Security Audit Report

For

Prosper (Web Applications)

Date of Creation	13-September-2025

Contents

1 Executive Summary

1.1 Objective

The primary objective of this assessment was to identify security vulnerabilities, pinpoint business logic errors, and address lapses in best security practices within the Prosper Web Applications. Simulating the perspective of an attacker or malicious user, comprehensive tests were conducted without causing any disruption to the application or network functionality. This document encapsulates the outcomes of a thorough security audit, featuring scan results that encompass discovered vulnerabilities, their severity and detailed insights into each vulnerability. The report covers issue definition, impact analysis, recommendations for resolution, and a detailed Proof of Concept (PoC). The intention is to empower stakeholders with actionable insights to enhance the security posture of the application.

1.2 Scope Definition

Applications in Scope:
User Portal: https://uat-audit.letsprosper.ae/
Agency Portal: https://agency.uat-audit.letsprosper.ae/
Admin Portal: https://admin.uat-audit.letsprosper.ae/

All the operational functionalities associated with the mentioned applications are exclusively considered in scope.

1.3 Vulnerabilities Identified

#	Vulnerability	Severity
1	Authentication Bypass via Forgot Password Functionality	High
2	Stored Cross-Site Scripting via Malicious File Upload	High
3	Unrestricted File Upload	High
4	Stored XSS via SVG upload	High
5	Stored XSS via input field	High
6	Insecure Direct Object Reference (IDOR)	High
7	Broken Access Control – Unrestricted Direct File Access	High
8	Improper Authorization – Auth Token Misuse Across Roles	High
9	HTML Injection in Input Fields	Medium
10	Insecure Direct Object Manipulation – Missing OTP Verification for Email Update	Medium

11	Using Components with Known Vulnerabilities	Medium
12	OTP Flooding	Medium
13	Email Flooding	Medium
14	Account Lockout Not Implemented	Medium
15	Password Reset Link Reusable After First Use	Medium
16	Insecure Design – Replayable Publish Property Request	Medium
17	Email / Phone Number Enumeration	Medium
18	Clickjacking Attack	Medium
19	Insecure Session Management	Medium
20	Absence of Web Application Firewall	Medium
21	Sensitive Information Exposure via Referrer Header	Low
22	Autocomplete Enabled for Password Fields	Low
23	Exposure of phpMyAdmin Interface and Documentation	Low
24	Server Banner Information Disclosure	Low
25	Cleartext Submission of Password	Low
26	Improper Error Handling	Informational
27	Session and XSRF Token Regeneration on Every Request	Informational

Graphical Summary:

This section presents a visual overview of identified vulnerabilities, utilizing a two-column table showcasing the severity levels and corresponding counts.

Vulnerability Severity	No. of Vulnerabilities
High	8
Medium	12
Low	5
Informational	2

Distribution of Identified Vulnerabilities Based on Severity Levels

2 Vulnerability Description

2.1 Authentication Bypass via Forgot Password Functionality

Severity: High

Issue Related to: User Portal

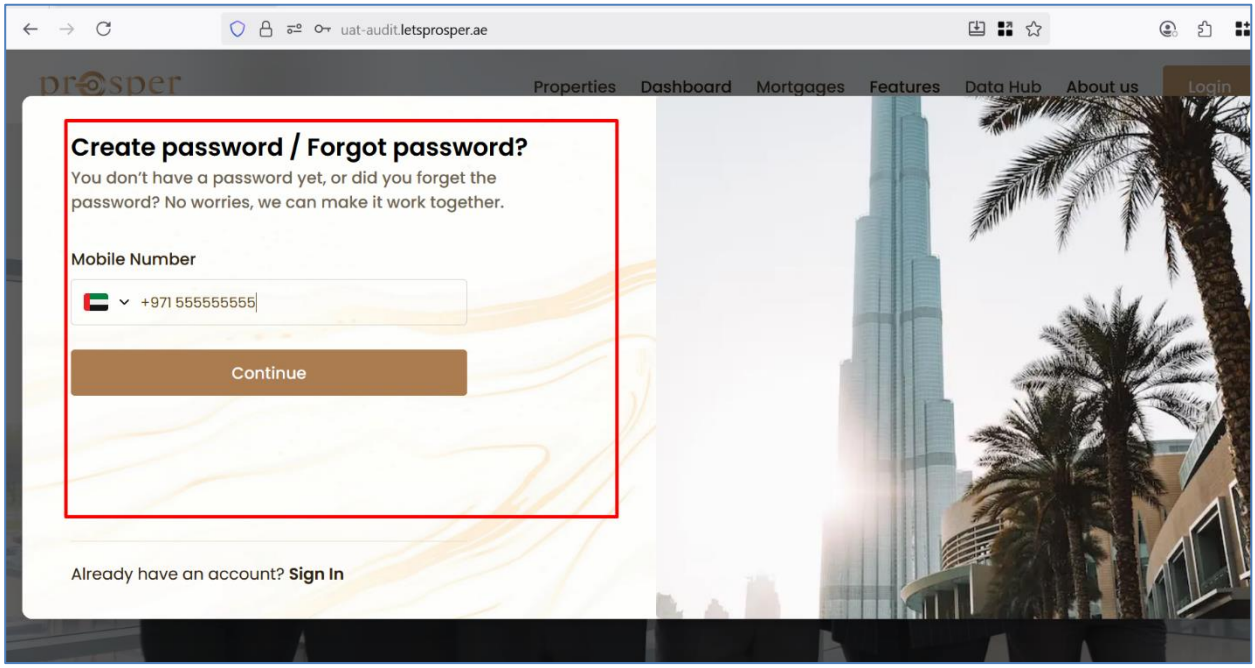
Issue Definition:
In the User Portal, the "Forgot Password" functionality allows attackers to reset passwords by manipulating the server response. When an incorrect OTP is submitted, the server returns a 422 Unprocessable Entity error. Changing this response to a 200 OK allows access to the new password creation page, enabling password resets for any user.

Analysis: This vulnerability allows attackers to hijack user accounts by resetting passwords without proper authorization. It completely undermines the security of the password reset process, posing a severe risk to user accounts. An attacker can gain unauthorized access to user data and perform malicious activities on behalf of the user.

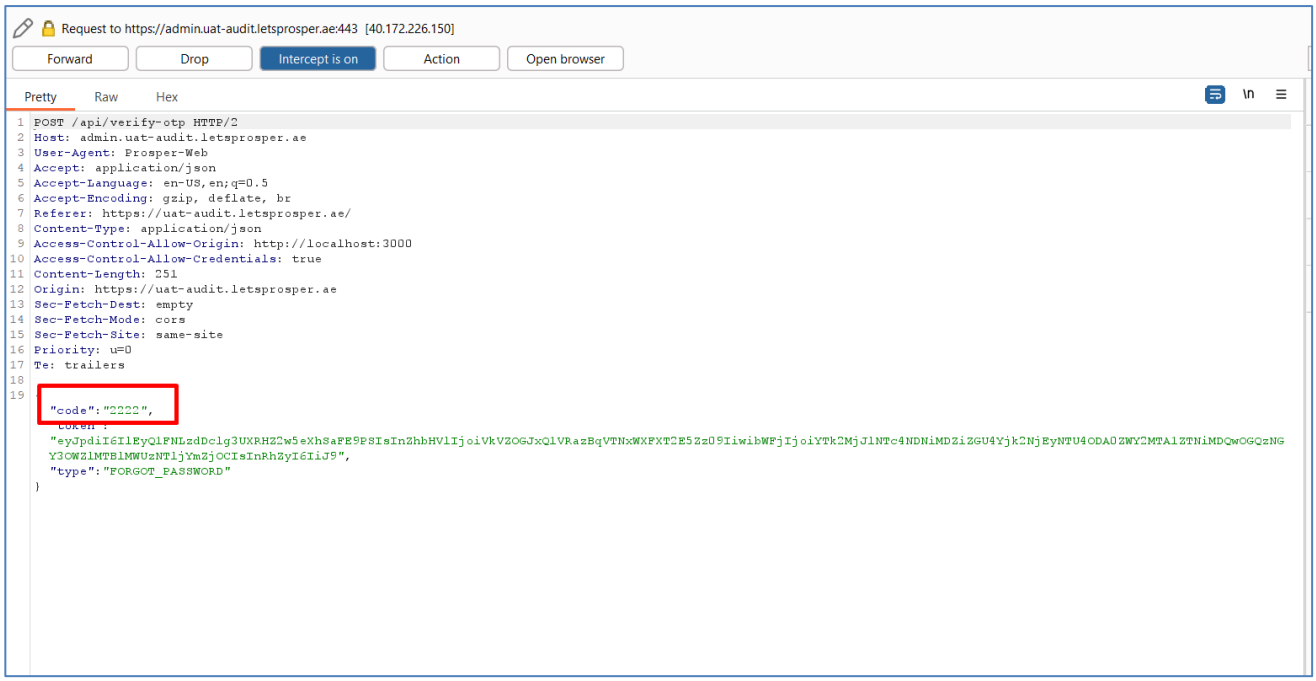
Remediation: It is recommended to ensure that all authentication and validation processes are handled on the server side. Client-side code should not be used for authentication purposes. It is also recommended to implement proper defense mechanisms to protect against response manipulation attacks.

Proof of Concept:

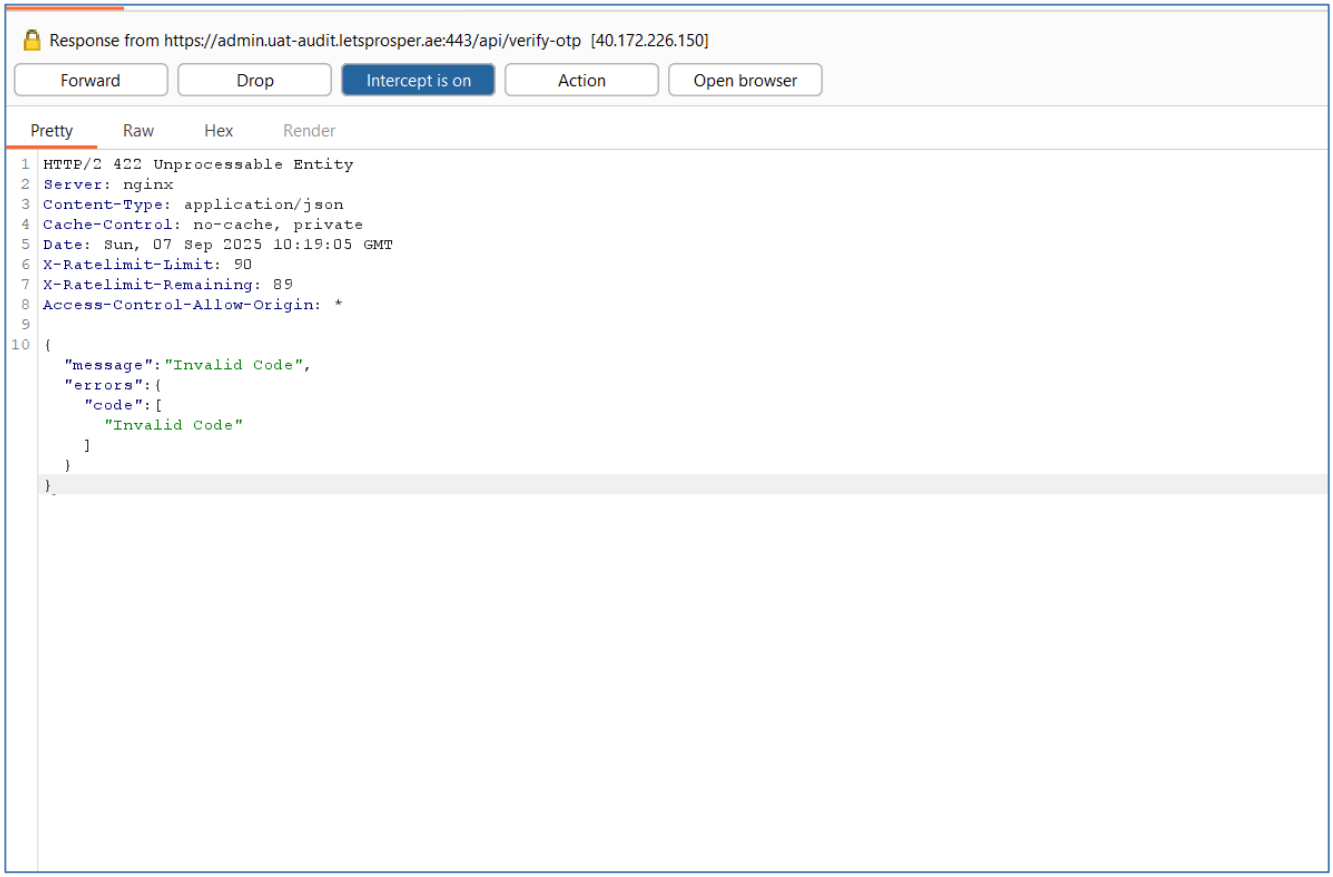
1. Enter a number in the mobile number field of the forgot password functioanlity.



