

جامعة جدة
University of Jeddah

WEB VULNERABILITIES DETECTOR

Student information

Names	IDs
Abdullah Hussain Al-Zahrani	1845691
Mohannad Yousef Al-Hazmi	1845220
Saud Abdullatif Al-Zahrani	1845060

Supervisor: Dr. Naif Saeed Al-Zahrani

Abstract

Internet and web applications have become an essential part of our life. Nowadays, we have started doing our everyday life activities on the internet. For example, most people now prefer to do online shopping instead of going to a mall. Also, during the COVID-19 pandemic and the Quarantine, all services were shifted to be online. And The number of websites has increased during the pandemic. So people start to put sensitive information on these websites, but are these websites secure?

Web security is the essential step in creating a web application. Because if the attacker gets successful exploitation of a vulnerability on your web application, this would cost a lot of damage to your business. The impact of loss depends on the severity level of the exposure. In order to solve this problem and secure web applications, web vulnerability detectors exist. A web vulnerability detector is a platform that combines and integrates multiple open-source tools and some scripts that aim to discover various web application vulnerabilities. These open-source tools and scripts will be supported with GUI. The Graphical User Interface (GUI) will help non-technical computer users scan their web applications. The platform will scan the web application for several vulnerabilities and generate a detailed report about the problems found.

The scanning process consists of three phases. The first phase is Reconnaissance. In this phase, the platform will try to gather as much information about the target. The second phase is the scanning phase. All the information collected from phase one will be used to scan web applications and find every possible vulnerability. Finally, the third phase is reporting. The platform will generate a detailed report about all the problems encountered during the scanning phase.

This project was developed using Python programming language. The python program has several functions. These functions are used to combine and integrate the open-source tools and scripts.

This paper will discuss detailed information about the web vulnerabilities detector project.

Acknowledgment

University study was a delightful and full of educational experience. College of Computer Science and Engineering technology gives us a chance to learn and improve our skills in Cyber Security under a professional and good experienced teacher.

Senior projects allow us to practice and develop an actual project. In the senior project, we were given a chance to get a lot of information about several techniques to build a real project under the advice of Dr. Naif Saeed AL Zahrani. As a result, we learned to be more creative in solving the problem by looking at issues from different perspectives and generating multiple solutions, especially with non-expected errors and commands in the terminal. In addition, a senior project provides an opportunity to gain insight into the programs and techniques used by companies to develop and monitor their programs and work.

We are very thankful and grateful to the College of Computer Science and Engineering Technology for giving us the chance to practice our skills. Also, we are very thankful and great full to Dr. Naif Saeed Al Zahrani for advising us to complete our project.

Table of Contents

COVER PAGE	1
Abstract	2
Acknowledgment	3
Chapter 1: Planning.....	7
1.1 Description of the context	7
2.2 Aims and objective	7
3.3 Project Plan	8
Chapter 2: Problem Understanding	9
2.1. Stakeholders' definition	9
2.2. Detailed Description of the background (project domain).....	10
2.2.1. Problem Definition	10
2.2.2 why it is important	11
2.2.3 The Solution.....	12
2.2.4 what makes your solution different from other solutions.....	12
2.3 Literature Review.....	12
2.3.1 Cross-Site Scripting Attack (XSS).....	12
2.3.2 SQL injection Attack.....	13
2.3.3 Comparing the solutions	13
2.4 Comparison criteria definition	14
2.5 Comparison results and the feasibility	14
Chapter 3: Analysis.....	15
3.1. Functional and Non-functional requirement	15
3.1.1. Functional requirement	15
3.1.2. Non-functional requirement.....	16
3.2 Hardware Requirements.....	16
3.3. Use case diagram	17
3.4 Data collection instruments (Datalogger).....	17
Chapter 4: Design	18
4.1. System Architecture	18
4.1.1 Phase 1: Reconnaissance.....	18
4.1.2 Phase 2: Scanning.....	19
4.1.3 Phase 3: Reporting	19
4.2 Diagrams	20
4.3. User Interface Design	21

Chapter 5: Development.....	24
5.1 APIs and Plugins	24
5.1.1 Phase 1: Reconnaissance.....	25
5.1.2 Phase 2: Scanning.....	26
5.2 Implementation/Coding	28
5.2.1 Platform interface	28
5.2.2 Front-End JavaScript.....	31
5.2.3 Back-End PHP	32
5.2.4 Server-side	35
5.3 Testing.....	42
Chapter 6: Testing the developed artifact.....	43
6.1 Data collection.....	43
6.2 Clean the data and prepare for analysis.....	43
6.3 Analyze the collected data using the appropriate method and tool	43
6.4 Visualize the results	44
References	45

Table of Figures

Figure1 : Project plan.....	8
Figure2 : US-CERT (Computer Emergency Readiness Team) record of the vulnerabilities over the years 2011-2020 [4], [5]	10
Figure3 UseCase Diagram	17
Figure 4 System Architecture.....	20
Figure 5 Main-Menu Page	21
Figure 6 Target Options Page	22
Figure 7 Dashboard Page (Statistical Page).	23
Figure8 Home page	29
Figure 9 Scan page	30
Figure10 Result page	30
Figure 11 Code screenshot	31
Figure 12 Code screenshot	31
Figure 13 Code screenshot	32
Figure 14 Code screenshot	32
Figure 15 Code screenshot	32
Figure 16 Target Options Page	33
Figure 17 Code screenshot	33
Figure 18 Code screenshot	33
Figure19 Code screenshot	34

Figure 20 Code screenshot	35
Figure 21 Code screenshot	35
Figure 22 Code screenshot	36
Figure 23 Code screenshot	36
Figure 24 Code screenshot	37
Figure 25 Code screenshot	37
Figure 26 Code screenshot	38
Figure 27 Code screenshot	39
Figure 28 Code screenshot	40
Figure 29 Code screenshot	40
Figure 30 Code screenshot	41
Figure 31 Code screenshot	41
Figure 32 Code screenshot	44

Table of Tables

Table1: Stakeholder	9
Table 2.....	15
Table3 : Non-functional	16
Table 4 APIs	24
Table 5 Performance	42

Chapter 1: Planning

1.1 Description of the context

[1] Nowadays, there are many challenges on the internet. One of these challenges is web security. [2] Web security is protocols and measures organizations implement to protect from cybercriminals and threats. Web security should be a top priority for every organization. Along with email, the web is one of the primary cyber-attack mediators. The web and DNS services are part of 91% of all malware attacks, and email and the web are an essential part of 99% of successful breaches. While the importance of web security is undisputed, protecting against web security threats is becoming more complex every day. IT security organizations face serious challenges when trying to secure the web. Hackers will always find a new way to intrude on the web to breach websites, steal sensitive data, or stop organization business.

2021 statistics of cyber security breaches [3]:

- Thirty thousand websites get hacked every day.
- Every 39 seconds, there is a new attack on the web.
- 63% of cyber security breaches were financially motivated

During the COVID-19 pandemic, the government locked down, and the services shifted to be online. People use web applications for everything, from food delivery to governmental services. Users shared their sensitive information on web applications, such as personal information and credit cards. After the pandemic began, the security breaches increased, and people started to find their personal information leaked on the internet. The cause behind this is the little knowledge about web application security from organizations and individuals. The security design of these web applications is weak and does not fulfill security standards. Our project aims to increase web application security by creating a website to scan other websites for vulnerabilities.

2.2 Aims and objective

We will develop an integrated platform that combines open-source tools and locally written scripts supported with GUI to discover various vulnerabilities. As described above, the primary purpose of this project is to help web administrators to detect and prevent the exposure before the adversary does.

- Ensure that systems, services, and information must be accessible whenever required
- Addressing web-based vulnerability and risks

- Identify weaknesses points that could be a compromise
- Identify the safeguards that should be implemented to deal with such threats
- Create a single platform that can act behave of several other tools and provide multiple services
- facilitate the process of discovering and addressing the vulnerability
- Facilitate the usage of vulnerabilities detection tools

