

# APPLICATION PENETRATION TESTING REPORT

**Cyber Test**

For:

**AppSec Labs**



Performed by:  
AppSec Labs

2024-10-22



## TABLE OF CONTENTS:

|   |           |
|---|-----------|
| <b>TABLE OF CONTENTS: .....</b>   | <b>2</b>  |
| <b>CHAPTER A - INTRODUCTION TO APPSEC LABS .....</b>                    | <b>3</b>  |
| <b>CHAPTER B – EXECUTIVE SUMMARY .....</b>                              | <b>4</b>  |
| GENERAL DETAILS   | 4         |
| TEST SCOPE AND DETAILS  | 4         |
| LIMITATIONS AND DISCLAIMERS   | 4         |
| CONCLUSIONS   | 5         |
| GRAPHS OF FINDINGS  | 6         |
| <b>CHAPTER C – TESTING METHODOLOGY .....</b>                            | <b>7</b>  |
| RECOMMENDED STEPS   | 8         |
| CAUTIONARY NOTE   | 8         |
| THREAT LEVEL OF THE VULNERABILITIES                                     | 9         |
| <b>CHAPTER D – SUMMARY OF VULNERABILITIES .....</b>                     | <b>10</b> |
| SUMMARY OF FINDINGS   | 10        |
| <b>CHAPTER E – SECURITY VULNERABILITIES EXPOSED DURING THE PT .....</b> | <b>11</b> |
| <b>1. SQL INJECTION</b>   | 12        |
| <b>2. STORAGE OF SENSITIVE LOGIN CREDENTIALS</b>                        | 16        |
| <b>3. DEPRECATED HTTP SECURITY HEADERS</b>                              | 19        |
| <b>APPENDIX A - LIST OF ATTACKS AND TESTS .....</b>                     | <b>22</b> |

## CHAPTER A - INTRODUCTION TO APPSEC LABS

**AppSec Labs** is an expert application security company, whose mission is a proactive attitude towards application security.

**AppSec Labs** services are led by **Erez Metula** MSC, a world-renowned application security expert lecturing regularly at major international security conferences and the author of the book *Managed Code Rootkits*.

Our experience has been gathered over many years of servicing hundreds of clients of all sizes around the globe.

AppSec Labs' main resource is its high level of expertise in identifying security vulnerabilities, implementing secure development practices in complex applications, and the in-depth knowledge needed for integrating security at the code level.

### Our Services:

- **Application Security Testing**– application penetration testing for web, desktop, cloud, mobile and IoT applications
- **Training** – the **AppSec Labs Academy** offers **hands-on** courses in application hacking and secure coding
- **SDLC Consulting** – secure development lifecycle implementation
- **R&D** – security consulting throughout the R&D and design stages

## CHAPTER B – EXECUTIVE SUMMARY

### General details

AppSec Labs was requested by AppSec Labs to perform an application security test for the Cyber Test system/service. AppSec Labs hereby confirms that the tests have been completed and the results were delivered to AppSec Labs.

The following document summarizes the results of this test.

### Test scope and details

The following components were covered and included in the testing scope:

- www.testappseclabs.test/home (Example)
- www.testappseclabs.test/dashboard (Example)
  
- Environment tested: QA
- Application Version Number: 1.0.1

Users:

- user1
- user2
- admin

Additional Scope info:

None.

### Limitations and Disclaimers

**The following aspects were out of scope:**

No specific components were out of scope.

**System stability level:**

The system was stable for testing.

**Infrastructure testing:**

Infrastructure was in scope for the current test.

The goal of the penetration test is to provide a list of issues that jeopardize the security of the system. The report does not necessarily cover all instances of each vulnerability and therefore the suggested mitigations should be implemented throughout the entire application, and not only for the provided examples.