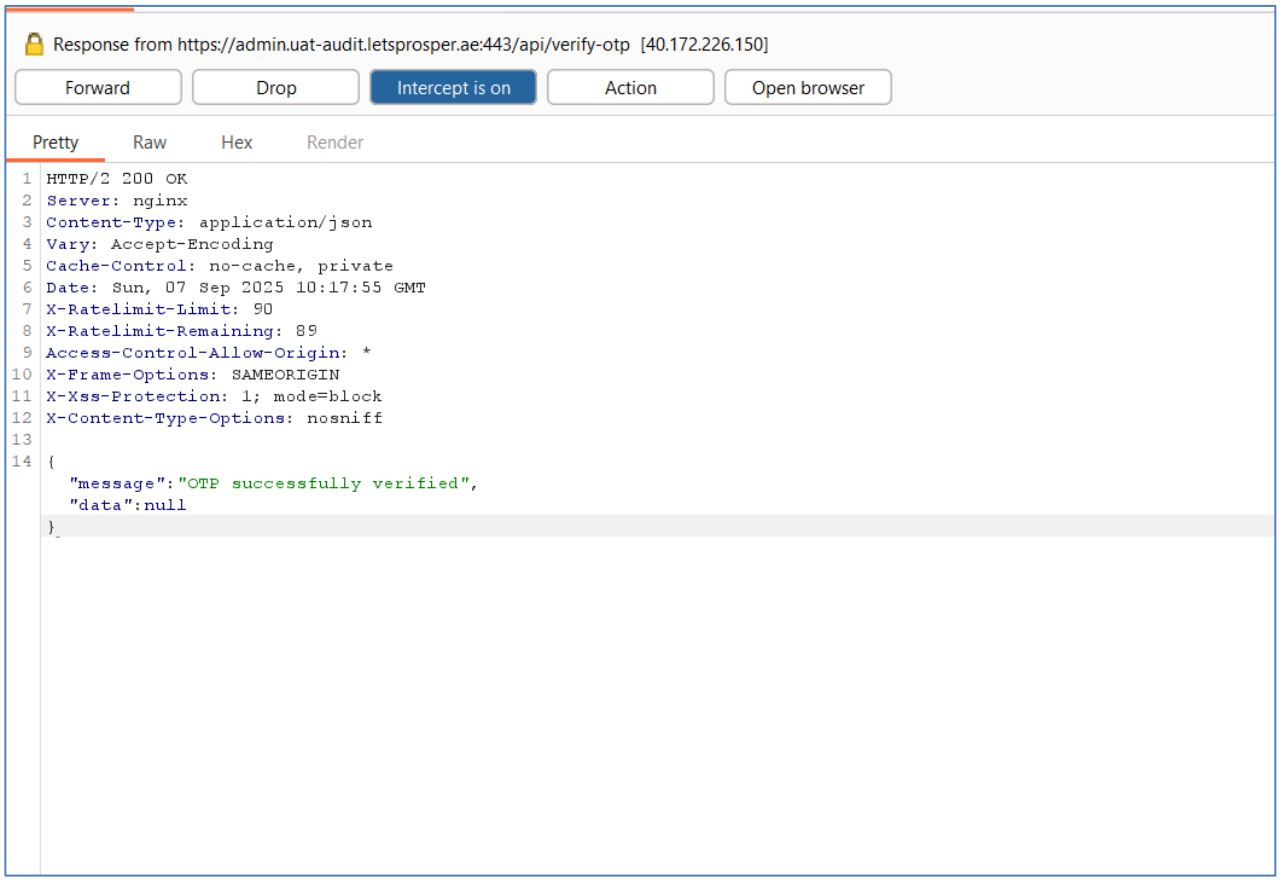
2. Try an incorrect OTP and capture the request.



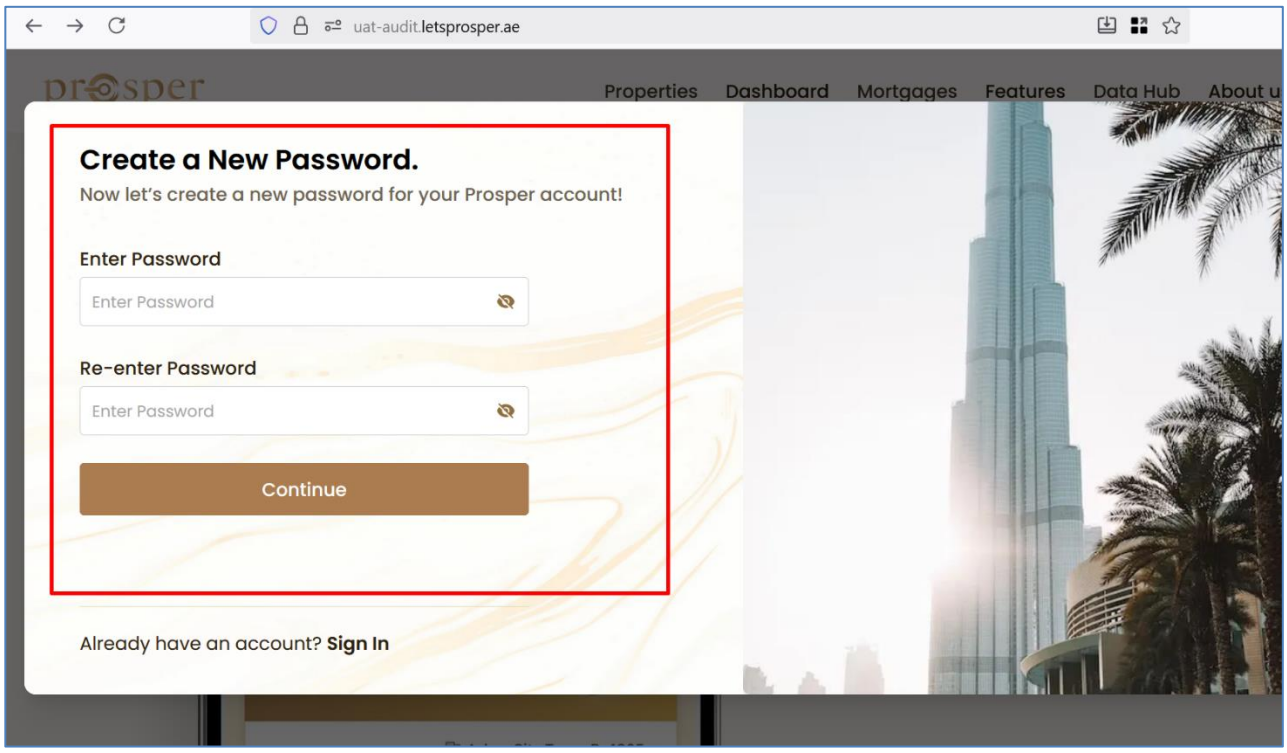
3. 422 Unprocessable Entity Response is received from the server.



4. Change this response with a valid 200 OK response.



5. Forward the response to the browser and the reset password page is accessible.



2.2 Stored Cross-Site Scripting via Malicious File Upload

Severity: High

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: The application is vulnerable to stored cross-site scripting (XSS) through its file upload and preview feature. When a user uploads a PDF containing embedded JavaScript, the application renders it directly into the DOM without sanitization, allowing the script to execute in the viewer's browser.

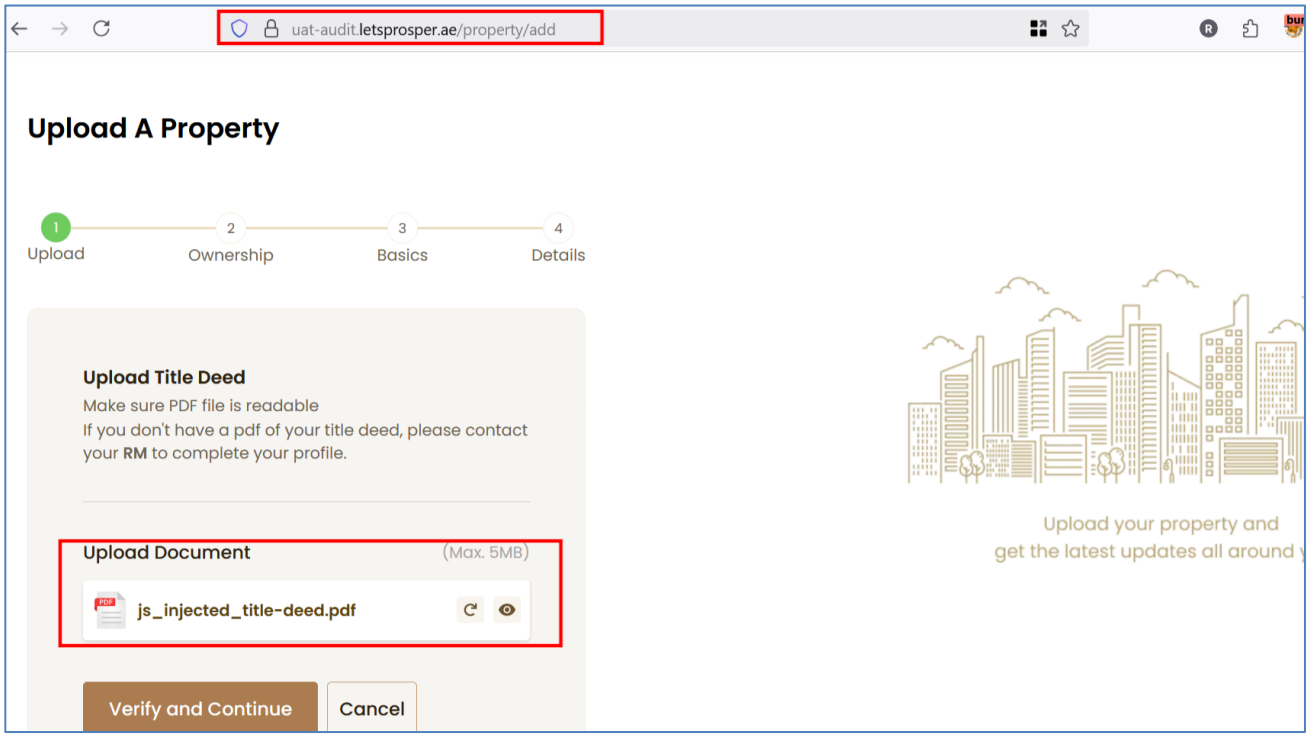
Analysis: At various file upload endpoints, malicious PDFs were successfully uploaded and previewed. Upon viewing, embedded JavaScript executed within the browser context, confirming a stored XSS. This occurs because the application does not sanitize the content or disable scripting during preview. The vulnerability is persistent and can be exploited to steal session data, perform unauthorized actions, or target privileged users.

