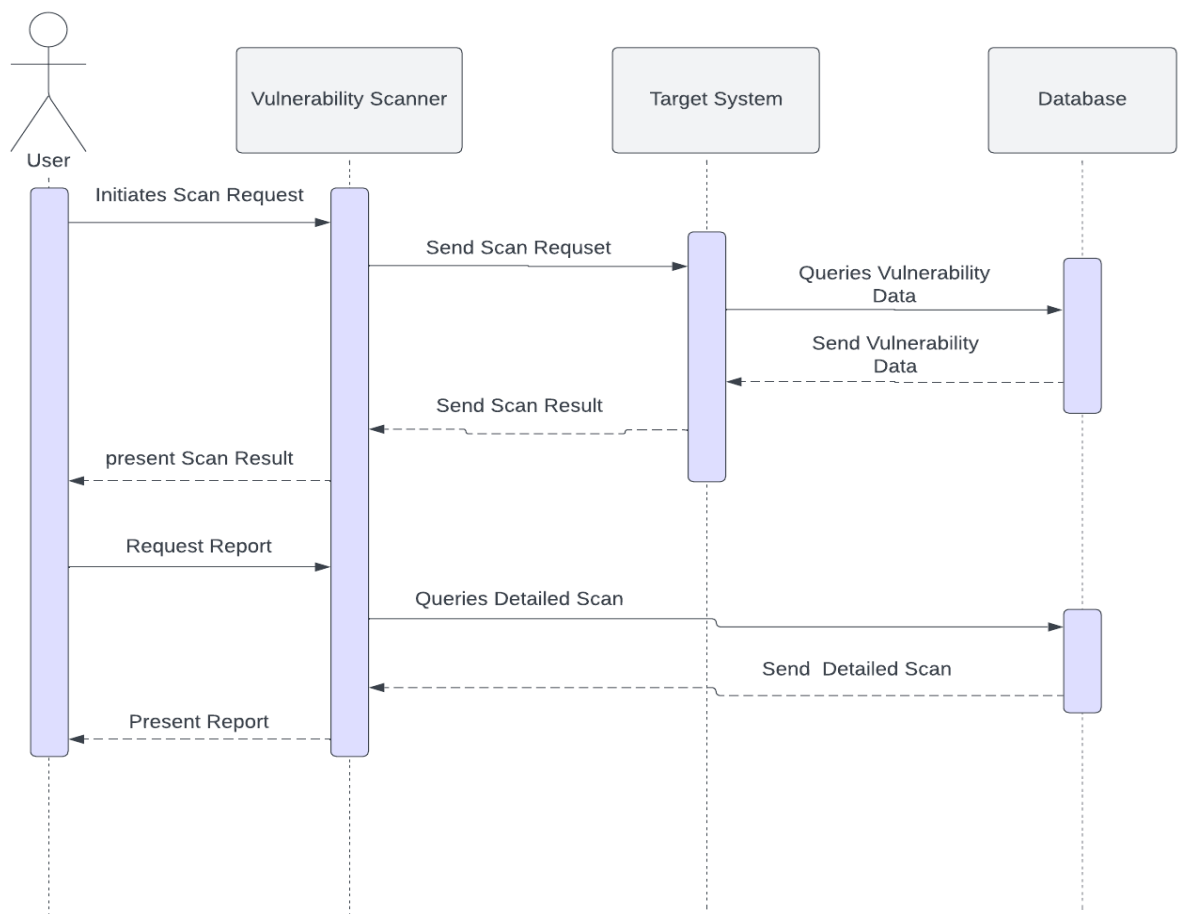
## Scenario Steps:

1	Login to the Vulnerability Scanner System	<p>The Freelance Web Developer accesses the login page of the Vulnerability Scanner System.</p> <p>They enter their username and password to authenticate their identity.</p>
2	Access Web Scan Functionality	<p>After successful authentication, the Freelance Web Developer navigates to the web scanning section of the Vulnerability Scanner System.</p>
3	Enter Website URL	<p>The Freelance Web Developer inputs the URL of the website or web application they are tasked with assessing.</p>
4	Scan Execution	<p>The Vulnerability Scanner System begins scanning the specified website or web application.</p> <p>It systematically checks for vulnerabilities across the web assets.</p>
5	Analysis and Report Generation	<p>As the scan progresses, the Vulnerability Scanner System analyzes the website or web application for vulnerabilities.</p> <p>It categorizes the vulnerabilities based on severity and generates a detailed report.</p>

## Vulnerability Scanner Sequence Diagram :

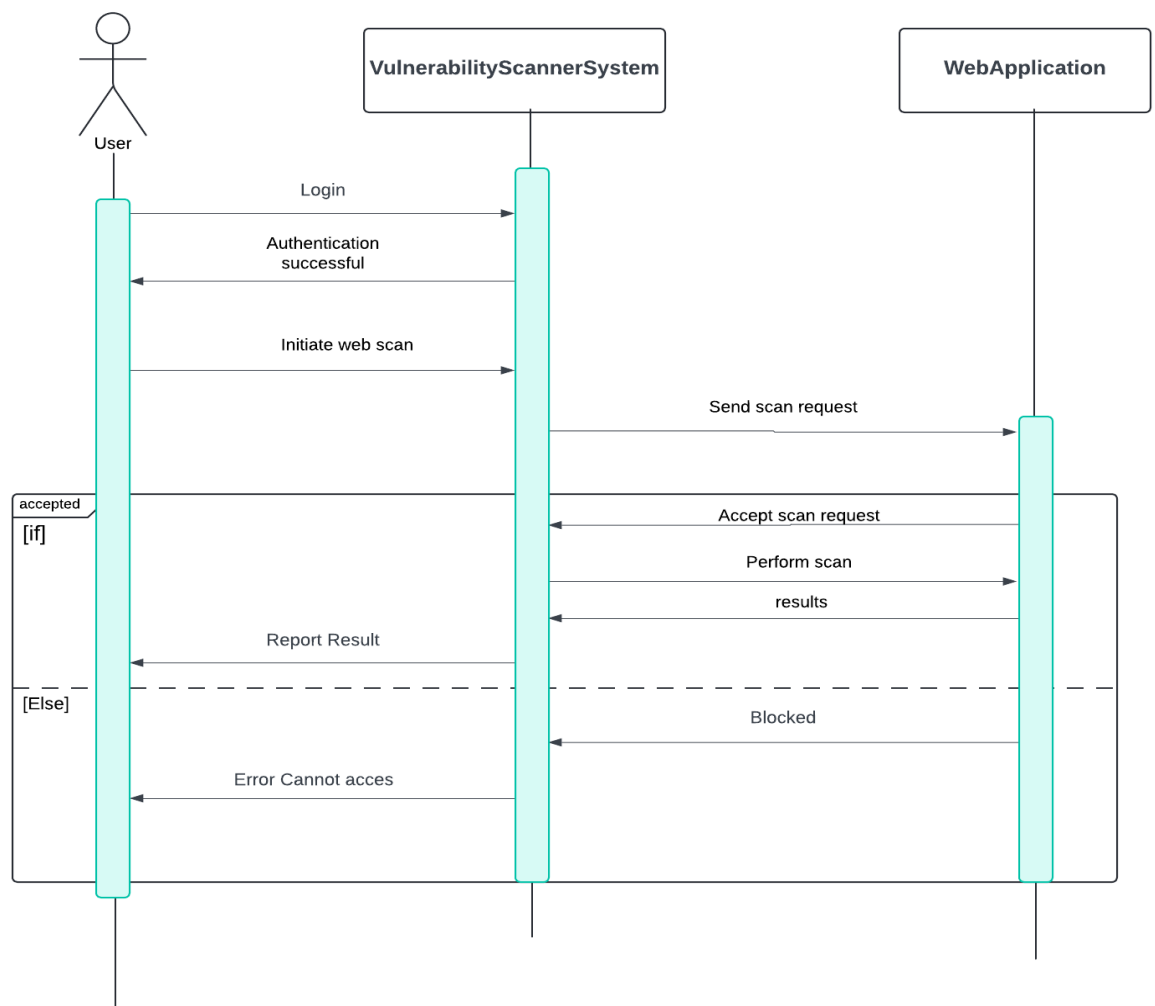
In the Vulnerability Scanner Sequence Diagram, we outline the series of interactions between the user, the vulnerability scanner system, the target system under scan, and the vulnerability database. This succinct representation encapsulates the step-by-step process of conducting a scan, querying vulnerability data, receiving scan results, requesting a detailed report, and presenting it to the user.

- the diagram link: [vulnerability-scanner sequence diagram](#).
- the diagram picture:



## Web Scan Sequence Diagram:

1. The User initiates the process by logging into the Vulnerability Scanner System.
2. The Vulnerability Scanner System authenticates the user and allows them to proceed.
3. The User then initiates a web scan through the system.
4. The Vulnerability Scanner System sends a scan request to the Web Application.
5. The Web Application processes the scan request and responds with the scan results.
6. Finally, the Vulnerability Scanner System presents the scan results to the User.



the diagram link: [Web-Scanner -Sequence Diagram](#)

## Network Scan Sequence Diagram:

### **-Log In**

- User -> VulnerabilityScannerSystem: Log in
- VulnerabilityScannerSystem -> User: Authentication successful

### **-Enter IP Address and Initiate Scan**

- User -> VulnerabilityScannerSystem: Enter IP address for scan
- User -> VulnerabilityScannerSystem: Initiate network scan

### **-Send Scan Request**

- VulnerabilityScannerSystem -> NetworkDevice: Send scan request to IP

### **-Scan Request Accepted**

- NetworkDevice -> VulnerabilityScannerSystem: Scan request accepted
- VulnerabilityScannerSystem -> User: Scan request accepted

