WhiteRook Cyber

Continuous Vulnerability Assessment Platform

Web App Pen Test Report

test App 1

21 Oct 2024

Copyright

This report is intended solely for the use of the individual or entity to whom it is addressed. Unauthorised distribution, reproduction, or use of this document, in whole or in part, is strictly prohibited. The information contained herein is confidential and may contain sensitive information regarding security vulnerabilities and assessments.

For permission to use or distribute this document, please email contact@whiterookcyber.com.au

contact@whiterookcyber.com.au
1300794777

Document Details

Action	Date
Completed On	21 Oct 2024
Generated on	30 Nov 2024

Table of Contents

Introduction

The data present in this document is the result of the penetration test conducted on "**https://beaglehack.com/**". The test generates a checklist of potential compliance issues by checking "**test App 1**" for vulnerabilities including insecure data collection forms, insecure cookies, third-party links, cross site scripting vulnerabilities, SQL injection, etc.

Using this data, you can then proactively filter and organize identified issues to ensure that your organization's most critical regulatory compliance concerns are addressed as soon as possible. While this information is intended to greatly improve the efficiency with which you can remediate compliance issues, it does not intend to represent the full scope of compliance with the Payment Card Industry Data Security Standard (PCI DSS) regulations.

These results are only a subset of the entire PCI DSS compliance. In addition, "**White Rook Cyber**" holds no responsibility for any use or misuse of information presented in this report.

The next section provides the non-technical team with a summary of all key findings and relates the impact it will have on your business. Section 3 provides the technical team with a detailed report of individual vulnerabilities along with its mitigation procedures. This detailed report will help your development team to improve the overall security of the system.

Executive Summary

The chart, tables, and graphs displayed below is to provide you the summary of all the vulnerabilities present in "**test App 1**" based on the status and severity. The severity of each vulnerability is calculated based on its occurrence, frequency, and impact upon the asset.

By examining these representations generated by "**White Rook Cyber**", you can gain insights into state of "**test App 1**"'s current security posture.

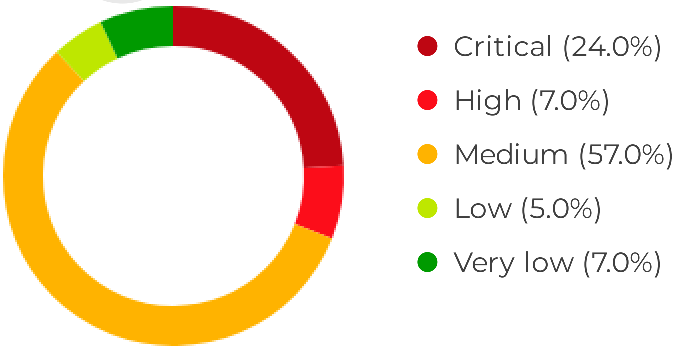
Catalog

Status	Count
New	41
Not Fixed	0
Reopened	0
Fixed	0

Graphical Summary

2

Vulnerability Distribution



Tabular Summary

Category	Count
Critical	10
High	3
Medium	23
Low	2
Very Low	3

PCI DSS Compliance Summary

Sl No	PCI DSS Sub-requirements	PASS/FAIL
1	Requirement 1.2: Network security controls (NSCs) are configured and maintained.	PASSED
2	Requirement 1.4: Network connections between trusted and untrusted networks are controlled.	PASSED
3	Requirement 2.2: System components are configured and managed securely.	PASSED
4	Requirement 3.2: Storage of account data is kept to a minimum.	PASSED
5	Requirement 4.2: PAN is protected with strong cryptography during transmission.	PASSED
6	Requirement 6.2: Bespoke and custom software are developed securely.	FAILED
7	Requirement 6.3: Security vulnerabilities are identified and addressed.	PASSED
8	Requirement 6.4: Public-facing web applications are protected against attacks.	FAILED

	managed securely.	PASSED
10	Requirement 7.2: Access to system components and data is appropriately defined and assigned.	FAILED
11	Requirement 8.2: User identification and related accounts for users and administrators are strictly managed throughout an account’s lifecycle.	PASSED
12	Requirement 8.3: Strong authentication for users and administrators is established and managed.	PASSED
13	Requirement 8.6: Use of application and system accounts and associated authentication factors is strictly managed.	PASSED
14	Requirement 11.6: Unauthorized changes on payment pages are detected and responded to.	PASSED

Application Info

The details of the application are as listed below:

Project name	test App 1
Application name	Test project
URL	https://beaglehack.com/
Test completed on	21 Oct 2024

Domain Details

Name	Value
Domain name	beaglehack.com
Domain status	Valid
Created on	11 Jan 2023
Updated on	17 Feb 2023
Expires on	11 Jan 2028
Days to expire	1177

Technical Summary

PCIDSS Compliance Failed Controls

Develop and Maintain Secure Systems and Software

Requirement 6.2: Bespoke and custom software are developed securely.

Software engineering techniques or other methods are defined and in use by software development personnel to prevent or mitigate common software attacks and related vulnerabilities in bespoke and custom software, including but not limited to the following:

- Injection attacks, including SQL, LDAP, XPath, or other command, parameter, object, fault, or injection-type flaws.
- Attacks on data and data structures, including attempts to manipulate buffers, pointers, input data, or shared data.
- Attacks on cryptography usage, including attempts to exploit weak, insecure, or inappropriate cryptographic implementations, algorithms, cipher suites, or modes of operation.
- Attacks on business logic, including attempts to abuse or bypass application features and functionalities through the manipulation of APIs, communication protocols and channels, client-side functionality, or other system/application functions and resources. This includes cross-site scripting (XSS) and cross-site request forgery(CSRF).
- Attacks on access control mechanisms, including attempts to bypass or abuse identification, authentication, or authorization mechanisms, or attempts to exploit weaknesses in the implementation of such mechanisms.
- Attacks via any “high-risk” vulnerabilities identified in the vulnerability identification process, as defined in Requirement 6.3.1.