Remediation: It is recommended to use specialized libraries to inspect PDF files and verify that they do not contain any embedded JavaScript or active objects before accepting uploads. In addition, sanitization tools like **QPDF** or **PDFBox** can also be used to remove any potentially malicious scripts or active content from the files. To further enhance security, serve PDF files with appropriate headers such as Content-Disposition: attachment and X-Content-Type-Options: nosniff. If inline viewing of PDFs is necessary, use secure viewers (e.g., **PDF.js**) with scripting

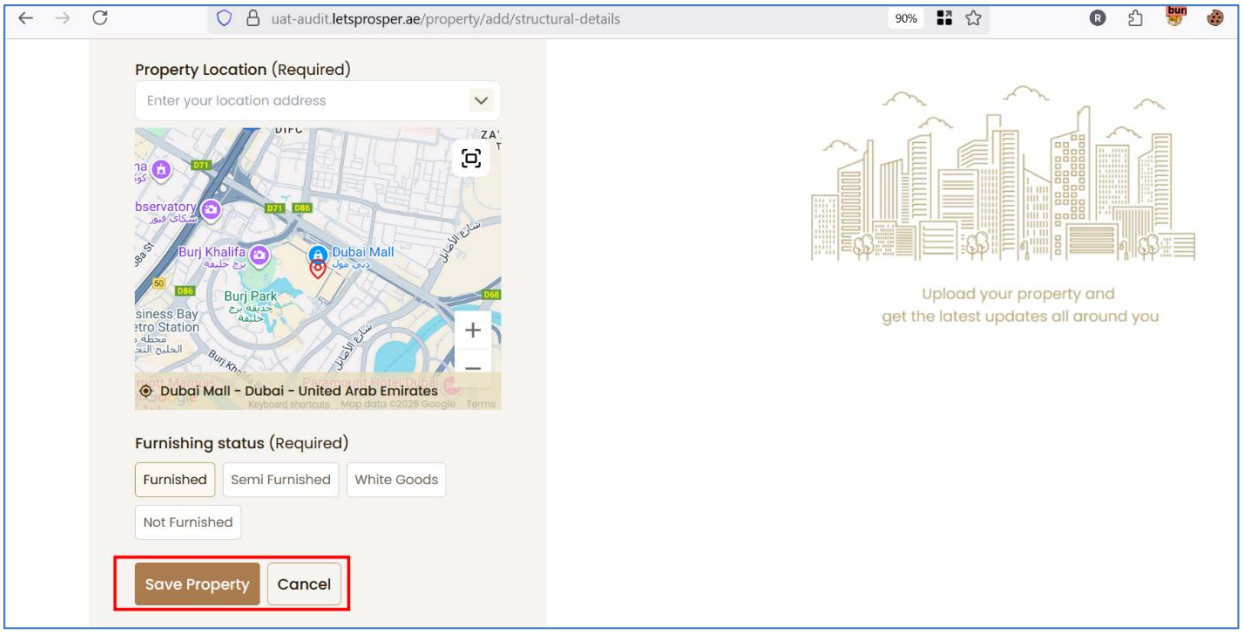
disabled to prevent script execution in the browser context.

Proof Of Concept:

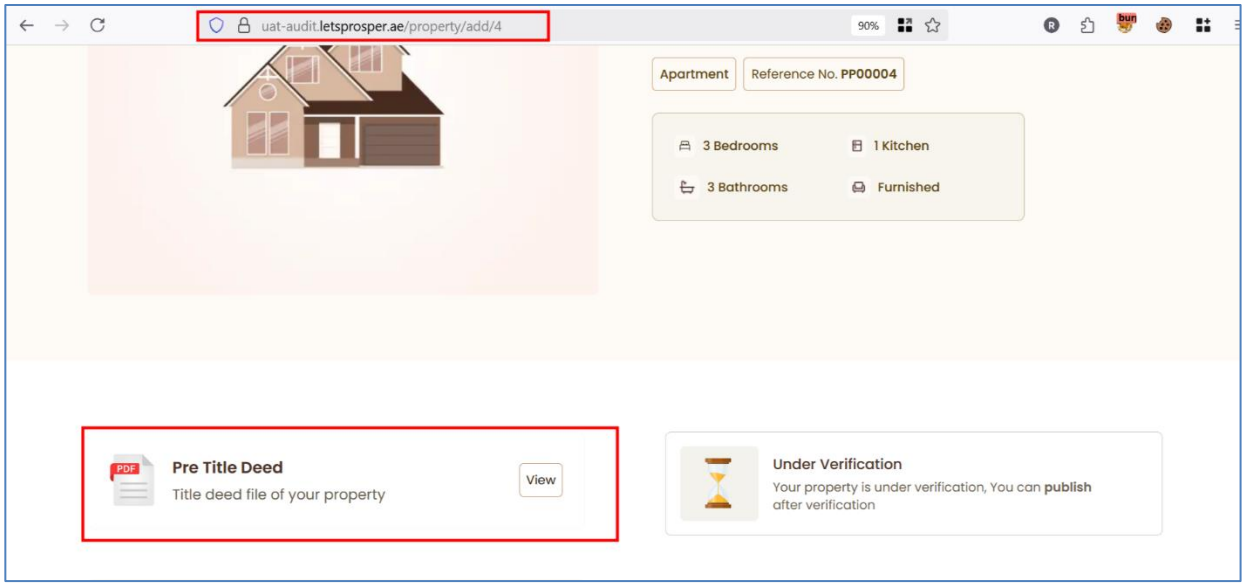
1. Upload a js injected pdf as a title deed.



2. The OCR has verified the pdf and we can continue to save the property.



3. The property is saved. We can continue to view the uploaded title deed.



4. The payload is triggered upon viewing the file.

Note:

The above proof of concept describes a single scenario where a PDF with embedded JavaScript leads to XSS. However, similar issues can occur across multiple functionalities of all the 3 portals where file upload is available.

It is therefore recommended to apply the same remediation consistently across all file upload features to ensure comprehensive protection.

Some of the affected endpoints include, but may not be limited to:

- https://admin.uat-audit.letsprosper.ae/api/title-deed-ocr-data
- https://admin.uat-audit.letsprosper.ae/api/agencies/developer-projects/{id}

- https://admin.uat-audit.letsprosper.ae/api/agencies/developer-properties/{id}
- https://admin.uat-audit.letsprosper.ae/admin/relationship-managers/{id}
- https://admin.uat-audit.letsprosper.ae/admin/agencies/{id}
- https://admin.uat-audit.letsprosper.ae/api/agencies/secondary-property/fetch-form-a-document
- https://admin.uat-audit.letsprosper.ae/admin/share-feedbacks

2.3 Unrestricted File Upload

Severity: High

Issue Related to: Agency Portal, Admin Portal

Issue Definition: The application suffers from unrestricted file upload vulnerabilities across multiple functionalities, allowing attackers to upload files of any type, including potentially dangerous executable files (.exe), without proper validation or restriction of file types and extensions.

Scenario 1: In the admin portal, certain file upload functionalities were identified where files such as agency logo and agency contract can be uploaded without restriction on file type or extension. Similarly, in the agency portal, user admin profile photos can be replaced by uploading files of any type, including .exe files or other potentially dangerous formats.
This occurs because the application does not implement strict server-side validation of file types or enforce allowed extensions, relying solely on client-side checks which can be bypassed easily.
Affected Endpoints:
https://admin.uat-audit.letsprosper.ae/admin/agencies/{id}
https://admin.uat-audit.letsprosper.ae/api/agencies/agency-users/{id}
https://admin.uat-audit.letsprosper.ae/api/agencies/property-specialists/{id}

Scenario 2: In the agency portal, within the property listings section, there is a file upload feature intended for uploading property videos. While there is a client-side restriction that allows only .mp4 files, this validation can be bypassed, allowing attackers to upload files such as .svg or any other extension. This happens because there is no server-side validation to enforce allowed file types or MIME types, making the system vulnerable to storing arbitrary files.
Affected Endpoints:
https://admin.uat-audit.letsprosper.ae/api/agencies/secondary-properties/{id}
https://admin.uat-audit.letsprosper.ae/admin/share-feedbacks

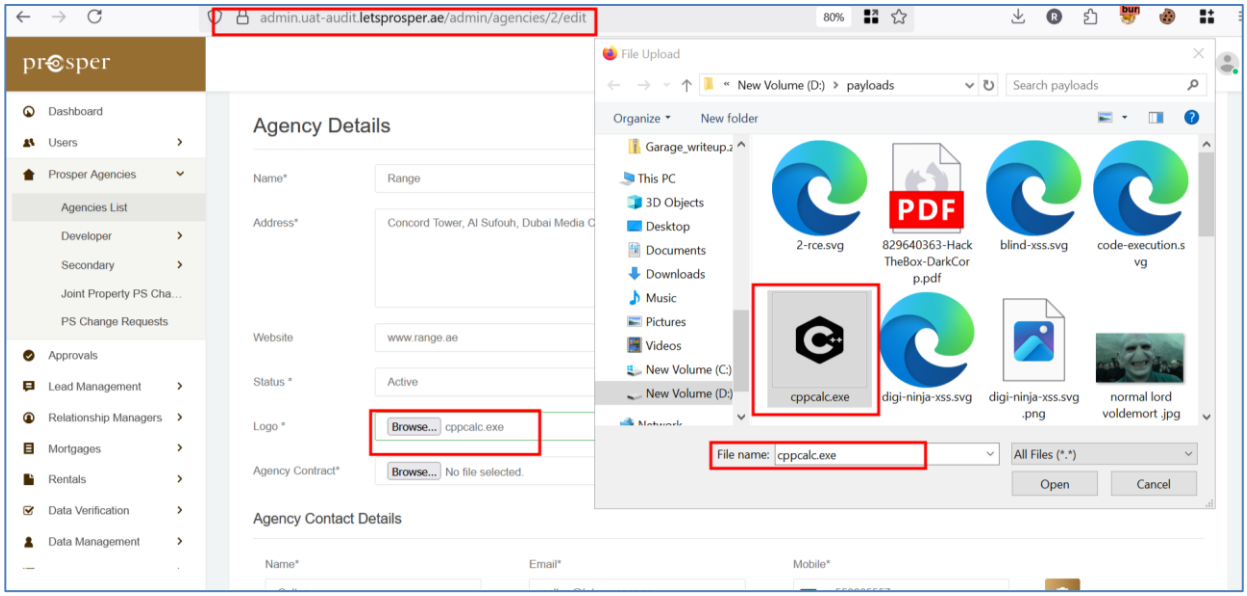
Analysis: Unrestricted file uploads allow attackers to upload executable or malicious files, which can lead to remote code execution, malware distribution, or manipulation of application behavior. The lack of server-side validation increases the risk of system compromise and affects overall application security.

Remediation: It is recommended to implement strict server-side validation to allow only specific file types and extensions. Verify MIME types and file signatures, enforce size limits, and store files securely outside the web root. Serve files with headers like Content-Disposition: attachment and X-Content-Type-Options: nosniff. Do not rely solely on client-side restrictions.

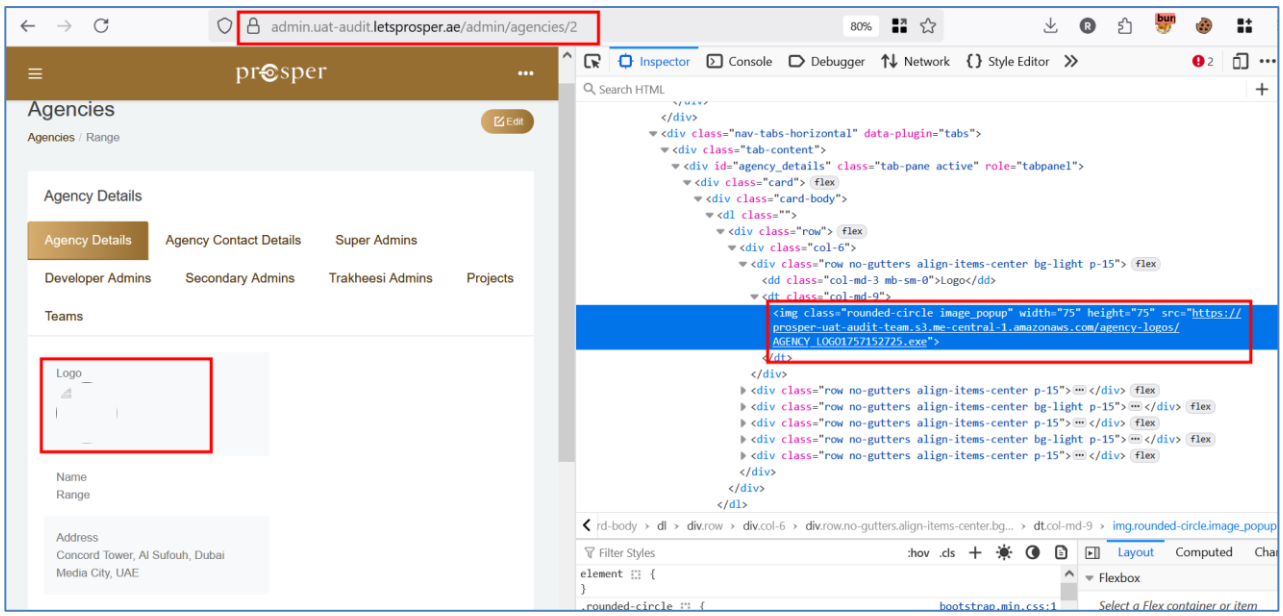
Proof Of Concept:

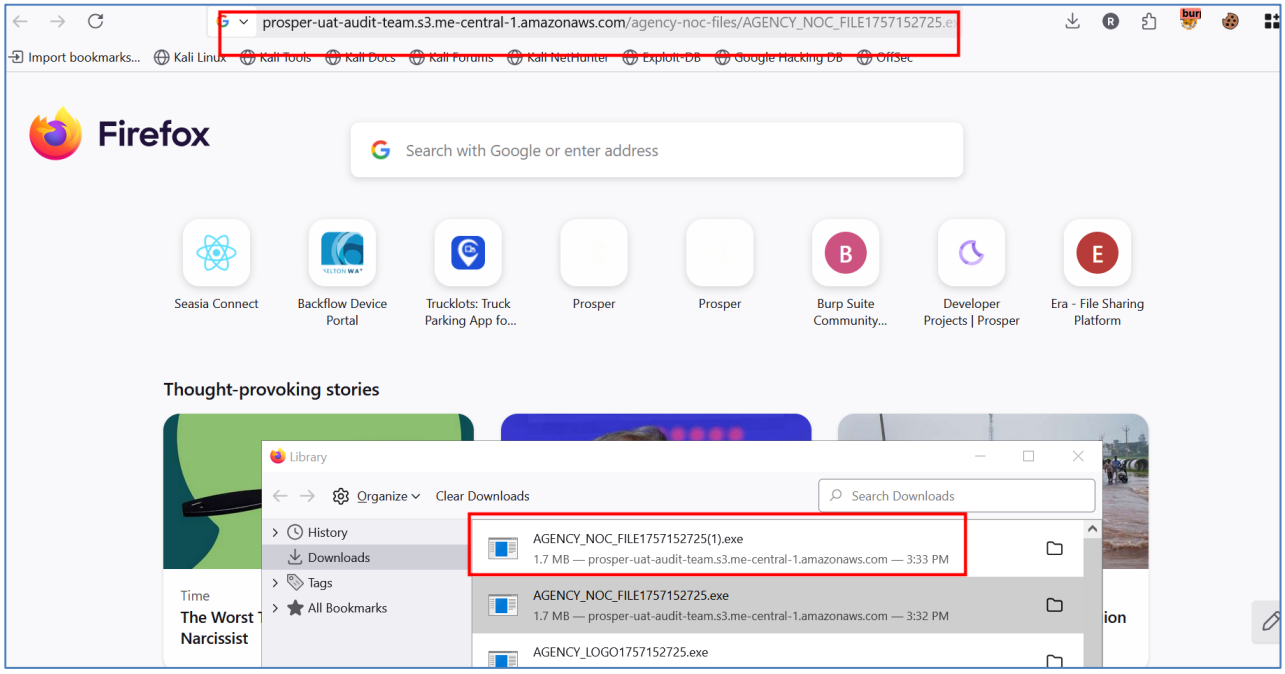
Scenario 1

1.1 Try to upload an exe file as agency logo or contract.

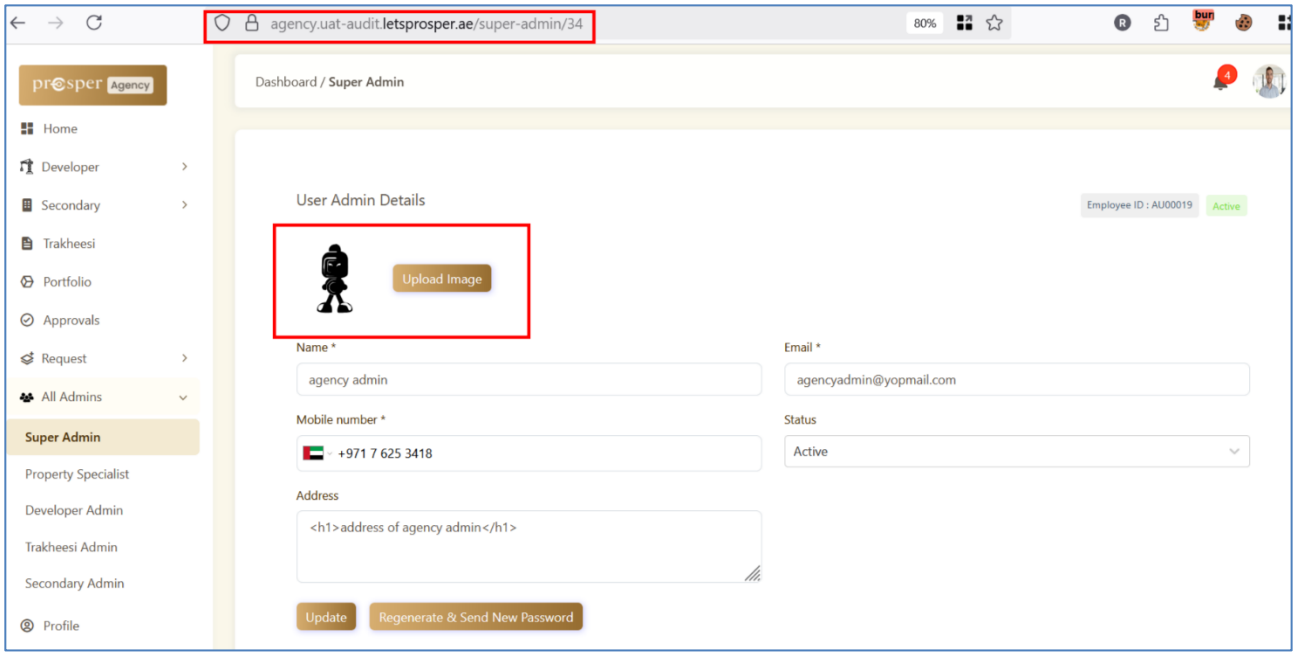


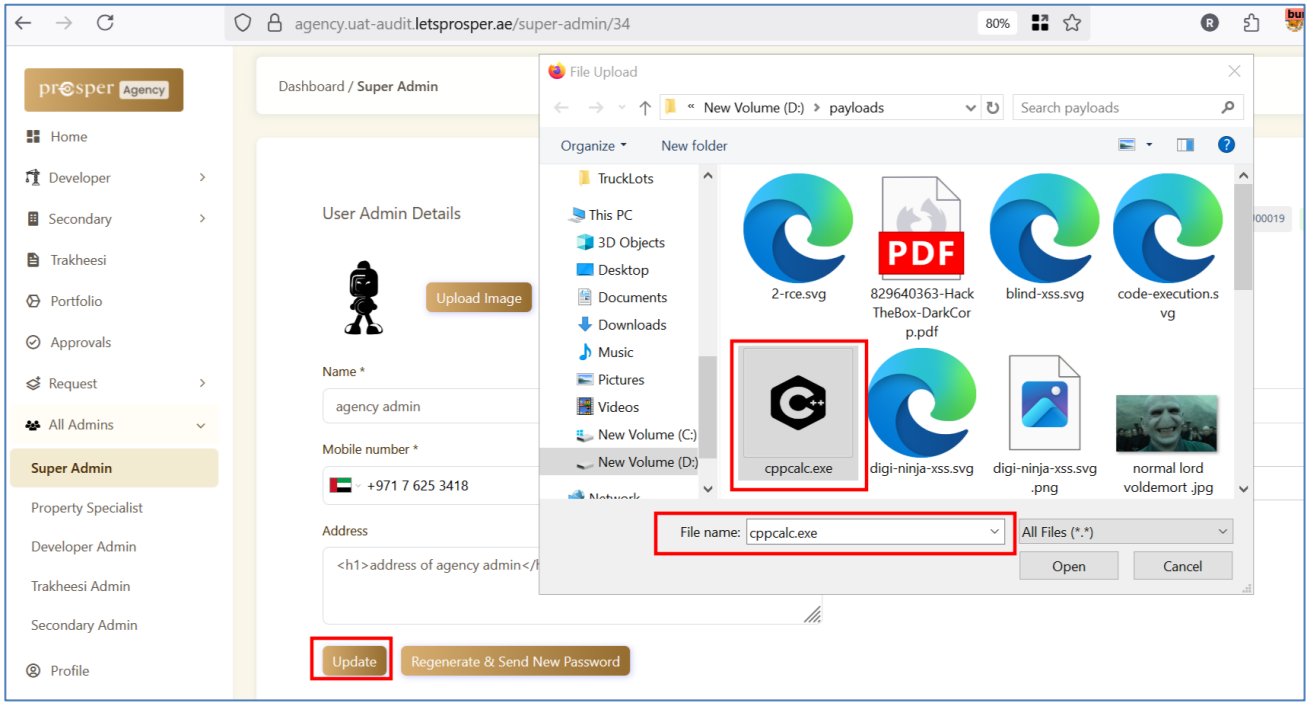
1.2 Go to agency details section and notice that the exe was successfully uploaded as agency logo or agency contract.