### **-Perform Scan**

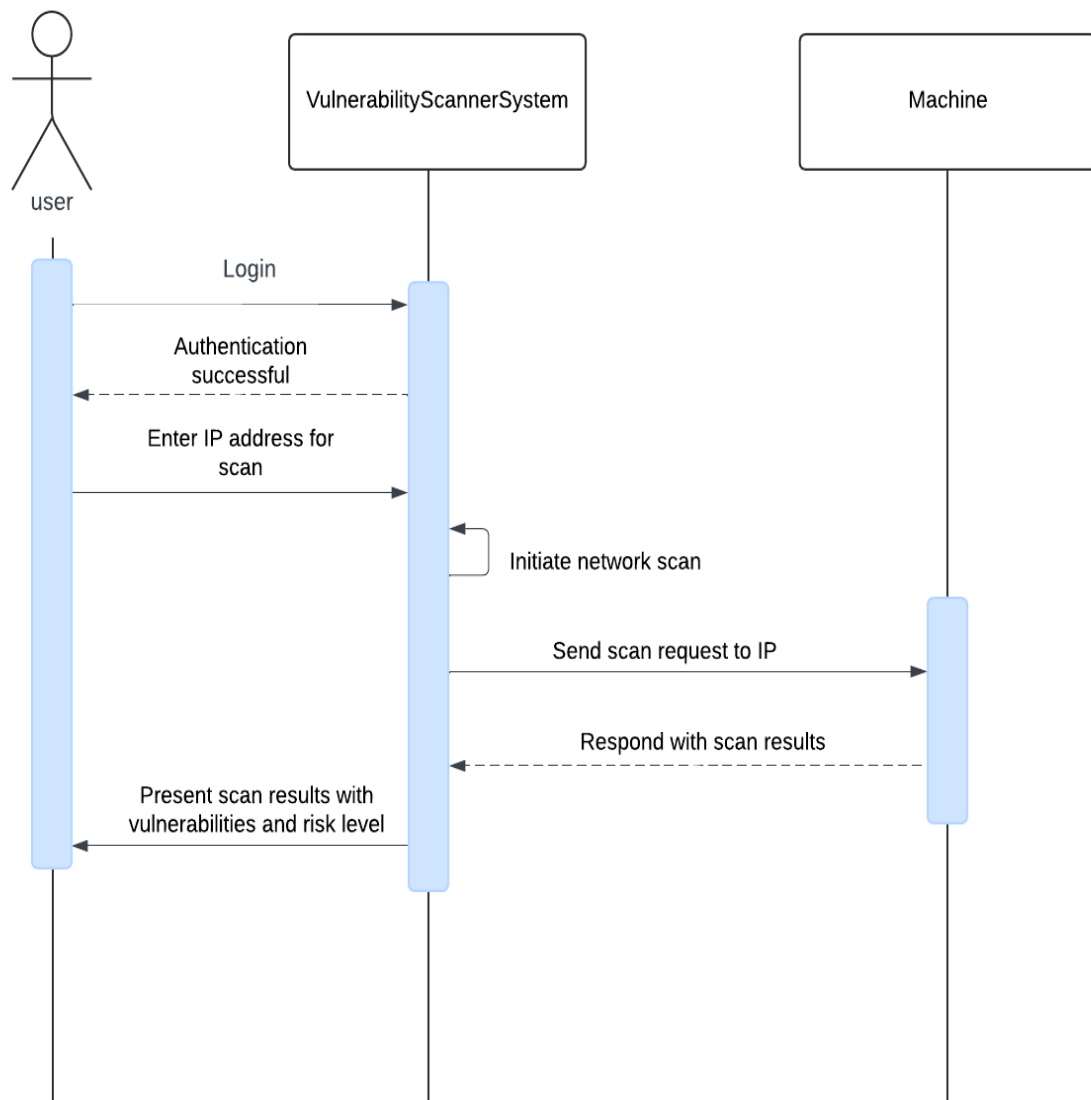
- NetworkDevice -> VulnerabilityScannerSystem: Perform scan

### **-Receive Scan Results**

- NetworkDevice -> VulnerabilityScannerSystem: Respond with scan results

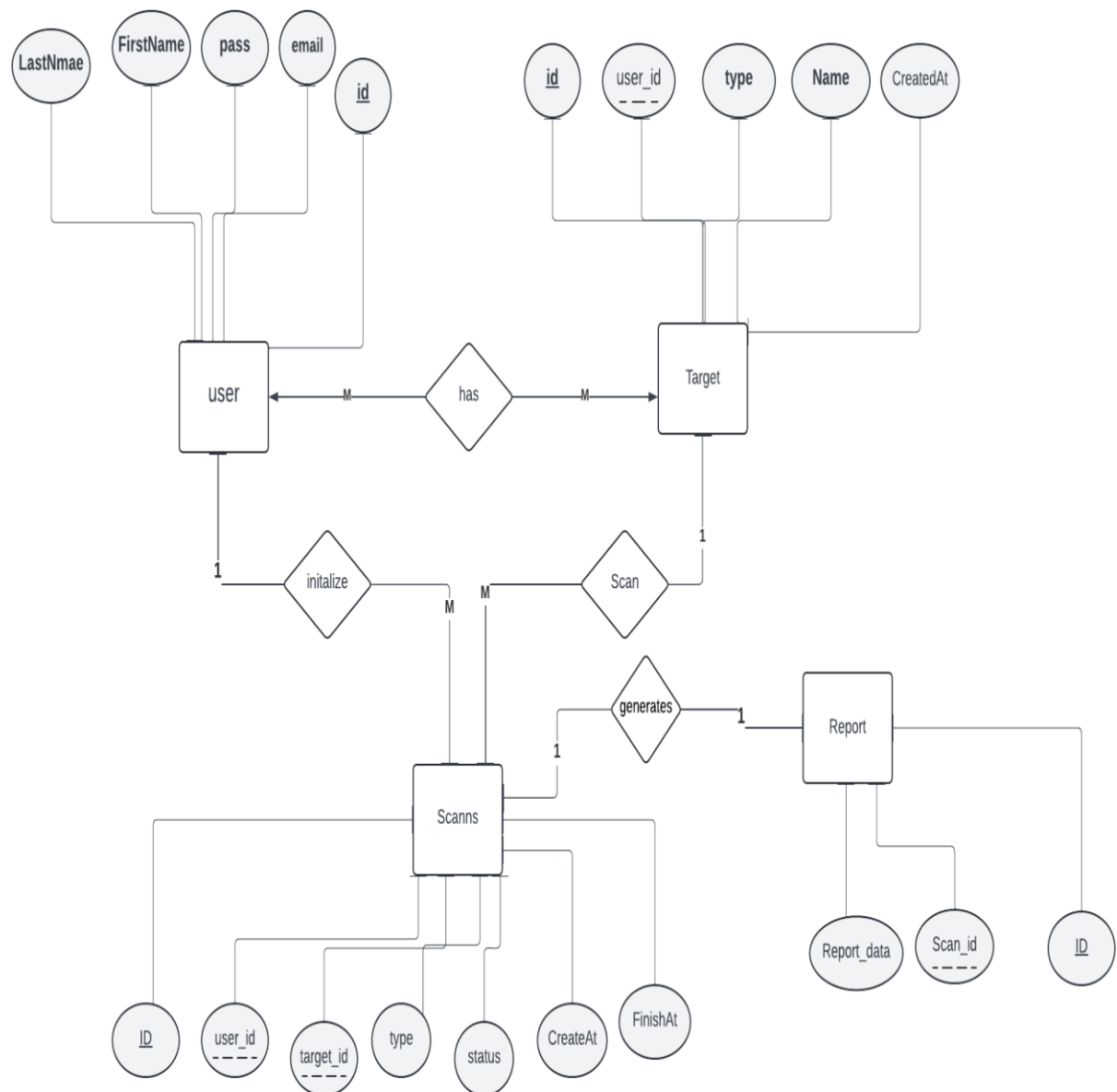
### **-Present Results**

- VulnerabilityScannerSystem -> User: Present scan results with vulnerabilities and risk level

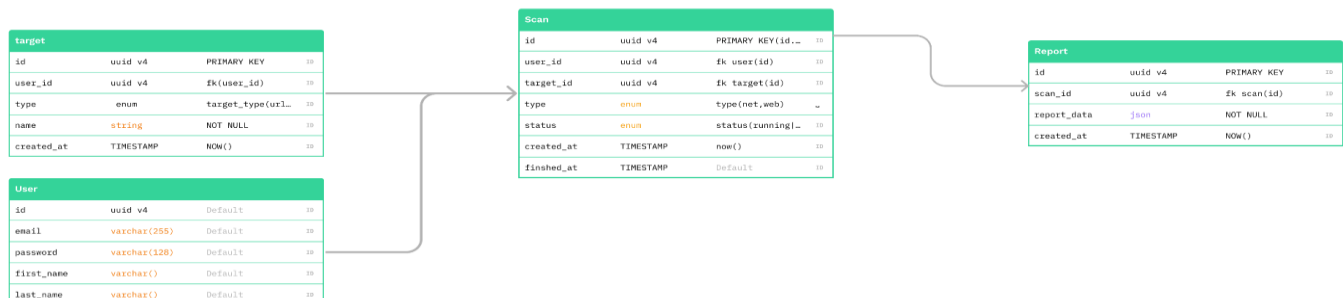


the diagram link: [Network-Scanner-Sequence-Diagram](#)

## ERD Diagram For Vulnerability Scanner:



# Vulnerability Scanner Schema:




target				
id	uuid v4	PRIMARY KEY	ID	
user_id	uuid v4	fk(user_id)	ID	
type	enum	target_type(url...	ID	
name	string	NOT NULL	ID	
created_at	TIMESTAMP	NOW()	ID	

User				
id	uuid v4	Default	ID	
email	varchar(255)	Default	ID	
password	varchar(128)	Default	ID	
first_name	varchar()	Default	ID	
last_name	varchar()	Default	ID	
created_at	TIMESTAMP	NOW()	ID	



Report			
id	uuid v4	PRIMARY KEY	ID
scan_id	uuid v4	fk scan(id)	ID
report_data	json	NOT NULL	ID
created_at	TIMESTAMP	NOW()	ID