Cross Site Scripting

OWASP 2013-A3 OWASP 2017-A7 OWASP 2021-A3 PCI V3.2-6.5.7 OWASP PC-C4 CAPEC-19 CWE-79 Subpart C, HIPAA-164.306(a)(2) ISO27001-A.14.2.5 WASC-8 WSTG-INPV-02 A.12.2.1 PCI v4.0-6.2.4

- Likelihood: High
- Impact: High
- Risk level: Critical

Issue Description:

Cross-site Scripting (XSS) is a client-side code injection attack. Using this technique, an attacker can execute malicious scripts into a legitimate website or web application. This server has a vulnerability that allows an attacker to send malicious code to the user. A browser cannot foresee the script on the website. So, it cannot judge if a website should be trusted or not. The browser will execute the script allowing the attacker to access any cookie or session token retained by the browser.

Recommendations:

Step 1: Validate and Sanitize User Inputs

The first step in mitigating Cross Site Scripting (XSS) vulnerabilities is to ensure all user inputs are properly validated and sanitized. This can be done by implementing a whitelist of acceptable inputs and rejecting any input that does not match the whitelist.
For example, if the application is expecting a name, only accept alphanumeric characters and spaces.

```
function validateName(name) {
const regex = /^[A-Za-z0-9\s]+$/;
return regex.test(name);
}
```

Step 2: HTML Encoding

The second step in mitigating XSS is to HTML encode all user inputs before displaying them on the page. This prevents malicious scripts from executing in the browser.
For example, if the application is displaying a user's name on the page, it should be HTML encoded before rendering.

```
function htmlEncodeName(name) {
return name.replace(/&/g, '&amp;')
.replace(/</g, '&lt;')
.replace(/>/g, '&gt;')
.replace(/"/g, '&quot;')
.replace(/'/g, '&#x27;')
.replace(/\//g, '&#x2F;');
}
```

Step 3: Content Security Policy (CSP)

The third step in mitigating XSS is to implement a Content Security Policy (CSP). A CSP is a set of rules that specify what resources a browser is allowed to load. This prevents malicious scripts from being loaded from untrusted sources.
For example, the following CSP will only allow scripts and styles to be loaded from the same domain as the application.

```
Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self';
```

Occurrences:

Occurrence 001 Status: New

Vulnerable Parameter	:	include
Payload	:	-->'"/></sCript><deTailS open x=">" ontoggle=(confirm)("beagle_injection_attack")``>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/csp/

Occurrence 002 Status: New

Vulnerable Parameter	:	mtxMessage
Payload	:	<script>alert('beagle_injection_attack')</script>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 003 Status: New

Vulnerable Parameter	:	txtName
Payload	:	<script>alert('beagle_injection_attack')</script>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 004 Status: New

Vulnerable Parameter	:	default
Payload	:	<script>alert('beagle_injection_attack')</script>
Method	:	GET
URL	:	https://beaglehack.com/vulnerabilities/xss_d/?default=French

Occurrence 005 Status: New

Vulnerable Parameter	:	name
Payload	:	<script>alert('beagle_injection_attack')</script>
Method	:	GET
URL	:	https://beaglehack.com/vulnerabilities/xss_r/?name=sampletext

Remote OS Command Injection

OWASP 2013-A1 OWASP 2017-A1 OWASP 2021-A3 PCI V3.2-6.5.1 OWASP PC-C5 CAPEC-88 CWE-78 Subpart C, HIPAA-164.306(a)(2) ISO27001-A.14.2.5 WASC-31 CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H WSTG-INPV-11 A.10.4.1 PCI v4.0-6.2.4

- Likelihood: High

- Impact: High
- Risk level: Critical

Issue Description:

This server passes unsafe user-supplied data to the system shell. The attacker can supply operating system commands and can execute with the privileges of the server application.

Recommendations:

Step 1: Understand the Vulnerability The first step in mitigating any vulnerability is to fully understand it. In this case, Remote OS Command Injection is a type of vulnerability where an attacker can execute arbitrary operating system commands on a web server. This can lead to a complete compromise of the server and potentially the entire network.
Step 2: Identify and Validate the Vulnerable Code The next step is to identify and validate the code that is vulnerable to Remote OS Command Injection. This can be done through manual code review or using automated tools such as a vulnerability scanner. Once the vulnerable code is identified, it is important to validate that it is indeed vulnerable by attempting to exploit it.
Step 3: Sanitize User Input The root cause of Remote OS Command Injection is improper handling of user input. Therefore, the most effective way to mitigate this vulnerability is to properly sanitize all user input before using it in a command. This includes both form inputs and URL parameters.
Step 4: Use Parameterized Queries Instead of directly concatenating user input into a command, it is recommended to use parameterized queries. This ensures that the user input is treated as data and not as a command to be executed. Parameterized queries can be implemented in different ways depending on the programming language and framework being used.
Example in PHP:

```
// Using PDO (PHP Data Objects)
$stmt = $pdo->prepare('SELECT * FROM users WHERE username = :username');
$stmt->execute(['username' => $_POST['username']]);
$user = $stmt->fetch();

// Using mysqli
$stmt = $db->prepare('SELECT * FROM users WHERE username = ?');
$stmt->bind_param('s', $_POST['username']);
$stmt->execute();
$user = $stmt->get_result()->fetch_assoc();
```