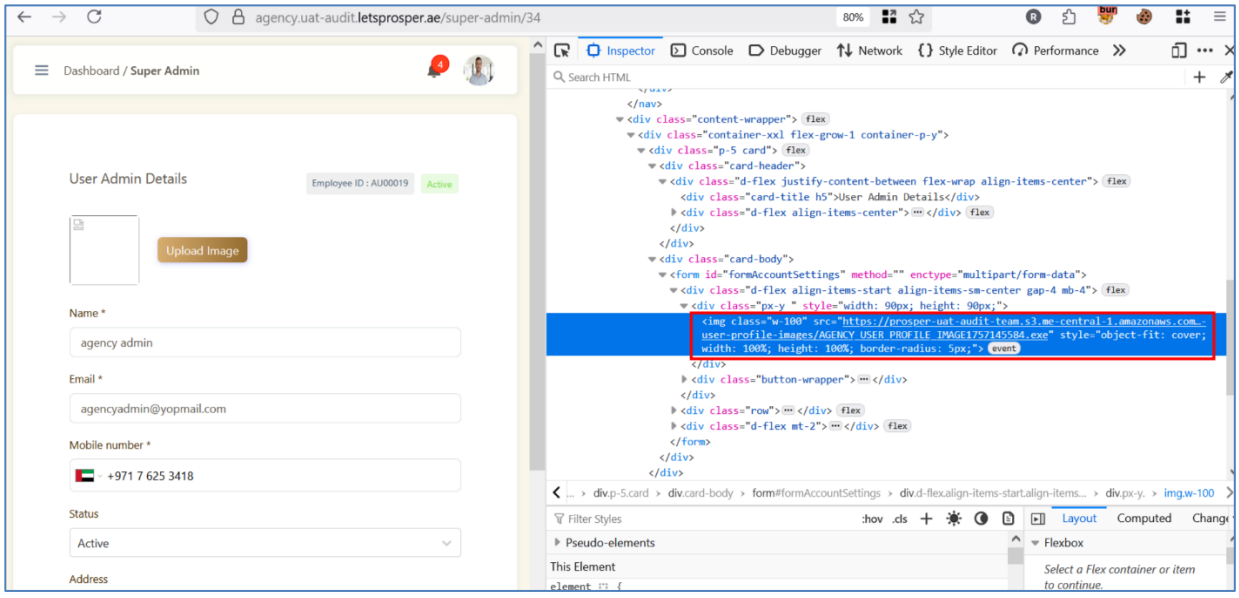


2.1 Similarly try upload an exe file as a profile photo of an admin.





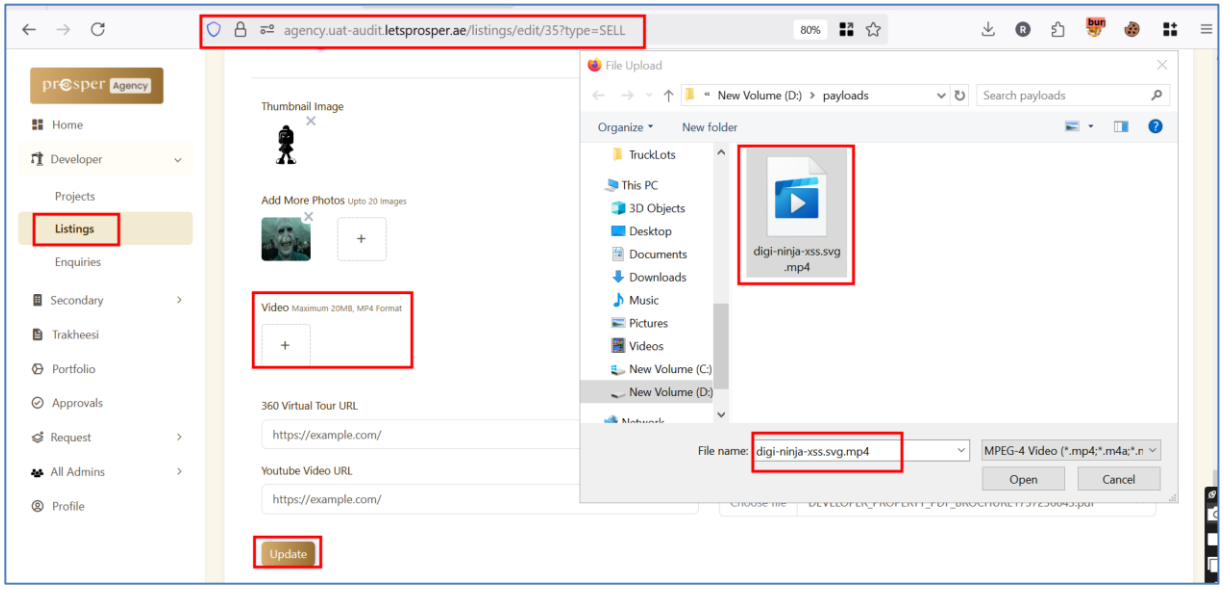
2.2 The exe is successfully uploaded as a profile photo.



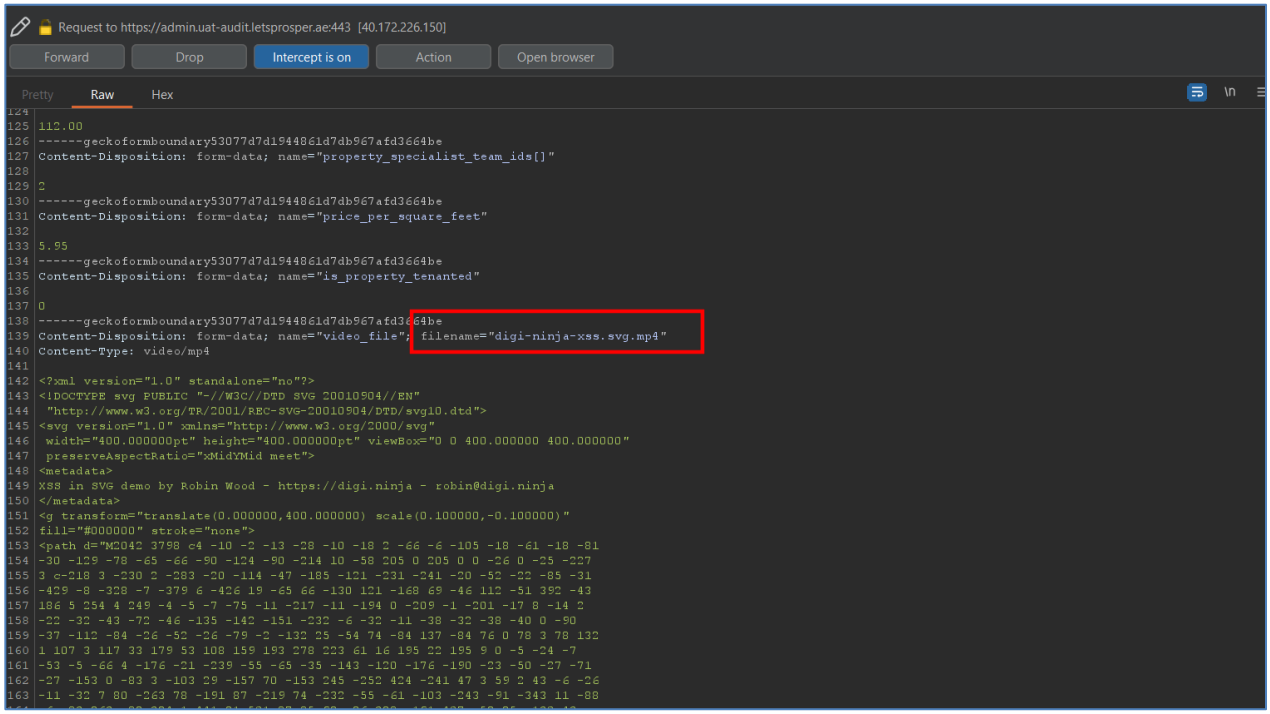
Security Audit Report

Scenario 2

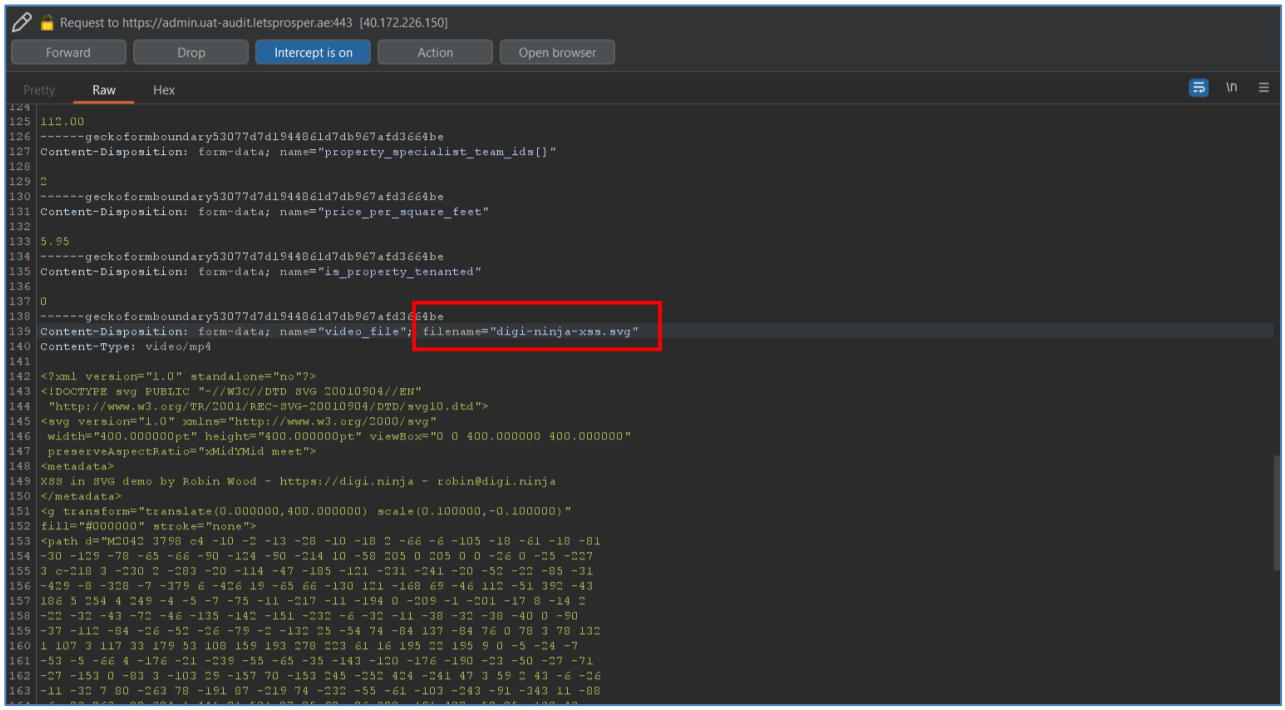
1. Upload a svg or any other extension file masked as an mp4 file.



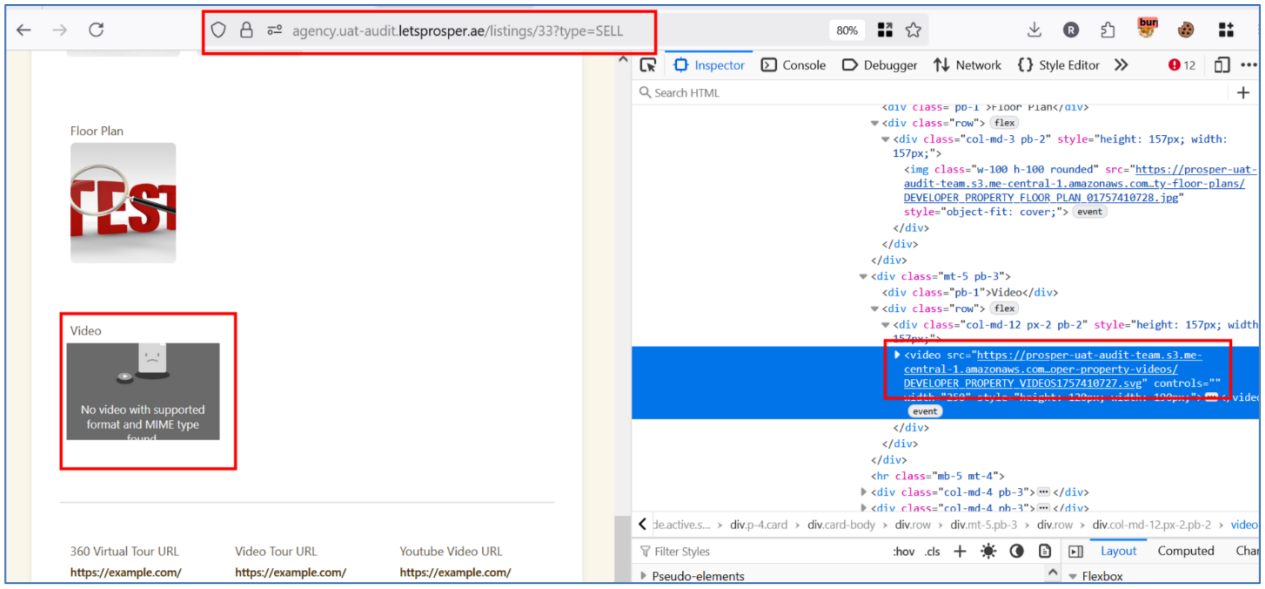
2. Intercept the upload request in the proxy intercept tool.



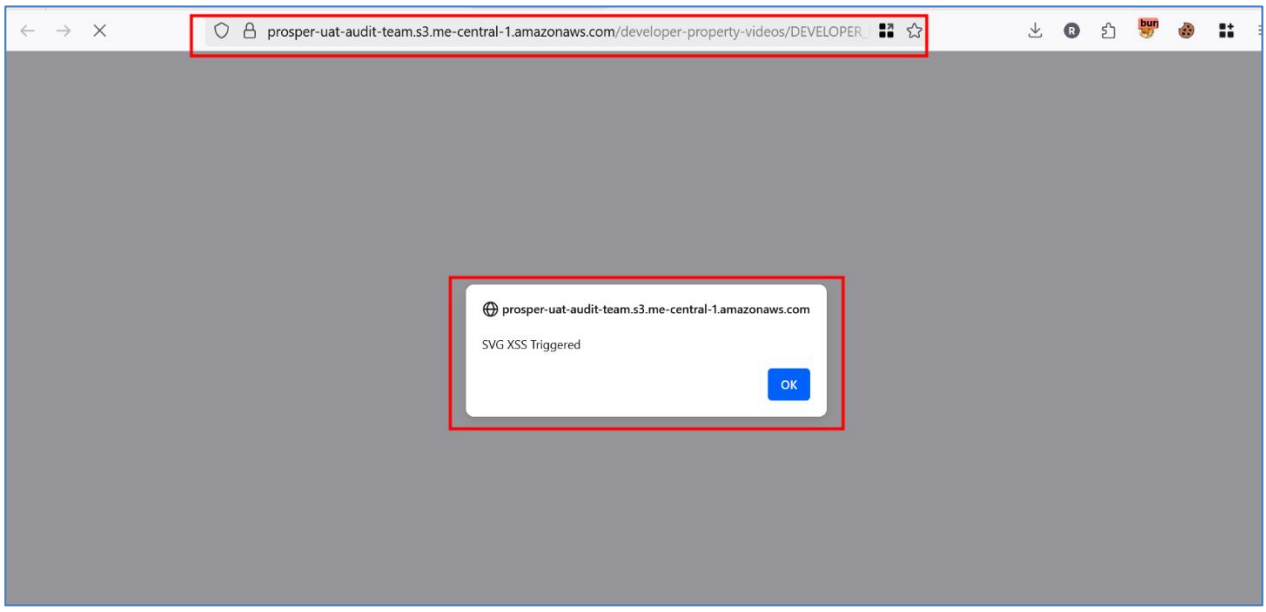
3. Change the extension of the mp4 to the original extension, in this case svg.