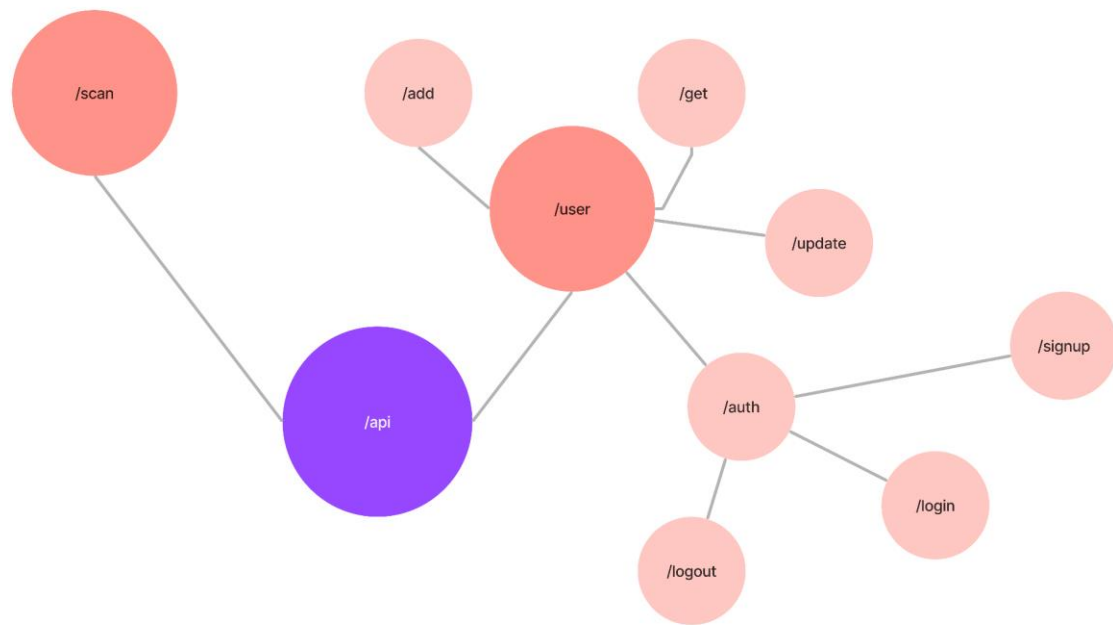


Scan			
id	uuid v4	PRIMARY KEY(id,...	ID
user_id	uuid v4	fk user(id)	ID
target_id	uuid v4	fk target(id)	ID
type	enum	type(net,web)	ID
status	enum	status(running ...	ID
created_at	TIMESTAMP	now()	ID
finshed_at	TIMESTAMP	Default	ID

vulnerabilities			
id	uuid v4	PRIMARY KEY	ID
name	varchar(64)	NOT NULL	ID
description	TEXT	NOT NULL	ID
severity	enum	[low,mid,high]	ID
recomendation	TEXT	NOT NULL	ID
cwe	int	NOT NULL	ID

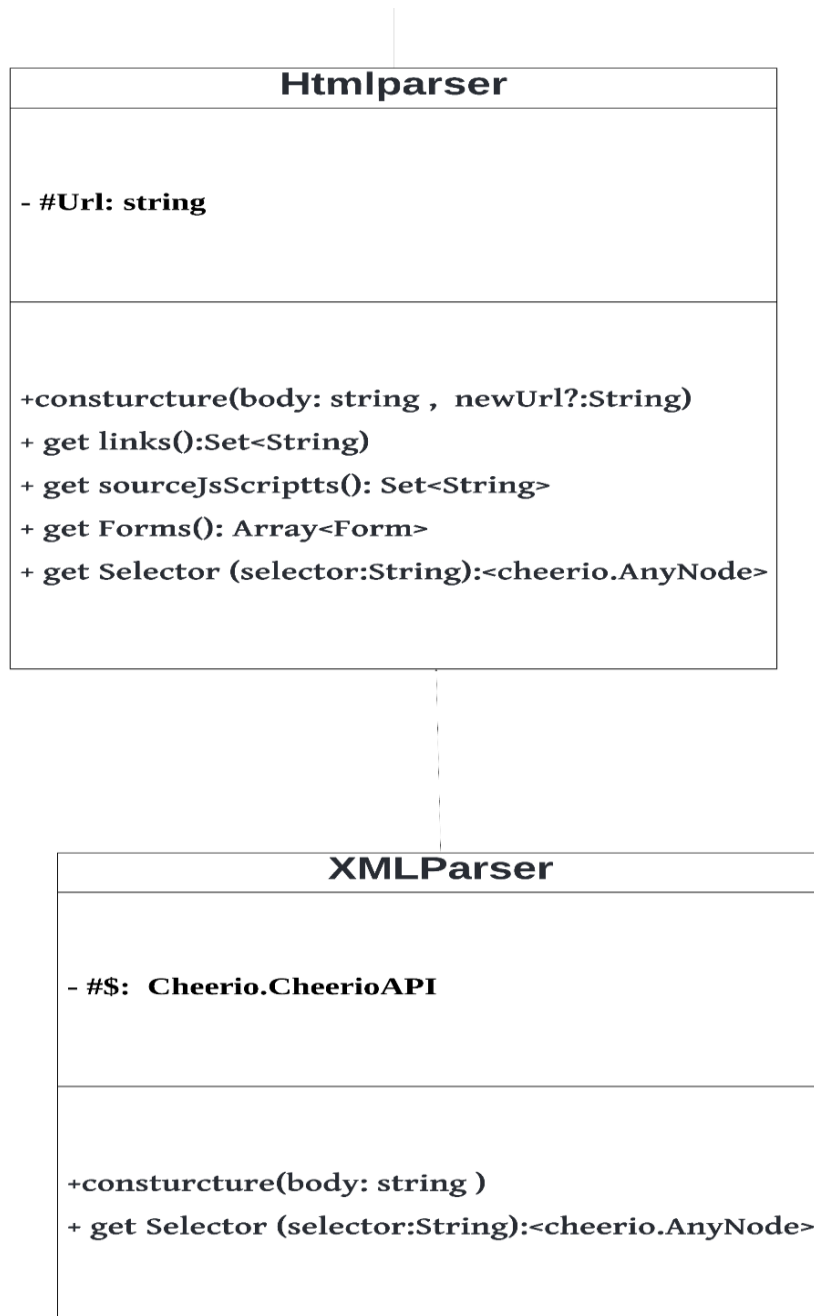
## schema for Api:

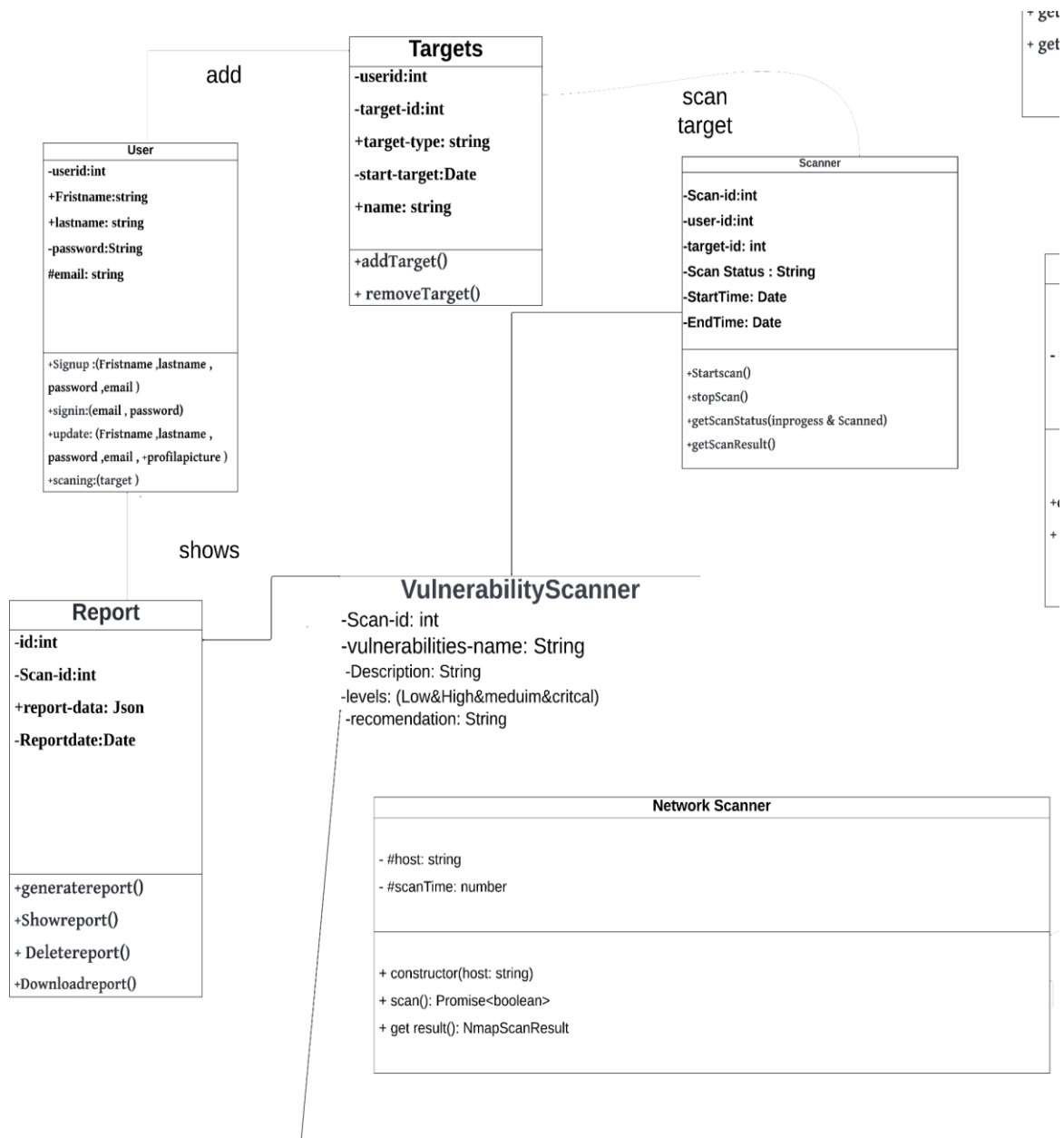
endpoint	method	description	request body	response body
/api/auth/register	post	user registration	user_info	user registration confirmation
/api/auth/login	post	user login	user credit	jwt
/api/auth/logout	get	user logout		
/api/websites/scans	post	create new website scan	website url and other info	scan_id
/api/websites/scans/{scan_id}	get	get scan details	scan_id	scan details
/api/networks/scans	post	create new network scan	network ip and other info	scan id
/api/networks/scans/{scan_id}	get	get scan details	scan_id	scan details



the Schema Diagram Link :[Schema\\_link](#)