3.3 Project Plan

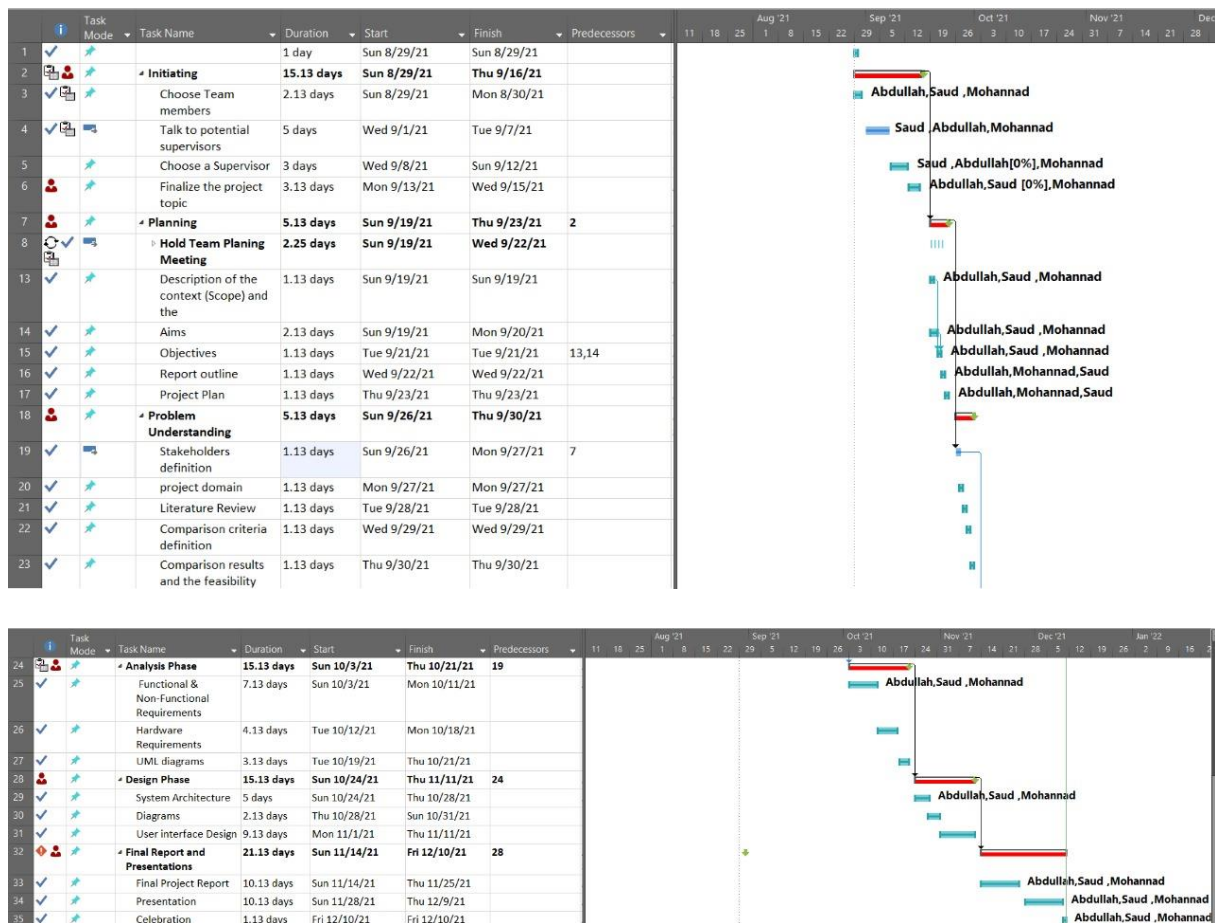


Figure1 : Project plan

Chapter 2: Problem Understanding

2.1. Stakeholders' definition

Stakeholder	Role	Interest	Communication methods	Responsibility
Jeddah University	Owner	<ul style="list-style-type: none"> - Deliver phases with time-consideration - Deliver the project at the end of the semester 	<ul style="list-style-type: none"> - Online meetings - Presentation - Written communication 	<ul style="list-style-type: none"> - Project Guidelines - Project Assessment
Dr. Naif Al-Zahrani	Supervisor	<ul style="list-style-type: none"> - Complete the project within specified criteria - Achieve target 	<ul style="list-style-type: none"> - Online meeting - Presentation - University meeting - Written communication 	<ul style="list-style-type: none"> - Project Supervision - Project Guidelines - Project Clarification - Project Assessment
Team members	Developer	<ul style="list-style-type: none"> - Complete the project - Meet all requirements 	<ul style="list-style-type: none"> - Communication Group - Written communication 	<ul style="list-style-type: none"> - Project Development - Project Delivery

Table1: Stakeholder

2.2. Detailed Description of the background (project domain)

2.2.1. Problem Definition

The internet-made web-based application is the most cyberattack suffered because of the ease of accessibility of the internet nowadays. However, due to the lack of security education, training, and awareness among the user, attackers may conduct many attacks to exploit vulnerabilities within the environment. Furthermore, there are no professional requirements to perform web-based attacks because of tools, frameworks, and written-exploitable code. Therefore, attackers can easily use it and compromise web applications in the network. Therefore, to detect and prevent these attacks, there is a high need to implement a defense mechanism for such attacks. However, this produces a new problem: the complexity of the usage, which is the primary purpose of our project.

Many governments and non-profit cybersecurity organizations have conducted research and statistics on vulnerabilities on the internet over the years. Unfortunately, these statistics have shown that attacks constantly increase and find it difficult to control, as shown in figure 1.

CVSS Severity Distribution Over Time

This visualization is a simple graph which shows the distribution of vulnerabilities by severity over time. The choice of LOW, MEDIUM and HIGH is based upon the CVSS V2 Base score. For more information on how this data was constructed please see the NVD CVSS page .

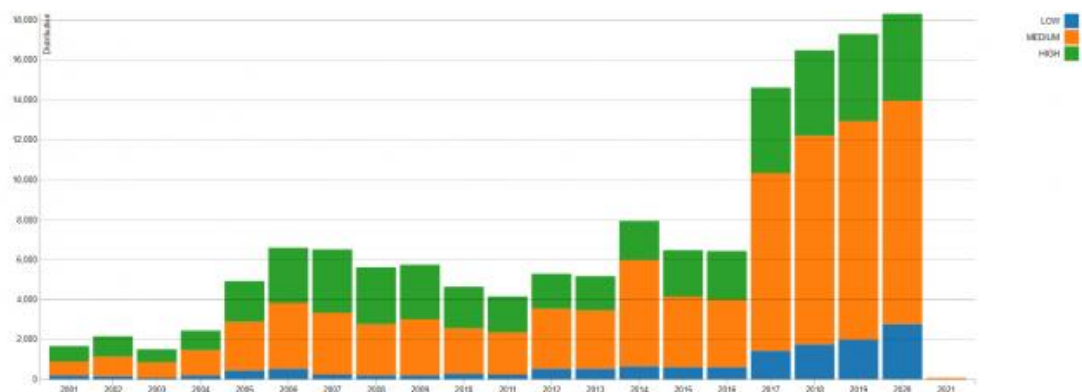


Figure2 : US-CERT (Computer Emergency Readiness Team) record of the vulnerabilities over the years 2011-2020 [4], [5]

2.2.2 why it is important

The importance consists of three categories: confidentiality, integrity, and availability. Confidentiality could be exposed when the attacker gets unauthorized access and escalate privilege in the web application. Thus, integrity would be threatened by the attacker's data modifications, and the availability wouldn't exist when the attacker denies the service from the web application.

Data leakage is a critical threat to organizations and governments. Loss of sensitive data can lead to reputational and financial losses that may affect an organization's long-term stability. The most common type of data leakage is customer/employee information such as National IDs, Credit cards, and personal credentials. According to IBM's 2016 Cost of Data Breach Study, the average cost loss of data leakage is \$4 million. The researcher Juniper Research's forecast expects that the global cost losses of data breaches in 2019 will be more than \$2.1 trillion[6]. Due to the technological leap and the number of records stored on the internet has increased.

Moreover, data breaches cost organizations millions of dollars. In 2016 Yahoo reported that 500 million accounts were stolen in 2014. In 2020 over 280 million Microsoft customer support records were stored unsecured in the database. In 2020 over 500,000 zoom accounts were found for sale on the dark web[7].

Since the data volume stored on the internet is growing and has become a critical threat to every person, cybercriminals always search for new techniques and methods to compromise systems, web applications, and networks. Cybercriminals will be aware of the most common security vulnerability in the web application. Why web application? The web application is the end-user interface. A regular user will use the features of web applications without knowing the security level of these futures.

On the other hand, cybercriminals scan these features looking for vulnerability. For instance, the typical user will enter his credentials and log in on the login page. Still, the attacker will use the credentials parameter and perform a SQL injection to access the database.

Nowadays, companies implant new futures to their web application to attract users or market businesses. So these futures become an essential part of project succession, but More Codes = More bugs. Hence, the higher furthers in a web application, the higher chance of getting comprised.

2.2.3 The Solution

First, we will develop a platform that combines multiple open source tools and locally written scripts supported with GUI that aims to discover various web application vulnerabilities. Web application vulnerability would be addressed through multiple tools integrated to work automatically without the user's tech knowledge. Furthermore, the Graphical User Interface will provide different scan options to users. These options adjust the methodology of the scan techniques involved with speed scan adjustment, pressure level, and vulnerability detection, allowing the user to scan for a specific type of vulnerability. Finally, the platform will generate a detailed report with all the problems found in the target system. Further, it will provide the right solution to fix the flaw.

The solution will be based on several types of scanning that will enable the user to choose the suitable configuration depending on their web application. First, the user will enter his web application URL in the input field and select the type of scan. After that, the scan will start and generate a report containing results discovered about web application vulnerability.

2.2.4 what makes your solution different from other solutions

Through this platform, users will be able to work on several tools that include several types of scan techniques instead of using each tool separately, which will be time-consuming. Also, any user can work on this platform and use the scan utility regardless of technical knowledge.

Scanning web applications is the most challenging phase in creating a web application. Because it requires a high knowledge of computer science, GUI will instruct the users and let them perform several types of scanning with simple clicks.

2.3 Literature Review

2.3.1 Cross-Site Scripting Attack (XSS)

XSS stands for cross-site scripting attack. XSS is performed in JavaScript language. The problem is that all the browsers can understand JavaScript, so the attacker can inject malicious JavaScript code into the website's visitors.