4. The file gets successfully uploaded.



5. Even the malicious payload in the svg file gets triggered.



2.4 Stored XSS via SVG upload

Severity: High

Issue Related to: Agency Portal, Admin Portal

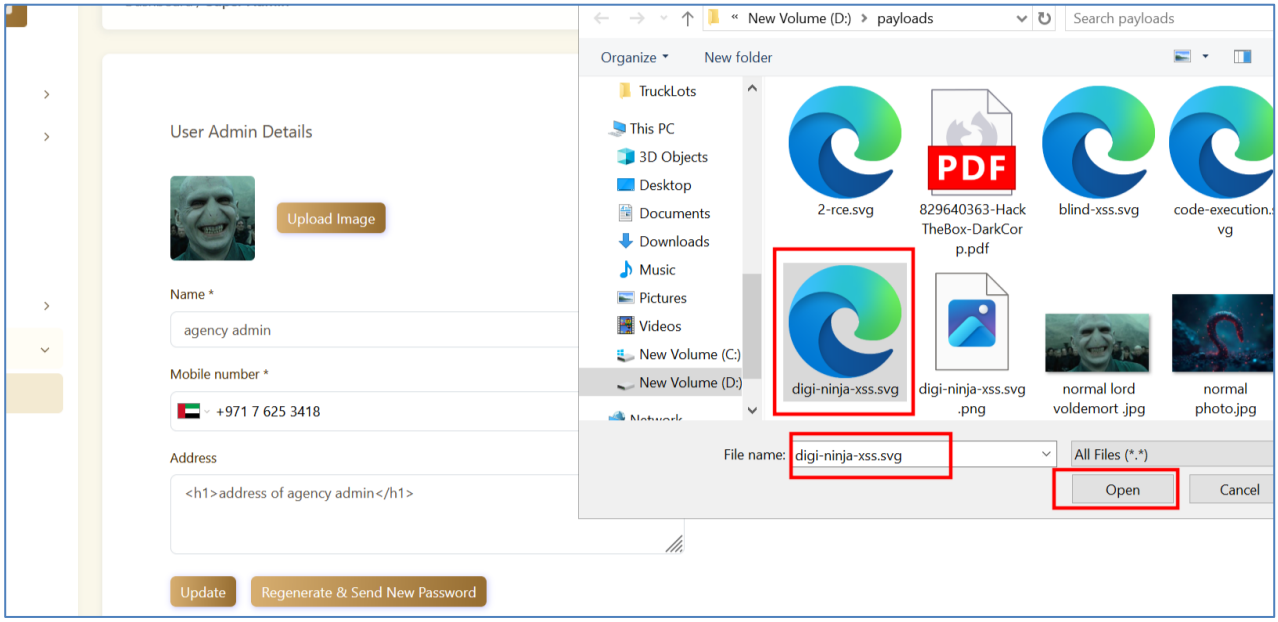
Issue Definition: The application allows users to upload SVG (Scalable Vector Graphics) files without sanitizing them. SVG files can contain embedded JavaScript, which executes when the file is viewed in the browser. This leads to stored XSS, as the malicious script executes in the context of other users or administrators when viewing the uploaded SVG.

Analysis: An attacker can upload an SVG file containing malicious JavaScript code. When this file is rendered in the browser, the script executes, potentially stealing session cookies, performing unauthorized actions, or targeting privileged users. This compromises both the security of the application and the users interacting with it.

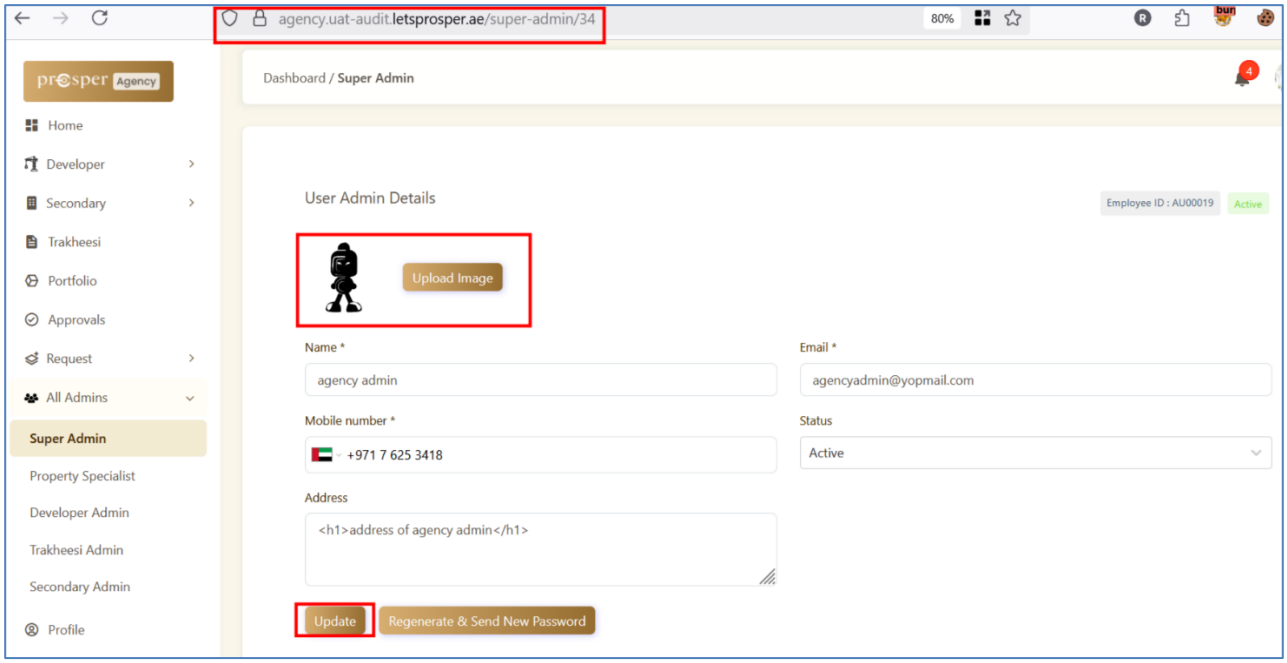
Remediation: It is recommended to restrict file uploads to allow only safe image types like .jpg, .png, or .gif. If SVG uploads are required, sanitize them using libraries such as SVG Sanitizer to strip any embedded scripts or active content before storing or serving.

Proof of Concept:

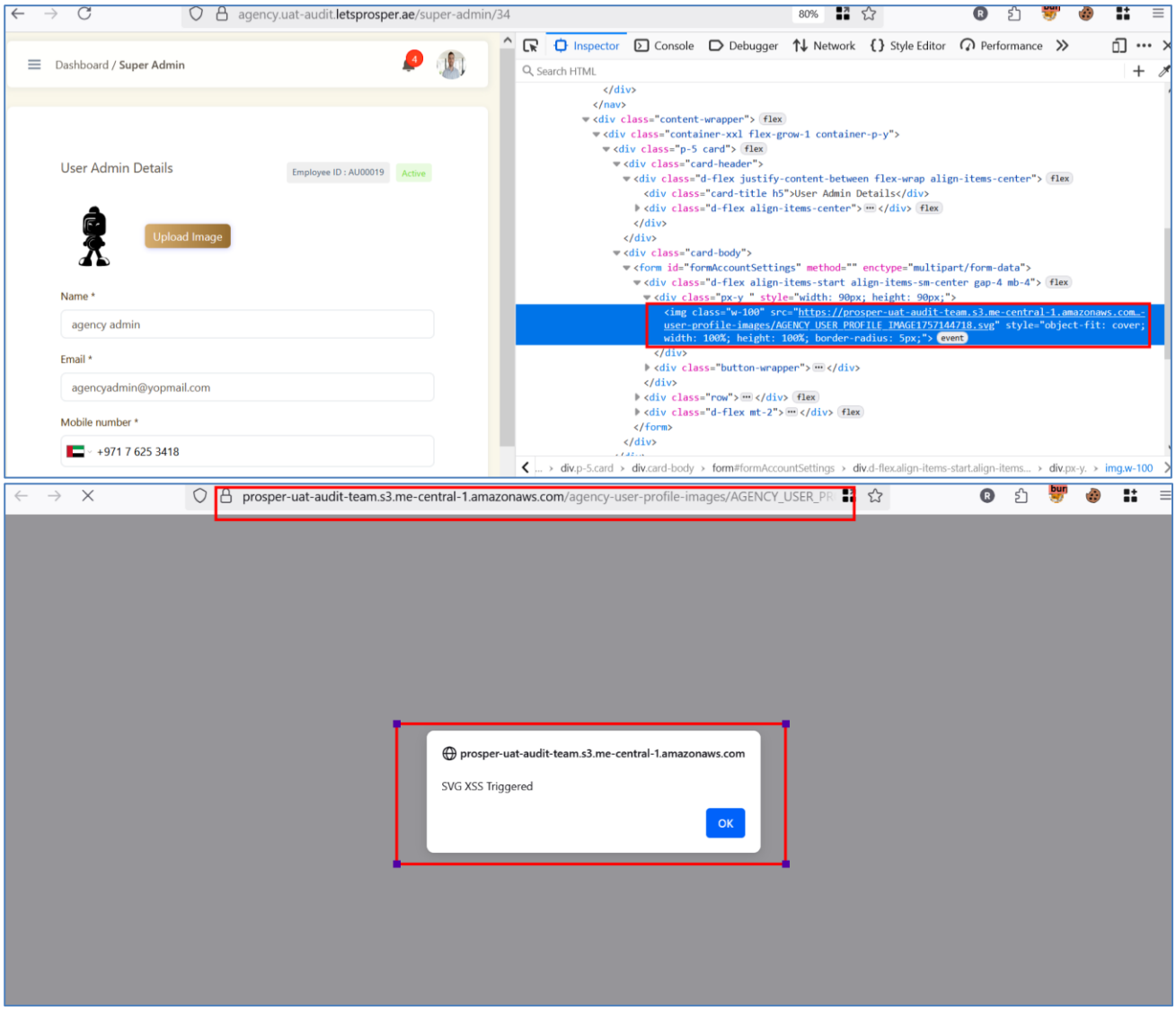
1. In the agency portal, update the profile photo of a user admin with the malicious svg with js code embedded into it.



2. The svg gets uploaded successfully.



3. The js code gets triggered upon accessing the profile photo.



Note:

The above proof of concept describes a single scenario where a SVG with embedded JavaScript leads to XSS. However, similar issues can occur across multiple functionalities where photo upload is available.
It is therefore recommended to apply the same remediation consistently across all photo upload features to ensure comprehensive protection.