Step 5: Use Whitelisting Another approach to mitigating Remote OS Command Injection is to use whitelisting. This means only allowing a specific set of characters or values to be accepted as user input. This can be done by implementing input validation and rejecting any input that does not match the specified criteria. Example in Java:

```
// Only allow alphanumeric characters
if (!input.matches("^[a-zA-Z0-9]*$")) {
// Reject input and display an error message
}
```

Step 6: Implement Least Privilege Principle In addition to sanitizing user input, it is important to follow the principle of least privilege. This means giving only the necessary permissions to the user or process that is executing the command. For example, if a web application only needs to read data from a database, it should not have permissions to write or execute commands on the server.
Step 7: Keep Software and Libraries Up to Date Outdated software and libraries often contain known vulnerabilities that can be exploited by attackers. Therefore, it is important to keep all software and libraries used in the web application up to date. This includes the web server, programming language, and any third-party libraries.

Occurrences:

Occurrence 006 Status: New

Parameter	:	ip
Attack	:	;id;
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/exec/

Directory Traversal

OWASP 2013-A7 OWASP 2017-A5 OWASP 2021-A1 CAPEC-213 CWE-22

Subpart C, HIPAA-164.308(a)(1)(ii)(B) WASC-33 WSTG-ATHZ-01 A.10.7.4 PCI v4.0-6.2.4

- Likelihood: High

- Impact: High
- Risk level: Critical

Issue Description:

Directory traversal is an HTTP attack that allows attackers to access restricted directories. It also executes commands outside of the web server’s root directory. The access to files is not limited by system operational access control. This leads to Directory traversal attacks, that aims to access files and directories that are stored outside the web root folder. The server having this vulnerability will allow an attacker to read any files from the server. This vulnerability will also allow the attacker to read or include files server or directories in the server. This can have major repercussions for the web applications.

Recommendations:

1. **Enforce Input Validation**: All user input should be validated for malicious characters like ../ and ..\ which are used to traverse directories. If any of these characters are detected, the application should reject the input and return an appropriate error message.
2. **Restrict Access to Directories**: It is important to ensure that only authorized users have access to sensitive directories. This can be done by configuring the web server to deny access to certain directories. For example, in Apache, the .htaccess file can be used to deny access to certain directories.

```
<Directory /var/www/html/securedir>
Order Deny,Allow
Deny from all
</Directory>
```

3. **Enforce Strong Authentication**: It is important to ensure that only authenticated users have access to sensitive directories. This can be done by implementing strong authentication mechanisms like two-factor authentication.
4. **Implement Security Logging**: It is important to monitor the application for any suspicious activities. This can be done by implementing logging mechanisms that log all user activities. This will help in detecting any malicious activities in the application.

Occurrences:

Occurrence 007 Status: New

Parameter	:	page
Attack	:	/etc/passwd
Method	:	GET
URL	:	https://beaglehack.com/vulnerabilities/fi/?page=%2Fetc%2Fpasswd

MySQL Boolean-based Blind SQL Injection (SQLi)

OWASP 2013-A1 OWASP 2017-A1 OWASP 2021-A3 PCI V3.2-6.5.1 OWASP PC-C3 CAPEC-66
CWE-89 Subpart C, HIPAA-164.306(a)(2) ISO27001-A.14.2.5 WASC-19
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H WSTG-INPV-05 A.12.2.1 PCI v4.0-6.2.4

- Likelihood: High
- Impact: High
- Risk level: Critical

Issue Description:

It is a technique which relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result. Depending on the result, the content within the HTTP response will change, or remain the same. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. Even though it is a slow attack this will help the attacker to enumerate the database.

Recommendations:

- Use of prepared statements (with parameterized queries)
- Use of stored procedures
- Whitelist input validation
- Escaping all user-supplied input
- Enforcing least privilege
- Performing whitelist input validation as a secondary defence

Occurrences:

Occurrence 008 Status: New

Proof	:	dvwa information_schema
Parameter	:	id
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/sqli

Occurrence 009 Status: New

Proof	:	dvwa information_schema
Parameter	:	id
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/sqli_blind

OUT OF BAND VULNERABILITY

Blind Remote Code Execution

OWASP 2013-A1 OWASP 2017-A1 OWASP 2021-A3 PCI V3.2-6.5.1 OWASP PC-C3 CAPEC-88 CWE-78 Subpart C, HIPAA-164.306(a)(2) ISO27001-A.14.2.5 WASC-31 WSTG-INPV-12 A.10.4.1 PCI v4.0-6.2.4

- Likelihood: High
- Impact: High
- Risk level: Critical

Issue Description:

Out of Band(OOB) Remote Code Execution is performed by sending a DNS request to a server, which occurs when input data is interpreted as an operating system command. By this, an attacker can execute arbitrary commands on the system and gain unauthorized access.