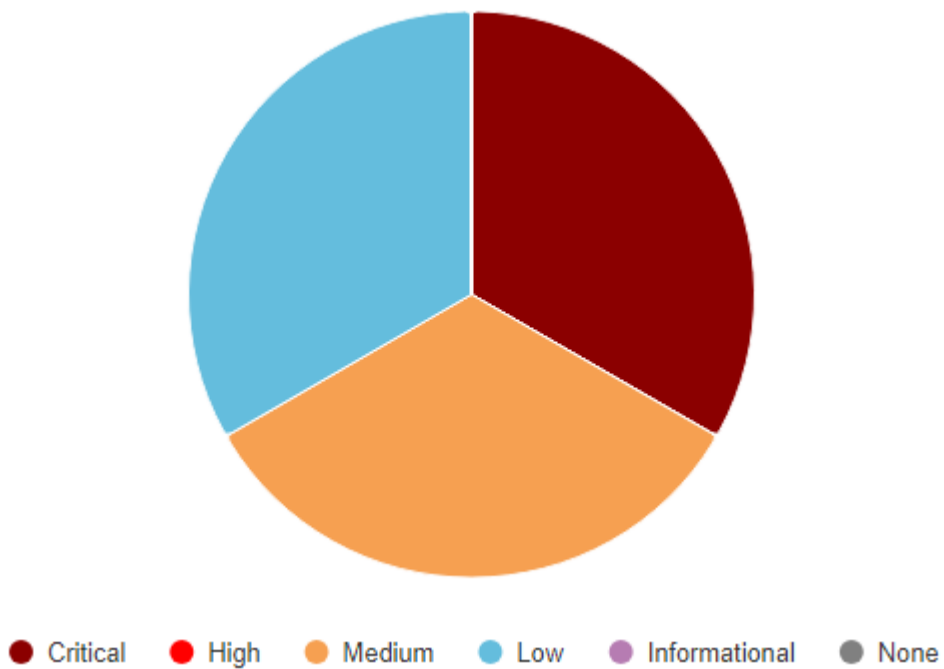
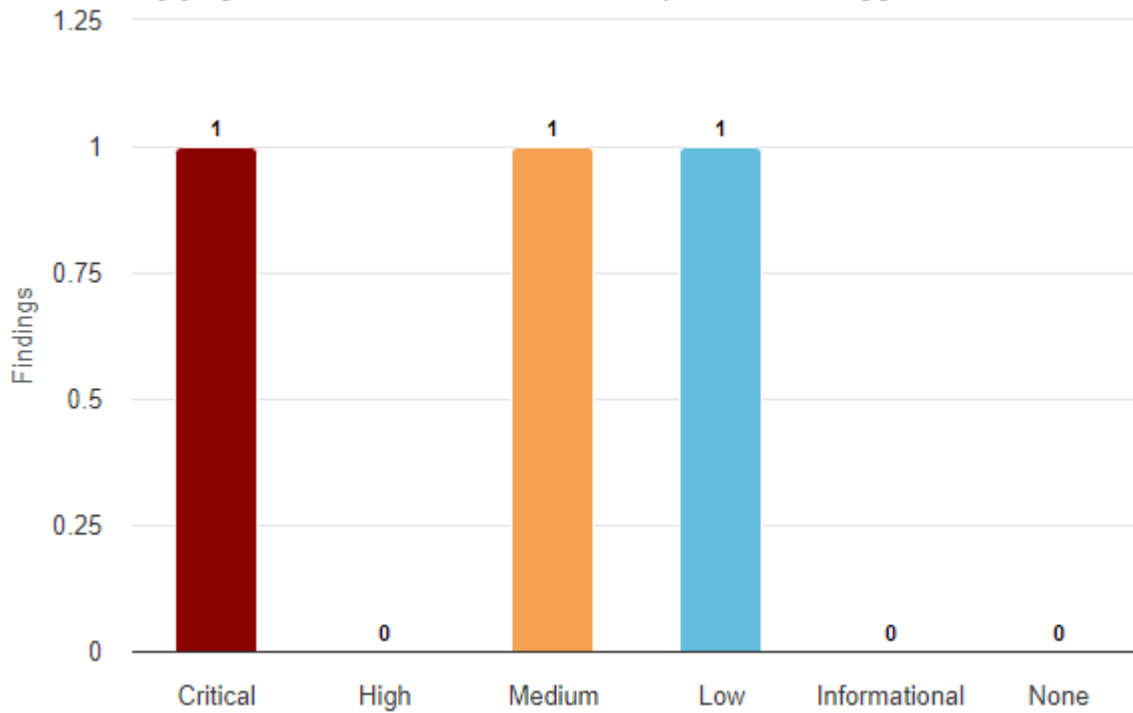
In addition, during a retest, the scope is limited to findings that were detected in the previous full round of tests and have been reported to be fixed since then by the client. The test scenarios only include exploitation of the attack scenarios as described in the original testing cycle, and in some cases basic attempts to bypass the fix. In addition, the test verified that the fix did not encounter any new security issues. However, the retest does not include any efforts to find new security findings.