## UML Diagram For Vulnerability Scanner :





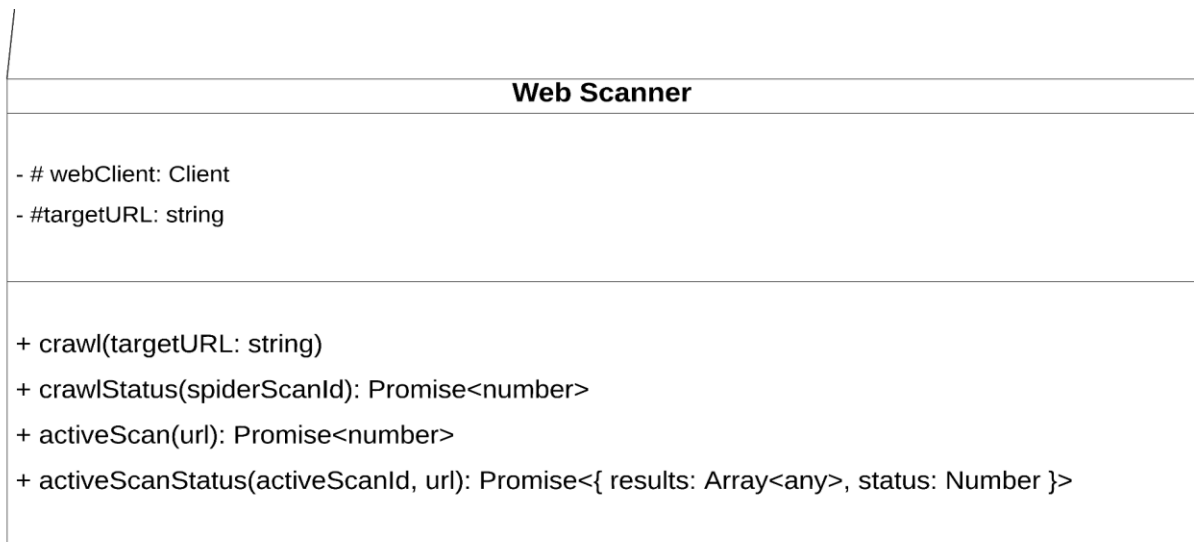
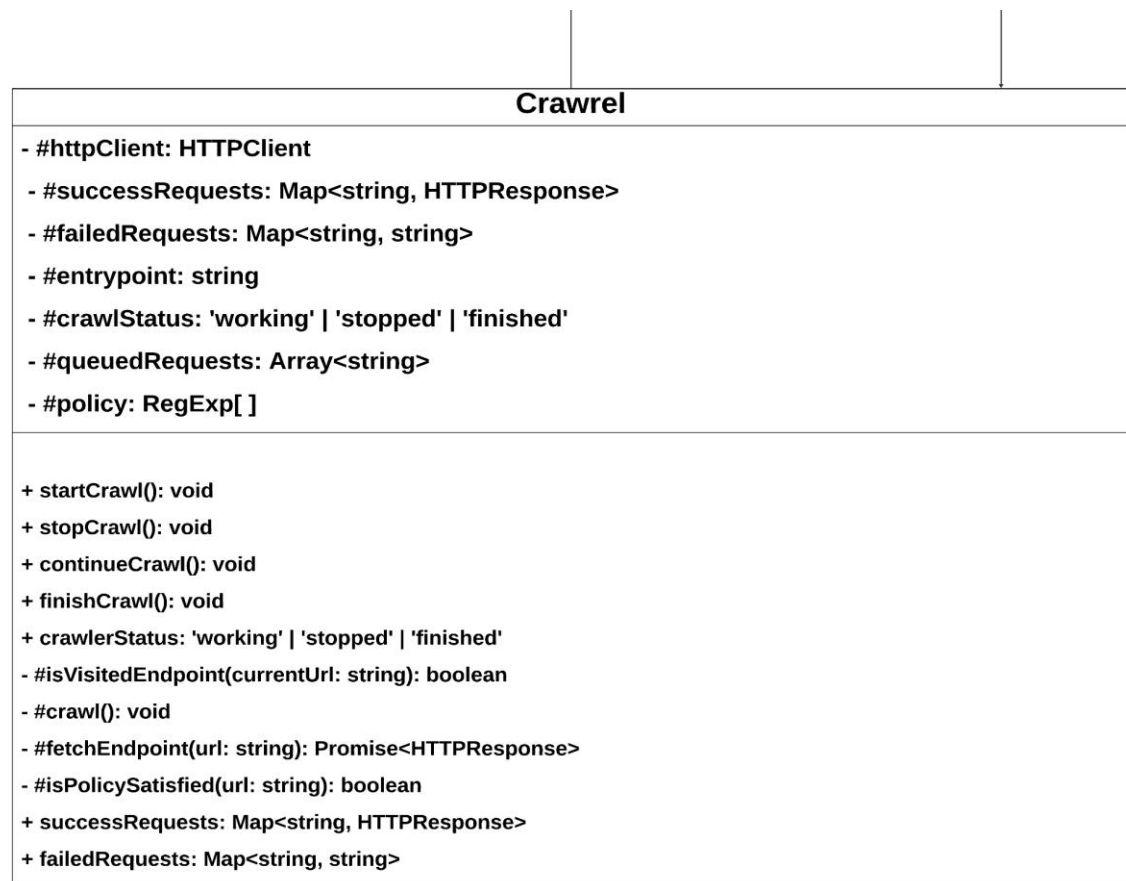
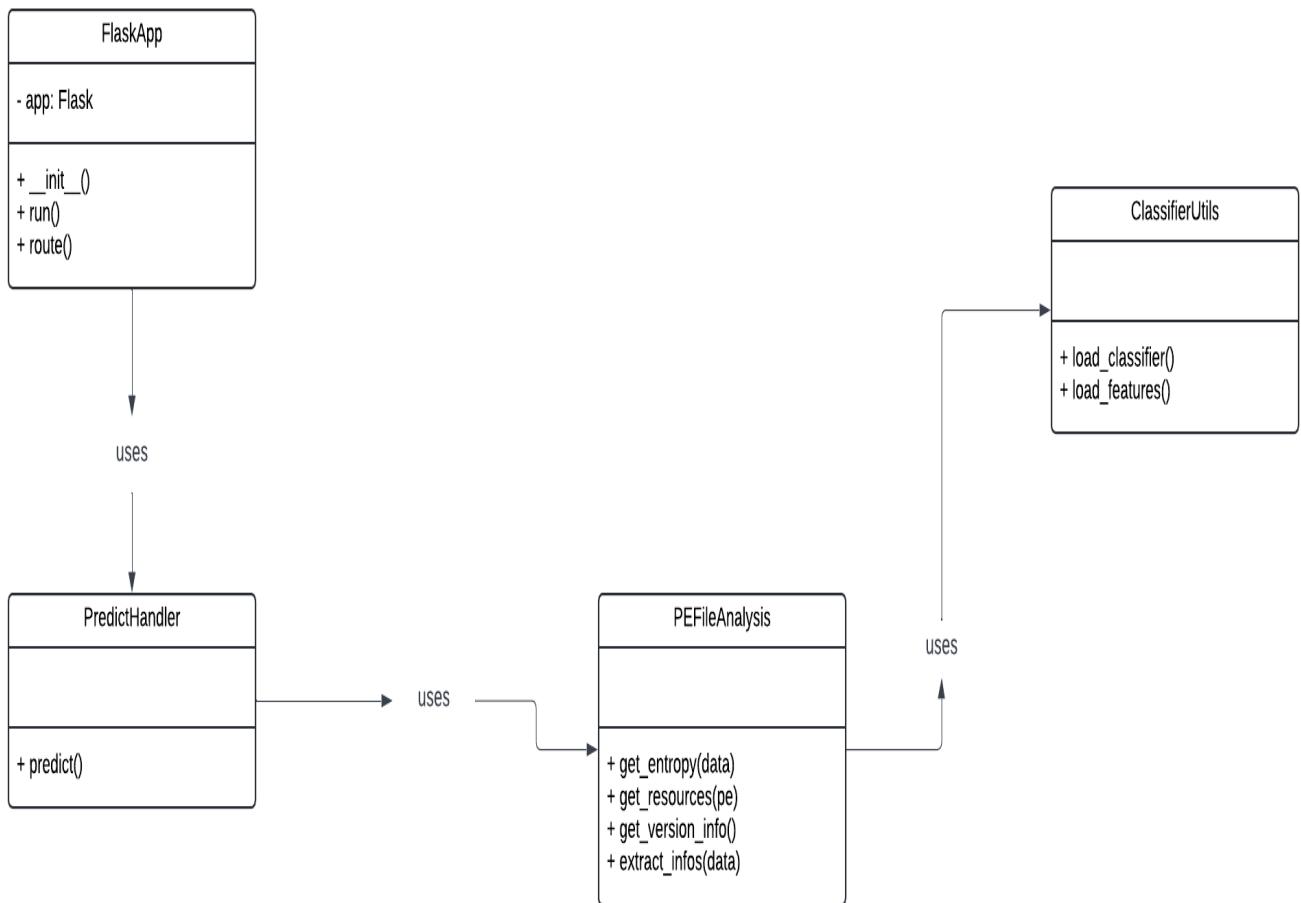


Diagram Link: [https://lucid.app/lucidchart/c648d193-ad37-41e8-b50d-cbcc1839cc27/edit?viewport\\_loc=-5897%2C-14425%2C13621%2C7046%2CHWEp-vi-RSFO&invitationId=inv\\_6dfaa99b-57c0-4572-849c-ae0112b07583](https://lucid.app/lucidchart/c648d193-ad37-41e8-b50d-cbcc1839cc27/edit?viewport_loc=-5897%2C-14425%2C13621%2C7046%2CHWEp-vi-RSFO&invitationId=inv_6dfaa99b-57c0-4572-849c-ae0112b07583)

## AntiVirus Diagram:



Link: [https://lucid.app/lucidchart/d7e670d3-27fa-4bfd-a0a3-150182568047/edit?viewport\\_loc=-92%2C-419%2C2415%2C1464%2C0\\_0&invitationId=inv\\_be49d047-08e7-4bc2-90fl-5724dc9cd190](https://lucid.app/lucidchart/d7e670d3-27fa-4bfd-a0a3-150182568047/edit?viewport_loc=-92%2C-419%2C2415%2C1464%2C0_0&invitationId=inv_be49d047-08e7-4bc2-90fl-5724dc9cd190)

## Chapter 4: AI(ANTI VIRUS)



## introduction:

The AI-based antivirus system leverages machine learning to enhance traditional malware detection methods. This chapter details the integration of the AI-based antivirus into the existing vulnerability scanner, providing a comprehensive approach to identifying and mitigating potential threats. This chapter explores the use of machine learning for malicious file detection, focusing on a practical implementation using Python and popular libraries such as scikit-learn and Flask.