Recommendations:

Remote code execution attacks can exploit various vulnerabilities, so protecting against them requires a multi-faceted approach. Here are some best practices to detect and mitigate RCE attacks:

- Sanitize inputs—attackers often exploit deserialization and injection vulnerabilities to perform RCE. Validating and sanitizing user-supplied input before allowing the application to use it will help prevent various RCE attack types.
- Manage memory securely—attackers can exploit memory management issues like buffer overflows. It is important to run regular vulnerability scans for all applications to identify buffer overflow and memory-related vulnerabilities to remediate issues before an attacker can perform RCE.
- Inspect traffic—RCE attacks involve attackers manipulating network traffic by exploiting code vulnerabilities to access a corporate system. Organizations should implement a network security solution that detects remote access and control of their systems and blocks attempted exploits of vulnerable applications.
- Control access—RCE gives attackers a foothold in the target network that they can use to expand access and execute more damaging attacks. Access controls and techniques like network segmentation, zero trust policies, and access management platforms can help prevent lateral movement, ensuring that attackers cannot escalate an attacker after gaining initial access to the target system.

Occurrences:

Occurrence 010 Status: New

Request Body	:	ip=sampletext&Submit=Submit
Parameter	:	ip
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/exec/

Local File Inclusion

OWASP 2013-A4 OWASP 2017-A5 OWASP 2021-A1 PCI V3.2-6.5.8 CAPEC-252 CWE-22 Subpart C, HIPAA-164.306(a)(2) ISO27001-A.14.2.5 WASC-33 WSTG-INPV-11 A.10.4.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: High
- Risk level: High

Issue Description:

This server allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. This is due to the use of user-supplied input without proper validation.

Recommendations:

1. **Enforce a Whitelist**

The best way to mitigate the risk of Local File Inclusion (LFI) is to create a whitelist of acceptable files and directories that can be included in the application. This will ensure that only legitimate files are included in the application and no malicious files can be included.
For example, in PHP you can use the is_file function to check if a file exists and is a regular file before including it:

```
if (is_file($_GET['file'])) {
include($_GET['file']);
}
```

2. **Sanitize User Inputs**

In order to prevent malicious inputs from being passed to the application, it is important to sanitize user inputs. This should be done using a whitelist approach, where only specific characters are allowed.
For example, in PHP you can use the preg_replace function to strip any characters that are not allowed in the user input:

```
$file = preg_replace('/[^a-zA-Z0-9_\-\.]/', '', $_GET['file']);
```

3. **Disable PHP File Uploads**

If your application allows users to upload files, it is important to disable the ability to upload PHP files. This will prevent malicious files from being uploaded and included in the application.
For example, in PHP you can use the upload_max_filesize setting to limit the maximum size of uploaded files:

```
upload_max_filesize = 0
```

4. **Disable Remote File Inclusion**

It is also important to disable the ability to include remote files in the application. This will prevent malicious remote files from being included in the application.
For example, in PHP you can use the allow_url_include setting to disable remote file inclusion:

```
allow_url_include = 0
```

5. **Disable Directory Traversal**

Finally, it is important to disable the ability to traverse directories in the application. This will prevent malicious files from being included in the application.
For example, in PHP you can use the open_basedir setting to restrict access to specific directories:

```
open_basedir = /var/www/html
```

Occurrences:

Occurrence 011 Status: New

Parameter	:	page
Payload	:	/etc/passwd
Method	:	GET
URL	:	https://beaglehack.com/vulnerabilities/fi/?page=file1.php

High Memory Limit in PHP Info Page

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 CAPEC-346 CWE-213 Subpart C, HIPAA-164.308(a)(1)(i) ISO27001-A.18.1.3 WASC-13 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: High
- Risk level: High

Issue Description:

In this sever the code-execution vulnerability may occur because the PHP module fails to properly handle memory_limit request termination. The attacker can leverage this issue by exploiting Memory Allocation Denial Of Service Vulnerability and premature termination during critical code execution.

Recommendations:

- Updating the PHP to the latest version

Occurrences:

Occurrence 012 Status: New

Findings	:	The phpinfo memory_limit is set to a high value: 128M
URL	:	https://beaglehack.com/phpinfo.php

Upload Temp Directory accessible is Everyone in PHP Info Page

OWASP 2013-A1 OWASP 2017-A1 OWASP 2021-A3 Subpart A, HIPAA-164.105 WASC-14 A.10.4.1 PCI v4.0-6.2.4

- Likelihood: High

- Impact: Medium
- Risk level: High

Issue Description:

This server has a vulnerability that the Upload tmp dir is accessible for everyone. The upload_tmp_dir allows you to specify where uploaded files should be saved until the handling script moves them to a more permanent location. If this file is within the document root of the website and accessible to system users other than PHP's user, it could be modified or overwritten while PHP is processing it. By default, upload_tmp_dir is set to the system's standard temporary directory, which can typically be accessed by all system users.

Recommendations:

- Upload tmp dir outside the document root of your web site.
- Upload tmp dir to not readable or writable by any other system users.

Occurrences:

Occurrence 013 Status: New

URL : https://beaglehack.com/phpinfo.php

X-Frame-Options header not implemented

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 OWASP PC-C1 CAPEC-103 CWE-693 Subpart C, HIPAA-164.308(a)(1)(i) ISO27001-A.14.2.5 WASC-14 WSTG-CLNT-09 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

In this webpage, X-Frame-Options header is not found. Without X Frame-Options header the browser cannot decide the page to render in <frame> or <iframe> and thus the site cannot ensure that their contents are not embedded in other sites. This vulnerability leads to many attacks like Clickjacking.

Recommendations:

Step 1:
Identify the application or website which needs to be secured.
Step 2:
Add the following code to the web page header to prevent the page from being loaded in an iFrame:

```
<meta http-equiv="X-Frame-Options" content="deny">
```

Step 3:
If the application or website is using Apache, add the following code to the .htaccess file:

```
Header always append X-Frame-Options DENY
```

Step 4:
If the application or website is using Nginx, add the following code to the server configuration file:

```
add_header X-Frame-Options DENY;
```

Step 5:
Test the changes to ensure that the page is not loaded in an iFrame.