## **Conclusions**

Based on the results of testing and verification process, and in accordance with the testing scope, details and limitations as stated in this document, AppSec Labs confirms that the system maximum risk of the findings in this report is: Critical

## Graphs of findings

The following graphs illustrate the current security state of the application:



## CHAPTER C – TESTING METHODOLOGY

The test was performed using a combination of automated and manual tools, in order to cover a wide range of applicative vulnerabilities as recommended by the OWASP and WASC methodologies.

A white box approach was used during the tests. This tests a system with full knowledge and access to system resources, including meetings and interviews with system architects and access to the source code and file system.

The test included access to the following resources:

- User credentials
- Admin access
- Interviews with developers

## Recommended steps

In order to improve the overall security state of the product it is recommended to take the following actions:

1. Fix vulnerabilities according to mitigation recommendations.
2. Patch management: install new patches and keep the system updated with latest releases (servers, databases, external libraries, etc.)
3. Penetration test retest – perform another cycle of testing to check whether the fixes were applied properly and with no security-related side effects.
4. Code review – a full analysis of the code to detect security vulnerabilities
5. Training (secure coding for QA/architects/developers, security awareness)

## Cautionary note

*The penetration testing that AppSec Labs performed was based on past experiences, currently available information, and known threats as of the date of testing. Given the constantly evolving nature of information security threats and vulnerabilities, there can be no assurance that any assessment will identify all possible vulnerabilities, or provide exhaustive and operationally viable recommendations to mitigate those exposures.*

*The statements relevant to the security of the system in this report reflect the conditions found at the completion of testing. In accepting our report, the recipient has acknowledged the validity of the above cautionary statement.*



## Threat level of the vulnerabilities

The severity of the vulnerabilities detected during the test was determined using OWASP and WASC methodologies. The following describes the impact of each threat level:

### **Critical**

- ☐ A security vulnerability that poses a major security risk with a direct exploit (not requiring user involvement). If exploited, the security threat might cause major damage to the application and/or have major impact on the company. The likelihood of such an attack occurring is high, considering the architecture/business-logic/complexity of the exploit.

### **High**

- ☐ The weakness identified has the potential to directly compromise the confidentiality, integrity and/or availability of the system or data, but the likelihood of its occurrence is not high, considering the architecture/business-logic/complexity of the exploit. The possible damage to the application or the company is high, but not a total disaster.
- ☐ In applications involving sensitive data, the risk might be considered high if the weakness by itself is against common regulations (e.g. PCI).

### **Medium**

- ☐ A medium security issue that imposes some affect/damage to the application. Often it cannot be used directly but can assist an attacker to launch further attacks.

### **Low**

- ☐ No direct threat exists. It is a risk, rather than a threat and does not cause damage by itself. The vulnerability may be leveraged together with other vulnerabilities in order to launch further attacks.
- ☐ The risk reveals technical information which might assist an attacker in launching or more accurately targeting future attacks.