### AI-Based Malware Detection:

The AI-based antivirus system employs machine learning algorithms to analyze file attributes and behavior patterns, enabling it to identify potentially malicious files with greater accuracy. Unlike signature-based detection methods, which rely on predefined patterns, AI-based detection can adapt and learn from new data, making it more effective against zero-day threats and polymorphic malware.

### the Main Topics in the Model :

#### **Feature Extraction and Selection:**

For malicious file detection, relevant features may include file metadata, entropy values, section characteristics, and resource information. However, not all features contribute equally to the classification task, making feature selection essential for improving model efficiency and interpretability.

### **Model Training and Evaluation:**

We proceed to train multiple classification algorithms on the labeled dataset. These algorithms include decision trees, random forests, gradient boosting, AdaBoost, and Gaussian Naive Bayes. We split the dataset into training and testing sets to assess each model's performance and choose the best-performing algorithm based on accuracy metrics.

### **File Analysis and Prediction:**

We focus on analyzing PE files to extract relevant features for prediction. We develop functions to compute entropy, extract resource information, and obtain version data from PE files using the pefile library. These features provide valuable insights into a file's characteristics, helping our machine learning models distinguish between malicious and legitimate files.

## **ANTiVirus Process:**

- **File Analysis:**

When a file is submitted for scanning, the AI-based antivirus system performs an in-depth analysis of its attributes, including file type, size, and metadata.

- **Behavioral Analysis:**

Next, the antivirus system executes the file within a controlled environment to observe its behavior. This dynamic analysis helps identify suspicious activities indicative of malware behavior.

- **Machine Learning:**

the trained machine learning model, the antivirus system compares the file's characteristics and behavior against known malware patterns. This allows it to make informed decisions regarding the file's trustworthiness.

- **Threat Classification:**

Based on the analysis results, the file is classified as either unmalicious or malicious.

"In our AI model , the selection of appropriate machine learning algorithms is paramount. Each algorithm brings unique capabilities and characteristics to the table, allowing for a comprehensive approach to malware detection and classification. Ultimately, the algorithm yielding the highest accuracy in classification emerges as the winner, ensuring optimal performance in identifying and mitigating potential threats. To Achieve it **we used a five Algorithms :**

1. Decision Trees
2. Random Forests
3. Gradient Boosting
4. AdaBoost
5. Gaussian Naive Bayes (GNB)"

## Explanation of Algorithm Selection:

### **1. Decision Trees:**

Decision trees are chosen for their simplicity and interpretability. They partition the feature space into a set of simple decision rules, making them easy to understand and visualize. Despite their tendency to overfit, ensemble methods like random forests and gradient boosting can mitigate this issue.

## **2. Random Forests:**

Random forests are an ensemble learning method that builds multiple decision trees during training. By averaging the predictions of individual trees, random forests reduce overfitting and improve generalization. They are robust to noise and outliers and can handle high-dimensional data effectively.

## **3. Gradient Boosting:**

Gradient boosting builds an ensemble of weak learners (typically decision trees) in a sequential manner, where each new tree corrects errors made by the previous ones. This iterative process minimizes the overall error and results in a powerful predictive model. Gradient boosting is known for its high accuracy and robustness.

## **4. AdaBoost:**

AdaBoost, short for Adaptive Boosting, is another ensemble method that combines multiple weak classifiers to create a strong classifier. It assigns higher weights to misclassified instances, forcing subsequent classifiers to focus on the difficult cases. AdaBoost is effective in handling imbalanced datasets and can adapt to complex decision boundaries.

## **5. Gaussian Naive Bayes (GNB):**

Naive Bayes is a probabilistic classifier based on Bayes' theorem with strong independence assumptions between features.

Gaussian Naive Bayes assumes that continuous features follow a Gaussian distribution, making it suitable for classification tasks with numerical data. It is computationally efficient and performs well on large datasets with high dimensionality.

## **conclusion:**

Our AI-based antivirus system significantly enhances malware detection by using machine learning to analyze file attributes and behavior. This approach improves accuracy and adaptability, crucial for defending against evolving threats.

By focusing on effective feature extraction and optimal algorithm selection based on classification accuracy, our system is efficient and reliable. The use of a Flask web service ensures easy integration and real-time analysis.

In summary, our AI-based antivirus system provides a robust and dynamic defense, essential for protecting digital assets in today's complex threat landscape.

# Chapter 5: Implementation

# Web Scanner Components :

## 1)Parser Class Documentation:

### Introduction:

The `Parser` class is designed to extract specific patterns, such as IP addresses, emails, URLs, phone numbers, credit cards, and various types of secrets, from a given text body.

### 1-Constructor:

#### **constructor(body: string)**

Creates a new instance of the `Parser` class with the provided text body.

### 2-Properties

#### **ipAddresses: Set<string>**

Returns a set of unique IP addresses found in the text body.

#### **emails: Set<string>**

Returns a set of unique email addresses found in the text body.

#### **urls: Set<string>**

Returns a set of unique URLs found in the text body.

#### **phoneNumbers: Set<string>**

Returns a set of unique phone numbers found in the text body.

#### **creditCards: Set<string>**

Returns a set of unique credit card numbers found in the text body.

### **topSecrets: Set<PatternResult>**

Returns a set of top-secret patterns, including various API keys, tokens, and credentials, along with their confidence levels.

### **secrets: Set<PatternResult>**

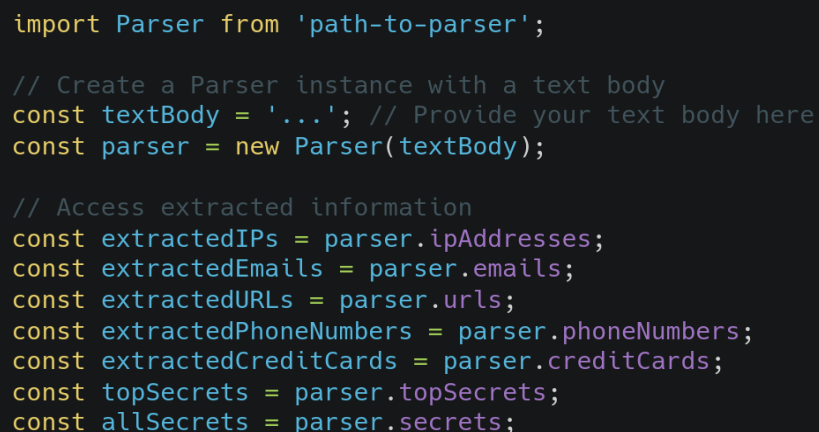
Returns a set of all secret patterns and their corresponding values, along with confidence levels.

### 3-Methods:

#### **#extractByPattern(pattern: string): Set<string>**

Private method used to extract values based on a given pattern. It takes a pattern name as an argument and returns a set of unique matches.

### 4-Example Usage:



```
import Parser from 'path-to-parser';

// Create a Parser instance with a text body
const textBody = '...'; // Provide your text body here
const parser = new Parser(textBody);

// Access extracted information
const extractedIPs = parser.ipAddresses;
const extractedEmails = parser.emails;
const extractedURLs = parser.urls;
const extractedPhoneNumbers = parser.phoneNumbers;
const extractedCreditCards = parser.creditCards;
const topSecrets = parser.topSecrets;
const allSecrets = parser.secrets;
```



# HtmlParser Class Documentation

## Introduction

The `HtmlParser` class extends the functionality of the `Parser` class, specializing in parsing HTML content. It provides methods to extract links, source JavaScript scripts, and information about HTML forms from the given HTML body.

## 1-Constructor

### **constructor(body: string, newUrl?: string)**

Creates a new instance of the `HtmlParser` class with the provided HTML body and an optional new URL. It utilizes the Cheerio library for HTML parsing.

## 2-Properties

### **links: Set<string>**

Returns a set of unique links found within the HTML content. Resolves relative URLs using the base URL provided during instantiation.

### **sourceJsScripts: Set<string>**

Returns a set of unique URLs for JavaScript scripts included in the HTML content. Resolves relative script source URLs using the base URL provided during instantiation.

### **forms: Array<Form>**

Returns an array of objects representing HTML forms found in the content. Each form object contains information such as URL, action, method, and parameters.

### 3-Methods

**getNodeBySelector(selector: string):**

**cheerio.Cheerio<cheerio.AnyNode>**

Returns a Cheerio object representing the HTML elements matching the provided selector.

**#extractByPattern(pattern: string): Set<string>**

Inherited from the parent Parser class, this private method extracts values based on a given pattern. It takes a pattern name as an argument and returns a set of unique matches.

### 4-Example Usage:

```
import HtmlParser from 'path-to-html-parser';
// Create an HtmlParser instance with HTML content and an optional URL
const htmlContent = '...'; // Provide your HTML content here
const baseUrl = 'https://example.com'; // Optional base URL for relative URL resolution
const htmlParser = new HtmlParser(htmlContent, baseUrl);

// Access extracted information
const extractedLinks = htmlParser.links;
const extractedJsScripts = htmlParser.sourceJsScripts;
const extractedForms = htmlParser.forms;

// Access specific HTML elements by selector
const specificElement = htmlParser.getNodeBySelector('your-selector');

// Example iteration through extracted forms
for (const form of extractedForms) {
  console.log('Form URL:', form.url);
  console.log('Form Action:', form.action); console.log('Form Method:', form.method);
  console.log('Form Parameters:', form.parameters);
  console.log('-----');
}
```

# XmlParser Class Documentation

## Introduction

The `XmlParser` class extends the functionality of the `Parser` class, specializing in parsing XML content. It uses the Cheerio library with XML-specific options to parse XML data and provides a method to retrieve XML nodes by a given selector.