The authors [8] developed ETSSDetector. Web vulnerability scanner that automatically analyzes web applications to find XSS vulnerabilities. It can identify and analyze all data entry points of the application and generate specific code injection tests for each one by simulating the browser's behavior and loading the page, extracting its components, testing it, and finally analyzing them.

The authors from the [9] *Laboratory of Distributed Systems and Software Engineering, Institute of Computing UNICAMP, State University of Campinas, Brazil*, researched Cross-site Scripting vulnerability and explained the risk of the exposure. The author found a solution to prevent and detect XSS vulnerability.

The authors decided to implement the Fault injection technique in their approach. Fault injection is a testing technique used to understand how the system behaves when facing unusual errors. This technique ensures that the system can recover from these errors.

The author's final approach is a composite of two techniques first is analyzing the existence of the vulnerability in a web application through two tools, the vulnerability scanner SoapUI and the fault injector WSInject. These tools will emulate the XSS injection attack between the web application and the end-users and analyze the attack and the weak points of the web application. The benefits of this process are getting higher coverage of XSS attacks and reducing the false-positive result from the scan.

2.3.2 SQL injection Attack

The authors [10] discussed the detection and prevention of SQL injection attacks (SQLIA) and described SQL injection attacks' functionality, detection techniques, and risks. They examined multiple detection techniques such as WASP (Web Application SQL injection protector), Ardilla, SecuriFly, JDBC-Checker, CANDID, etc. These techniques are efficient and effective in terms of detecting SQL injection attacks

The authors [11] used a tool to perform SQL Injection, Havij. Havij is an automated SQL Injection tool that helps penetration testers to find and exploit SQL Injection vulnerabilities on a website page. The tester can perform back-end database fingerprinting, retrieve DBMS login names password hashes, dump or delete tables and columns, retrieve data from the database, execute SQL statements in the server, and access the underlying file system and execute operating system commands

2.3.3 Comparing the solutions

Nowadays, web application vulnerabilities have become more aggressive and diversifiable. Unfortunately, these techniques aren't enough for detecting various web application vulnerabilities to make the web environment more secure. Solving one problem out of many does not mean solving the problem. Furthermore, the ordinary computer user doesn't know about installing these tools and scanning his website.

Compared to these solutions, our solution combines more than one technique to detect multiple web application vulnerabilities and provides a simple website that any ordinary computer user can work on.

2.4 Comparison criteria definition

According to a literature review in the above sub-section, the author described part of the solutions to the main problem: web application vulnerabilities. We would contain a more significant part of these problems and conduct an easy and flexible solution in terms of standards. As we mentioned, our solution will combine most of these functionalities and other features to deal with web application attacks based on business standards and criteria. Instead of dealing with each vulnerability separately and consuming more time and resources, our solution will handle these issues in a single united platform.

2.5 Comparison results and the feasibility

We assume that our solution would solve the central issue of being aware of the vulnerability within web applications. The result is to improve and secure the web environment and facilitate the discovery mechanism of vulnerabilities. We don't have specific results yet, but our project is feasible and applicable based on what we have read, resources, time limitations, budget, and collected tools.

Chapter 3: Analysis

3.1. Functional and Non-functional requirement

3.1.1. Functional requirement

The functional requirements define how the system should behave based on user interaction and how to respond to inputs. In addition, functional requirements describe what the system must do and not do. For example, in the below table, these are the functional

ID	Requirement Definition
FR1	Create an account
FR1.1	The system shall enable a user to create an account
FR2	Login
FR2.1	The system shall enable the user to log in with his account
FR3	Website URL
FR3.1	The system shall allow the user to insert the URL of the target website
FR4	Type of scan
FR4.1	The system shall provide multi-type of scan
FR5	Configuration
FR5.1	The system shall enable the user to add the proper configuration before starting the scan, e.g., a particular cookie
FR6	Tracking the process of scan
FR6.1	The system will provide the verbosity of the process to the user
FR7	Create report
FR7.1	The system shall create a report of the possible identified vulnerabilities to the user
FR8	Type of the report
FR8.1	The system shall allow the user to choose the report file type (e.g., PDF, Excel, etc.)
FR9	Submit report (users)
FR9.1	The system shall allow users to submit a report to technical support

Table 2

3.1.2. Non-functional requirement

As discussed, functional requirements are what the system must do or not do. Non-functional requirements describe how the system should do the functional requirement. In the below table, these are the non-functional requirement for our solution:

User Interface	UI1: The system shall provide certain functionalities in the user interface.
	UI2: The system shall provide a user-friendly, Interactive, and responsive interface
	UI3: The user interface shall be easy to handle and clear
Software Requirements	SR1: The system shall communicate with the system's database to retrieve and use specific security payloads.
	SR2: The system shall guarantee high performance in the scan process
Security Requirements	SE1: The system shall provide a login page.
	SE2: The system shall allow users to access only the services were allowed to access.
	SE2.1: The system shall allow only the admin role to make, edit, and delete the software.
	SE3: The system shall validate all user input.

Table3 : Non-functional

3.2 Hardware Requirements

A dynamic website requires more resources than a static website. The dynamic website stores and processes data require large space and a powerful CPU. The best practice is to have more resources above the actual usage. This will enable the server to prevent overflow. These are the minimum requirement for our solution:

- Processor: 4 x 1,6 GHz CPU
- RAM: 7 GB RAM
- SSD: 1x 50 GB

3.3. Use case diagram

The use case diagram discusses how the user will interact with the web application. The following figure illustrates the use case diagram of the solution:

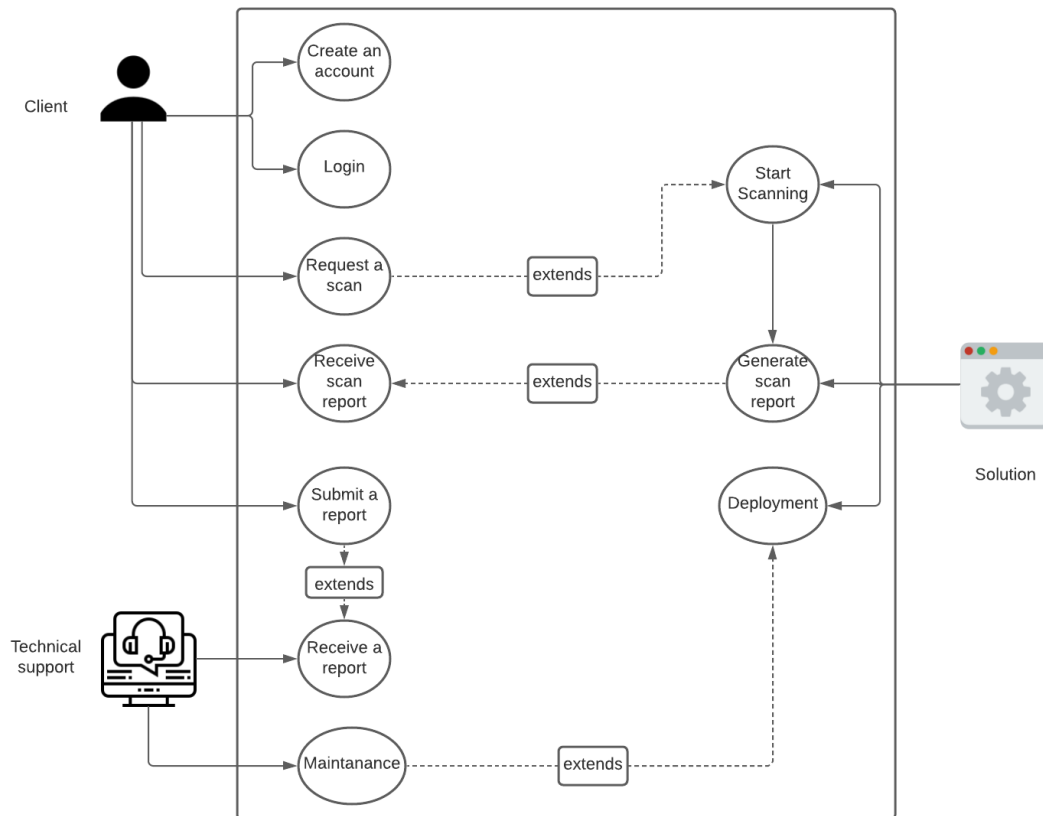


Figure3 UseCase Diagram

3.4 Data collection instruments (Datalogger)

The platform will conduct several techniques to gather the domain/sub-domain of the target web application. First, therefore, distinguished the alive domain/sub-domain from non-alive. After that, the platform will start deep searching to locate public files across the website's contents. Then, it will start analyzing the files to discover sensitive information.

The platform will identify the target's Operating System using specific methodologies to apply appropriate scanning. Back-end language will be determined to apply proper scanning. Finally, the platform will gather information about the various web technologies to identify the weakness in the target

Chapter 4: Design

4.1. System Architecture

The platform will be developed as an integrated platform that combines open source tools and locally written scripts supported with GUI. The platform aims to discover various vulnerabilities. The primary purpose of this project is to help the community to detect and prevent exposure before the adversary does. The system architecture consists of three main phases to accomplish the process of discovering web application vulnerability. These phases are Reconnaissance, scanning, and reporting.