### **Informational**

- ☐ This is a vulnerability which is either not currently exploitable or it currently has no actual impact.
- ☐ For example, the vulnerability cannot be exploited because of some other unrelated element or design feature of application that, if it was changed, would suddenly make the vulnerability exploitable.
- ☐ Alternatively, this element of the application may not currently be considered to be security sensitive but this may change in the near future.
- ☐ As such, this finding is being included to raise awareness of this possibility. The finding description should include the relevant circumstances.

## CHAPTER D – SUMMARY OF VULNERABILITIES

### Summary of findings

The system has been found to be vulnerable to the attacks detailed below, as at the date of the test and taking into account the test conditions and environment.

| # | Threat level    | Finding description                                    | Status             |
|---|-----------------|--|--------------------|
| 1 | <b>Critical</b> | <a href="#">SQL Injection</a>                          | Not Fixed          |
| 2 | <b>Medium</b>   | <a href="#">Storage of Sensitive Login Credentials</a> | Waiting for Retest |
| 3 | <b>Low</b>      | <a href="#">Deprecated HTTP Security Headers</a>       | Open Finding       |

## CHAPTER E – SECURITY VULNERABILITIES EXPOSED DURING THE PT

In the upcoming section of this report, we delve into the pivotal findings gathered during the penetration testing process. Each identified vulnerability will be thoroughly documented, encompassing its severity level, potential repercussions on the organization, and any associated data. Additionally, corroborating evidence of each vulnerability, such as screen captures, logs, or exploit code, will be provided to substantiate the associated risks. Where relevant, the methods employed for exploiting and the root causes of each vulnerability will also be detailed. This exhaustive analysis is aimed at delivering an overview of the system existing security stance, pinpointing areas necessitating urgent intervention, and proposing suitable remediation strategies.

Moreover, each finding will also include a risk assessment, evaluating the potential business and operational impacts if the vulnerability were to be exploited. This includes potential financial losses, damage to the organization's reputation, legal implications, and effects on business continuity. Also, we will categorize each vulnerability to provide a standardized assessment of its severity. Following the detailed presentation of each finding, we will offer strategic recommendations tailored to mitigate the identified risks and improve the target's defense mechanisms. Our ultimate objective is to equip the relevant team members with the necessary insights and tools to fortify its security posture, safeguarding its critical assets from potential cyber threats.

## 1. SQL Injection

### Threat level

Critical

### Business Impact

An attacker can manipulate the database, for example: extract sensitive information from the database, bypass authentication, and more.

### Description

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.

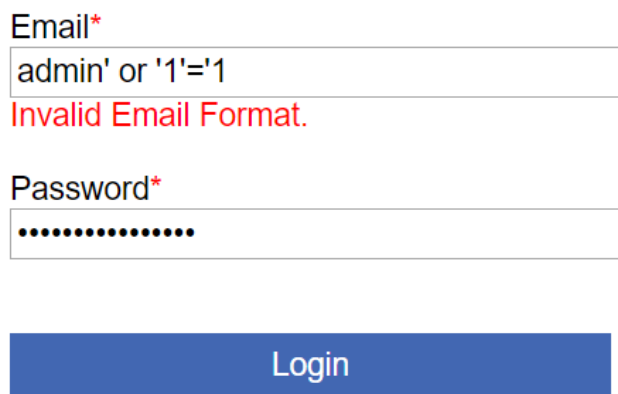
A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines.

## Proof of concept - SQL Injection

During the test, it was noted that the application is vulnerable to SQL injection.

The following screenshots provide details regarding the SQL injection vulnerability that was found in the system:

An attacker sends the following request with SQL injection payload:



Email\*

Invalid Email Format.

Password\*

Login

By utilizing the specific parameter vulnerable to SQL injection, the attacker can use automatic tools and steal the entire database.

