PROJECT NAME: ONLINESHOPPINGSYSTEM

There are two types of analysis

- **Static Analysis**
- **Dynamic Analysis**

Now First we see the static analysis.

Static Analysis

Static analysis refers to the process of analyzing software code or other artifacts without executing them. It is a type of software testing technique that aims to identify potential defects, vulnerabilities, or quality issues in the codebase before the software is run or deployed.

Static analysis tools look at a program's source code, bytecode, or binary representation to find different kinds of problems. These tools analyze the code's structure, syntax, and semantics in search of patterns that point to potential issues using a set of predefined rules. The analysis can address a variety of problems, such as coding errors, security vulnerabilities, maintainability issues, and coding standards compliance.

Static analysis can be applied at different stages of the SDLC (software development life cycle), like during coding, and when the code is reviewed. It helps developers identify and fix issues early on, reducing the chances of defects and vulnerabilities making their way into the final software release.

Most tools used in static analysis

- **SonarQube**
- **EsLint**
- **Checkstyle**
- **PMD**
- **FindBugs**
- **Coverity**
- **Synk**
- **ReSharper**
- **Infer**

To find the vulnerability in PHP code we use the Synk (Synk CLI)

Synk

Snyk is a developer-first security platform that focuses on helping developers find and fix vulnerabilities in their code and open-source libraries. It offers a range of tools and services to help organizations proactively address security concerns during the development and deployment of their software.

By enabling developers to identify and address vulnerabilities as soon as they are discovered, Snyk's primary objective is to make security an integral part of the development process. To help developers find and fix vulnerabilities without interfering with their workflow, it offers seamless integration with well-known development tools, including IDEs, build systems, and CI/CD pipelines.

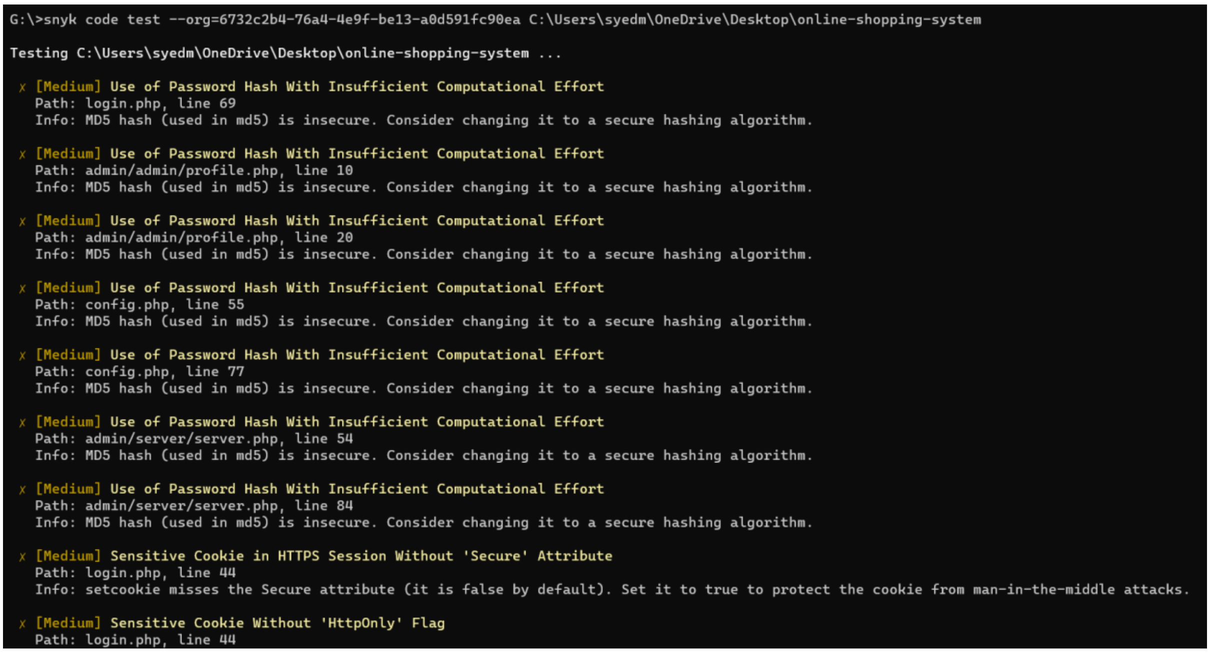
Snyk CLI (Command-Line Interface) is a command-line tool provided by Snyk that allows you to interact with the Snyk platform and perform various security-related tasks directly from your terminal or command prompt. It provides a convenient way to integrate security checks and vulnerability management into your development workflow.

You can consult the official Snyk documentation at https://docs.snyk.io/snyk-cli for more in-depth information about Snyk CLI, its installation, usage, and available commands.

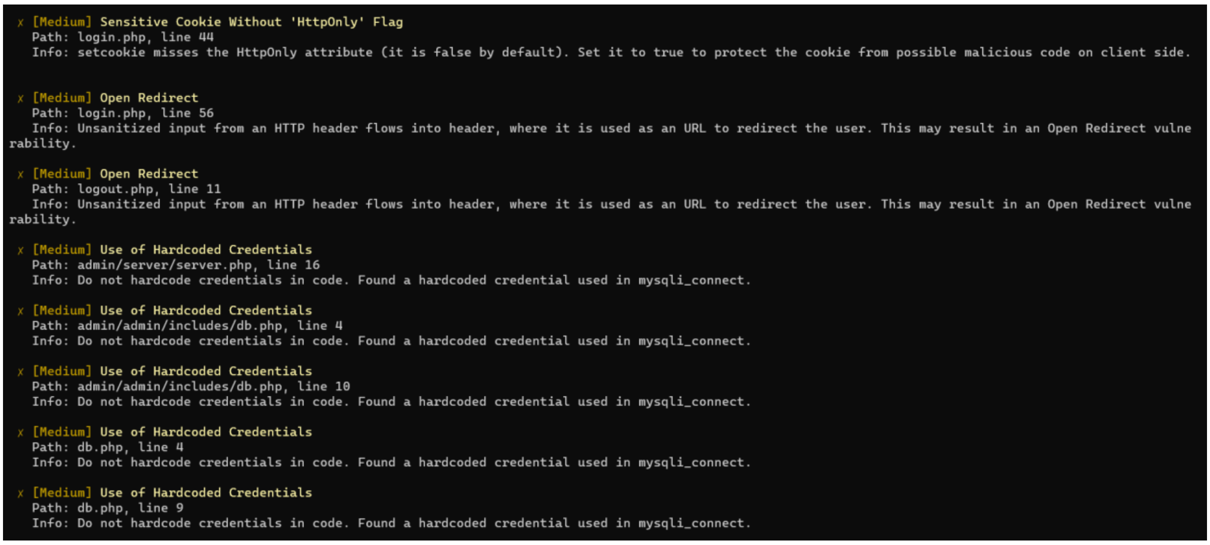
Now next run this command in the command prompt, it analyzes the code.

Command: "**snyk code test --org=6732c2b4-76a4-4e9f-be13-a0d591fc90ea C:\Users\syedm\OneDrive\Desktop\online-shopping-system**"

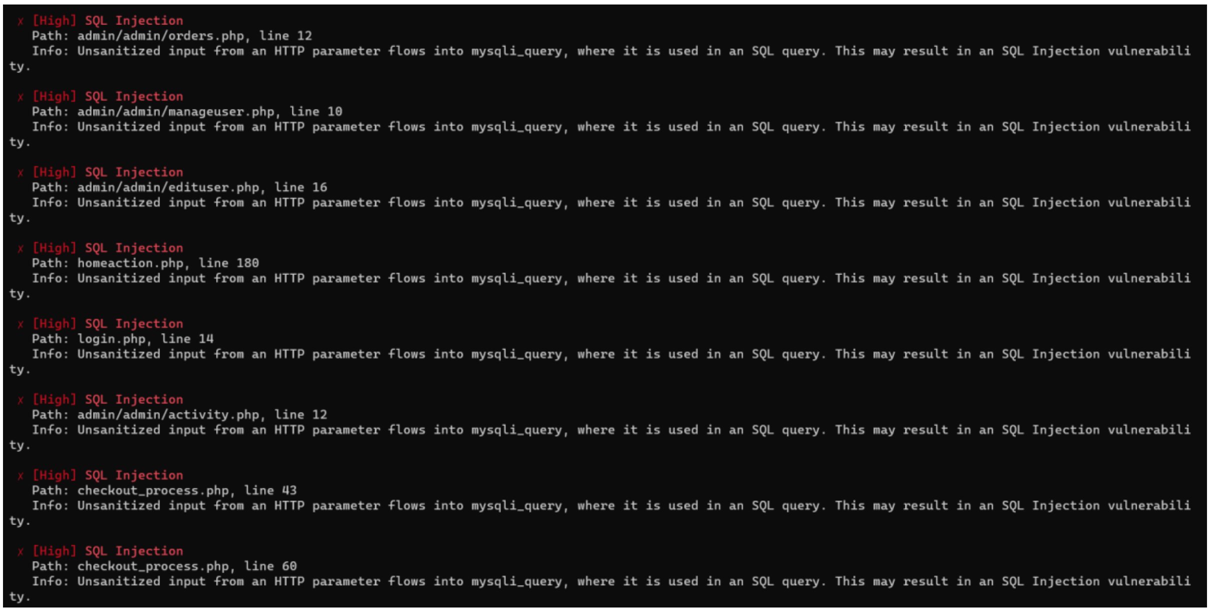
After this command it show the vulnerability in the project