4.1.1 Phase 1: Reconnaissance

Reconnaissance is the first phase in the platform. In this phase, the platform gathers as much information about the target to identify vulnerabilities.

4.1.1.1 Domain and sub-domains:

The platform will conduct several techniques to gather domain/sub-domain of the target web application, using open source tools and deeply enumerating sub-domains using locally written scripts. Therefore, the platform will distinguish the alive domain/sub-domain from non-alive.

4.1.1.2 Website hierarchy

The platform will start deep searching to locate public files across the website's contents. Then, it will start analyzing the files to discover sensitive information. There are two types of searching for public files guessing and analyzing.

First, the guessing technique will use a dictionary word stored on the platform. Then, it will start discovering system hierarchy, such as files and folder that exists in the webserver.

Second, analyzing techniques are more flexible than guessing and consume fewer resources. It will inspect all webserver's pages by using a web crawler. The web crawler will target all webserver's files such as (JavaScript, CSS, etc..).

4.1.1.3 Web server Operating System

The platform will identify the target's Operating System using specific methodologies to apply appropriate scanning.

4.1.1.4 Back-end Language

Back-end language will be identified to apply appropriate scanning. The platform will send a crafted payload to the target website to determine the Back-end language.

4.1.1.5 Information collected

all the information collected from previous steps will be saved in a unique form. To prepare for the next phase, which is scanning.

4.1.2 Phase 2: Scanning

Scanning is the second phase. In this phase, the platform will use the previously collected data to perform the scanning process to discover vulnerabilities in the target web application.

After identifying the target and performing the initial Reconnaissance, thus, the platform will begin to search for an entry point into the target system. Notably, the scanning itself is not the actual intrusion but an extended form of Reconnaissance in which the platform learns more about the target web application. The information obtained from such Reconnaissance helps select strategies to identify the target.

Scanning is the process of gathering additional detailed information about the target using highly complex and aggressive reconnaissance techniques. Scanning refers to a set of procedures used for identifying weaknesses. It is one of the essential phases of intelligent gathering, enabling them to create a profile of the target web application.

4.1.2.1 Web technologies

This step will perform scanning on various web technologies such as (proxies, IDS, IPS, etc..) to identify the weakness in the target.

4.1.2.2 Locating and Addressing vulnerabilities

In this step, we will be ready to identify and locate the vulnerabilities based on the collected information from the previous steps. This step will be automated with various tools and open-sourced and locally written scripts.

4.1.3 Phase 3: Reporting

This the final phase, the platform will generate a report with all the vulnerabilities found in the target web application. The report shall include complete detailed information

about vulnerabilities. Moreover, a suggestion will be provided along with references. The report will be generated in multiple types (pdf, excel, HTML, etc..).

The vulnerabilities found on the target website will be classified with universal standards: CVSS v3.0.

4.2 Diagrams

This diagram will summarize the system architecture process.



Figure 4 System Architecture

4.3. User Interface Design

We have designed three photos to help us imagine what the platform will look like in the feature.

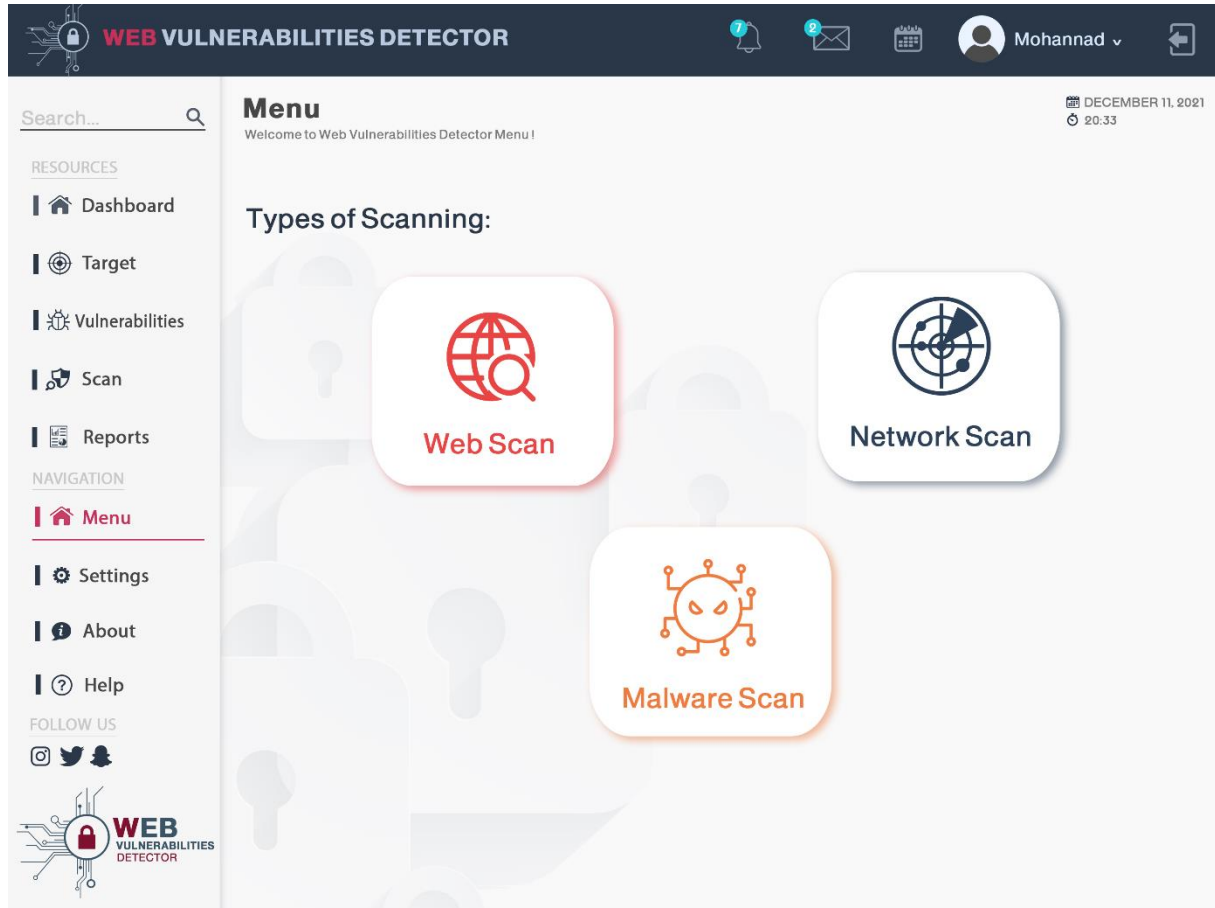









Figure 5 Main-Menu Page



WEB VULNERABILITIES DETECTOR






Mohannad ▾



Search...


Target
Welcome to Web Vulnerabilities Detector Target Page !



Target



Description



Crawl Depth


Scan Speed


Scan Type


Report


Schedule


Special Headers




RESOURCES


- Dashboard
- Target**
- Vulnerabilities
- Scan
- Reports

NAVIGATION

- Menu
- Settings
- About
- Help

FOLLOW US

- 
- 
- 


WEB VULNERABILITIES DETECTOR

DECEMBER 11, 2021
20:33

Figure 6 Target Options Page



Figure 7 Dashboard Page (Statistical Page).

Chapter 5: Development

5.1 APIs and Plugins

As mentioned above, the platform will integrate multiple open-source tools. The system architecture consists of three phases (Reconnaissance, scanning, and reporting). Each phase will use its own APIs. The table below lists all APIs used in the web vulnerabilities detector. The table consists of three columns:

- Tool: the name of the tool
- Function: the function of the tool
- Fully developed: the tool is fully developed by us
- Partially developed: the tool is partially developed

Tool	Function	Fully developed	Partially developed
SubFinder [13]	Subdomain enumeration		
AssetFinder [14]	Subdomain enumeration		
WayBackUrls [15]	Web Crawling		
Gau [16]	Web Crawling		
Gf [17]	Filtering output		
Open redirect	Scans for open redirect	✓	
Bolt [18]	Scans for Cross-Site Request Forgery (CSRF)		
Corsy [19]	Scans for Cross-Origin Resource Sharing (CORS)		
FDsploit [20]	Scans for Local File Inclusion (LFI)		✓
RFI	Request for Information (RFI)	✓	
SQLi Scanner [21]	SQL Injection		✓
XSS	Cross-Site Scripting (XSS)	✓	
WAFW00F [22]	Web Application Firewall (WAF)		
SRI Checker [23]	Subresource Integrity (SRI)		

Table 4 APIs

5.1.1 Phase 1: Reconnaissance

The platform will gather as much information as possible about the target website in this phase. The process of gathering information will be as follow:

5.1.1.1 Subdomain Enumeration

Subdomain enumeration is the process of finding a valid subdomain of one or more domains. It's an essential part of the reconnaissance phase to increase the chances of finding vulnerabilities and prepping the scanning stage requirement. The platform uses two tools for finding subdomains:

- **Subfinder**

Subfinder tool is used to discover a valid subdomain for the website by using passive online sources. The tool is written in GO programming language. Subfinder is one of the highest-rated tools in GitHub to do subdomain enumeration. [13]

- **Assetfinder**

Assetfinder is an open-source tool used to find the subdomains that are related to a specific domain. The tool is developed via go programming language. Assetfinder is one of the famous tools to do subdomain enumeration. [14]

5.1.1.2 Web crawling

A Web crawler starts with a list of URLs to test. The crawler tests these URLs by communicating with web servers that respond to those URLs. It identifies all the hyperlinks in the retrieved web pages and adds them to the list of URLs to visit.