## 1-Constructor

### **constructor(body: string)**

Creates a new instance of the `XmlParser` class with the provided XML body. It utilizes Cheerio with XML-specific settings (`xml: true, xmlMode: true`) for proper XML parsing.

## 2-Properties

### **getNodeBySelector(selector: string): cheerio.Cheerio<cheerio.AnyNode>**

Returns a Cheerio object representing the XML elements matching the provided selector.

## 3-Methods

### **#extractByPattern(pattern: string): Set<string>**

Inherited from the parent `Parser` class, this private method extracts values based on a given pattern. It takes a pattern name as an argument and returns a set of unique matches.

## 4-Example Usage:



```
import XmlParser from 'path-to-xml-parser';

// Create an XmlParser instance with XML content
const xmlContent = '...'; // Provide your XML content here
const xmlParser = new XmlParser(xmlContent);

// Access specific XML elements by selector
const specificElement = xmlParser.getNodeBySelector('your-selector');

// Example of using the extractByPattern method
const extractedValues = xmlParser.extractByPattern('your-pattern');

// Iterate through extracted values
for (const value of extractedValues) {
  console.log('Extracted Value:', value);
}
```

## HTTP Client:

The HTTPClient module is a comprehensive HTTP client library designed to facilitate making HTTP requests using either Axios or Puppeteer. It provides an abstraction over the underlying HTTP clients, allowing for easy integration into various projects. This documentation outlines the key components and functionalities of the HTTP client.

## HTTP Request and Response Types:

### HTTPRequest:

The HTTPRequest type defines the structure of an HTTP request.

```
type HTTPRequest = {  
  method: string;  
  url: string;  
  headers?: Record<string, string>;  
  body?: any;  
};
```

- method: The HTTP method for the request (e.g., "GET", "POST").
- url: The URL of the request.
- headers: Optional headers for the request.
- body: Optional request body.

### HTTPResponse:

The HTTPResponse type defines the structure of an HTTP response.

```
type HTTPResponse = {  
  statusCode: number;  
  headers: Record<string, string>;  
  body: any;  
};
```

- statusCode: The HTTP status code of the response.
- headers: Response headers.
- body: Response body.

## HTTP Client Configuration:

### HTTP ClientConfig:

The HTTPClientConfig interface specifies the configuration options for the HTTP client.

```
interface HTTPClientConfig {  
    client: "axios" | "puppeteer";  
    proxyUrl?: string;  
}
```

- client: Specifies the HTTP client to use, either "axios" or "puppeteer".
- proxyUrl: Optional. The URL of the proxy server to be used for requests.

## HTTP Client:

### Constructor:

The HTTPClient class is initialized with an optional configuration.

```
class HTTPClient {  
    constructor(config?: HTTPClientConfig);  
}
```

- config: An optional configuration object of type HTTPClientConfig.

### Axios Configuration:

- The #setupAxios method configures Axios based on the provided options.

## Puppeteer Configuration:

The `#setupPuppeteer` method configures Puppeteer based on the provided options.

## Request Method:

The request method sends an HTTP request and returns a promise that resolves to an `HTTPResponse`.

```
async request(req: HTTPRequest): Promise<HTTPResponse>;
```

-req: The HTTP request object.

## Conclusion:

The `HTTPClient` module simplifies making HTTP requests by providing a unified interface and supporting both `Axios` and `Puppeteer`. It is configurable and flexible, making it suitable for a variety of use cases.

## 2)Crawler Class Documentation:

### Introduction:

The `Crawler` class is designed to crawl web pages starting from a specified entrypoint, fetching and parsing HTML content, and following links according to a set of policies. It utilizes an `HTTPClient` for making HTTP requests and an `HtmlParser` for extracting links from HTML content.

### 1-Constructor:

**`constructor(entrypoint: string, policy?: RegExp[], httpClientConfig?: HTTPClientConfig)`**

Creates a new instance of the `Crawler` class with the provided entrypoint URL, optional crawl policies (`RegExp` array), and an optional configuration for the underlying `HTTPClient`.

## 2-Properties:

**status: 'working' | 'stopped' | 'finished'**

Returns the current status of the crawler, indicating whether it is working, stopped, or finished.

**successRequests: Map<string, HTTPResponse>**

Returns a map containing successful HTTP requests with URLs as keys and corresponding HTTPResponse objects as values.

**failedRequests: Map<string, string>**

Returns a map containing failed HTTP requests with URLs as keys and error messages as values.

## 3-Methods:

**- startCrawl()**

Starts the crawl process, fetching and parsing web pages and following links according to the defined policies.

**-stopCrawl()**

Stops the crawl process, pausing the ongoing crawl.

**-continueCrawl()**

Resumes the crawl process if it was previously stopped.

**-finishCrawl()**

Finishes the crawl process, clearing the list of queued requests.

## Private Methods:

**#isVisitedEndpoint(currentUrl: string): boolean**

Checks if a given URL has already been visited (successfully or unsuccessfully).



## **#crawl()**

The main crawl loop that iteratively fetches and processes web pages until there are no more queued requests or the crawl is stopped.

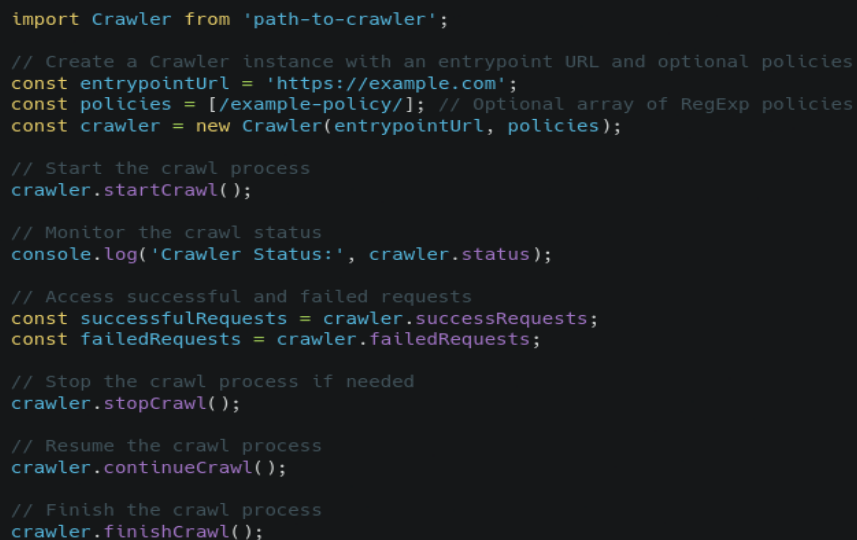
## **#fetchEndpoint(url: string): Promise<HTTPResponse>**

Fetches the content of a specified URL using the underlying HTTPClient instance.

## **#isPolicySatisfied(url: string): boolean**

Checks if the provided URL satisfies any of the crawl policies.

## 4-Example Usage:



```
import Crawler from 'path-to-crawler';

// Create a Crawler instance with an endpoint URL and optional policies
const endpointUrl = 'https://example.com';
const policies = [/example-policy/]; // Optional array of RegExp policies
const crawler = new Crawler(endpointUrl, policies);

// Start the crawl process
crawler.startCrawl();

// Monitor the crawl status
console.log('Crawler Status:', crawler.status);

// Access successful and failed requests
const successfulRequests = crawler.successRequests;
const failedRequests = crawler.failedRequests;

// Stop the crawl process if needed
crawler.stopCrawl();

// Resume the crawl process
crawler.continueCrawl();

// Finish the crawl process
crawler.finishCrawl();
```

## PathTraversalScanner Class Documentation:

The PathTraversalScanner class is designed to scan web applications for **path traversal vulnerabilities**. It leverages the Crawler class to handle HTTP requests and responses, utilizing a list of payloads to test various URLs for potential vulnerabilities.

### 1-Constructor:

#### **constructor(baseTestUrl: string)**

Creates a new instance of the `PathTraversalScanner` class with the provided base URL for testing.

- **Parameters:** `baseTestUrl` (string): The base URL of the web application to be scanned.

### 2-Properties

- **#crawler** (`Crawler`): An instance of the `Crawler` class used to manage HTTP requests and responses.
- **#baseTestUrl** (`string`): The base URL of the web application to be scanned.

### 3-Methods

#### **scan(): Promise<void>**

Starts the scanning process, reading payloads from a dictionary file and testing each payload against the base URL.

- Returns:
  - `Promise<void>`: A promise that resolves when the scan is complete.

## Private Methods:

### **#appendPayloadToUrl(baseUrl: string, payload: string): string**

Appends a payload to the base URL, modifying URL parameters to test for path traversal vulnerabilities.

#### **-Parameters:**

- **baseUr1** (string): The base URL to which the payload will be appended.
- **payload** (string): The payload to be appended to the URL.

#### **-Returns:**

- **string**: The modified URL with the appended payload.

### **#testPathTraversal(url: string): Promise<void>**

Tests a URL for path traversal vulnerabilities by sending an HTTP request and logging the response status.

#### **-Parameters:**

- **url** (string): The URL to be tested for path traversal vulnerabilities.

#### **-Returns:**

- **Promise<void>**: A promise that resolves when the test is complete.

## 4-Example Usage:

```
import PathTraversalScanner from 'path-to-scanner';

// Define the base URL for testing
const baseTestUrl = 'http://testphp.vulnweb.com';

// Create a PathTraversalScanner instance with the base URL
const scanner = new PathTraversalScanner(baseTestUrl);

// Start the scanning process
scanner.scan();
```

## 3)SQLi Class Documentation

### Introduction:

The **SQLi** class is designed to scan web applications for SQL injection vulnerabilities. It uses a combination of HTTP requests and HTML form parsing to identify and test potential injection points.

### 1-Constructor

#### **constructor(httpClientConfig?: HTTPClientConfig)**

Creates a new instance of the **SQLi** class with an optional configuration for the HTTP client.