The following image shows the impact of the SQL injection on the application:

```
Database: exercises
Table: users
[4 entries]
```

| id | groupid | age | name  | passwd  |
|----|---------|-----|-------|---------|
| 1  | 10      | 10  | admin | admin   |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret  |
| 5  | 5       | 2   | user2 | azerty  |

The above image shows that the attacker can leverage the sql injection vulnerability.

### Recommended mitigation

There may be other locations and parameters that are vulnerable to SQL Injection. As such, the mitigation should be implemented throughout the entire application and not only for the given examples.

- Use a parameterized query: using a parameterized query will assist the programmer in determining an SQL query and then pass the parameters of this query in runtime. This method allows the database engine to decide which part is the query and which parts are the parameters sent by the client:

```
String query = "SELECT * FROM users WHERE email = ?";
```

```
OleDbCommand cmd = new OleDbCommand(query, connection);
```

```
cmd.Parameters.Add("@email_address", SqlDbType.VarChar, 25);
```

```
cmd.Parameters["@email_address"].Value = Email.Text; OleDbDataReader reader =
```

```
command.ExecuteReader();
```

- In addition, it is also recommended to use stored procedures to store the SQL queries as procedures in the database. The SQL code for a stored procedure is defined and stored in the database itself and then called from the application.

## Status

Not Fixed

During the retest, it was found that the same finding exists in the system.

## 2. Storage of Sensitive Login Credentials

### Threat level

Medium

### Business Impact

An attacker might gain access to the client authentication credentials.

### Description

The application stores the client's login credentials. These credentials might get extracted easily from the device by an attacker. Compromising login credentials can endanger the account security since the attacker using the login credentials can impersonate a victim's identity and perform actions on his behalf without the victim knowing.



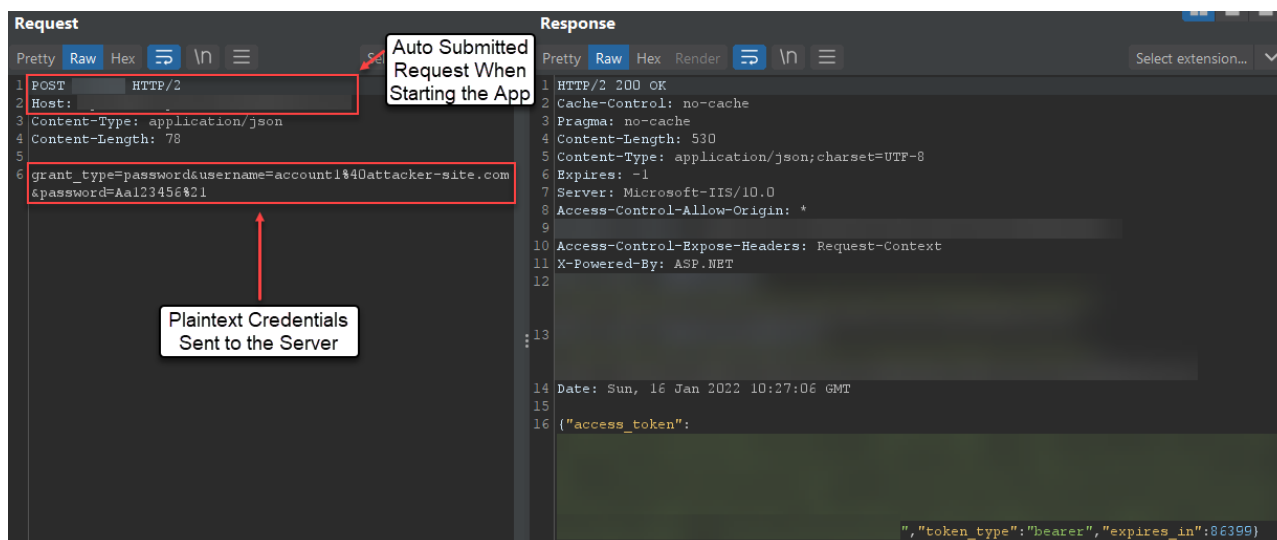
## Proof of concept - Stored Encrypted Credentials

The following image shows the stored encrypted login credentials:

```
generic_x86_arm:/data/data/ /files/.config/.isolated-storage # ls -la
total 12
drwx----- 2 u0_a90 u0_a90 4096 2022-01-16 11:44 .
drwx----- 3 u0_a90 u0_a90 4096 2022-01-16 11:44 ..
-rw----- 1 u0_a90 u0_a90 910 1970-01-20 00:12 PropertyStore.forms
```

Encrypted Credentials Saved Here

The following image shows the decrypted login credentials:



As can be seen above, encrypted login credentials are stored on the client-side and can be decrypted.