- **Waybackurls**

Waybackurls is a tool used to perform web crawling. The tool will identify all possible URLs in a specific domain.[15]

- **Gau**

Gau is a web crawling tool used to determine all possible links in a specific domain. [16]

5.1.1.3 Output Filtering

GF

GF is a tool used to filter output files that are generated by web crawlers. The tool will filter a txt file that contains a list of URLs. This list of URLs will be filtered based on the type of possible vulnerability on the URL. [17]

5.1.2 Phase 2: Scanning

In this phase, the real work begins. The scanning phase focuses on finding vulnerabilities on the target website. The platform will scan the website for several types of vulnerabilities as follow:

5.1.2.1 Open redirect:

Open redirect is a vulnerability that will redirect user requests from the intended site to another site that could be malicious. The open redirect could help the attacker further than phishing and redirect users to another website as it could be combined with server-side request forgery and XSS. The platform will use a python script that we write. The script will scan all possible open redirect vulnerabilities.

5.1.2.2 Cross-Site Request Forgery (CSRF):

CSRF vulnerability allows an attacker to create a request within a web application where the user is logged into by sending an HTTP request whenever the user opens a website containing malicious code. An attacker could manipulate the user to execute unwanted actions such as transferring funds or changing their email. The most concern is if the user has an administration privilege because that could compromise the entire web application. The platform uses a bolt tool to scan for CSRF vulnerability. A bolt is a python tool that scans for CSRF. [18]

5.1.2.3 Cross-Origin Resource Sharing (CORS) [CVE-2021-26991]:

CORS is an HTTP-header-based request that allows restricted resources of the web page to be requested from another domain. Misconfiguration could compromise the confidentiality and integrity of the data by allowing the third party to retrieve user information such as user settings and payment cards. The platform uses Corsy tool to scan for CORS. Corsy is a python program that scans the website for all possible CORS. [19]

5.1.2.4 Local File Inclusion (LFI):

LFI is a preprocessor directive that manipulates web applications to expose content on a web server by including files on a server through the web browser. The attack could expose sensitive information and can lead to cross-site scripting and remote code execution. The platform used FDSexploit to scan for LFI vulnerabilities. FDSexploit is a famous python program script that scans LFI vulnerability with different payloads. [20]

5.1.2.5 Remote file inclusion (RFI):

Remote file inclusion is an attack that occurs in a web application that dynamically includes external scripts or files. If an RFI attack succeeds, the impact could lead to the disclosure of sensitive information to full system compromise. The platform will use a python script that we write. The script will scan all possible RFI vulnerabilities.

5.1.2.6 SQL Injection:

SQL injection attack is a vulnerability that allows an attacker to insert malicious code to manipulate the database. By performing an SQL injection attack, the attacker could delete data from the database, retrieve sensitive information such as usernames and passwords, and modify the data within the database. To perform an SQL injection attack, the attacker needs to locate an input that is vulnerable to SQL injection. Then attacker tries to insert an SQL query to perform this attack. [21]

5.1.2.7 Cross-Site Scripting (XSS):

Cross-site scripting attack is a type of injection where the attacker inserts a malicious script into dynamic web pages that other users could view. For example, the attacker can send an email that contains a link with a malicious script to the user. When the user clicks on the link, the script will get executed. This script could be deployed to perform a specific function, such as retrieving a user's cookies. The platform will use a python script that we write. The script will scan all possible XSS vulnerabilities.

5.1.2.8 Web Application Firewall (WAF):

Having a web application firewall is to protect the web application from attacking attempts by inspecting and filtering traffic between the internet and the web application. Web application firewalls help to defend against several attacks such as cross-site scripting, file inclusion, SQL injection, and cross-site request forgery. The platform will use a python script

to determine the WAF type. WAFW00F is a python script that scans the website and determines its firewall. [22]

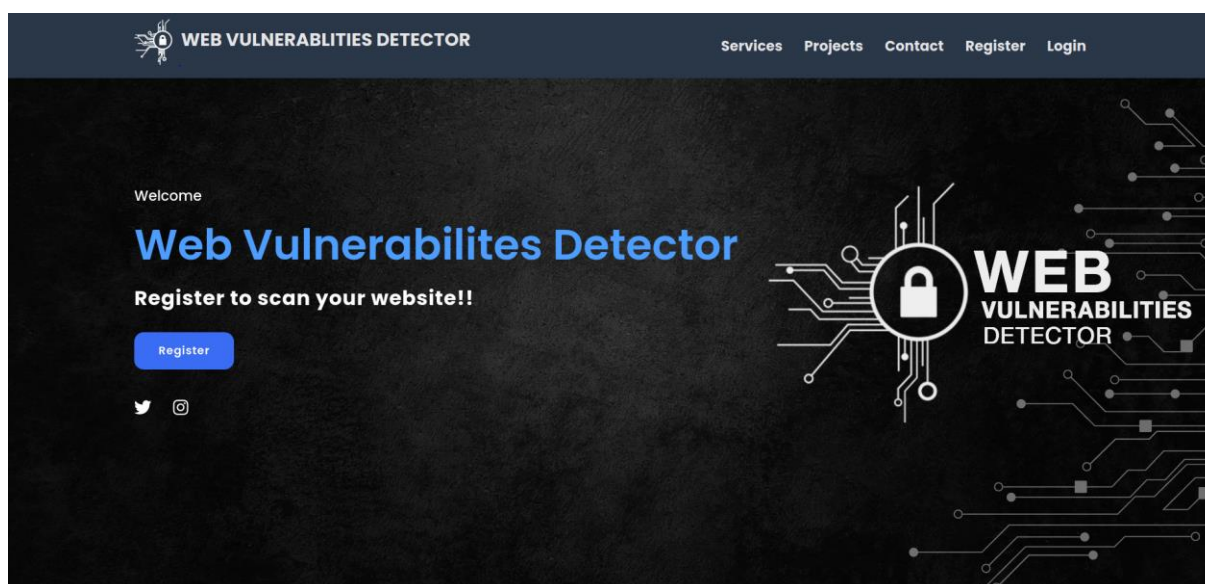
5.1.2.9 Subresource Integrity (SRI):

Subresource Integrity Is a method that allows browsers to verify that resources are hosted in a third party without manipulation by using hash comparing to compare the hash value of the resources on a web server and third-party server. [23]

5.2 Implementation/Coding

Implantation and coding are the essential phases of every project. The implantation phase is called the start phase because it is the phase where you are going to start creating your project. The development team will start creating the project based on the specified requirements in implantation. In order to successfully implement the web vulnerabilities detector requirement, we have chosen python programing language as the main language. The python program language has been the most used programming language in the past years. It has many functions and packages that will help us integrate the open-source tool and develop web vulnerabilities detector project. For the graphical user interface (GUI), we have chosen HTML, CSS, and JavaScript. In the back-end, we used PHP programing language with the support of python. We divided the development part into four categories:

5.2.1 Platform interface



Services



Vulnerabilities



Figure8 Home page

This is the main page of our platform. It contains information about the services that the platform provides.

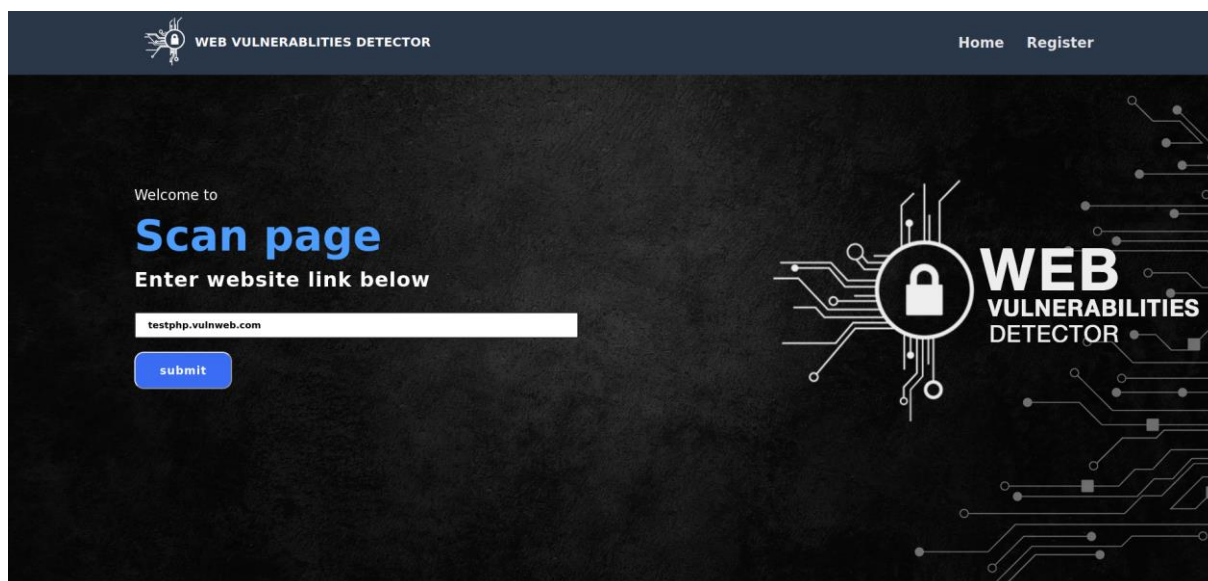


Figure 9 Scan page

This is the scanning from here. The user can scan his website. The user will input his domain and press the scan button.