Some of the affected endpoints include, but may not be limited to:

- https://admin.uat-audit.letsprosper.ae/admin/banners/{id}
- https://admin.uat-audit.letsprosper.ae/admin/share-feedbacks
- https://admin.uat-audit.letsprosper.ae/api/agencies/agency-users/{id}
- https://admin.uat-audit.letsprosper.ae/api/agencies/secondary-properties/{id}
- https://admin.uat-audit.letsprosper.ae/admin/relationship-managers/{id}

2.5 Stored XSS via input field

Severity: High

Issue Related to: User Portal, Agency Portal, Admin Portal

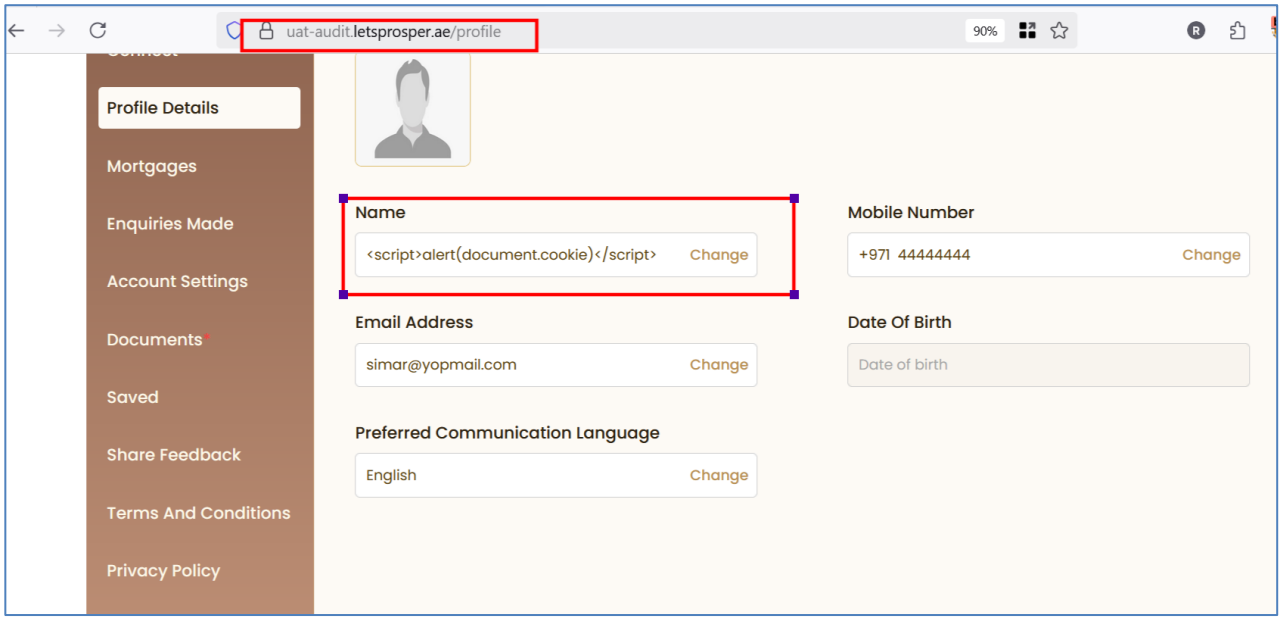
Issue Definition: The application is vulnerable to stored cross-site scripting (XSS) through input fields where user-supplied data is saved and later rendered in the web application without proper sanitization or escaping. Malicious JavaScript code injected into input fields gets executed in the browser of any user who views the affected data.

Analysis: An attacker can inject JavaScript payloads into input fields like “Name” in Profile Details. When the application displays the saved input in the browser, the script executes in the context of any user viewing it. This can lead to session theft, unauthorized actions on behalf of another application user, severely impacting application integrity and user trust.

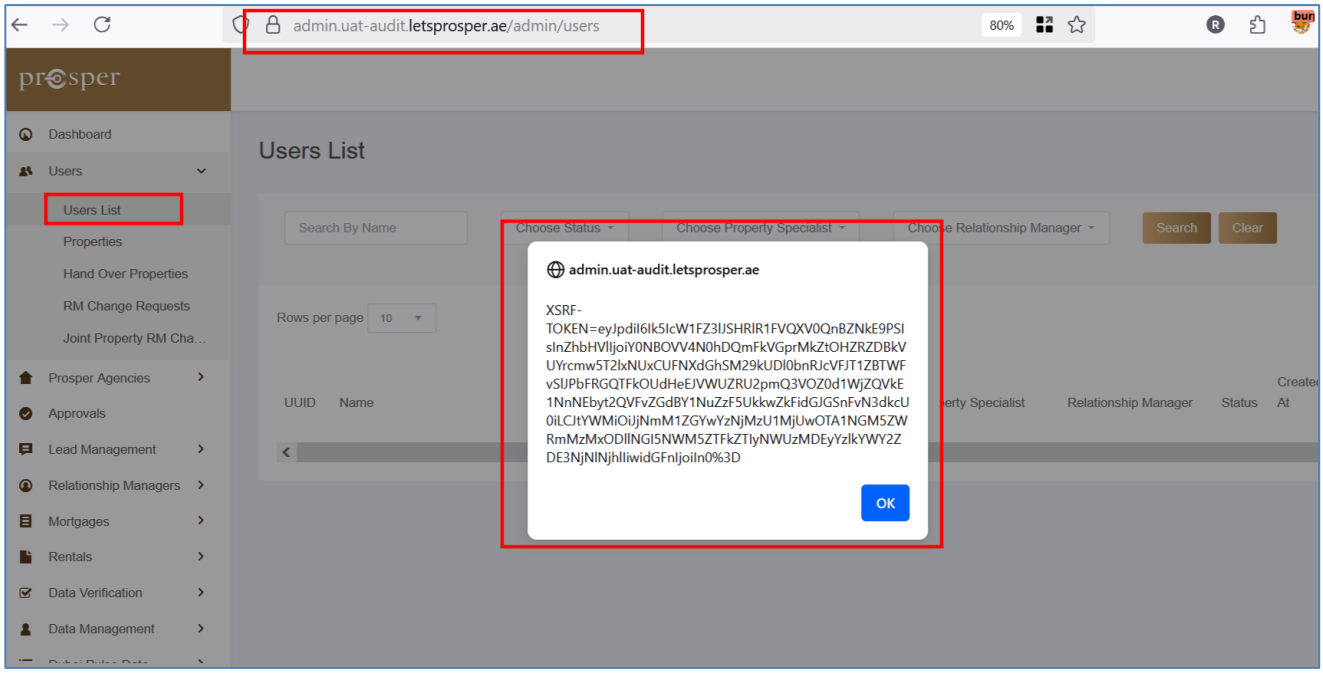
Remediation: It is recommended that all user-supplied input should be properly sanitized and escaped before being displayed in the application to prevent script execution. Input validation should be enforced to block suspicious or unexpected content at the point of data entry. Additionally, a strong Content Security Policy (CSP) should be implemented to limit the ability of any malicious scripts from executing, reducing the impact of potential XSS attacks.

Proof of Concept:

1. Enter a javascript payload in the name under the profile details section.



2. In the admin portal, under the users list section where all the usernames are displayed, the payload gets triggered.



Note:

The above proof of concept describes a single scenario where a javascript input leads to XSS. However, similar issues can occur across multiple functionalities.

It is therefore recommended to apply the same remediation consistently across all input fields to ensure comprehensive protection.

Some of the affected endpoints include, but may not be limited to:

- https://admin.uat-audit.letsprosper.ae/api/agencies/developer-properties/{id}
- https://admin.uat-audit.letsprosper.ae/api/agencies/agency-users/{id}
- https://admin.uat-audit.letsprosper.ae/admin/relationship-managers/{id}
- https://admin.uat-audit.letsprosper.ae/api/users
- https://admin.uat-audit.letsprosper.ae/admin/share-feedbacks

2.6 Insecure Direct Object Reference (IDOR)

Severity: High

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: The application is vulnerable to Insecure Direct Object Reference (IDOR), allowing unauthorized access to sensitive user data by manipulating guessable identifiers in the URL or request parameters. This enables attackers to access or modify other users' data without proper authorization checks.

Security Audit Report

Scenario 1: Users can view other users' added properties simply by changing the property ID in the /property/add endpoint. Additionally, one user can view another user's dashboard via the /dashboard/owner-property endpoint by modifying the user or property ID in the request, due to a lack of proper ownership validation.

Scenario 2: The Title Deed documents are stored in a predictable manner, where each document is assigned a large numeric ID. However, the ID pattern reveals that only the last 4 digits change incrementally for new documents. This allows an attacker to brute-force the last few digits and gain unauthorized access to other users' title deeds.

Scenario 3: Agency admins (such as secondary admins, tarkheesi, developers, and property specialists) are able to view and perform actions on properties assigned to other admins. This occurs due to missing role-based access control checks that should restrict each admin's access only to properties they are responsible for. Additionally, similar IDOR vulnerabilities may also be present in the Relationship Managers (RMs) and Data Verification team functionalities, allowing unauthorized access or actions on data belonging to others.

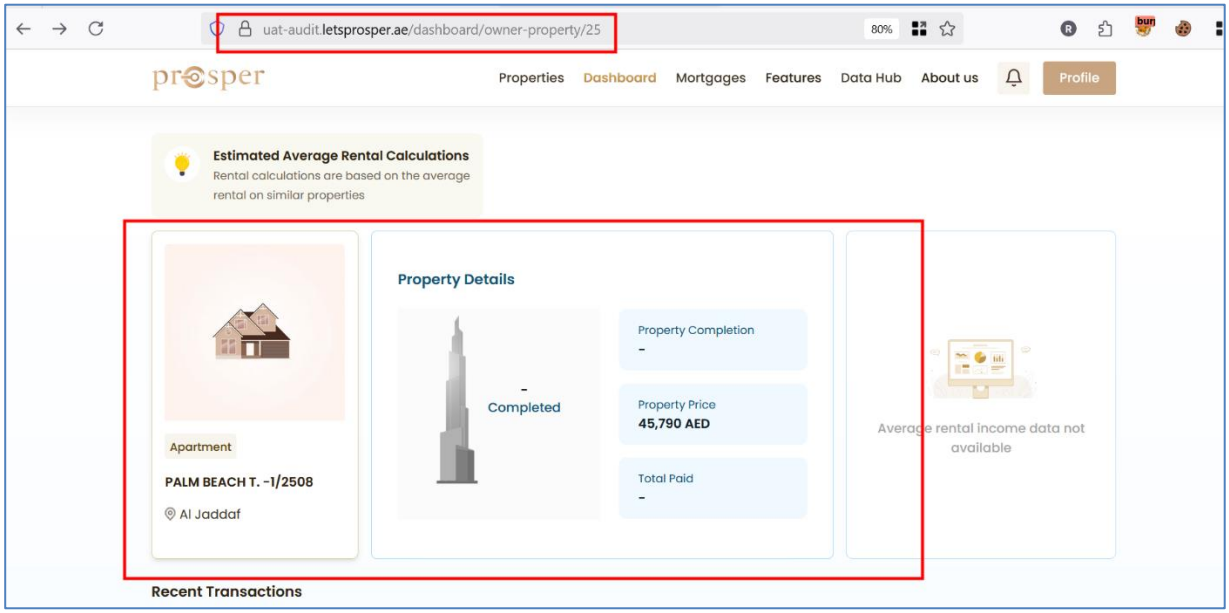
Analysis: An attacker can access or modify data belonging to other users, such as property details, dashboards, or sensitive documents like title deeds. This leads to exposure of private data, loss of confidentiality, privacy violations, and potential data manipulation or deletion, severely impacting application security and user trust.

Recommendation: It is recommended to implement strict authorization checks on the server side to ensure that users can only access or modify resources they own or are permitted to view. Avoid exposing predictable identifiers and consider using non-guessable unique identifiers (UUIDs) instead of sequential numbers. Ensure that direct access to sensitive files is only possible via authorized mechanisms and never by relying on obscured URLs. Additionally, apply rate limiting mechanisms like reCAPTCHA to prevent automated brute-force attempts.