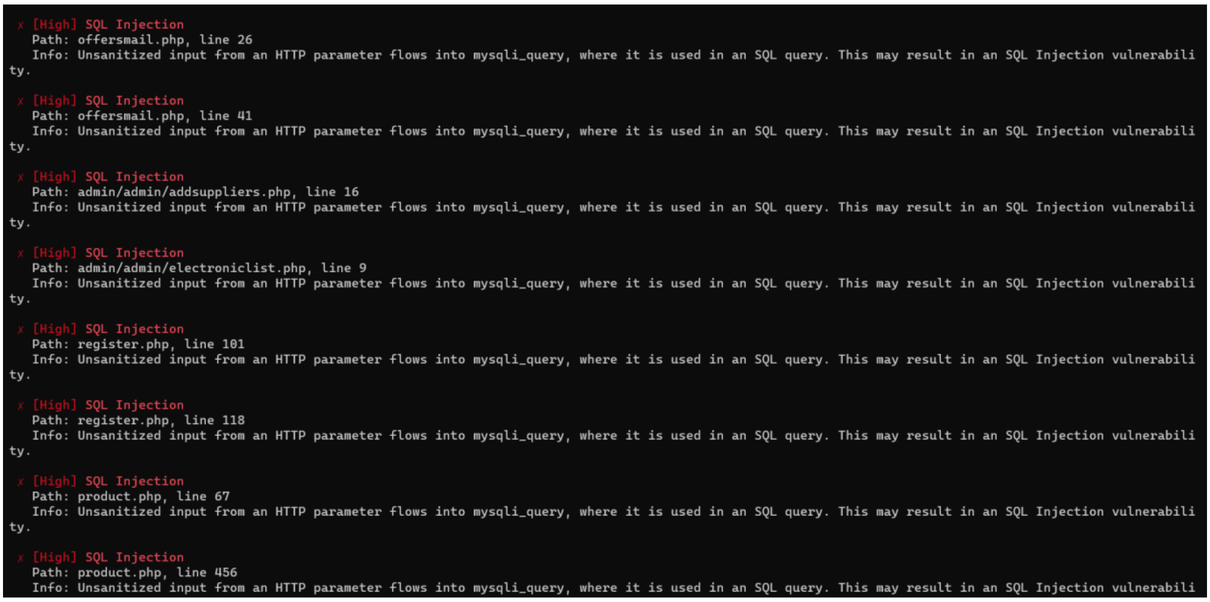
```
G:\>snyk code test --org=6732c2b4-76a4-4e9f-be13-a0d591fc90ea C:\Users\syedm\OneDrive\Desktop\online-shopping-system

Testing C:\Users\syedm\OneDrive\Desktop\online-shopping-system ...

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: login.php, line 69
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: admin/admin/profile.php, line 10
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: admin/admin/profile.php, line 20
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: config.php, line 55
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: config.php, line 77
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: admin/server/server.php, line 54
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Use of Password Hash With Insufficient Computational Effort
   Path: admin/server/server.php, line 84
   Info: MD5 hash (used in md5) is insecure. Consider changing it to a secure hashing algorithm.

 ✗ [Medium] Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
   Path: login.php, line 44
   Info: setcookie misses the Secure attribute (it is false by default). Set it to true to protect the cookie from man-in-the-middle attacks.

 ✗ [Medium] Sensitive Cookie Without 'HttpOnly' Flag
   Path: login.php, line 44
```



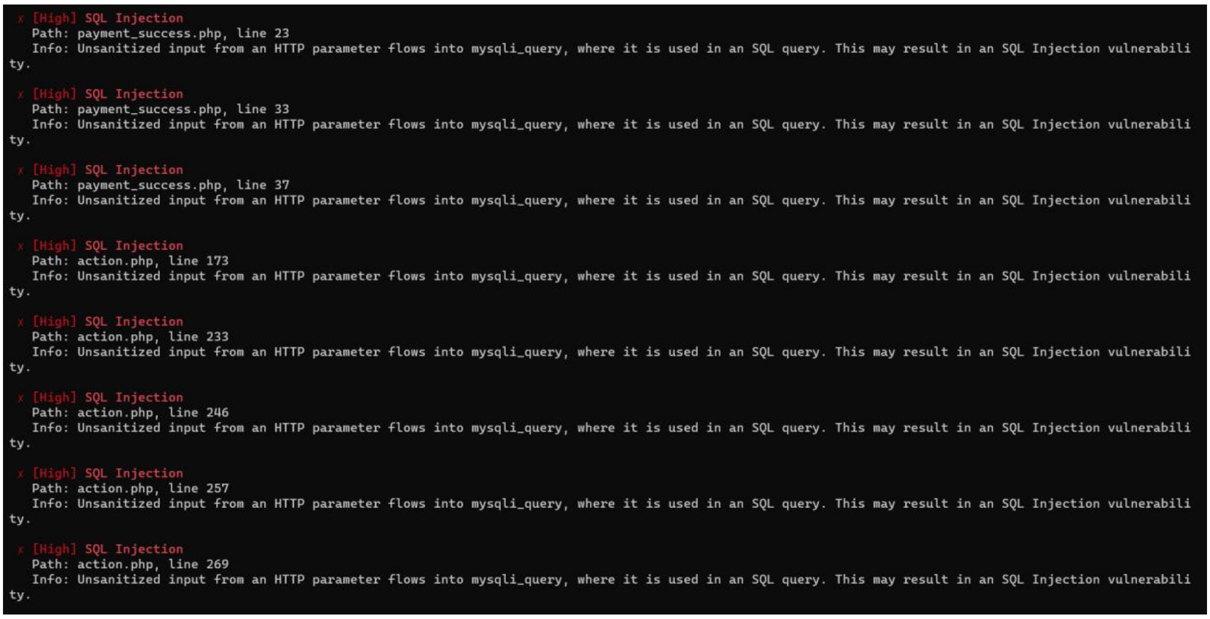
```
 ✗ [Medium] Sensitive Cookie Without 'HttpOnly' Flag
   Path: login.php, line 44
   Info: setcookie misses the HttpOnly attribute (it is false by default). Set it to true to protect the cookie from possible malicious code on client side.


 ✗ [Medium] Open Redirect
   Path: login.php, line 56
   Info: Unsanitized input from an HTTP header flows into header, where it is used as an URL to redirect the user. This may result in an Open Redirect vulne
rability.

 ✗ [Medium] Open Redirect
   Path: logout.php, line 11
   Info: Unsanitized input from an HTTP header flows into header, where it is used as an URL to redirect the user. This may result in an Open Redirect vulne
rability.

 ✗ [Medium] Use of Hardcoded Credentials
   Path: admin/server/server.php, line 16
   Info: Do not hardcode credentials in code. Found a hardcoded credential used in mysqli_connect.

 ✗ [Medium] Use of Hardcoded Credentials
   Path: admin/admin/includes/db.php, line 4
   Info: Do not hardcode credentials in code. Found a hardcoded credential used in mysqli_connect.

 ✗ [Medium] Use of Hardcoded Credentials
   Path: admin/admin/includes/db.php, line 10
   Info: Do not hardcode credentials in code. Found a hardcoded credential used in mysqli_connect.

 ✗ [Medium] Use of Hardcoded Credentials
   Path: db.php, line 4
   Info: Do not hardcode credentials in code. Found a hardcoded credential used in mysqli_connect.

 ✗ [Medium] Use of Hardcoded Credentials
   Path: db.php, line 9
   Info: Do not hardcode credentials in code. Found a hardcoded credential used in mysqli_connect.
```



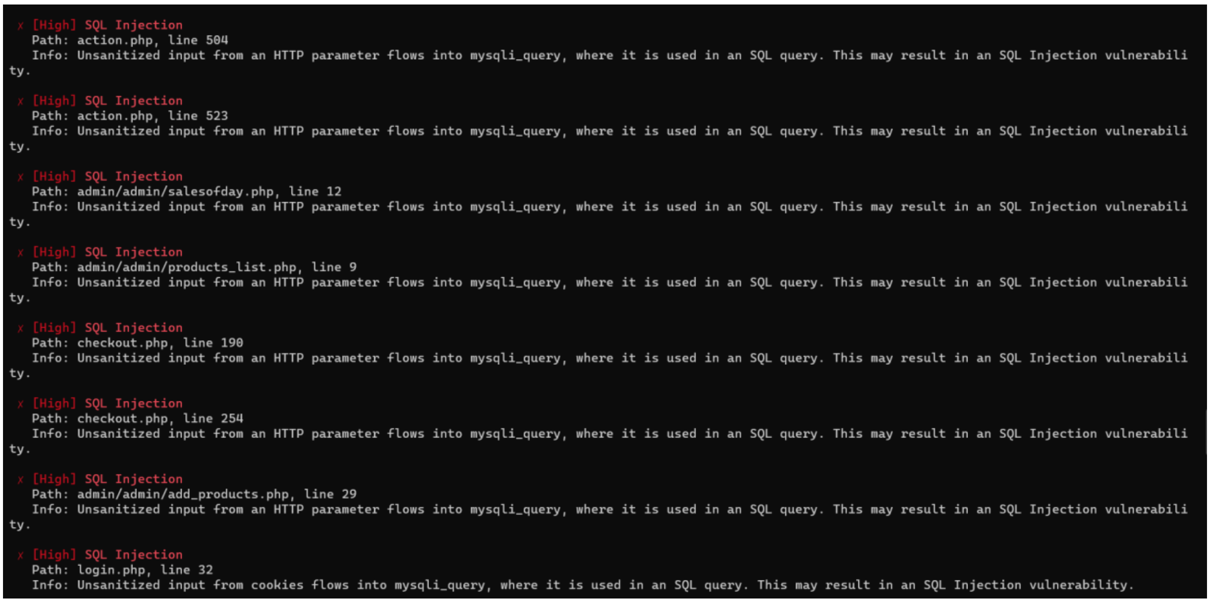
```
x [High] SQL Injection
   Path: admin/admin/orders.php, line 12
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: admin/admin/manageuser.php, line 10
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: admin/admin/edituser.php, line 16
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: homeaction.php, line 180
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: login.php, line 14
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: admin/admin/activity.php, line 12
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: checkout_process.php, line 43
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: checkout_process.php, line 60
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.
```