Occurrences:

Occurrence 014 Status: New

URL : https://beaglehack.com/

Clickjacking

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 CAPEC-103 CWE-1021

Subpart C, HIPAA-164.308(a)(1)(i) WASC-15 WSTG-CLNT-09 A.10.4.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

Clickjacking is a malicious technique of tricking a user into clicking on a link, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages. On this page, the attackers will use multiple clear or opaque layers to trick a user into clicking on a button or link on another page when they were aiming to click on the top level page. This leads to leakage of sensitive information.

Recommendations:

Step 1: Understand Clickjacking Clickjacking is a type of web application vulnerability where an attacker tricks a user into clicking on a hidden or invisible element on a webpage, which can lead to unintended actions being performed by the user. This can include clicking on buttons, links, or even entering sensitive information.
Step 2: Implement X-Frame-Options Header The X-Frame-Options header is a security feature that helps prevent clickjacking attacks by limiting the ability of a webpage to be embedded in an iframe. This header can be implemented in your web server configuration or by adding the following code to the header section of your web page:
X-Frame-Options: SAMEORIGIN
This will restrict the page from being loaded in an iframe from a different origin, preventing clickjacking attacks.
Step 3: Use Content Security Policy (CSP) Content Security Policy (CSP) is an additional layer of security that helps prevent clickjacking attacks. It allows web developers to specify which resources can be loaded on a webpage, thereby preventing malicious scripts from being loaded. To implement CSP, add the following code to the header section of your web page:
Content-Security-Policy: frame-ancestors 'self'
This will restrict the page from being loaded in an iframe from any other origin, except for the same origin.
Step 4: Implement Frame-Busting Script A frame-busting script is a piece of code that can be added to your web page to prevent it from being loaded in an iframe. This can be done by adding the following code to the header section of your web page:

if (top.location != self.location) { top.location = self.location; }
This script will redirect the page to the top-level window if it is being loaded in an iframe, thereby preventing clickjacking attacks.
Step 5: Use X-Content-Type-Options Header The X-Content-Type-Options header is a security feature that helps prevent clickjacking attacks by limiting the ability of a webpage to be loaded in a different content type. To implement this header, add the following code to the header section of your web page:
X-Content-Type-Options: nosniff
This will prevent the browser from guessing the content type of the page, thereby preventing clickjacking attacks.
Step 6: Implement Frame-Killer Script A frame-killer script is a piece of code that can be added to your web page to prevent it from being loaded in an iframe. This can be done by adding the following code to the header section of your web page:
if (window.top != window.self) { window.top.location = window.self.location; }
This will redirect the page to the top-level window if it is being loaded in an iframe, thereby preventing clickjacking attacks.

Occurrences:

Occurrence 015 Status: New

URL : https://beaglehack.com/

HTML Injection

OWASP 2013-A1 OWASP 2017-A1 OWASP 2021-A3 OWASP PC-C4 CAPEC-242 CWE-80
Subpart C, HIPAA-164.306(a)(2) WASC-08 WSTG-CLNT-03 A.12.2.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

This server is vulnerable to HTML injection that occurs when an attacker is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page using metacharacters. This may lead to consequences like disclosure of a user's session cookies or it can allow the attacker to modify the page content seen by the victims.

Recommendations:

1. **Validate User Input**: User input should be validated and sanitized before being used in HTML output.
2. **Encode Output**: Output should be HTML encoded before being sent to the user. This can be done using the htmlspecialchars() method in PHP, or the HtmlEncode() method in .NET.
3. **Use Parameterized Queries**: Parameterized queries should be used when accessing the database. This will ensure that user input is not interpreted as part of the query.
4. **Disable Client-Side Scripts**: Client-side scripts such as JavaScript should be disabled in user input. This can be done by using the strip_tags() method in PHP, or the DisableScripts() method in .NET.
5. **Disable HTML Tags**: HTML tags should be disabled in user input. This can be done by using the strip_tags() method in PHP, or the DisableHtmlTags() method in .NET.

Occurrences:

Occurrence 016 Status: New

Vulnerable Parameter	:	include
Payload	:	<h1><a href="https://xssiframeloader.pug.gs">beagle_injection_attack</a></h1>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/csp/

Occurrence 017 Status: New

Vulnerable Parameter	:	txtName
Payload	:	<h1><a href="https://xssiframeloader.pug.gs">beagle_injection_attack</a></h1>

Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 018 Status: New

Vulnerable Parameter	:	mtxMessage
Payload	:	<h1><a href="https://xssiframeloader.pug.gs">beagle_injection_attack</a></h1>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 019 Status: New

Vulnerable Parameter	:	name
Payload	:	<h1><a href="https://xssiframeloader.pug.gs">beagle_injection_attack</a></h1>
Method	:	GET
URL	:	https://beaglehack.com/vulnerabilities/xss_r/?name=sampletext

Iframe Injection

OWASP 2013-A1 OWASP 2017-A1 OWASP 2021-A3 PCI V3.2-6.5.1 OWASP PC-C5
Subpart C, HIPAA-164.306(a)(2) WSTG-INPV-05 A.12.2.4 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

The IFRAME element may be a security risk if any page on your site contains an XSS vulnerability which can be exploited. An attacker may leverage this issue to execute arbitrary script code in the browser of an unsuspecting user in the context of the affected site. This can allow the attacker to steal cookie-based authentication credentials and launch other attacks.