## Recommended mitigation

- It is recommended to avoid storing user credentials. Login credentials should be used once to retrieve an access token. Upon receiving the access token, the credentials must be removed from the memory.
- It is recommended to set up PIN/Biometric access control for Keychain/Keystore stored credentials will limit the attack and lower the overall finding's severity. As such, for each application access to the secure item, The user will be prompted to authenticate (PIN or biometrics). After successful authentication, the key will be accessible to the application.
- A Refresh API should be used to keep the session alive if necessary.

## Status

Waiting for Retest

Customer fixed the finding and demand for retest.

### 3. Deprecated HTTP Security Headers

#### Threat level

Low

#### Business Impact

Usage of deprecated X-XSS-Protection header reduces protection against cross-site scripting (XSS) attacks, potentially exposing the web application to increased risks of malicious code injection.

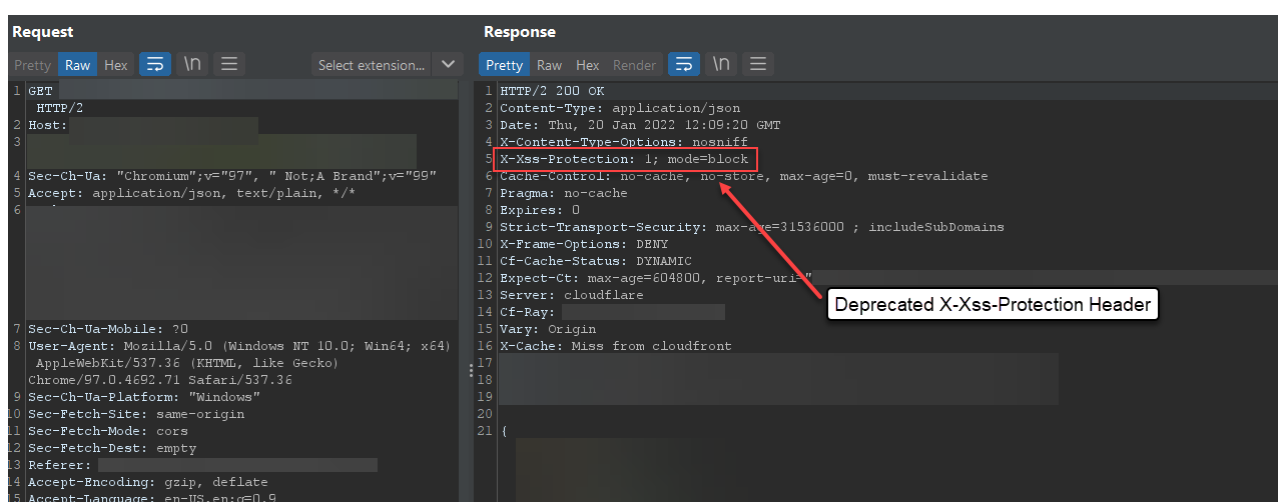
#### Description

Deprecated HTTP security headers are directives that were once used to improve a website's or web application's security, but have since been replaced by more effective or efficient alternatives. These headers may still be in use on some websites, but they are no longer considered best practices and should be updated to more current security measures. To maintain robust security, it is crucial for developers and website administrators to replace the deprecated X-XSS-Protection header with more modern and effective security measures. This includes employing strict input validation, output encoding, and utilizing "Content Security Policy" (CSP) to mitigate the risks associated with XSS attacks and ensure the overall integrity of their web applications. Leaving deprecated security headers in place can create vulnerabilities in a website or web application, as they may not provide the same level of protection as more current measures. It is important to regularly review and update security headers to ensure that a website or web application is protected against potential threats.