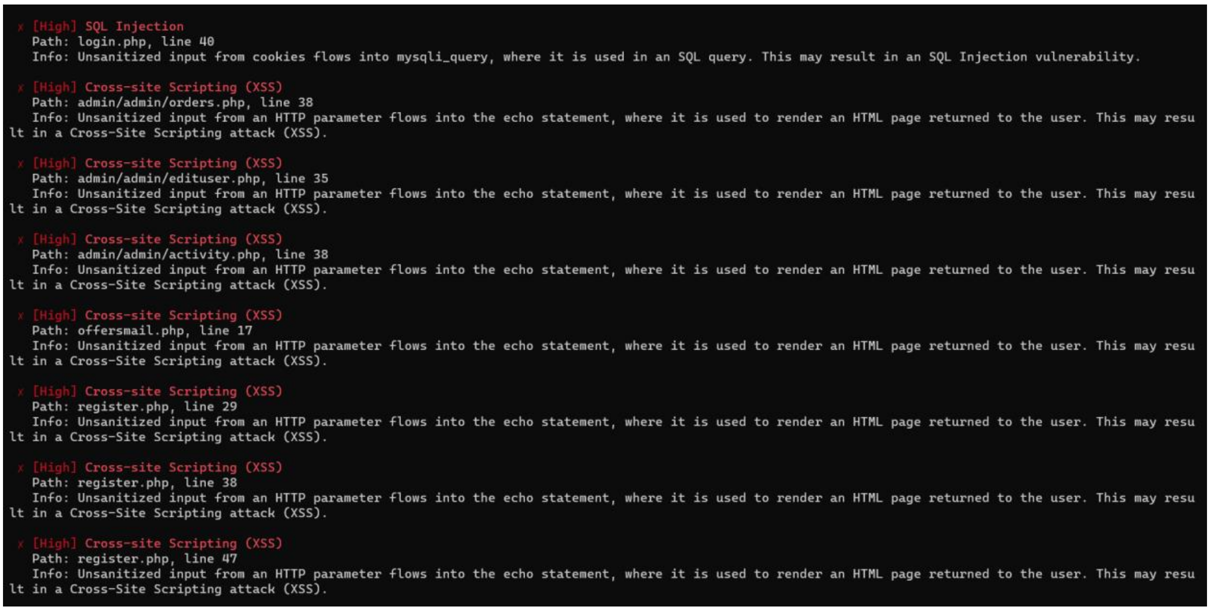
```
x [High] SQL Injection
   Path: offersmail.php, line 26
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: offersmail.php, line 41
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: admin/admin/addsuppliers.php, line 16
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: admin/admin/electroniclist.php, line 9
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: register.php, line 101
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: register.php, line 118
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: product.php, line 67
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: product.php, line 456
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
```



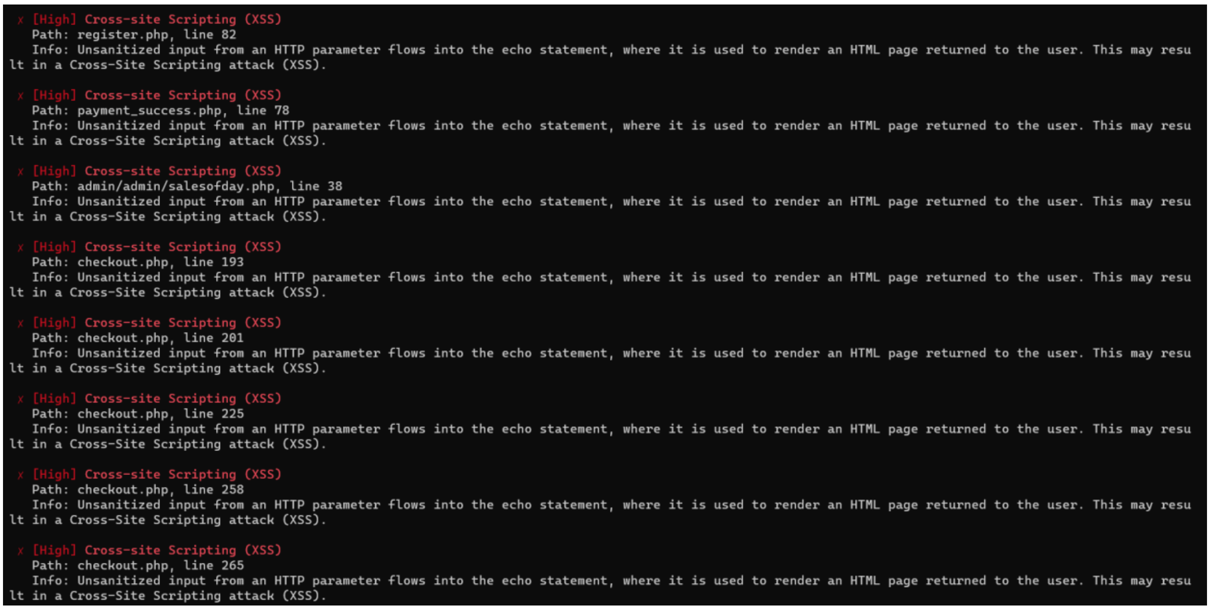
```
x [High] SQL Injection
   Path: payment_success.php, line 23
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: payment_success.php, line 33
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: payment_success.php, line 37
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: action.php, line 173
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: action.php, line 233
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: action.php, line 246
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: action.php, line 257
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

x [High] SQL Injection
   Path: action.php, line 269
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.
```



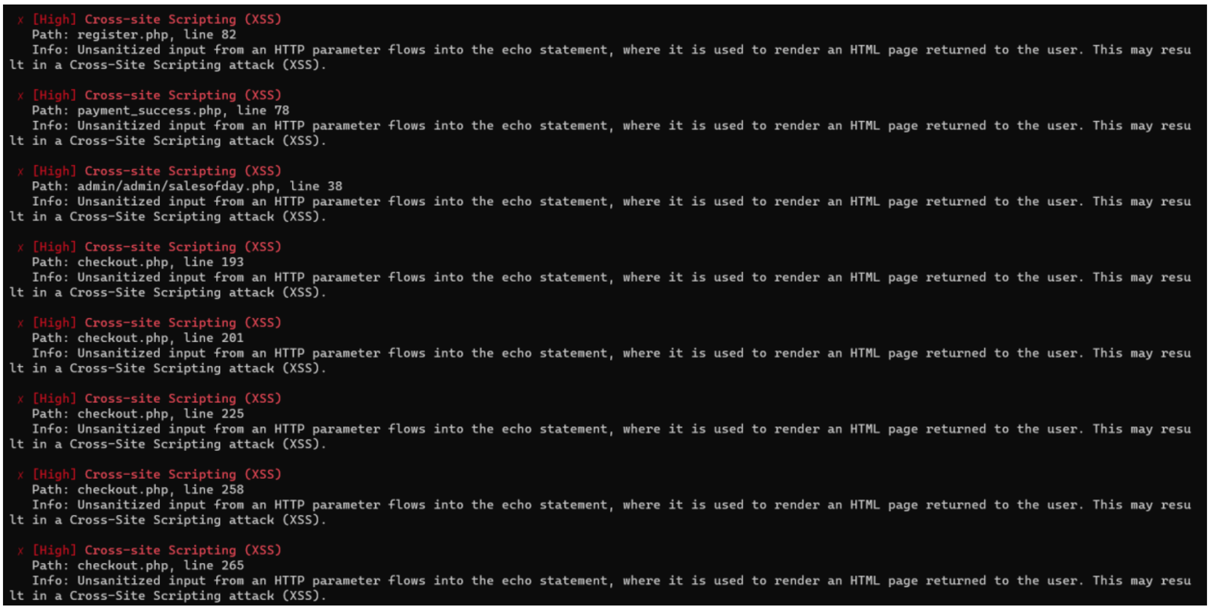
```
 ✗ [High] SQL Injection
   Path: action.php, line 504
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: action.php, line 523
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: admin/admin/salesofday.php, line 12
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: admin/admin/products_list.php, line 9
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: checkout.php, line 190
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: checkout.php, line 254
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: admin/admin/add_products.php, line 29
   Info: Unsanitized input from an HTTP parameter flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerabili
ty.

 ✗ [High] SQL Injection
   Path: login.php, line 32
   Info: Unsanitized input from cookies flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerability.
```



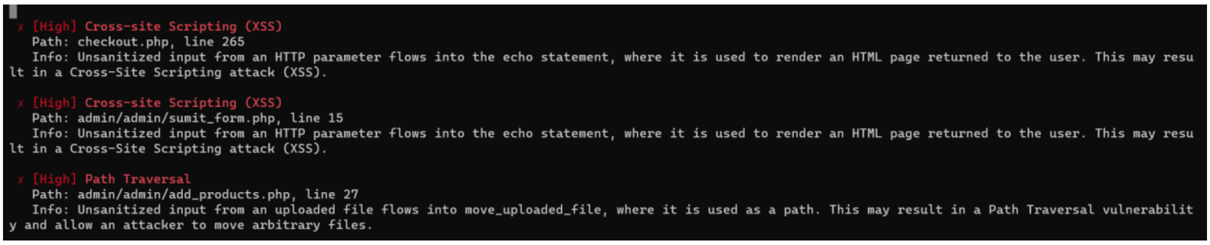
```
 ✗ [High] SQL Injection
   Path: login.php, line 40
   Info: Unsanitized input from cookies flows into mysqli_query, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

 ✗ [High] Cross-site Scripting (XSS)
   Path: admin/admin/orders.php, line 38
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: admin/admin/edituser.php, line 35
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: admin/admin/activity.php, line 38
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: offersmail.php, line 17
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: register.php, line 29
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: register.php, line 38
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: register.php, line 47
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).
```