Recommendations:

1. Validate user input: Ensure that all user input is validated for malicious content. This can be done by using whitelisting techniques, such as only allowing specific characters, and blacklisting techniques, such as blocking certain strings.
2. Sanitize user input: Sanitize user input by using a combination of server-side and client-side validation techniques. Server-side validation should be used to check for malicious content, while client-side validation should be used to ensure the data is in the correct format.
3. Use X-Frame-Options: Use the X-Frame-Options header to prevent your web page from being loaded in an iframe. This header can be set to either DENY or SAMEORIGIN.
4. Use CSP: Use Content Security Policy (CSP) to prevent iframe injection. CSP can be used to block certain types of content, such as iframes, from being loaded on your web page.
5. Monitor for suspicious activity: Monitor your web application for suspicious activity, such as unexpected requests or responses. If you detect any suspicious activity, investigate it immediately.

Occurrences:

Occurrence 020 Status: New

Vulnerable Parameter	:	txtName
Payload	:	<iframe src="https://xssiframeloader.pug.gs"></iframe>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 021 Status: New

Vulnerable Parameter	:	btnSign
Payload	:	<iframe src="https://xssiframeloader.pug.gs"></iframe>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 022 Status: New

Vulnerable Parameter	:	mtxMessage
Payload	:	<iframe src="https://xssiframeloader.pug.gs"></iframe>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/xss_s/

Occurrence 023 Status: New

Vulnerable Parameter	:	include
Payload	:	<iframe src="https://xssiframeloader.pug.gs"></iframe>
Method	:	POST
URL	:	https://beaglehack.com/vulnerabilities/csp/

Content Security Policy (CSP) header not implemented

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 CWE-16 Subpart A, HIPAA-164.105

ISO27001-A.14.2.5 WASC-15 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

Content Security Policy (CSP) is a computer security standard. It was introduced to prevent cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the trusted web page context. In this application, the Content Security Policy header is not implemented. This leads to vulnerabilities like Cross-site Scripting and related attacks. Not implementing Content Security Policy this application missing out this extra layer of security.

Recommendations:

- The mitigation for this vulnerability is to enable CSP on your website by sending the Content-Security-Policy in HTTP response headers. The header must instruct the browser to apply the policies you specified.

Occurrences:

Occurrence 024 Status: New

URL : https://beaglehack.com/

allow_url_fopen Enabled in PHP Info Page

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 CWE-16 Subpart C, HIPAA-164.308(a)(1)(i) WASC-13 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

In this server, the allow_url_fopen is enabled. The allow_url_fopen setting carries the risk of enabling Remote File Execution, Access Control Bypass or Information Disclosure attacks. If an attacker can inject a remote URI of their choosing into a file function they could manipulate an application into executing, storing or displaying the fetched file including those from any untrusted remote source

Recommendations:

- Disable allow_url_fopen from php.ini or .htaccess.

Occurrences:

Occurrence 025 Status: New

URL : https://beaglehack.com/phpinfo.php

file_uploads in on in PHP Info Page

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 CAPEC-17 CWE-434 Subpart A, HIPAA-164.105 WASC-14 WSTG-BUSL-09 A.10.4.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

In this server, the PHP file upload is on. This leads to unrestricted files to upload to the server. This vulnerability may cause attacks like denial of service.

Recommendations:

- Restricting file types accepted for upload.
- Checking the file extension.
- Using the whitelist approach.

Occurrences:

Occurrence 026 Status: New

URL : https://beaglehack.com/phpinfo.php

Weak MD5 Session Hash Algorithm

OWASP 2013-A9 OWASP 2017-A9 OWASP 2021-A6 CWE-328 Subpart C, HIPAA-164.308(a)(1)(i) WASC-11 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

This server uses the MD5 algorithm for session hash function. This algorithm is vulnerable. The attacker can easily crack these hash value using a brute-force attack.

Recommendations:

- Use Slow Password Hash such as BCrypt, PBKDF2, SCrypt etc

Occurrences:

Occurrence 027 Status: New

URL : https://beaglehack.com/phpinfo.php

Cookie set without 'Secure' flag

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 CAPEC-102 CWE-614 ISO27001-A.14.1.2 WASC-15 WSTG-SESS-02 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Low
- Risk level: Low

Issue Description:

If the secure flag is set on a cookie, then browsers will not submit the cookie in any requests that use an unencrypted HTTP connection, thereby preventing the cookie from being trivially intercepted by an attacker monitoring network traffic. If the secure flag is not set, then the cookie will be transmitted in clear-text if the user visits any HTTP URLs within the cookie's scope.

Recommendations:

- The secure flag should be set on all cookies that are used for transmitting sensitive data when accessing content over HTTPS

Occurrences:

Occurrence 028 Status: New

Cookie	:	PHPSESSID
URL	:	https://beaglehack.com/

HTTP Strict Transport Security (HSTS) header not implemented

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 OWASP PC-C1 CAPEC-217 CWE-523

Subpart C, HIPAA-164.312(e)(1) ISO27001-A.14.1.2 WASC-4 WSTG-CONF-07 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Medium
- Impact: Low
- Risk level: Low

Issue Description:

In this website, the HTTP Strict Transport Security (HSTS) policy is not implemented. This website is being served from not only HTTP but also HTTPS and it lacks HSTS policy implementation.HTTP Strict Transport Security is a web security policy mechanism to interact with complying user agents such as a web browser using only secure HTTP connections.

Recommendations:

- Configure your web server to use HSTS to redirect HTTP requests to HTTPS.