## Proof of concept - X-XSS-Protection

Modern browsers no longer use XSS filtering, and usage of the deprecated header can introduce additional security issues on the client-side.

As can be seen from this image below, the X-XSS-Protection security header is set and returned from the server:



```
Request
1 GET
2 HTTP/2
3 Host:
4 Sec-Ch-Ua: "Chromium";v="97", " Not;A Brand";v="99"
5 Accept: application/json, text/plain, */*
6
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/97.0.4692.71 Safari/537.36
9 Sec-Ch-Ua-Platform: "Windows"
10 Sec-Patch-Site: same-origin
11 Sec-Patch-Mode: cors
12 Sec-Patch-Dest: empty
13 Referer:
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9

Response
1 HTTP/2 200 OK
2 Content-Type: application/json
3 Date: Thu, 20 Jan 2022 12:09:20 GMT
4 X-Content-Type-Options: nosniff
5 X-Xss-Protection: 1; mode=block
6 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
7 Pragma: no-cache
8 Expires: 0
9 Strict-Transport-Security: max-age=31536000 ; includeSubDomains
10 X-Frame-Options: DENY
11 Cf-Cache-Status: DYNAMIC
12 Expect-Ct: max-age=604800, report-uri="
13 Server: cloudflare
14 Cf-Ray:
15 Vary: Origin
16 X-Cache: Miss from cloudfront
17
18
19
20
21 {
```

Deprecated X-Xss-Protection Header

## Recommended mitigation

**Note:** The mitigation should be implemented throughout the entire application and not only for the given examples.

- It is recommended to configure the following security header: X-XSS-Protection: 0
- For better protection against XSS (Cross-Site Scripting) it's recommended to implement the following header: Content-Security-Policy
- For more information in regards to the Content-Security-Policy header and proper configuration: <https://owasp.org/www-project-secure-headers/#content-security-policy>
- For more information in regards to implementing security headers: <https://owasp.org/www-project-secure-headers/>

## APPENDIX A - LIST OF ATTACKS AND TESTS

The following table is a generic list of tests performed by AppSec Labs during the security test (and **NOT** a list of vulnerabilities detected in the system). The list includes known attacks valid at the time of the production of this report.

In addition to this list, personalized testing is performed on the system, to check the application's business logic and to counter new attacks according to the most recent research carried out internally by AppSec Labs as well as those published by other, well known security companies.

The system is vulnerable to the findings that were described in the report. Subject to the nature of the test (methodology, scope, limitations, etc.), other vulnerabilities may exist but were not discovered under these conditions.

| Category                                    | Test Name  |
|---|--|
| Information Gathering                       | Search engine discovery / reconnaissance   |
|   | Web application fingerprint  |
|   | Review Webpage Comments and Metadata for Information Leakage                                 |
|   | Application entry points Identification  |
|   | Execution paths mapping  |
|   | Web application framework fingerprinting   |
|   | Web application fingerprinting   |
|   | Application architecture mapping   |
|   | Information Disclosure by error codes  |
|   | SSL Weakness - SSL/TLS Testing (SSL Version, Algorithms, Key length, Digital Cert. Validity) |
| Configuration and Deploy Management Testing | Application Configuration management weakness  |
|   | File extensions handling - sensitive information   |
|   | Old, Backup and Unreferenced Files - Sensitive Information                                   |
|   | Unauthorized Admin Interfaces access   |
|   | HTTP Methods enabled, XST permitted, HTTP Verb   |
|   | Http strict transport security   |
|   | RIA cross domain policy  |
|   | Role definitions enumeration   |
| Vulnerable user registration process        |  |