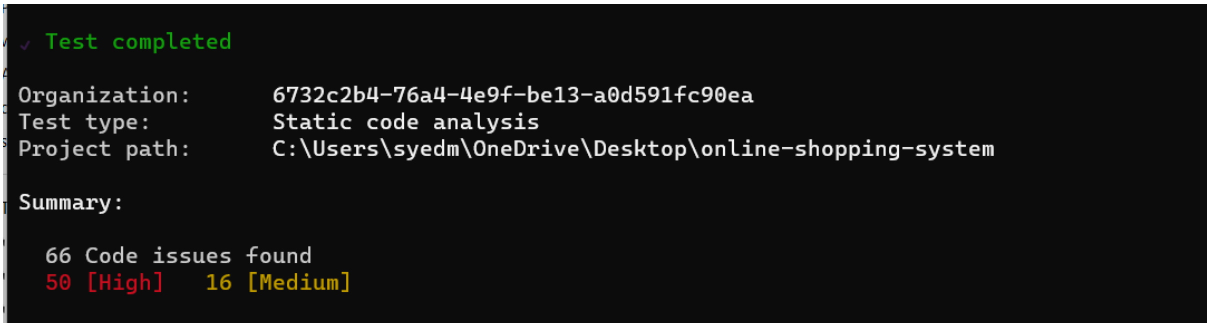
```
 ✗ [High] Cross-site Scripting (XSS)
   Path: register.php, line 82
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: payment_success.php, line 78
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: admin/admin/salesofday.php, line 38
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 193
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 201
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 225
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 258
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 265
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).
```



```
 ✗ [High] Cross-site Scripting (XSS)
   Path: register.php, line 82
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: payment_success.php, line 78
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: admin/admin/salesofday.php, line 38
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 193
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 201
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 225
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 258
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 265
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).
```



```
 ✗ [High] Cross-site Scripting (XSS)
   Path: checkout.php, line 265
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Cross-site Scripting (XSS)
   Path: admin/admin/sumit_form.php, line 15
   Info: Unsanitized input from an HTTP parameter flows into the echo statement, where it is used to render an HTML page returned to the user. This may resu
lt in a Cross-Site Scripting attack (XSS).

 ✗ [High] Path Traversal
   Path: admin/admin/add_products.php, line 27
   Info: Unsanitized input from an uploaded file flows into move_uploaded_file, where it is used as a path. This may result in a Path Traversal vulnerabilit
y and allow an attacker to move arbitrary files.
```



```
✓ Test completed

Organization:      6732c2b4-76a4-4e9f-be13-a0d591fc90ea
Test type:         Static code analysis
Project path:      C:\Users\syedm\OneDrive\Desktop\online-shopping-system

Summary:

  66 Code issues found
  50 [High]   16 [Medium]
```

Now you can see a we found many vulnerabilities in this project.

According to OWSAP ZAP we see what is CWE no of these vulnerabilities and how to prevent it

VULNERABILTES

- **CROSS SITE SCRIPTING**
- **SQL INJECTION**
- **SENSITIVE COOKIE IN HTTPS SESSION WITHOUT SECURE ATTRIBUTE**
- **USE OF PASSWORD HASH WITH INSUFFICIENT COMPUTATIONAL EFFORT**
- **SENSITIVE COOKIE WITHOUT HTTPONLY FLAG**

CROSS SITE SCRIPTING

CWE ID: CWE-79

Description: Cross-Site Scripting (XSS) occurs when an application allows untrusted data to be included in web pages without proper sanitization or validation. This allows an attacker to inject malicious scripts that are executed by victims' browsers, leading to various attacks such as session hijacking, defacement, or stealing sensitive information.

Prevention: To prevent XSS, input validation and output encoding should be used. Input validation ensures that user-supplied data matches the expected format, while output encoding prevents interpreted data from being treated as code. Implementing a Content Security Policy (CSP) and using security frameworks that handle XSS vulnerabilities can also help mitigate the risk.

SQL INJECTION

CWE ID: CWE-89

Description: SQL Injection occurs when an attacker is able to insert malicious SQL statements into a query executed by an application. This can lead to unauthorized access, data manipulation, or even complete control of the underlying database.

Prevention: The recommended prevention techniques include using parameterized queries or prepared statements with bound parameters, as well as using stored procedures or ORM (Object-Relational Mapping) frameworks. Input validation and sanitization should also be performed to ensure that user-supplied data doesn't contain malicious SQL code.

SENSITIVE COOKIE IN HTTPS SESSION WITHOUT SECURE ATTRIBUTE

CWE ID: CWE-614

Description: This vulnerability refers to a situation where an application fails to mark sensitive cookies as "Secure" when using HTTPS. This can allow an attacker to intercept the traffic and steal the cookie, potentially compromising the user's session.

Prevention: All sensitive cookies should have the 'Secure' attribute enabled, which ensures that they are only sent over encrypted connections. This can be achieved by configuring the web server or the application to enforce the 'Secure' attribute for cookies that contain sensitive information.

USE OF PASSWORD HASH WITH INSUFFICIENT COMPUTATIONAL EFFORT

CWE ID: CWE-916

Description: This vulnerability occurs when an application uses weak or insufficient computational effort in password hashing. Weak password hashing makes it easier for attackers to crack passwords using techniques like brute-forcing or rainbow table attacks.

Prevention: To prevent this vulnerability, applications should employ strong and slow hashing algorithms designed for password storage, such as bcrypt, scrypt, or Argon2. Additionally, the use of salts and iterating the hashing process multiple times (with appropriate parameters) can significantly increase the computational effort required to crack passwords.

SENSITIVE COOKIE WITHOUT HTTPONLY FLAG

CWE ID: CWE-1004

Description: The CWE-1004 vulnerability refers to a situation where a sensitive cookie is missing the 'HttpOnly' flag. The 'HttpOnly' flag is an attribute that can be set when sending a cookie from the server to the client. Its purpose is to prevent client-side scripting languages, such as JavaScript, from accessing the cookie. Without the 'HttpOnly' flag, an attacker could exploit cross-site scripting (XSS) vulnerabilities to steal the cookie and potentially compromise the user's session.

Prevention: To prevent this vulnerability, it is essential to ensure that sensitive cookies are properly marked with the 'HttpOnly' flag. Here are some prevention measures:

1. Enable 'HttpOnly' flag: When setting a cookie containing sensitive information, ensure that the 'HttpOnly' flag is enabled. This can be done through server-side code or configuration, depending on the web application framework being used.
2. Secure development practices: Employ secure coding practices to prevent XSS vulnerabilities that could be used to steal cookies. This includes input validation, output encoding, and proper sanitization of user-supplied data.
3. Content Security Policy (CSP): Implement a Content Security Policy that restricts the execution of scripts from untrusted sources. A properly configured CSP can help mitigate the impact of XSS attacks by limiting the ability of malicious scripts to execute.
4. Regular security testing: Perform regular security assessments, including vulnerability scanning and penetration testing, to identify and remediate any vulnerabilities in the application, including potential XSS issues.

LINK FOR SYNK TOOL: https://snyk.io/

****

DYANMIC ANALYSIS

Dynamic analysis refers to the examination of a program's behavior during runtime. It involves studying how the program executes, interacts with its environment, and handles various inputs or events.

Most tool used in dynamic analysis

- **Xdebug**
- **Blackfire**
- **Blackfire**
- **PHP CodeSniffer**
- **ASST**

Here we using the ASST which is also command line tool but before running this tool we setup the online shopping project

ASST: ASST is an Open Source, Source Code Scanning Tool, it is a CLI (Command Line Interface) application, developed with JavaScript (Node.js framework).

Currently concentrates on PHP and MySQL programming languages, but since its core functionalities are ready and available for everyone, programmers can contribute and add plugins or extensions to it, to add features and make it scan for other programming languages such as Java, C#, Python, etc and their frameworks. So, its infrastructure is designed to be contributed with other programmers to make it better and more novel.

The best of our knowledge, ASST is the only tool that scans PHP language according to OWASP Top 10 Web Application Security Risks.

Install these tools in your system

XAMPP: https://www.apachefriends.org/download.html

ASST: https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools

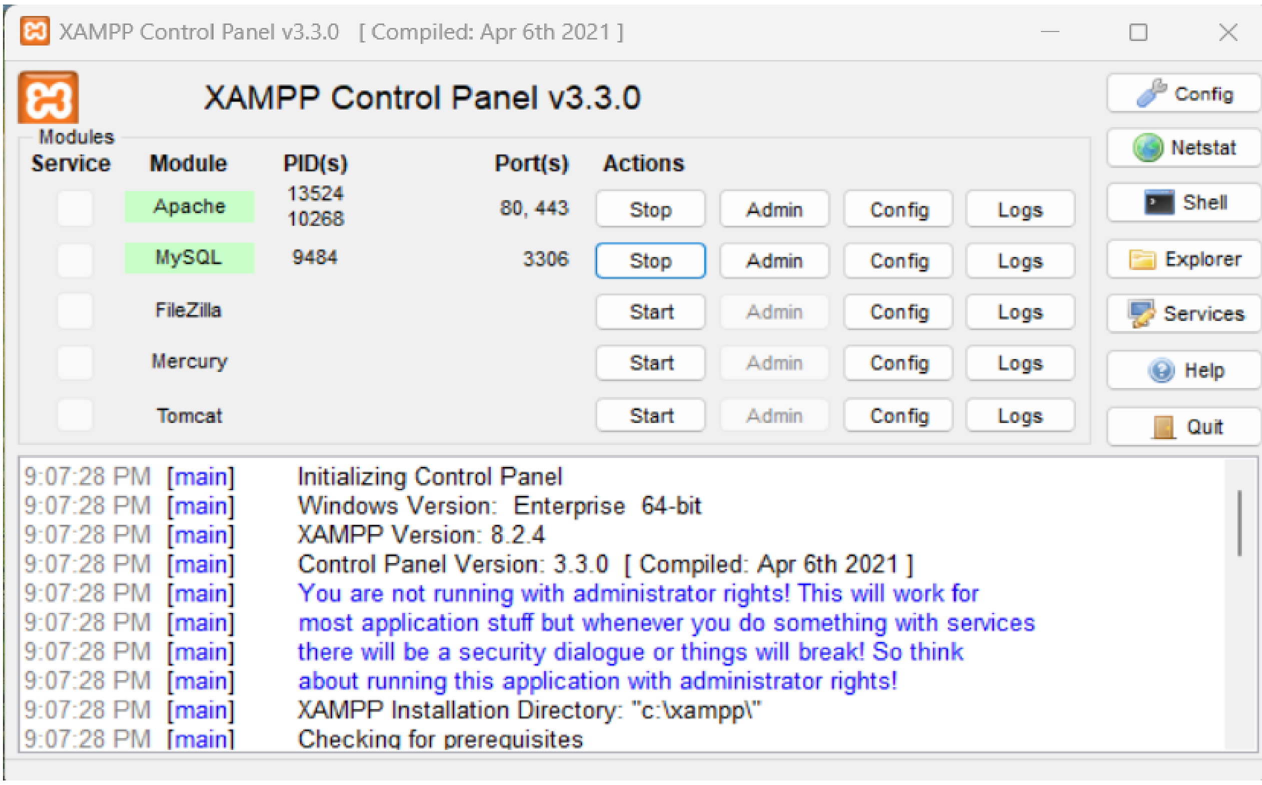
SETUP

Fist run the **XAMPP**

Open the Firefox enter this command: command “**localhost/phpmyadmin**”

Up the file of “**SQL**” and run it cmd to check vulnerabilities.