Occurrences:

Occurrence 029 Status: New

URL : https://beaglehack.com/

X-Content-Type-Options header not implemented

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 OWASP PC-C1 CWE-16

Subpart C, HIPAA-164.308(a)(1)(i) ISO27001-A.14.1.2 WASC-15 A.12.1.1 PCI v4.0-6.2.4

- Likelihood: Low
- Impact: Low
- Risk level: Info

Issue Description:

The X-Content-Type-Options response HTTP header is a marker used by the server to indicate that the Multipurpose Internet Mail Extensions types advertised in the Content-Type headers should not be changed and be followed. It is a way to say that the webmasters knew what they were doing. In this webpage, X Content Type Options is not found. This application is vulnerable to Multipurpose Internet Mail Extensions sniffing attacks. These vulnerabilities can occur when a website allows users to upload content to a website. This can give them the opportunity to perform cross-site scripting and compromise the website.

Recommendations:

Step 1: Add the X-Content-Type-Options header to your web application

The X-Content-Type-Options header is used to indicate that the browser should not interpret the response as something other than the specified content type.
For example, if you are serving HTML, you should add the following header:

```
X-Content-Type-Options: nosniff
```

Step 2: Configure your web server to add the X-Content-Type-Options header

Depending on your web server, you need to configure it to add the X-Content-Type-Options header to the response.

Apache

In Apache, you can add the X-Content-Type-Options header by adding the following to your Apache configuration:

```
Header set X-Content-Type-Options "nosniff"
```

Nginx

In Nginx, you can add the X-Content-Type-Options header by adding the following to your Nginx configuration:

```
add_header X-Content-Type-Options "nosniff";
```

Step 3: Test the configuration

Once the X-Content-Type-Options header has been added, you should test the configuration to make sure that it is working correctly.
You can use a tool like curl to test the response headers:

```
$ curl -I https://example.com

HTTP/2 200
Content-Type: text/html; charset=utf-8
X-Content-Type-Options: nosniff
[...]
```

If the X-Content-Type-Options header is present in the response, then the configuration has been successful.

Occurrences:

Occurrence 030 Status: New

URL : https://beaglehack.com/

Possible BREACH Attack

OWASP 2013-A9 OWASP 2017-A9 OWASP 2021-A6 OWASP PC-C1 CWE-310 Subpart C, HIPAA-164.312(e)(1) WASC-04 WSTG-CLNT-10 A.12.3.1 PCI v4.0-6.2.4

- Likelihood: Low
- Impact: Low
- Risk level: Info

Issue Description:

The BREACH (Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext) attack is a security vulnerability that specifically targets websites using HTTPS with compression enabled. This attack exploits the compression algorithms employed in HTTPS to extract sensitive information, such as authentication tokens or session cookies, from encrypted traffic.
For a website to be vulnerable to the BREACH attack, the following conditions must be met:

- HTTP Compression Enabled : The website must use HTTP compression.
- User Input Reflected in Response Body : The application must reflect user input in its HTTP response body.
- Exposure of Secrets in Response Body : The application must expose sensitive information (e.g., CSRF tokens) in the response body.

Recommendations:

Step 1: Disable Compression

The BREACH attack exploits the compression of HTTPS responses to extract sensitive information. To mitigate the BREACH attack, you should disable compression of HTTPS responses.
If you are using Apache, you can add the following configuration to your .htaccess file:

```
<IfModule mod_headers.c>
RequestHeader append Accept-Encoding "identity;q=1.0"
</IfModule>
```

Step 2: Use Random CSRF Tokens

The BREACH attack relies on the attacker being able to detect when the same CSRF token is used multiple times. To mitigate this, you should generate random CSRF tokens for each request.
If you are using PHP, you can generate a random CSRF token with the following code:

```
<?php
$csrf_token = bin2hex(random_bytes(32));
?>
```

Step 3: Use HTTPS Everywhere

The BREACH attack is only possible over HTTPS connections. To ensure that the attack is not possible, you should ensure that all requests are sent over HTTPS.
If you are using Apache, you can add the following configuration to your .htaccess file:

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

Occurrences:

Occurrence 031 Status: New

URL	:	https://beaglehack.com/

Cookie set without 'HttpOnly' flag

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 OWASP PC-C1 CWE-104

Subpart A, HIPAA-164.105 WASC-14 WSTG-SESS-02 A.12.3.1 PCI v4.0-6.2.4

- Likelihood: Low
- Impact: Low
- Risk level: Info

Issue Description:

If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's value via an injected script.

Recommendations:

- It is recommended to set 'HttpOnly' for all session cookies.

Occurrences:

Occurrence 032 Status: New

Cookie	:	PHPSESSID
URL	:	https://beaglehack.com

Requirement 6.4: Public-facing web applications are protected against attacks.

For public-facing web applications, an automated technical solution is deployed that continually detects and prevents web-based attacks, with at least the following:

- Is installed in front of public-facing web applications and is configured to detect and prevent web-based attacks.
- Actively running and up to date as applicable.
- Generating audit logs.
- Configured to either block web-based attacks or generate an alert that is immediately investigated.

Application Error Disclosure

OWASP 2013-A6 OWASP 2017-A3 OWASP 2021-A2 OWASP PC-C10 CWE-200

Subpart A, HIPAA-164.105 WSTG-ERRH-01 A.12.2.2 PCI v4.0-6.4.2

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

This server has a vulnerability that it displays too much information about databases, bugs, and other technological components directly linked with the web application while showing error. The attacker can monitor these errors to be displayed by particular requests, either specially crafted with tools or created manually.