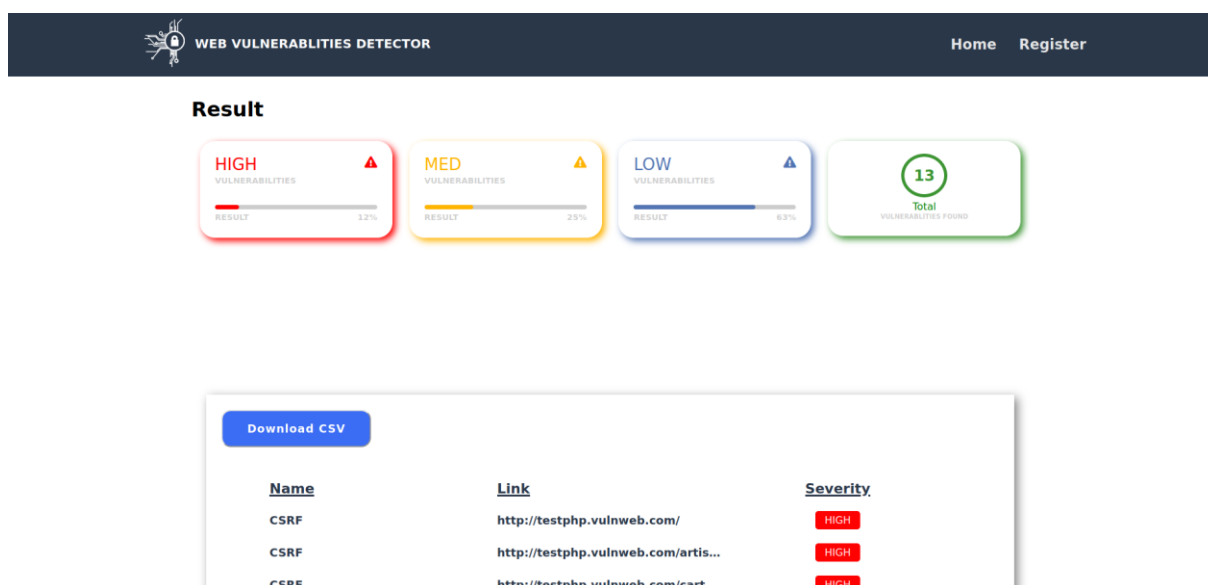


Figure10 Result page

This is the result of the page of the scanning process.

5.2.2 Front-End JavaScript

```

434 <script>
435     $(document).ready(function(){
436         $("#submit").on('click', function(){
437             $('#loading').fadeIn()
438             var DOM = $('#domain').val();
439             // send ajax
440             $.ajax({
441                 url:"API.php",
442                 method:"post",
443                 data:{d:DOM},
444                 //dataType:"json",
445                 success:function(data)
446                 {
447                     $('#design').fadeIn();
448                     $('#loading').fadeOut()
449                     console.log('success');
450                     $('html, body').animate({
451                         scrollTop: $("#design").offset().top
452                     }, 2000);
453                     const obj = JSON.parse(data);

```

Figure 11 Code screenshot

In JavaScript, from line 434 to 444, it will send the POST request method to the API.php page with the parameter d, which contains the target domain.

The success section line 453 will parse the JSON response into the obj variable to prepare for the next step.

```

465     // Iterate over the first level
466     // x // SEV
467     // y // name
468     // obj[x][y][i] // link
469     for (x in obj) {
470         // console.log(obj.hasOwnProperty(x));
471         // console.log(x);
472         // Iterate over the second level
473         for (y in obj[x]) {
474             // console.log(y);
475             // Iterate over the 3rd level
476             for (var i = 0; i < obj[x][y].length; i++)
477             {
478                 // console.log(obj[x][y][i]);
479                 $('#row').append('<tr><td>${y}</td><td>${obj[x][y][i]}</td><td><a href="#" class='${x}'>${x}</td></tr>');
480             }
481         }
482     }
483 }

```

Figure 12 Code screenshot

This section of code will fetch the JSON array and start appending the result into the table.

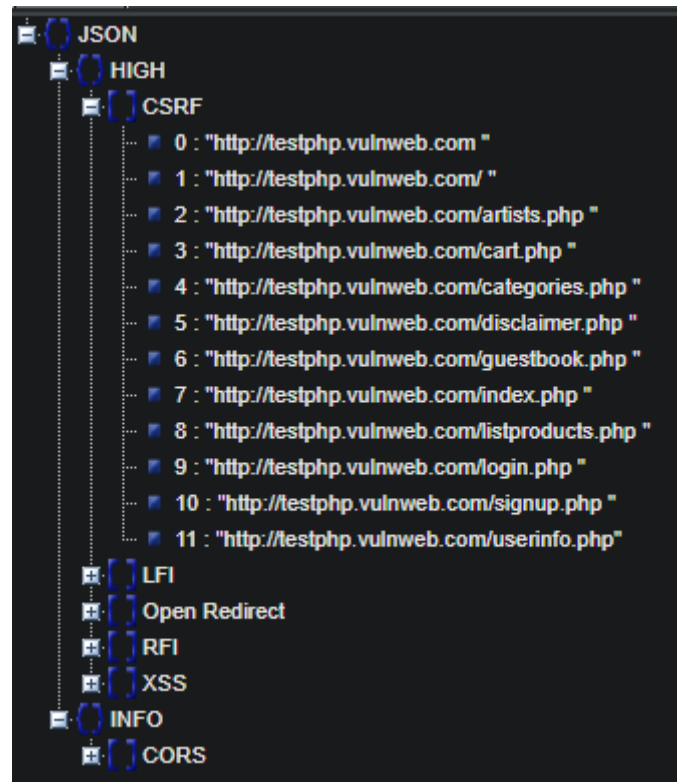


Figure 13 Code screenshot

The JSON response will be like the above picture

5.2.3 Back-End PHP

```
14  if(isset($_POST['d'])) && !empty($_POST['d']))
15  {
16
17  $domain = $_POST['d'];
```

Figure 14 Code screenshot

This section will check if the parameter POST(d) exists and is not empty

```
25  exec("/bin/python3.10 script/main.py $domain", $output, $return_var);
```

Figure 15 Code screenshot

This line of code will run the first argument into the terminal, where the second argument is the output, and the 3rd is for errors ← mn kese


```
33 // new code
34 $severities = array("HIGH", "MED", "LOW", "INFO");
```

Figure 16 Target Options Page

> HIGH
> INFO
> LOW
> MED

Figure 17 Code screenshot

This array will contain the severity that matches the server's files

```
48 foreach ($severities as $severity) {
49     // open the path that contain the result of the vulns
50     $files = glob("/var/www/html/wvd-copy/script/output/$domain/$severity/*.txt", GLOB_BRACE);
51     // variable that hold the severity value // ex. HIGH MED LOW INFO
52     $sev = $severity;
53
54     // read the name of the file
55     foreach($files as $file) {
56         // this code is to filter the name of the vulnerability
57         $parts = explode('.', $file)[1]; // make it CSRF.txt
58         $vuln_name = explode('.', $parts)[0]; // make it CSRF
59
60
61         // start adding the links of txt file that contain the result into array in json
62
63         // open the result file
64         $handle = fopen($file, "r");
65
66         // function to check if there is error on opening the file
67         if ($handle) {
68             // creating empty array, and reseting the array to empty every iteration
69             $testArray = [];
70             // this loop will read the lines of the result file and save into $testArray
71             while (($line = fgets($handle)) !== false) {
72                 $testArray[] = $line;
73             }
74
75
76
77             // adding the array into the json file; the array contains the result
78             $jsonArray[$severity][$vuln_name] = $testArray;
79
80             // closing the file
81             fclose($handle);
82         } else {
83             // error opening the file.
84
85
86     }
```

Figure 18 Code screenshot

The first foreach will fetch the directories by the target/severity directories, which the path will be /var/www/HTML/HOME/script/output/**target**/**HIGH**/*.txt , and it will save the txt files in the \$files line 50

The second foreach will take the txt file and start to filter its name, for example, www.target.com_CSRF.txt . Here the name of the file will be divided based on the red delimiters, and the final result will be CSRF, and it will be stored in the \$vuln_name variable line 58.

then from lines 64 to 74, it will start reading the lines inside the txt files, and it will save it into the \$test_array line 72

in line 78 the code will save the lines of \$test_array into \$jsonArray variable

```
92     echo json_encode($jsonArray);  
93     die();
```

Figure19 Code screenshot

Finally, at this line, it will encode the array to JSON encode and return to the website (user).