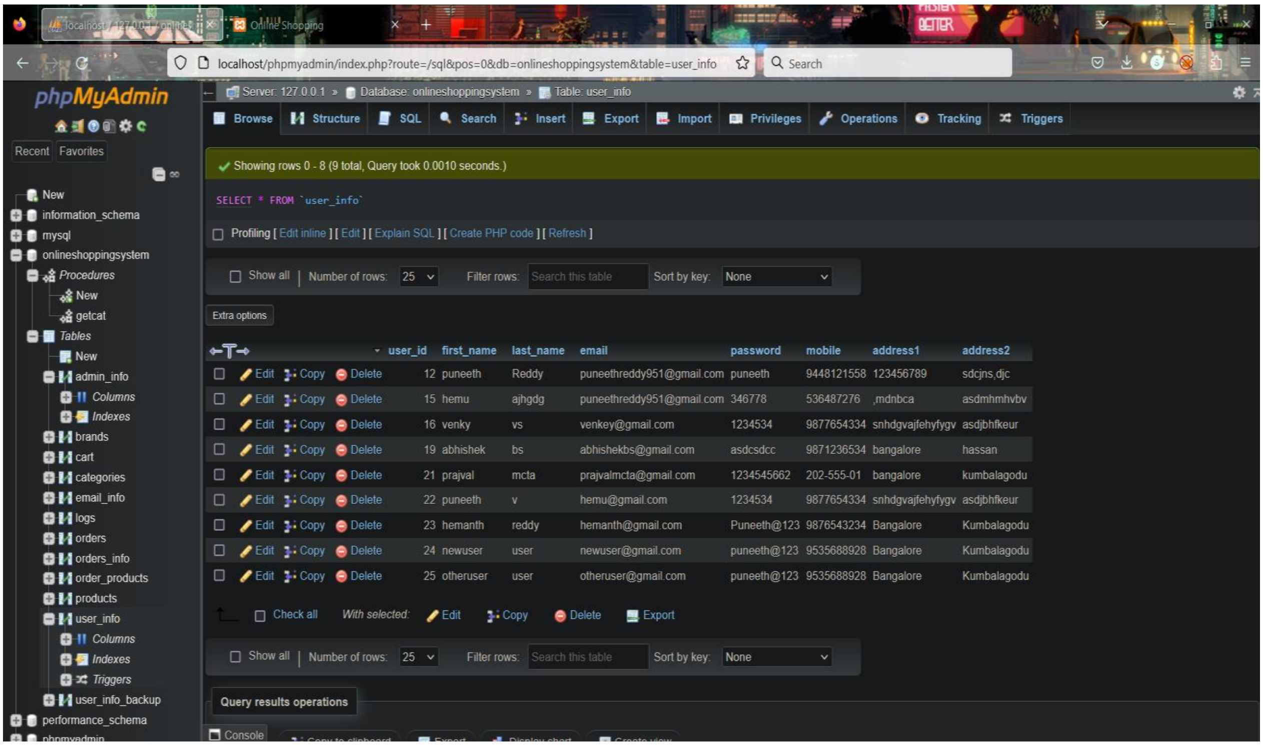
Next open the **db php** file and set according to the project



XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

Service	Module	PID(s)	Port(s)	Actions			
	Apache	13524 10268	80, 443	Stop	Admin	Config	Logs
	MySQL	9484	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
9:07:28 PM  [main]   Initializing Control Panel
9:07:28 PM  [main]   Windows Version:  Enterprise  64-bit
9:07:28 PM  [main]   XAMPP Version: 8.2.4
9:07:28 PM  [main]   Control Panel Version: 3.3.0  [ Compiled: Apr 6th 2021 ]
9:07:28 PM  [main]   You are not running with administrator rights! This will work for
9:07:28 PM  [main]   most application stuff but whenever you do something with services
9:07:28 PM  [main]   there will be a security dialogue or things will break! So think
9:07:28 PM  [main]   about running this application with administrator rights!
9:07:28 PM  [main]   XAMPP Installation Directory: "c:\xampp\"
9:07:28 PM  [main]   Checking for prerequisites
```



localhost/phpmyadmin/index.php?route=/sql&pos=0&db=onlineshoppingsystem&table=user_info

Server: 127.0.0.1 » Database: onlineshoppingsystem » Table: user_info

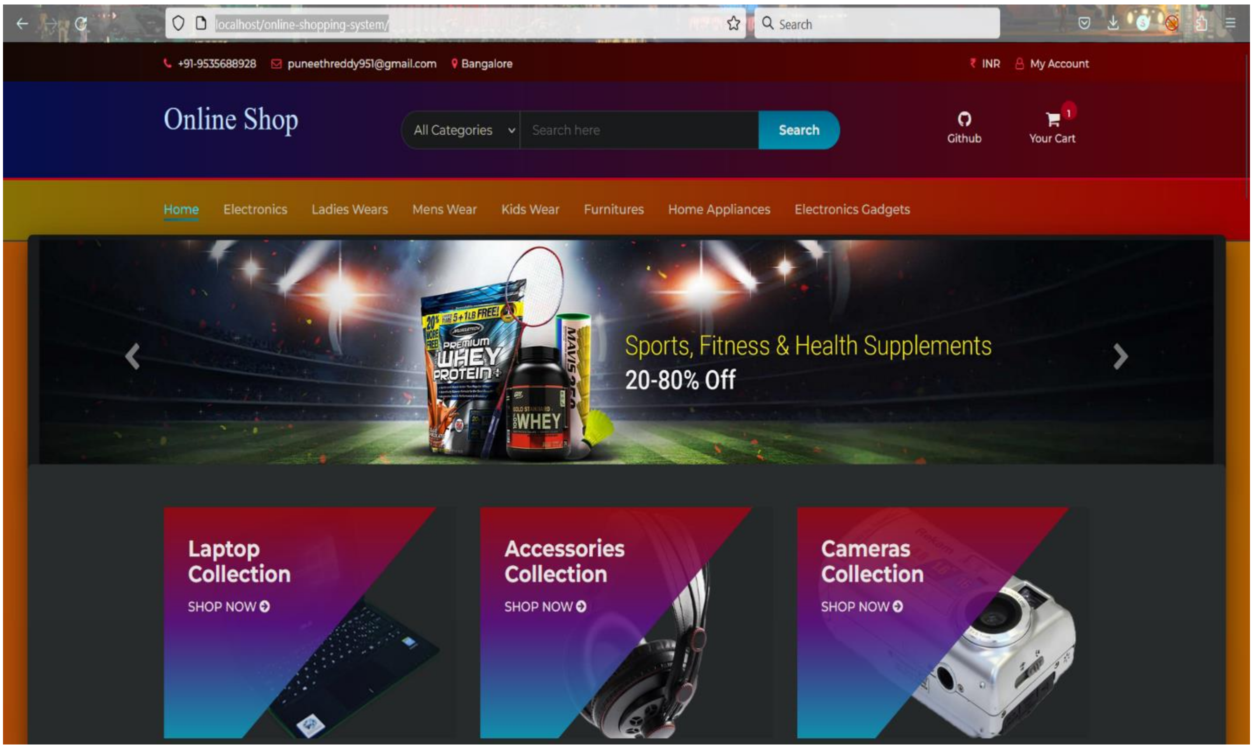
✓ Showing rows 0 - 8 (9 total, Query took 0.0010 seconds.)