|                            |   |
|----------------------------|---|
| Authentication Testing     | Vulnerable account provisioning process                             |
|                            | Permissions of Guest/Low Permission Accounts                        |
|                            | Account suspension/resumption process                               |
|                            | Credentials Transported over Unencrypted Channel                    |
|                            | User enumeration  |
|                            | Account lockout   |
|                            | Authentication bypass   |
|                            | "Remember password" functionality                                   |
|                            | Browser caching   |
|                            | Weak password policy  |
|                            | Weak password security mechanisms                                   |
|                            | Weak password change or reset flow                                  |
|                            | Race conditions   |
|                            | Weak multiple factors authentication                                |
|                            | Weak CAPTCHA implementation   |
| Authorization Testing      | Weaker authentication in alternative channel                        |
|                            | Directory traversal/file inclusion                                  |
|                            | Authorization schema bypass   |
|                            | Privilege escalation  |
|                            | Insecure direct object references                                   |
| Session Management Testing | Session management bypass   |
|                            | Cookies are set without 'HTTP Only', 'Secure', and no time validity |
|                            | Session fixation  |
|                            | Exposed session variables   |
|                            | Cross site request forgery (CSRF)                                   |
|                            | Logout management   |
|                            | Session timeout   |
|                            | Session puzzling  |
| Data Validation Testing    | Reflected cross site scripting                                      |
|                            | Stored cross site scripting   |
|                            | HTTP verb tampering   |

|                        |   |
|------------------------|---|
|                        | HTTP Parameter pollution / manipulation                       |
|                        | SQL injection   |
|                        | LDAP injection  |
|                        | ORM injection   |
|                        | XML injection   |
|                        | SSI injection   |
|                        | Xpath Injection   |
|                        | IMAP/SMTP injection   |
|                        | Code injection  |
|                        | Local/remote file inclusion                                   |
|                        | Command injection   |
|                        | Buffer overflow   |
|                        | Heap overflow   |
|                        | Stack overflow  |
|                        | Format string manipulation                                    |
|                        | Incubated vulnerabilities                                     |
|                        | HTTP splitting/smuggling                                      |
| Error Handling         | Analysis of Error Codes                                       |
|                        | Analysis of Stack Traces                                      |
| Cryptography           | Weak SSL/TLS ciphers, insufficient transport layer protection |
|                        | Padding oracle  |
|                        | Sensitive information sent via unencrypted channels           |
| Business Logic Testing | Business logic data validation                                |
|                        | Ability to Forge Requests                                     |
|                        | Integrity checks  |
|                        | Process timing  |
|                        | Replay attack   |
|                        | Circumvention of Work Flows                                   |
|                        | Abuse of Functionality  |
|                        | File upload vulnerabilities                                   |
| Client Side Testing    | DOM based Cross Site Scripting                                |
|                        | Javascript Execution  |



|                           |   |
|---------------------------|---|
| AJAX Testing              | Html/css injection                                    |
|                           | Client side url redirect                              |
|                           | Client side resource manipulation                     |
|                           | Cross origin resource sharing                         |
|                           | Cross site flashing                                   |
|                           | Clickjacking / UI rendering                           |
|                           | Web sockets   |
|                           | Web messaging   |
|                           | Local storage / session storage sensitive information |
|                           | AJAX weakness   |
| Denial of Service Testing | SQL Wildcard vulnerability                            |
|                           | Locking customer accounts                             |
|                           | Buffer overflows                                      |
|                           | User specified object allocation                      |
|                           | User Input as a Loop Counter                          |
|                           | Writing User Provided Data to Disk                    |
|                           | Failure to Release Resources                          |
| Web Services Testing      | Storing too Much Data in Session                      |
|                           | WS information gathering                              |
|                           | WSDL weakness   |
|                           | Weak xml structure                                    |
|                           | XML content-level                                     |
|                           | WS HTTP GET parameters/REST                           |
|                           | WS Naughty SOAP attachments                           |
|                           | WS replay testing                                     |