5.2.4 Server-side

On the server side, our main python tool is working. The python tool contains the integrated open source and the scripts. In addition, the tool contains several functions. These functions will be fully described in below figures:

5.2.4.1 Variables:

```
script > ! config.yaml
1  # Fill website_name variable with you website name, ex.WVDLOCAL.
2
3  website_name: 'wvd-copy'
4  sub_domain_path: '/var/www/html/HOME/script/subdomains/{}/'
5  output_path: '/var/www/html/HOME/script/output/{}/'
6  |
```

Figure 20 Code screenshot

```
27  #-----
28
29  #Target website
30  target = sys.argv[1]
31
32
33  website = yaml_data['website_name']
34  path = yaml_data['sub_domain_path'].replace('HOME',website).format(target)
35  path_out = yaml_data['output_path'].replace('HOME',website).format(target)
36  #-----
```

Figure 21 Code screenshot

As shown in the above two figures, we have three main variables:

- **Sub_domain_path:** this variable contains the subdomain path
- **Output_path:** this variable contains the output path
- **Target:** this variable will store a system argument. The tool will get the argument from the user, and the argument is the specified domain by the user.

5.2.4.2 Command function

```
41 #function need to be replaced with REGEX
42 def command(command):
43     proc = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
44     (out, err) = proc.communicate()
45     out = str(out)
46     err = str(err)
47     out = out.replace('"', '')
48     out = out.replace('b', '', 1)
49     out = out.replace('\\n', '\n')
50     out = out.replace('(', '')
51     out = out.replace(')', '')
52     out = out.replace(', ', ',')
53     out = out.replace("'", '')
54     return str(out), str(err)
```

Figure 22 Code screenshot

Command function is a function that allows the python program to write commands in the Linux terminal. The function has two variables:

- **Out:** the variable will store the desired output on the Linux terminal
- **Err:** the variable will store the expected errors on the Linux terminal

After storing the result and the errors, the tool will run several commands to filter the output.

5.2.4.3 RemoveEmptyLines function

```
58 def removeEmptyLines(file1):
59     with open(file1) as f_input:
60         data = f_input.read().rstrip('\n')
61     f_input.close()
62     with open(file1, 'w') as f_output:
63         f_output.write(data)
64     f_output.close()
```

Figure 23 Code screenshot

RemoveEmptyLines function is an essential function to perform output normalization. The function will delete any space or blank lines in the output files. We are using this function to normalize all output from all tools.

5.2.4.4 Creating folders function

```
88 #create folder with the name of target
89 def createFolder():
90     command("mkdir -p {}".format(path))
91
```

Figure 24 Code screenshot

CreateFolder function is a function that will create a folder with the user's name of the specified domain. The function contains the command function, which will write the command "mkdir -p {" in order to create a folder with the name of the target domain. The value {} is the variable path mentioned above in the variables section

```
96 #create output folder
97 def createFolder_output():
98     command("mkdir -p {}".format(path_out))
99     command("mkdir -p {}/HIGH".format(path_out))
100    command("mkdir -p {}/MED".format(path_out))
101    command("mkdir -p {}/LOW".format(path_out))
102    command("mkdir -p {}/INFO".format(path_out))
103    #- - - - -
```

Figure 25 Code screenshot

CreateFolder_output function is a function used to create a folder with the target domain in the output folder using the command function. Then the function will create four folders inside the target domain folder as follow (HIGH, MED, LOW, INFO). In the command function, the variable path_out is used to specify the path of the output folder.

5.2.4.5 Tools functions

Tools functions are the part of the code where we are going to run all the tools used in web vulnerabilities. Each tool has its own function. All of these functions will be described below figures:

SubDomainEnumeration

```

102 #Tools for subdomain scanning
103 subDomainEnumartion={
104     "assetFinder":'assetfinder {} -subs-only | grep {}$'.format(target,target),
105     "subFinder":'subfinder -d {} -all -silent'.format(target)
106 }
107 |
108 #start subdomain enumartion
109 def subDomainEnumeration():
110     file1 = open("{}{}.txt".format(path,target),"a")
111     for key in subDomainEnumartion:
112         out,err = command(subDomainEnumartion[key])
113         file1.write(out)
114
115     #remove duplication in the file
116     file1.close()
117     command("sort -u -o {}{}.txt {}{}.txt".format(path,target,path,target))
118

```

Figure 26 Code screenshot

SubDomainEnumeration is a function used to run two tools, assetfinder and subfinder. First, in line 103, we create an array that is going to store two values, as shown in the figure. Then, in line 109, we have created the subdomain enumeration function. The function contains:

- **Line 110:** This command will create a text file with the target domain's name in the subdomains path.
- **Line 111:** this for loop is used to run the command function that contains the specified array in line 103. The desired out will be stored in the created file in line 110. Then it will close the file at the end of the loop.
- **In line 117:** This command is used to normalize the output folder, which is going to remove the duplicated URLs in the text files.

URL Enumeration

```
123 def URL_Enumeration():
124     #start URL enumeration with waybackurls and gau
125     f = open("{}{}.txt".format(path,target),'r')
126     file1 = open("{}{}_URLS.txt".format(path,target),"a")
127     for line in f:
128         out,err = command("waybackurls {}".format(line))
129         file1.write(out)
130         out,err = command("gau {}".format(line))
131         file1.write(out)
132
133
134     file1.close()
135     command("sort -u -o {}{}_URLS.txt {}{}_URLS.txt".format(path,target,path,target))
136
```

Figure 27 Code screenshot

URL_Enumeration is a function used to run two tools, WayBackUrls and Gau. The function contains:

- **Line 125:** This command will open the text file that contains the output of the subdomain and start reading the content of the file
- **Line 126:** This command will create a text file with the target domain's name in the subdomains path.
- **Line 127:** this for loop is used to run two command functions. The desired out will be stored in the created file in line 126. Then it will close the file at the end of the loop.
- **In line 135:** This command is used to normalize the output folder, which is going to remove the duplicated URLs in the text files.

URL Filtration

```

139 #URL Filtration
140 def filter_gf():
141     command("gf xss {}_URLS.txt >> {}_xss.txt".format(path,target,path,target))
142     command("gf redirect {}_URLS.txt >> {}_redirect.txt".format(path,target,path,target))
143     command("gf idor {}_URLS.txt >> {}_idor.txt".format(path,target,path,target))
144     command("gf lfi {}_URLS.txt >> {}_lfi.txt".format(path,target,path,target))
145     command("gf rce {}_URLS.txt >> {}_rce.txt".format(path,target,path,target))
146     command("gf ssrf {}_URLS.txt >> {}_ssrf.txt".format(path,target,path,target))
147     command("gf sqli {}_URLS.txt >> {}_sqli.txt".format(path,target,path,target))
148     command("gf cors {}_URLS.txt >> {}_cors.txt".format(path,target,path,target))

```

Figure 28 Code screenshot

Filter_gf function is used to run the GF tool. This function is used to filter the result from the URL_Enumeration function. The function contains several commands that seem to each other. For example, the commands from lines 141 to 148 are used to filter the URLs into text files. Each file has a possible link with the vulnerability, as shown in the picture.

CORS function

```

158 #CSRF
159
160 def CSRF():
161     file1 = open("{}_HIGH/{}_CSRF.txt".format(path_out,target),"a")
162     out,err = command(["python3 /var/www/html/wvd-copy/script/tools/Bolt-master/bolt.py "+
163         "-u http://{} -l 3 | grep http | cut -d ' ' -f 2".format(target)])
164     file1.write(out)
165
166     file1.close()
167     command("sort -u -o {}_HIGH/{}_CSRF.txt {}_HIGH/{}_CSRF.txt".format(path_out,target,path_out,target))
168     command("sed -i '1d' {}_HIGH/{}_CSRF.txt".format(path_out,target,path_out,target))
169
170     removeEmptyLines("{}_HIGH/{}_CSRF.txt".format(path_out,target))
171

```

Figure 29 Code screenshot

CSRF is a function used to run bolt tool. The function is used to detect CSRF vulnerability. The function contains:

- **Line 161:** This command will create a text file with the name of the target domain and vulnerability name on the output path
- **Line 162:** running the tool command on a Linux terminal
- **Line 167-168:** These two commands will normalize the output file.

CSRF function

```

178 #CORS
179
180 def CORS():
181     file1 = open("{}_INFO/{}_CORS.txt".format(path_out,target),"a")
182     out,err = command("python3 /var/www/html/wvd-copy/script/tools/Corsy-master/corsy.py -u http://{}".format(target))
183     file1.write(out)
184
185     file1.close()
186     command("sort -u -o {}_INFO/{}_CORS.txt {}_INFO/{}_CORS.txt".format(path_out,target,path_out,target))
187     removeEmptyLines("{}_INFO/{}_CORS.txt".format(path_out,target))

```

Figure 30 Code screenshot

CORS is a function used to run the corsy tool. The function used to detect Cors. The function contains:

- **Line 161:** This command will create a text file with the name of the target domain and vulnerability name on the output path
- **Line 162:** running the tool command on a Linux terminal
- **Line 167-168:** These two commands will normalize the output file.