```
SELECT * FROM `user_info`
```

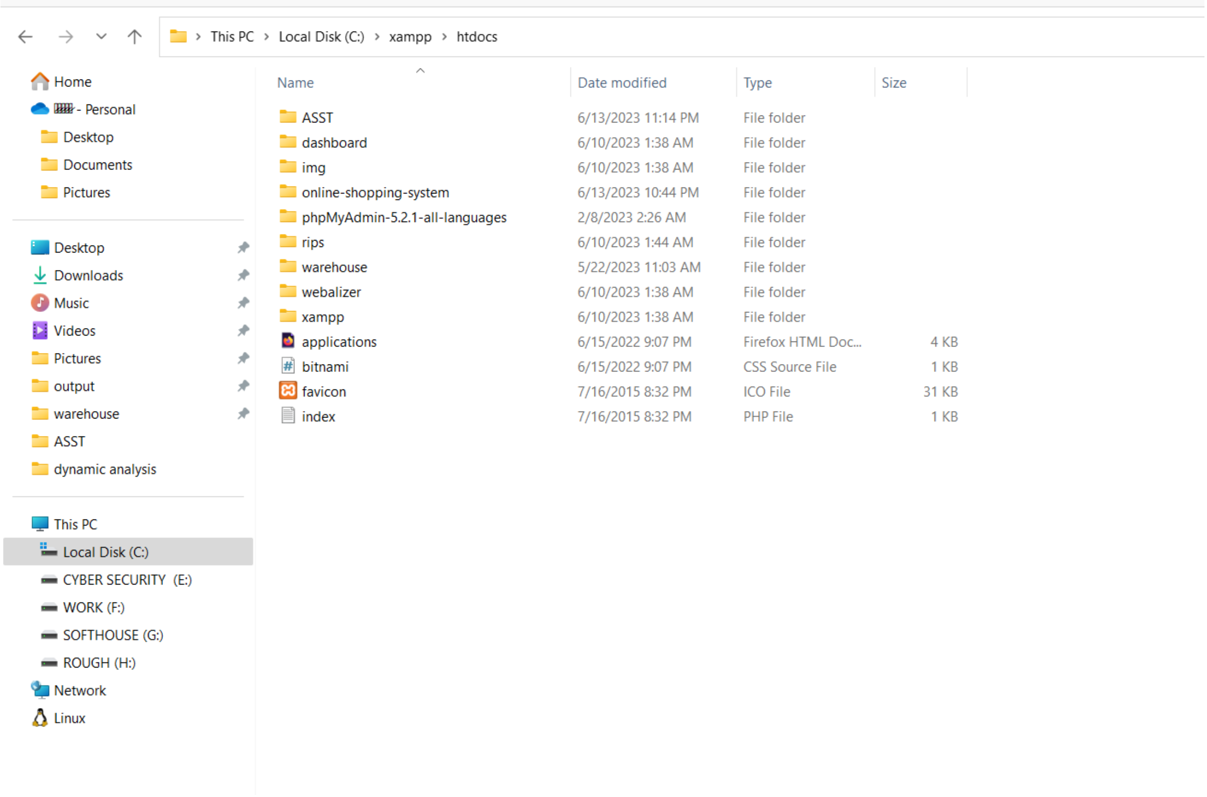
	user_id	first_name	last_name	email	password	mobile	address1	address2
□ Edit Copy Delete	12	puneeth	Reddy	puneethreddy951@gmail.com	puneeth	9448121558	123456789	sdcjns,djc
□ Edit Copy Delete	15	hemu	ajhgdg	puneethreddy951@gmail.com	346778	536487276	,mdnbca	asdmhmhvbv
□ Edit Copy Delete	16	venky	vs	venkey@gmail.com	1234534	9877654334	snhdgvajfehyfygv	asdjbhfkeur
□ Edit Copy Delete	19	abhishek	bs	abhishekbs@gmail.com	asdcsdcc	9871236534	bangalore	hassan
□ Edit Copy Delete	21	prajval	mcta	prajvalmcta@gmail.com	1234545662	202-555-01	bangalore	kumbalagodu
□ Edit Copy Delete	22	puneeth	v	hemu@gmail.com	1234534	9877654334	snhdgvajfehyfygv	asdjbhfkeur
□ Edit Copy Delete	23	hemanth	reddy	hemanth@gmail.com	Puneeth@123	9876543234	Bangalore	Kumbalagodu
□ Edit Copy Delete	24	newuser	user	newuser@gmail.com	puneeth@123	9535688928	Bangalore	Kumbalagodu
□ Edit Copy Delete	25	otheruser	user	otheruser@gmail.com	puneeth@123	9535688928	Bangalore	Kumbalagodu

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

PROJECT LOOK



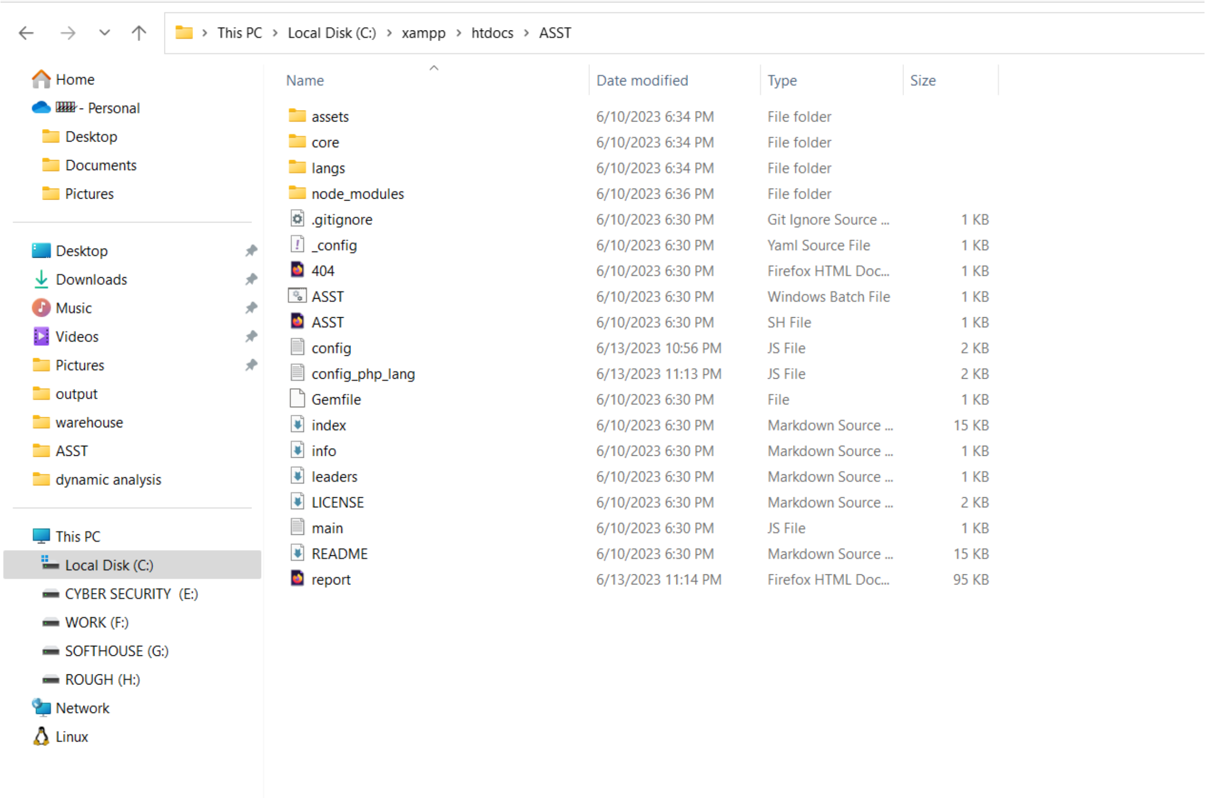
ASST FOLDER



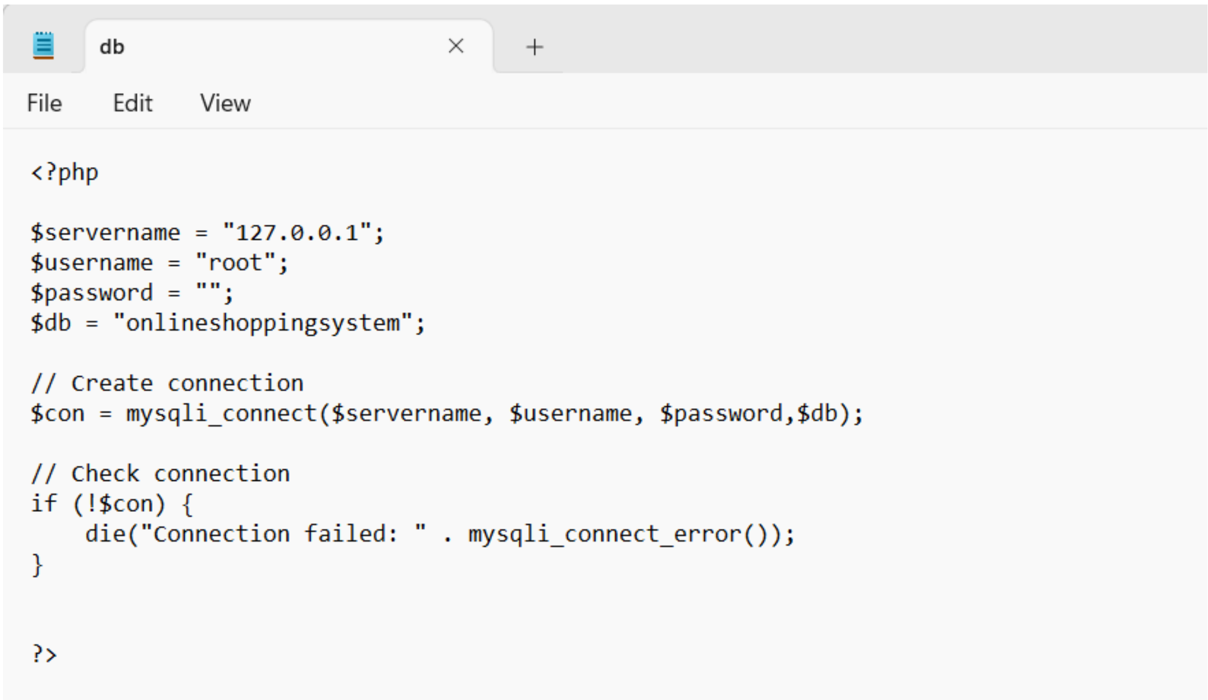
This PC › Local Disk (C:) › xampp › htdocs

Name	Date modified	Type	Size
ASST	6/13/2023 11:14 PM	File folder	
dashboard	6/10/2023 1:38 AM	File folder	
img	6/10/2023 1:38 AM	File folder	
online-shopping-system	6/13/2023 10:44 PM	File folder	
phpMyAdmin-5.2.1-all-languages	2/8/2023 2:26 AM	File folder	
rips	6/10/2023 1:44 AM	File folder	
warehouse	5/22/2023 11:03 AM	File folder	
webalizer	6/10/2023 1:38 AM	File folder	
xampp	6/10/2023 1:38 AM	File folder	
applications	6/15/2022 9:07 PM	Firefox HTML Doc...	4 KB
bitnami	6/15/2022 9:07 PM	CSS Source File	1 KB
favicon	7/16/2015 8:32 PM	ICO File	31 KB
index	7/16/2015 8:32 PM	PHP File	1 KB

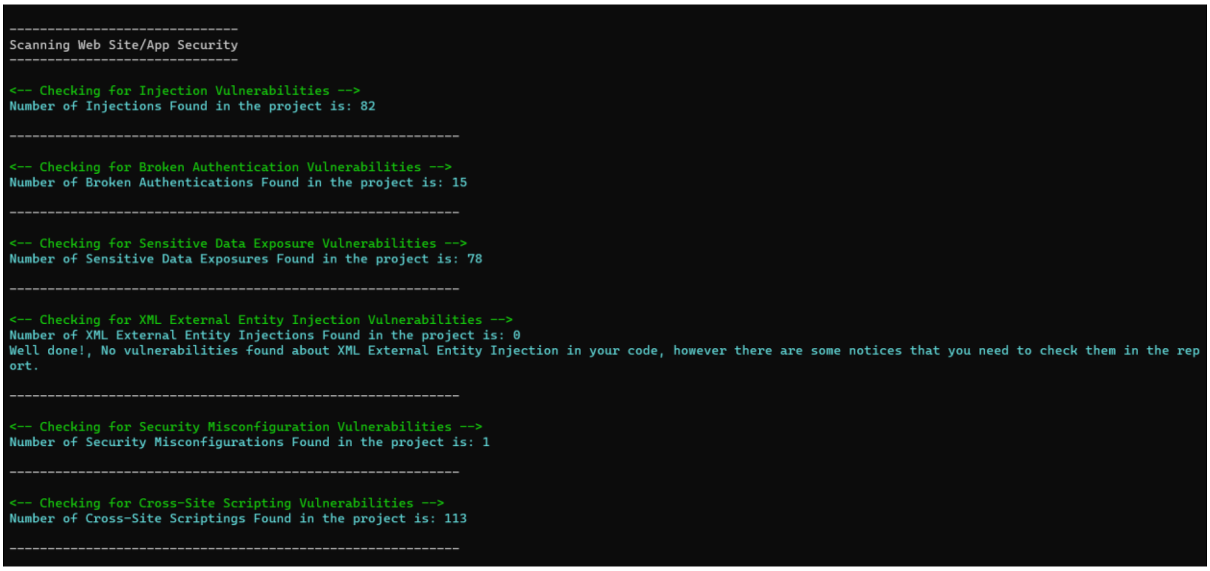
XAMPP FOLDER



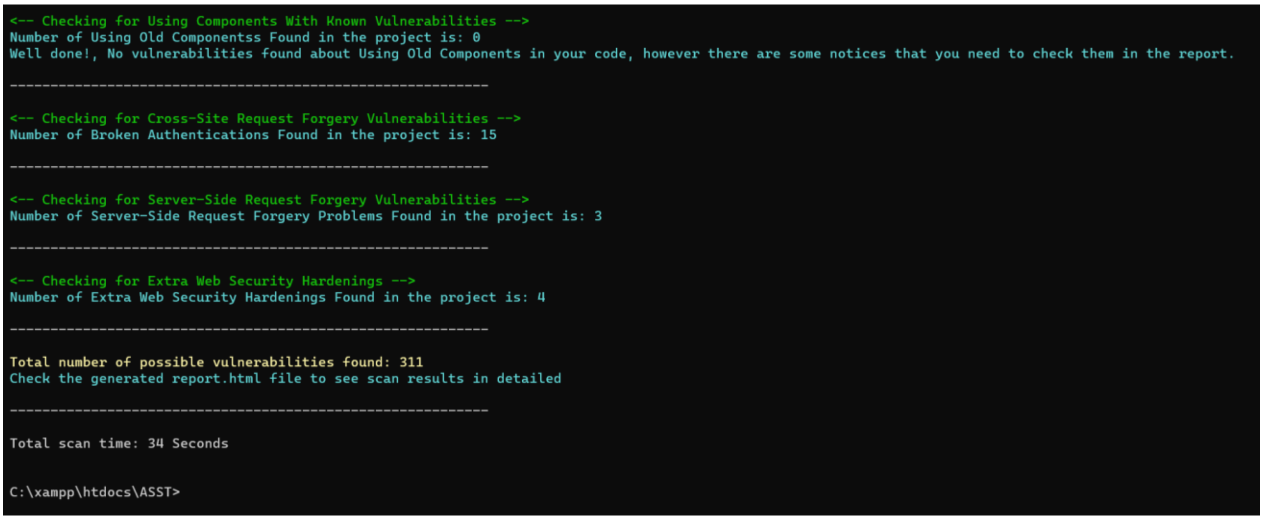
Name	Date modified	Type	Size
assets	6/10/2023 6:34 PM	File folder	
core	6/10/2023 6:34 PM	File folder	
langs	6/10/2023 6:34 PM	File folder	
node_modules	6/10/2023 6:36 PM	File folder	
.gitignore	6/10/2023 6:30 PM	Git Ignore Source ...	1 KB
_config	6/10/2023 6:30 PM	Yaml Source File	1 KB
404	6/10/2023 6:30 PM	Firefox HTML Doc...	1 KB
ASST	6/10/2023 6:30 PM	Windows Batch File	1 KB
ASST	6/10/2023 6:30 PM	SH File	1 KB
config	6/13/2023 10:56 PM	JS File	2 KB
config_php_lang	6/13/2023 11:13 PM	JS File	2 KB
Gemfile	6/10/2023 6:30 PM	File	1 KB
index	6/10/2023 6:30 PM	Markdown Source ...	15 KB
info	6/10/2023 6:30 PM	Markdown Source ...	1 KB
leaders	6/10/2023 6:30 PM	Markdown Source ...	1 KB
LICENSE	6/10/2023 6:30 PM	Markdown Source ...	2 KB
main	6/10/2023 6:30 PM	JS File	1 KB
README	6/10/2023 6:30 PM	Markdown Source ...	15 KB
report	6/13/2023 11:14 PM	Firefox HTML Doc...	95 KB



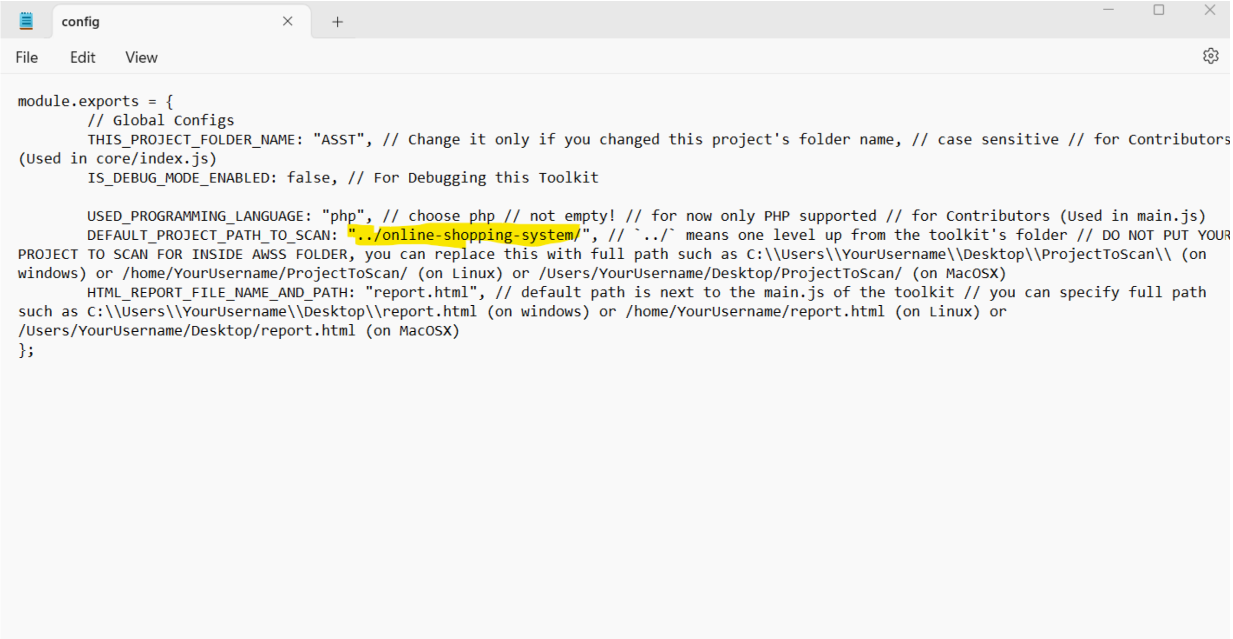
```
<?php