#### **-Parameters:**

- **httpClientConfig** (HTTPClientConfig): Optional configuration for the HTTP client. If not provided, a default configuration using **axios** will be used.

## 2-Properties

- **sqlErrors** (`string[]`): An array of SQL error messages used to identify SQL injection vulnerabilities.
- **httpClient** (`HttpClient`): An instance of the `HttpClient` class used to make HTTP requests.
- **payloads** (`string[]`): An array of payloads used to test for SQL injection vulnerabilities.

## 3-Methods:

### 1)Public Methods:

#### **sqlInjectionScan(url: string): Promise<void>**

Starts the SQL injection scanning process for the specified URL.

#### **-Parameters:**

- `url` (string): The URL of the web application to be scanned.

#### **-Returns:**

- `Promise<void>`: A promise that resolves when the scan is complete.

### 2)Private Methods:

#### **#getFormsFromCrawler(crawler: Crawler): Promise<any[]>**

Uses the `Crawler` class to crawl the specified URL and extract forms from the HTML content.

#### **-Parameters:**

- `crawler` (Crawler): An instance of the `Crawler` class used to crawl the URL.

### -Returns:

- `Promise<any[]>`: A promise that resolves with an array of forms found on the crawled pages.

### #vulnerable(response: HTTPResponse): boolean

Checks if a given HTTP response indicates a SQL injection vulnerability based on known SQL error messages.

### -Parameters:

- `response` (HTTPResponse): The HTTP response to be checked.

### -Returns:

- `boolean`: `true` if the response indicates a SQL injection vulnerability, otherwise `false`.

## 4-Example Usage

```
import SQLi from 'path-to-sqli';

// Define the URL for testing
const testUrl = 'http://testphp.vulnweb.com';

// Create an instance of the SQLi scanner
const scanner = new SQLi();

// Start the SQL injection scan
scanner.sqlInjectionScan(testUrl);
```

# Network Scanner Components :

## Port Scanner:

The PortScanner module is a utility for scanning TCP and UDP ports on a target host. It provides functions to check whether a specific port or a range of ports is open. This documentation outlines the key components and functionalities of the PortScanner class.

### Port Type:

The Port type defines the structure of a network port.

```
type Port = {  
    portType: 'tcp' | 'udp';  
    portNumber: number;  
};
```

- portType: Specifies the type of the port, either 'tcp' or 'udp'.
- portNumber: The port number.

### Constructor

The PortScanner class is initialized with the target host.

```
class PortScanner {  
    constructor(targetHost: string);  
}
```

- targetHost: The host to scan for open ports.

### TCP Port Scanning:

- The scanTCPPort method asynchronously scans a TCP port for openness.

```
async scanTCPPort(port: number, timeout: number = 1000): Promise<boolean>;
```

- port: The TCP port to scan.
- timeout: Optional timeout for the connection attempt in milliseconds.

## UDP Port Scanning:

The scanUDPPort method asynchronously scans a UDP port for openness.

```
async scanUDPPort(port: number): Promise<boolean>;
```

- port: The UDP port to scan.

## General Port Scanning:

The scanPort method asynchronously scans a port based on the specified protocol (TCP or UDP).

```
async scanPort(protocol: 'tcp' | 'udp', port: number, timeout: number = 1000):  
Promise<boolean>;
```

- protocol: The protocol to use for scanning ('tcp' or 'udp').
- port: The port to scan.
- timeout: Optional timeout for the connection attempt in milliseconds.

## Scan All Ports:

The scanAllPortsInRange and scanAllPorts methods asynchronously scan a range of ports or all ports for openness.

```
async scanAllPortsInRange(protocol: 'tcp' | 'udp', startPort: number, endPort:  
number, timeout: number = 1000): Promise<void>;
```

```
async scanAllPorts(timeout: number = 1000): Promise<void>;
```

- protocol: The protocol to use for scanning ('tcp' or 'udp').
- startPort: The starting port of the range.
- endPort: The ending port of the range.



- timeout: Optional timeout for the connection attempt in milliseconds.

## Open Ports:

The openPorts getter returns an array of open ports.

```
get openPorts(): Port[];
```

## Conclusion:

The PortScanner module simplifies the process of scanning ports on a target host, allowing developers to check for open ports based on TCP or UDP protocols. It provides flexibility in scanning individual ports or entire ranges.

```

class PortScanner {
  #openPorts: Array<Port> = [];
  #targetHost: string;

  constructor(targetHost: string) {
    this.#targetHost = targetHost;
  }

  async scanTCPPort(port: number, timeout: number = 1000): Promise<boolean> {
    return new Promise((resolve) => {
      const socket = new net.Socket();

      socket.on('connect', () => {
        this.#openPorts.push({ portType: 'tcp', portNumber: port });
        socket.end();
        resolve(true);
      });

      socket.on('error', (_) => {
        resolve(false);
      });

      socket.setTimeout(timeout, () => {
        socket.end();
        resolve(false);
      });

      socket.connect(port, this.#targetHost);
    });
  }

  async scanUDPPort(port: number): Promise<boolean> {
    return new Promise((resolve) => {
      const socket = dgram.createSocket('udp4');
      socket.on('message', () => {
        this.#openPorts.push({ portType: 'udp', portNumber: port });
        socket.close();
        resolve(true);
      });

      socket.on('error', (_) => {
        resolve(false);
      });

      const message = Buffer.from('Are You Opened? :)');
      socket.send(message, port, this.#targetHost);
    });
  }

  async scanPort(protocol: 'tcp' | 'udp', port: number, timeout: number = 1000): Promise<boolean> {
    if (protocol === 'tcp') {
      return await this.scanTCPPort(port, timeout);
    } else if (protocol === 'udp') {
      return await this.scanUDPPort(port);
    } else {
      return false;
    }
  }

  async scanAllPortsInRange(protocol: 'tcp' | 'udp', startPort: number, endPort: number, timeout: number = 1000): Promise<void> {
    for (let port = startPort; port <= endPort; port++) {
      await this.scanPort(protocol, port, timeout);
    }
  }

  async scanAllPorts(timeout: number = 1000): Promise<void> {
    await this.scanAllPortsInRange('tcp', 1, 65535, timeout);
    await this.scanAllPortsInRange('udp', 1, 65535, timeout);
  }

  get openPorts(): Port[] {
    return this.#openPorts;
  }
}

export default PortScanner;

```

# Network Vulnerability Scanner :

## NetworkScanner Class:

The **NetScanner** class is designed to perform network scans on a given host. Below are the signatures and brief descriptions of its methods and properties.

### Constructor

#### **constructor(host: string)**

**-Parameters:** **host:** The host to be scanned (type: **string**).

**-Description:** Initializes a new instance of the **NetScanner** class with the specified host. It also sets the scan time to the current date and time.

### Methods:

#### **scan(): Promise<boolean>**

**-Returns:** A **Promise<boolean>** that resolves to **true** if the scan is successful, or rejects with an error if the scan fails.

**-Description:** Performs a network vulnerability scan on the specified host using the OpenPorts Result **with** vulnerDB api various options and outputs the results to an XML file. The XML file is named based on the host and the scan time.

## Properties:

### **result: NetworkScanResult | undefined**

**-Returns:** A `NetworkScanResult` object if the scan results XML file exists and is valid, otherwise `undefined`.

**-Description:** Retrieves the results of the network scan from the XML file generated. The results include the host, OS name, and an array of port scan results, each containing the protocol, service, port number, and any associated CVE identifiers.

---

## Types:

### **NetworkScanResult:**

- **Properties:**
  - o `host`: The scanned host (type: `string`).
  - o `os`: The name of the operating system (type: `string`).
  - o `ports`: An array of `PortScanResult` objects.

### **PortScan Result:**

- **Properties:**
  - o `protocol`: The protocol used (type: `"tcp" | "udp"`).
  - o `service`: The name of the service running on the port (type: `string`).
  - o `portNumber`: The port number (type: `number`).
  - o `cve`: An array of CVE identifiers (type: `string[]`).

```

get result(): NetworkScanResult | undefined {
    if (fs.existsSync(`./${this.#host}-${this.#scanTime}.xml`)) {
        let xmlReport: string = fs.readFileSync(`./${this.#host}-${this.#scanTime}.xml`, "utf8");
        let jsonReport: object = JSON.parse(toJson(xmlReport));
        if (!jsonReport['nmaprun']['host']) {
            return
        }
        let osName: string = ''
        if (jsonReport['nmaprun']['host']['os'] && jsonReport['nmaprun']['host']['os']['name']) {
            osName = jsonReport['nmaprun']['host']['os']['name'];
        }
        let portsReport: object[] = jsonReport['nmaprun']['host']['ports']['port'];
        let PortScanResults: PortScanResult[] = []
        portsReport?.forEach((port) => {
            let service: string = port['service']['name'];
            let protocol: "tcp" | "udp" = port['protocol'];
            let portNumber = parseInt(port['portid']);
            let cve: string[] = []

            if (port['script'] && port['script']['table'] && port['script']['table']['table']) {
                let vulnersReport: object[] = port['script']['table']['table'];

                vulnersReport.forEach((result: any) => {
                    let id: string = ''
                    let isCve: boolean = false
                    result['elem'].forEach((element: object) => {

                        if (element['key'] == "id") {
                            id = element['$t']
                        }
                        if (element['key'] == "type" && element['$t'] == 'cve') {
                            isCve = true
                        }
                    })
                    if (isCve) {
                        cve.push(id as string)
                    }
                })
            }
            PortScanResults.push({
                protocol,
                service,
                portNumber,
                cve
            });
        })
        return {
            host: this.#host,
            os: osName,
            ports: PortScanResults
        }
    }
}

```

## NetWork Scan linked with front:

```
const socket = socketIOClient("http://localhost:3000");
const Target = () => {
  const { host } = useParams();
  const [item, setItem] = useState(null);
  const [loading, setLoading] = useState(true);
  const [scanFinished, setScanFinished] = useState(false);
  const [status, setStatus] = useState("");
  useEffect(() => {
    socket.emit("scanNet", host);
    setStatus("Scanning...");
    socket.on("scanNetResult", (data) => {
      if (data.message === "Scan Stopped") {
        setStatus("Scan Stopped");
      } else if (data.error) {
        setStatus(data.error);
      } else {
        setItem(data);
        setScanFinished(true);
        setStatus("Scan Completed");
      }
    });
    return () => {
      socket.off("scanNetResult");
    };
  }, [host]);
```