Running all functions

```

235     createFolder()
236     createFolder_output()
237     subDomainEnumeration()
238     URL_Enumeration()
239     filter_gf()
240     threading.Thread(target=xss.main,args=(path,path_out,target)).start()
241     threading.Thread(target=open_redirect.main,args=(path,path_out,target)).start()
242     threading.Thread(target=RFI.main,args=(path,path_out,target)).start()
243     threading.Thread(target=CSRF,args=()).start()
244     threading.Thread(target=CORS,args=()).start()
245     threading.Thread(target=SQLInjection,args=()).start()
246     threading.Thread(target=LFI,args=()).start()
247     threading.Thread(target=SRI,args=()).start()
248     threading.Thread(target=SH,args=()).start()

```

Figure 31 Code screenshot

This is the place to run all functions of the tools. As shown in the above figure, we used thread in order to enhance the speed of scanning

5.3 Testing

In the Testing phase, we have tested the platform's performance, allowing us to monitor and enhance the platform.

Performance Testing: in the performance testing, we have tested the time of execution of the main code and calculated its times. After that, we deployed the Threads concept into the code, and threads were the main part of the Performance testing that made the platform faster without affecting accuracy.

<i>Function</i>	Without Thread	With Thread
createFolder	0.005 s	0.005 s
createFolder_output	0.028 s	0.026 s
subDomainEnumeration	71.847 s	82.593 s
URL_Enumeration	114.901 s	49.136 s
filter_gf	0.105 s	0.113 s
xss.main	175.728 s	0.003 s
open_redirect.main	81.449 s	0.004 s
RFI.main	224.922 s	0.005 s
CSRF	54.609 s	0.006 s
LFI	40.418 s	0.009 s
Total	764.016 s	144.347 s

Table 5 Performance

Chapter 6: Testing the developed artifact

6.1 Data collection

The process of the data collection is divided into two phases. The first phase is sub-domain Enumeration which depends on Assetfinder and Subfinder tools. The Assetfinder tool will gather all sub-domain that are related to the specified domain. The Subfinder tool will use to discover the valid subdomain of the website. These processes work together to feed the platform with data. It is essential to increase the chances of finding vulnerabilities and prepping the scanning stage requirement. The second phase to accomplish data collection is web crawling, indexing the web pages' data by using Waybackurls and GAU. Waybackurls is used to identify URLs that belong to the target domain and GAU to determine the links of the domain. This is the platform's approach to data collection.

6.2 Clean the data and prepare for analysis

The data is processed and normalized before moving to the scanning stage. This process is important to distinguish the alive domain/sub-domain from non-alive. The first station is cleaning the URLs from noises that disrupt the value. Moreover, any empty line will remove from the collected data. In addition, the data will be checked if any data is duplicated. If so, the duplicated data will remove. Furthermore, each URL will be categorized separately based on the type of vulnerability and the severity level. In conclusion, the fill will be processed in the back-end and converted to JSON file format.

6.3 Analyze the collected data using the appropriate method and tool

In the analyzing phase. We have used the GF tool to analyze and categorize the target_URLS.txt and divide it into several files that carry only links of a specific vulnerability. the tool work along with WayBackURLS and GAU

6.4 Visualize the results

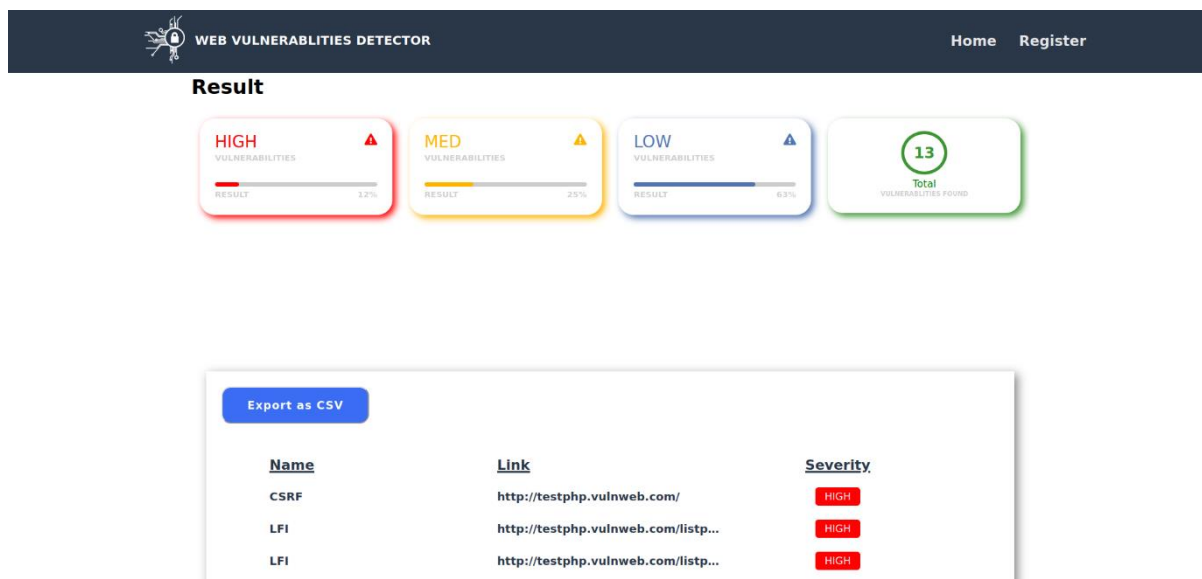


Figure 32 Code screenshot

The result screen will include four squares that summarize the finding of the platform. Underneath it, a table will consist of the details of the finding. It consists of 3 columns which are Name, Link, and Severity. The platform will provide a feature to the user to have a local CSV copy of the result.

References

- [1] "Module 2 – CEH Footprinting – CEH Exam Tools."
<http://cehexamtools.com/preparation/module-2-ceh-footprinting/> (accessed Dec. 12, 2021).
- [2] "What is web application security? | Web security | Cloudflare."
<https://www.cloudflare.com/learning/security/what-is-web-application-security/> (accessed Nov. 13, 2021).
- [3] Sai, "39+ Cyber Security Statistics That Should Worry You [2021]," *TechJury*, Feb. 20, 2019. <https://techjury.net/blog/cyber-security-statistics/> (accessed Nov. 13, 2021).
- [4] K. S. S. E. December 17 and 2020, "US-CERT Reports 17,447 Vulnerabilities Recorded in 2020," *Dark Reading*, Dec. 16, 2020. <https://www.darkreading.com/threat-intelligence/us-cert-reports-17-447-vulnerabilities-recorded-in-2020> (accessed Nov. 13, 2021).
- [5] "2020's Record Numbers of Vulnerabilities," *K2io*, Jan. 11, 2021. <https://www.k2io.com/2020s-record-numbers-of-vulnerabilities/> (accessed Nov. 13, 2021).
- [6] L. Cheng, F. Liu, and D. Yao, "Enterprise data breach: causes, challenges, prevention, and future directions: Enterprise data breach," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, p. e1211, Jun. 2017, doi: 10.1002/widm.1211.
- [7] R. S. Updated, "98 Must-Know Data Breach Statistics for 2021 | Varonis," *Inside Out Security*, Jan. 27, 2020. <https://www.varonis.com/blog/data-breach-statistics/> (accessed Nov. 13, 2021).
- [8] T. Rocha and E. Souto, "ETSSDetector: A Tool to Automatically Detect Cross-Site Scripting Vulnerabilities," Aug. 2014, pp. 306–309. doi: 10.1109/NCA.2014.53.
- [9] M. I. P. Salas and E. Martins, "Security Testing Methodology for Vulnerabilities Detection of XSS in Web Services and WS-Security," *Electronic Notes in Theoretical Computer Science*, vol. 302, pp. 133–154, Feb. 2014, doi: 10.1016/j.entcs.2014.01.024.
- [10] "[No title found]," *IJCSMC*.
- [11] B. Nagpal, N. Singh, N. Chauhan, and A. Panesar, "Tool based implementation of SQL injection for penetration testing," in *Communication Automation International Conference on Computing*, May 2015, pp. 746–749. doi: 10.1109/CCAA.2015.7148509.
- [12] "Functional vs Non-Functional Requirements - Understand the Difference," *ReQtest*, Apr. 04, 2012. <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/> (accessed Nov. 12, 2021).
- [13] *projectdiscovery/subfinder*. ProjectDiscovery, 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/projectdiscovery/subfinder>
- [14] T. Hudson, *assetfinder*. 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/tomnomnom/assetfinder>
- [15] T. Hudson, *waybackurls*. 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/tomnomnom/waybackurls>

- [16] C. Leo, *getallurls (gau)*. 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/lc/gau>
- [17] *GoFrame*. GoFrame, 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/gogf/gf>
- [18] S. Sangwan, *Bolt*. 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/s0md3v/Bolt>
- [19] S. Sangwan, *Corsy*. 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/s0md3v/Corsy>
- [20] “GitHub - chrispetrou/FDsploit: File Inclusion & Directory Traversal fuzzing, enumeration & exploitation tool.” <https://github.com/chrispetrou/FDsploit> (accessed May 12, 2022).
- [21] M. Stampar, *Damn Small SQLi Scanner*. 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/stamparm/DSSS>
- [22] *WAFW00F*. Enable Security, 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/EnableSecurity/wafw00f>
- [23] *SRI Checker*. 4ARMED, 2022. Accessed: May 12, 2022. [Online]. Available: <https://github.com/4ARMED/sri-check>