$servername = "127.0.0.1";
$username = "root";
$password = "";
$db = "onlineshoppingsystem";

// Create connection
$con = mysqli_connect($servername, $username, $password,$db);

// Check connection
if (!$con) {
    die("Connection failed: " . mysqli_connect_error());
}


?>
```



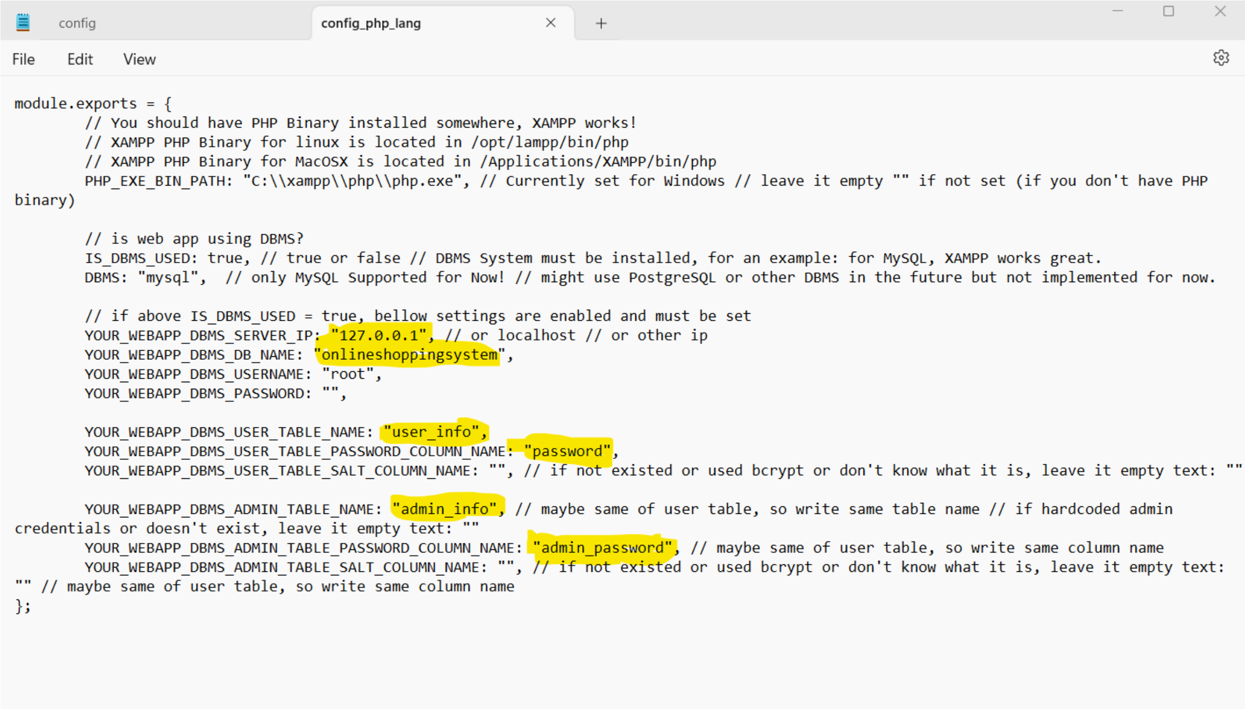
```
------------------------------
Scanning Web Site/App Security
------------------------------

<-- Checking for Injection Vulnerabilities -->
Number of Injections Found in the project is: 82

------------------------------------------------------------

<-- Checking for Broken Authentication Vulnerabilities -->
Number of Broken Authentications Found in the project is: 15

------------------------------------------------------------

<-- Checking for Sensitive Data Exposure Vulnerabilities -->
Number of Sensitive Data Exposures Found in the project is: 78

------------------------------------------------------------

<-- Checking for XML External Entity Injection Vulnerabilities -->
Number of XML External Entity Injections Found in the project is: 0
Well done!, No vulnerabilities found about XML External Entity Injection in your code, however there are some notices that you need to check them in the rep
ort.