Recommendations:

1. **Ensure Error Messages are Not Revealing**

It is important to ensure that the error messages returned by the application do not reveal sensitive information about the application, such as the version of the application, the database type, or the operating system. This can be done by configuring the application to return generic error messages instead of detailed error messages.
For example, in ASP.NET Core, the following configuration can be used to return generic error messages:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}
else
{
app.UseExceptionHandler("/Error");
app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>
{
endpoints.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");
});
}
```

2. **Disable Detailed Error Messages in Production**

In production environments, it is important to disable detailed error messages, as they can reveal sensitive information to attackers. This can be done by configuring the application to return generic error messages instead of detailed error messages.
For example, in ASP.NET Core, the following configuration can be used to return generic error messages:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}
else
{
app.UseExceptionHandler("/Error");
app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>
{
endpoints.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");
});
}
```

3. **Log Errors in Production**

It is important to log errors in production environments, as this will allow the development team to quickly identify and fix any issues. This can be done by configuring the application to log errors to a central logging system.
For example, in ASP.NET Core, the following configuration can be used to log errors to a central logging system:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
ILoggerFactory loggerFactory)
{
loggerFactory.AddNLog();

if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}
else
{
app.UseExceptionHandler("/Error");
app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>
{
endpoints.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");
});
}
```

Occurrences:

Occurrence 033 Status: New

Evidence	:	Parent Directory
Method	:	GET

URL : https://beaglehack.com/docs/

Restrict Access to System Components and Cardholder Data by Business Need to Know

Requirement 7.2: Access to system components and data is appropriately defined and assigned.

An access control model is defined and includes granting access as follows:

- Appropriate access depending on the entity's business and access needs.
- Access to system components and data resources that is based on users' job classification and functions.
- The least privileges required (for example, user, administrator) to perform a job function.

PHP info page disclosure

OWASP 2013-A6 OWASP 2017-A3 OWASP 2021-A2 PCI V3.2-6.5.5 CAPEC-346 CWE-213 Subpart C, HIPAA-164.306(a)(1) ISO27001-A.18.1.3 WASC-13 PCI v4.0-7.2.1

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

The "phpinfo" page displays information about the current PHP environment, such as version, configuration settings, and installed modules. This information can be useful for debugging and troubleshooting, but it can also be a security vulnerability if the page is accessible to unauthorized users. The page can reveal sensitive information about the server, such as paths to files, database credentials, and other configuration details that can be exploited by attackers. To prevent this, it is recommended to restrict access to the "phpinfo" page to only trusted users, or to remove the page altogether if it is not needed for your application.

Recommendations:

1. **Disable PHP Info Page**

Create a .htaccess file in the root of your web application and add the following code:

```
<Files "phpinfo.php">
Order Deny,Allow
Deny from all
</Files>
```

2. **Disable PHP Information Disclosure**

Edit your php.ini file and add the following line:

```
expose_php = Off
```

Occurrences:

Occurrence 034 Status: New

Finding : https://beaglehack.com/phpinfo.php

Directory Listing

OWASP 2013-A5 OWASP 2017-A6 OWASP 2021-A5 PCI V3.2-6.5.8 CWE-284 Subpart C, HIPAA-164.312(a)(1) ISO27001-A.18.1.3 WSTG-CONF-09 PCI v4.0-7.2.1

- Likelihood: Medium
- Impact: Medium
- Risk level: Medium

Issue Description:

The web server is configured to display the list of files contained in this directory. This is not recommended because the directory may contain files that are not normally exposed through links on the web site.

Recommendations:

You should make sure the directory does not contain sensitive information or you may want to restrict directory listings from the web server configuration.

Occurrences:

Occurrence 035 Status: New

URL : https://beaglehack.com/vulnerabilities/javascript/source/

Occurrence 036 Status: New

URL : https://beaglehack.com/vulnerabilities/

Occurrence 037 Status: New

URL : https://beaglehack.com/dvwa/

Occurrence 038 Status: New

URL : https://beaglehack.com/docs/

Occurrence 039 Status: New

URL : https://beaglehack.com/dvwa/js/

Occurrence 040 Status: New

URL	:	https://beaglehack.com/dvwa/css/

Occurrence 041 Status: New

URL	:	https://beaglehack.com/dvwa/images/

PCIDSS Compliance Passed Controls

Requirement 1.2: Network security controls (NSCs) are configured and maintained.

Requirement 1.4: Network connections between trusted and untrusted networks are controlled.

Requirement 2.2: System components are configured and managed securely.

Requirement 3.2: Storage of account data is kept to a minimum.

Requirement 4.2: PAN is protected with strong cryptography during transmission.

Requirement 6.3: Security vulnerabilities are identified and addressed.

Requirement 6.5: Changes to all system components are managed securely.

Requirement 8.2: User identification and related accounts for users and administrators are strictly managed throughout an account's lifecycle.

Requirement 8.3: Strong authentication for users and administrators is established and managed.

Requirement 8.6: Use of application and system accounts and associated authentication factors is strictly managed.

Requirement 11.6: Unauthorized changes on payment pages are detected and responded to.

Conclusion

This application complies with 11 out of 14 requirements in the PCI standards.

This PCI DSS compliance report is prepared following the penetration test on test App 1 and the result will only cover as far as the “black-box,” or “grey-box” testing methods allow.

Visit the official website of the PCI Security Council for a complete copy of the PCI-DSS: https://www.pcisecuritystandards.org/

The sub-requirements associated with the PCI DSS guidelines on the application security standards determine the passing or failure of tests mentioned in this document. All other sub-requirements that do not belong to the criteria are not considered in this report.