## Result on Front:

The screenshot shows a web application interface with a dark theme. At the top, there is a navigation bar with a back arrow, the text "Unixty", and a user profile icon labeled "User Name". Below the navigation bar, the main heading is "Scan Details For Target <192.168.1.1>". The interface is divided into two main sections. On the left, there is a table with two columns: "host" and "Status". The first row shows "192.168.1.1" and "Scan Completed". Below this, there are two more rows: "Scan-Type" with "Network Scan" and "Risk-Level" with "Safe". At the bottom of this section, there is a row for "Date" showing "5/28/2024, 2:11:03 AM" and a "Details" link. On the right, there is a "Details" tab. Below the tab, there are three rows of data, each representing a port scan result. Each row has four columns: "PortNumber:", "Protocol:", "Service:", and "CVEs:". The first row shows PortNumber: 53, Protocol: tcp, Service: tcpwrapped, and CVEs: null. The second row shows PortNumber: 80, Protocol: tcp, Service: http, and CVEs: null. The third row shows PortNumber: 443, Protocol: tcp, Service: https, and CVEs: null. At the bottom of the interface, there is a "Back to Home" button.

host	Status
192.168.1.1	Scan Completed

Scan-Type	Risk-Level
Network Scan	Safe

Date	Details
5/28/2024, 2:11:03 AM	

PortNumber:	Protocol:	Service:	CVEs:
53	tcp	tcpwrapped	null
80	tcp	http	null
443	tcp	https	null

[Back to Home](#)

## Webscanner Linked with front :

```
const socket = socketIO.connect("http://localhost:3000");

function WebScan() {
  const location = useLocation();
  const currentPath = location.pathname;
  const urlParts = currentPath.split("=");
  const lastPart = urlParts[urlParts.length - 1];
  const [status, setStatus] = useState("0");
  const [scanType, setScanType] = useState("");
  const [result, setResult] = useState([]);
  useEffect(() => {
    socket.emit("scanWeb", lastPart);
    socket.on("scanWebResult", (data) => {
      setScanType(data.scanType);
      setStatus(data.status);
      setResult(data.results);
      if (data.scanType === "Scan Finished" && data.status === 100) {
        socket.off("scanWebResult");
      }
    });
  });
}
```

## Result on Front:

3	http://testphp.vulnweb.com/categories.php	3 1 3	<a href="#">Show vulnerabilities details</a>																
4	http://testphp.vulnweb.com/robots.txt	1 1	<a href="#">Show vulnerabilities details</a>																
5	http://testphp.vulnweb.com/index.php	3 1 3	<a href="#">Hide vulnerabilities details</a>																
<div><div>Vulnerabilities that exist in this address</div><table><tr><td>Missing Anti-clickjacking Header</td><td>Medium</td><td>Charset Mismatch (Header Versus Meta Content-Type Charset)</td><td>Informational</td></tr><tr><td>Content Security Policy (CSP) Header Not Set</td><td>Medium</td><td>Absence of Anti-CSRF Tokens</td><td>Medium</td></tr><tr><td>Server Leaks Version Information via "Server" HTTP Response Header Field</td><td>Low</td><td>X-Content-Type-Options Header Missing</td><td>Low</td></tr><tr><td>Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)</td><td>Low</td><td></td><td></td></tr></table></div>				Missing Anti-clickjacking Header	Medium	Charset Mismatch (Header Versus Meta Content-Type Charset)	Informational	Content Security Policy (CSP) Header Not Set	Medium	Absence of Anti-CSRF Tokens	Medium	Server Leaks Version Information via "Server" HTTP Response Header Field	Low	X-Content-Type-Options Header Missing	Low	Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low		
Missing Anti-clickjacking Header	Medium	Charset Mismatch (Header Versus Meta Content-Type Charset)	Informational																
Content Security Policy (CSP) Header Not Set	Medium	Absence of Anti-CSRF Tokens	Medium																
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	X-Content-Type-Options Header Missing	Low																
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low																		
6	http://testphp.vulnweb.com/sitemap.xml	1 1	<a href="#">Show vulnerabilities details</a>																
7	http://testphp.vulnweb.com/cart.php	6 2 8	<a href="#">Show vulnerabilities details</a>																
8	http://testphp.vulnweb.com/disclaimer.php	3 1 3	<a href="#">Hide vulnerabilities details</a>																
<div><div>Vulnerabilities that exist in this address</div></div>																			



# AI Implementation :

## Description:

In this part, we will discuss the implementation details of a machine learning model designed to classify files as either unmalicious or malicious based on their features. The model utilizes a dataset containing various attributes extracted from files, such as header information, resource details, and section characteristics. We will cover the following aspects of the implementation:

### 1. Dataset :

#### Description:

The dataset provided is a comprehensive collection of executable files and their respective features, specifically designed to train and evaluate machine learning models for antivirus and malware detection. This dataset includes detailed attributes extracted from executable files, which are critical for distinguishing between legitimate software and potentially harmful malware.

	Name	md5	Machine	\
0	memtest.exe	631ea355665f28d4787448e442fbf5b8	332	
1	ose.exe	9d10f99a5712e28f8acd5641e3a7eae6b	332	
2	setup.exe	4092f518527353cedb88a70fddcfcd390	332	
3	Dw20.EXE	a41e524f8d45f0074fd07805ff8c9b12	332	
4	datafig20.exe	c87e561258f2f8550c0f990bf643a731	332	

	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	\
0	224	258	9	
1	224	3330	9	
2	224	3330	9	
3	224	258	9	
4	224	258	9	

	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	\
0	0	361984	115712	
1	0	138560	19968	
2	0	517120	621568	
3	0	585728	369152	
4	0	294912	247296	

	SizeOfUninitializedData	...	ResourcesNb	ResourcesMeanEntropy	\
0	0	...	4	3.262823	
1	0	...	2	4.258461	
2	0	...	11	4.426324	
3	0	...	10	4.364291	
4	0	...	2	4.386100	

	ResourcesMinEntropy	ResourcesMaxEntropy	ResourcesMeanSize	\
0	2.568844	3.537939	8797.000000	
1	3.420744	5.080177	837.000000	
2	2.846449	5.271813	31102.272727	
3	2.609314	6.400720	1457.000000	
4	3.421598	5.190683	1074.500000	

	ResourcesMinSize	ResourcesMaxSize	loadConfigurationSize	\
0	216	18032	0	
1	518	1156	72	
2	104	270376	72	
3	90	4264	72	
4	840	1308	72	

	VersionInformationSize	legitimate
0	16	1
1	18	1
2	18	1
3	18	1
4	18	1

## 2. Data Preparation:

- Loading the dataset from a CSV file.
- Handling missing values and ensuring data integrity.
- Splitting the dataset into features (X) and target labels (y).

```
data = pd.read_csv('data.csv', sep='|')
X = data.drop(['Name', 'md5', 'legitimate'], axis=1).values
y = data['legitimate'].values

print('Searching important feature based on %i total features\n' % X.shape[1])
```

## 3. Feature Selection:

- Identifying important features using a tree-based classifier.
- Selecting relevant features for model training.

```
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel

fsel = ExtraTreesClassifier().fit(X, y)
model = SelectFromModel(fsel, prefit=True)
X_new = model.transform(X)
```

## 3. Model Training:

- Comparing performance across multiple algorithms.

- Selecting the best-performing algorithm based on evaluation metrics.

```
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2)

algorithms = {
    "DecisionTree": tree.DecisionTreeClassifier(max_depth=10),
    "RandomForest": ske.RandomForestClassifier(n_estimators=50),
    "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=50),
    "AdaBoost": ske.AdaBoostClassifier(n_estimators=100),
    "GNB": GaussianNB()
}

results = {}
for algo in algorithms:
    clf = algorithms[algo]
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    results[algo] = score

winner = max(results, key=results.get)
```

#### **4. Model Evaluation:**

- Assessing the model's performance using metrics like accuracy, false positive rate, and false negative rate.

```
clf = algorithms[winner]
res = clf.predict(X_test)
accuracy = accuracy_score(y_test, res)
```

#### **5. Model Persistence:**

- Saving the trained model and the selected features for future use.

```
import joblib

joblib.dump(clf, 'classifier/classifier.pkl')
open('classifier/features.pkl', 'wb').write(pickle.dumps(features))
```

## Accuracy :

```
▶ Searching important feature based on 54 total features
⇄ 13 features identified as important:
1. feature DllCharacteristics (0.128396)
2. feature Characteristics (0.126882)
3. feature Machine (0.111380)
4. feature VersionInformationSize (0.070518)
5. feature SectionsMaxEntropy (0.063467)
6. feature Subsystem (0.058488)
7. feature ResourcesMaxEntropy (0.047736)
8. feature MajorSubsystemVersion (0.047481)
9. feature ImageBase (0.044280)
10. feature SizeOfOptionalHeader (0.042549)
11. feature ResourcesMinEntropy (0.030500)
12. feature SizeOfStackReserve (0.023660)
13. feature SectionsMinEntropy (0.019091)

Testing algorithms...
DecisionTree : 98.996740 %
RandomForest : 99.297356 %
GradientBoosting : 98.750453 %
AdaBoost : 98.518653 %
GNB : 69.898587 %

Winner algorithm is RandomForest with a 99.297356 % success
Saving algorithm and feature list in classifier directory...
Saved...
False positive rate : 0.601067 %
False negative rate : 0.938515 %
```

## Linked With frontend:

```

reader.onload = async () => {
  const base64File = reader.result.split(',')[1];
  try {
    const response = await axios.post('https://unixity-ai-safebuffer.onrender.com/predict', {
      file: base64File,
    });
    setUploadStatus(`File is ${response.data.result}`);
    setShowUploadMessage(true); // Show upload message
    // Hide upload message after 5 seconds
    setTimeout(() => {
      setShowUploadMessage(false);
    }, 5000);
  } catch (error) {
    setUploadStatus('Error uploading file');
  }
};
};

```

## the Result on Front :