------------------------------------------------------------

<-- Checking for Security Misconfiguration Vulnerabilities -->
Number of Security Misconfigurations Found in the project is: 1

------------------------------------------------------------

<-- Checking for Cross-Site Scripting Vulnerabilities -->
Number of Cross-Site Scriptings Found in the project is: 113

------------------------------------------------------------
```



```
<-- Checking for Using Components With Known Vulnerabilities -->
Number of Using Old Componentss Found in the project is: 0
Well done!, No vulnerabilities found about Using Old Components in your code, however there are some notices that you need to check them in the report.

------------------------------------------------------------

<-- Checking for Cross-Site Request Forgery Vulnerabilities -->
Number of Broken Authentications Found in the project is: 15

------------------------------------------------------------

<-- Checking for Server-Side Request Forgery Vulnerabilities -->
Number of Server-Side Request Forgery Problems Found in the project is: 3

------------------------------------------------------------

<-- Checking for Extra Web Security Hardenings -->
Number of Extra Web Security Hardenings Found in the project is: 4

------------------------------------------------------------

Total number of possible vulnerabilities found: 311
Check the generated report.html file to see scan results in detailed

------------------------------------------------------------

Total scan time: 34 Seconds


C:\xampp\htdocs\ASST>
```



```
module.exports = {
        // Global Configs
        THIS_PROJECT_FOLDER_NAME: "ASST", // Change it only if you changed this project's folder name, // case sensitive // for Contributors
(Used in core/index.js)
        IS_DEBUG_MODE_ENABLED: false, // For Debugging this Toolkit

        USED_PROGRAMMING_LANGUAGE: "php", // choose php // not empty! // for now only PHP supported // for Contributors (Used in main.js)
        DEFAULT_PROJECT_PATH_TO_SCAN: "../online-shopping-system/", // `../` means one level up from the toolkit's folder // DO NOT PUT YOUR
PROJECT TO SCAN FOR INSIDE AWSS FOLDER, you can replace this with full path such as C:\\Users\\YourUsername\\Desktop\\ProjectToScan\\ (on
windows) or /home/YourUsername/ProjectToScan/ (on Linux) or /Users/YourUsername/Desktop/ProjectToScan/ (on MacOSX)
        HTML_REPORT_FILE_NAME_AND_PATH: "report.html", // default path is next to the main.js of the toolkit // you can specify full path
such as C:\\Users\\YourUsername\\Desktop\\report.html (on windows) or /home/YourUsername/report.html (on Linux) or
/Users/YourUsername/Desktop/report.html (on MacOSX)
};
```



```
module.exports = {
        // You should have PHP Binary installed somewhere, XAMPP works!
        // XAMPP PHP Binary for linux is located in /opt/lampp/bin/php
        // XAMPP PHP Binary for MacOSX is located in /Applications/XAMPP/bin/php
        PHP_EXE_BIN_PATH: "C:\\xampp\\php\\php.exe", // Currently set for Windows // leave it empty "" if not set (if you don't have PHP
binary)

        // is web app using DBMS?
        IS_DBMS_USED: true, // true or false // DBMS System must be installed, for an example: for MySQL, XAMPP works great.
        DBMS: "mysql",  // only MySQL Supported for Now! // might use PostgreSQL or other DBMS in the future but not implemented for now.

        // if above IS_DBMS_USED = true, bellow settings are enabled and must be set
        YOUR_WEBAPP_DBMS_SERVER_IP: "127.0.0.1", // or localhost // or other ip
        YOUR_WEBAPP_DBMS_DB_NAME: "onlineshoppingsystem",
        YOUR_WEBAPP_DBMS_USERNAME: "root",
        YOUR_WEBAPP_DBMS_PASSWORD: "",

        YOUR_WEBAPP_DBMS_USER_TABLE_NAME: "user_info",
        YOUR_WEBAPP_DBMS_USER_TABLE_PASSWORD_COLUMN_NAME: "password",
        YOUR_WEBAPP_DBMS_USER_TABLE_SALT_COLUMN_NAME: "", // if not existed or used bcrypt or don't know what it is, leave it empty text: ""

        YOUR_WEBAPP_DBMS_ADMIN_TABLE_NAME: "admin_info", // maybe same of user table, so write same table name // if hardcoded admin
credentials or doesn't exist, leave it empty text: ""
        YOUR_WEBAPP_DBMS_ADMIN_TABLE_PASSWORD_COLUMN_NAME: "admin_password", // maybe same of user table, so write same column name
        YOUR_WEBAPP_DBMS_ADMIN_TABLE_SALT_COLUMN_NAME: "", // if not existed or used bcrypt or don't know what it is, leave it empty text:
"" // maybe same of user table, so write same column name
};
```

Now you can see a we found many vulnerabilities in this project.

According to OWSAP ZAP we see what is CWE no of these vulnerabilities and how to prevent it

VULNERABILTES

- **INJECTION VULNERABILITIES**
- **AUTHENICATION VULNERABILITES**
- **SENSITIVE DATA EXPOSURE VULNERABILITIES**
- **SECURITY MISCONFIGURATION VULNERABILITIES**
- **CROSS SITE REQUEST FORHERY VULNERABILTIES**
- **SERVER-SIDE REQUEST FORGERY VULNERABILITES**
- **EXTRA WEB SECUIRTY HARDENINGS**

INJECTION VULNERABILITIES

CEW ID: CEW-005

Description: Injection vulnerabilities occur when untrusted data is inserted into an application and interpreted as code or commands by the system. This can lead to unauthorized access, data loss, or even complete system compromise.

Prevention: To prevent injection vulnerabilities, use parameterized queries or prepared statements instead of concatenating user input into queries directly. Additionally, input validation and proper encoding or escaping techniques should be applied to user input to mitigate the risk of injection attacks.

AUTHENICATION VULNERABILITES

CEW ID: CEW-006

Description: Authentication vulnerabilities involve weaknesses in the mechanisms used for user authentication and session management. Common issues include weak or guessable passwords, inadequate password storage, session fixation, or session hijacking.

Prevention: To address authentication vulnerabilities, enforce strong password policies, implement multi-factor authentication (MFA), use secure session management techniques like session tokens or cookies with secure attributes, and protect against brute-force attacks.

SENSITIVE DATA EXPOSURE VULNERABILITIES

CEW ID: CEW-007

Description: Sensitive data exposure refers to situations where sensitive information, such as passwords, credit card details, or personal data, is not adequately protected. This vulnerability allows attackers to gain access to the data, leading to potential identity theft, financial fraud, or privacy breaches.

Prevention: To prevent sensitive data exposure, ensure sensitive data is properly encrypted at rest and in transit. Implement strong access controls, securely handle and store sensitive data, avoid storing unnecessary data, and follow industry-standard encryption practices.

SECURITY MISCONFIGURATION VULNERABILITIES

CEW ID: CEW-008

Description: Security misconfigurations occur when the application, server, or network infrastructure is not properly configured. Examples include default or weak passwords, unused services or ports, outdated software versions, or insecure access controls. Attackers can exploit these misconfigurations to gain unauthorized access or perform other malicious activities.

Prevention: To mitigate security misconfiguration vulnerabilities, follow secure configuration guides for your application, server, and framework. Remove unnecessary services, keep software and libraries up to date, use secure default configurations, and perform regular security assessments and audits.

CROSS SITE REQUEST FORHERY VULNERABILTIES

CEW ID: CEW-009

Description: CSRF vulnerabilities allow an attacker to trick a victim into performing unwanted actions on a website where they are authenticated. By leveraging the trust placed in the user's browser, the attacker can execute malicious actions, potentially leading to unauthorized data modification or account compromise.

Prevention: To prevent CSRF attacks, implement anti-CSRF tokens in forms and AJAX requests to validate the authenticity of requests. Use the SameSite attribute for cookies, enforce strict referer policies, and employ secure coding practices to ensure that sensitive actions require explicit user consent.

SERVER-SIDE REQUEST FORGERY VULNERABILITES

CEW ID: CEW-010

Description: SSRF vulnerabilities occur when an attacker can manipulate the server-side requests made by an application. By tricking the server into making requests to unintended or malicious resources, the attacker can perform actions such as reading internal files, accessing unauthorized services, or exploiting internal systems.

Prevention: To prevent SSRF vulnerabilities, validate and sanitize all user-supplied input, especially URLs. Whitelist allowed protocols and domains, implement strict server-side request filtering, and restrict access to sensitive internal resources by using firewalls and proper network segmentation.

EXTRA WEB SECUIRTY HARDENINGS

CEW ID: CEW-011

Description: Extra web security hardening refers to additional measures taken to strengthen the security of web applications beyond the basic security requirements. This can include implementing additional security controls, conducting regular security testing, applying secure coding practices, and keeping up with the latest security updates and patches.

Prevention: Implement security best practices such as using secure coding frameworks and libraries, employing secure development practices like threat modeling and code reviews, conducting regular security assessments, performing penetration testing, and staying up to date with security patches and updates for all software components.

****