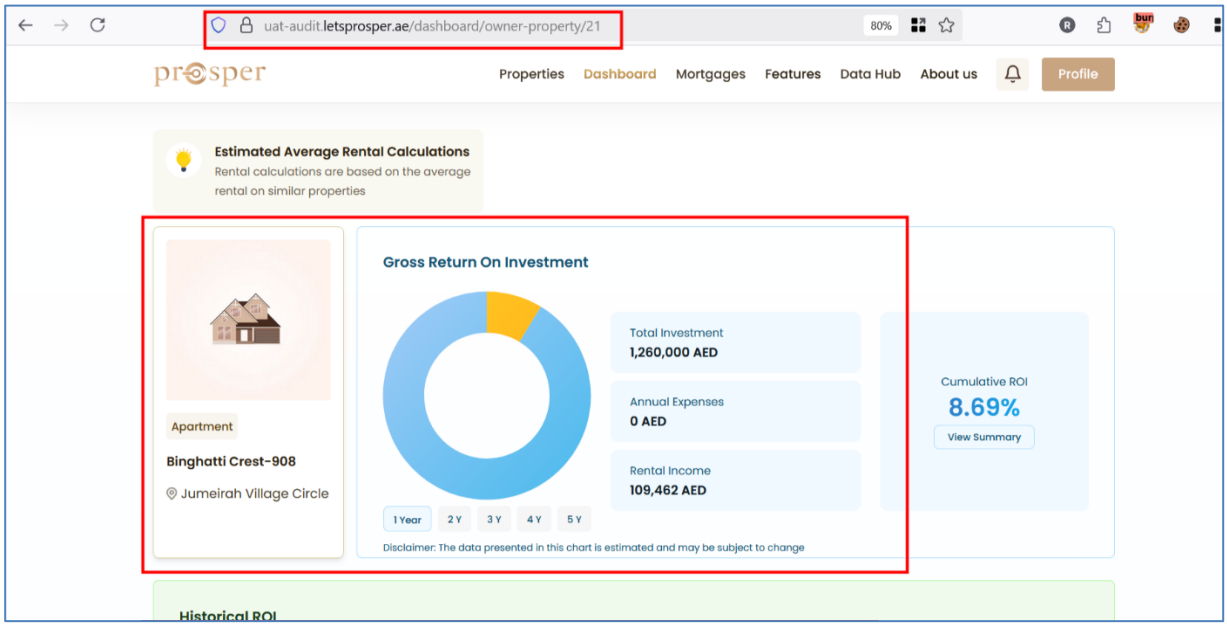
Proof of Concept:

Scenario 1

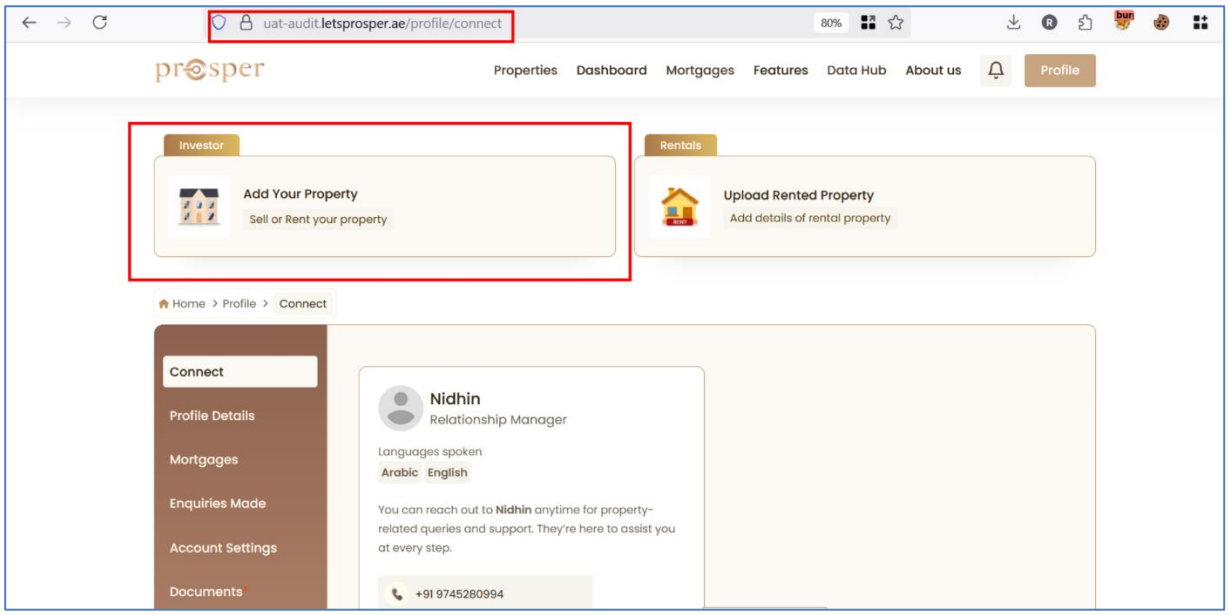
1. View a user's dashboard and his property which has been given property id 25.



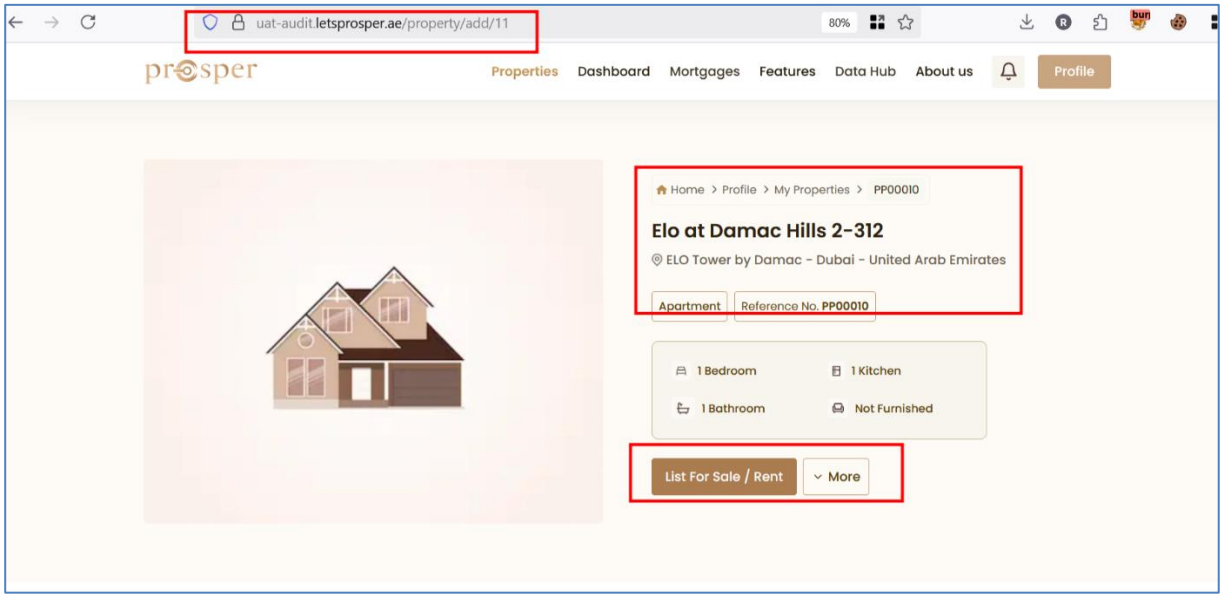
2. Change the property id to point it to someone else's property, in this case 21. The related dashboard becomes accessible.



3. Similar is the case for /add/property endpoint. Currently we have a new user who has not added any property in the prosper system.



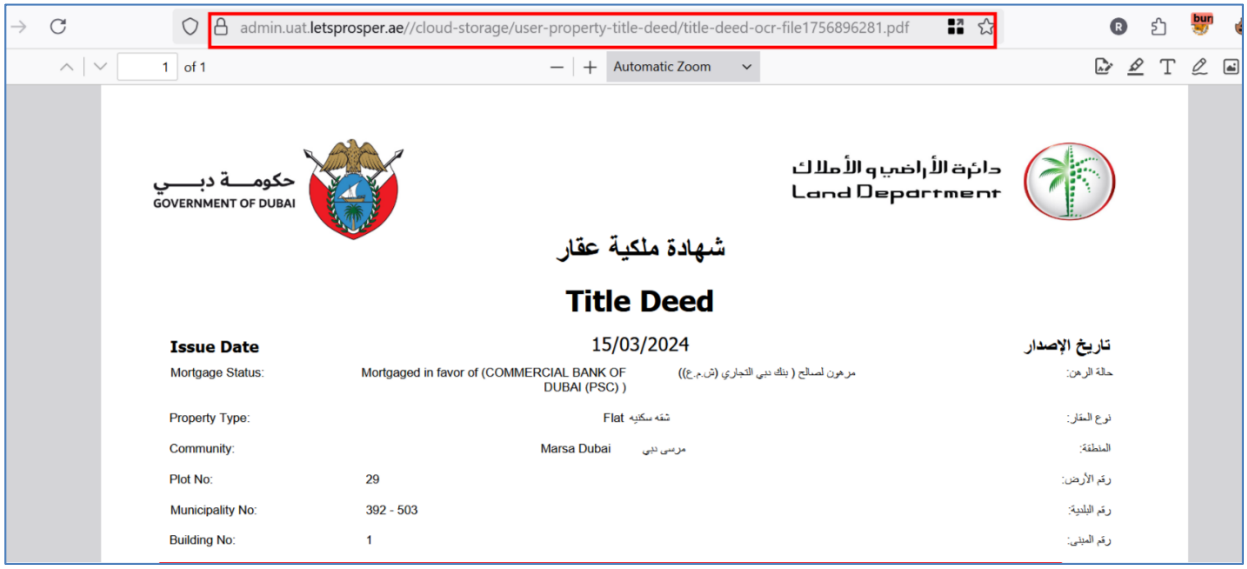
4. Just by accessing the /add/property endpoint with a valid property id, the related property becomes accessible.



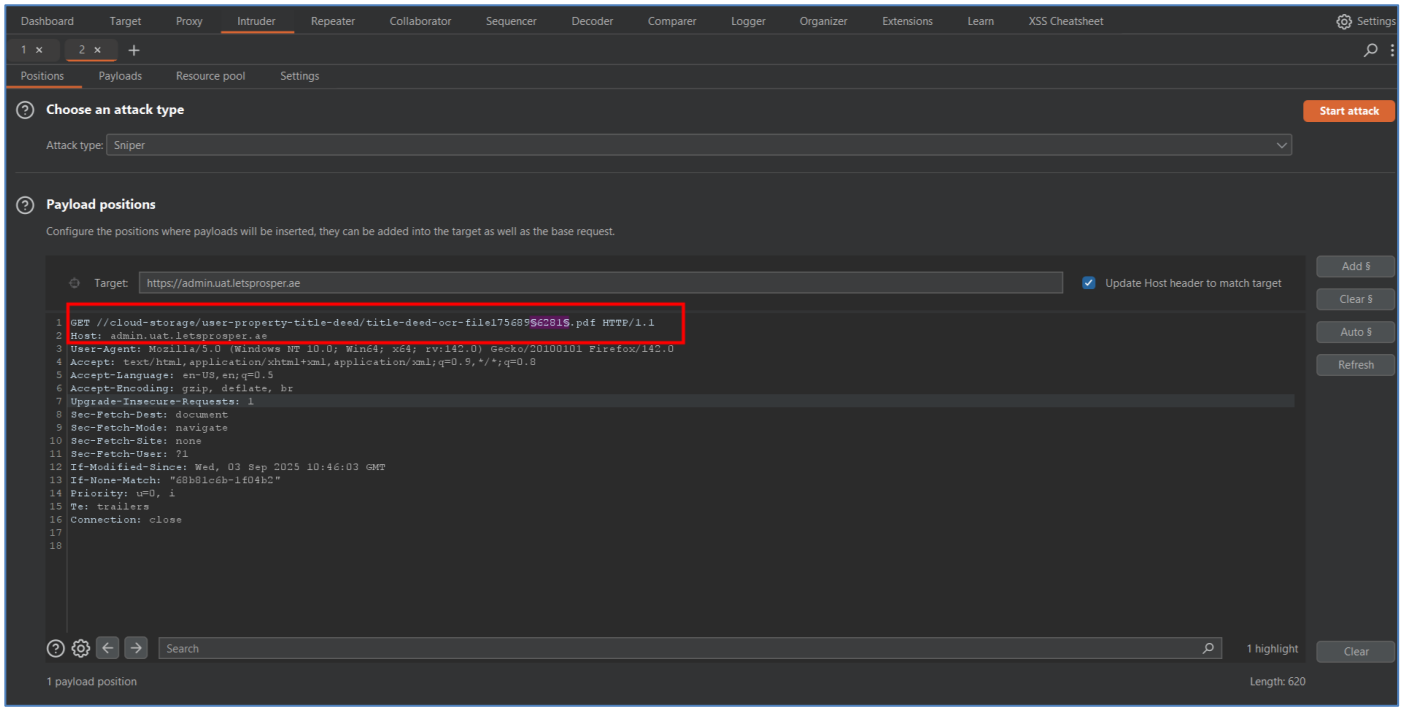
Security Audit Report

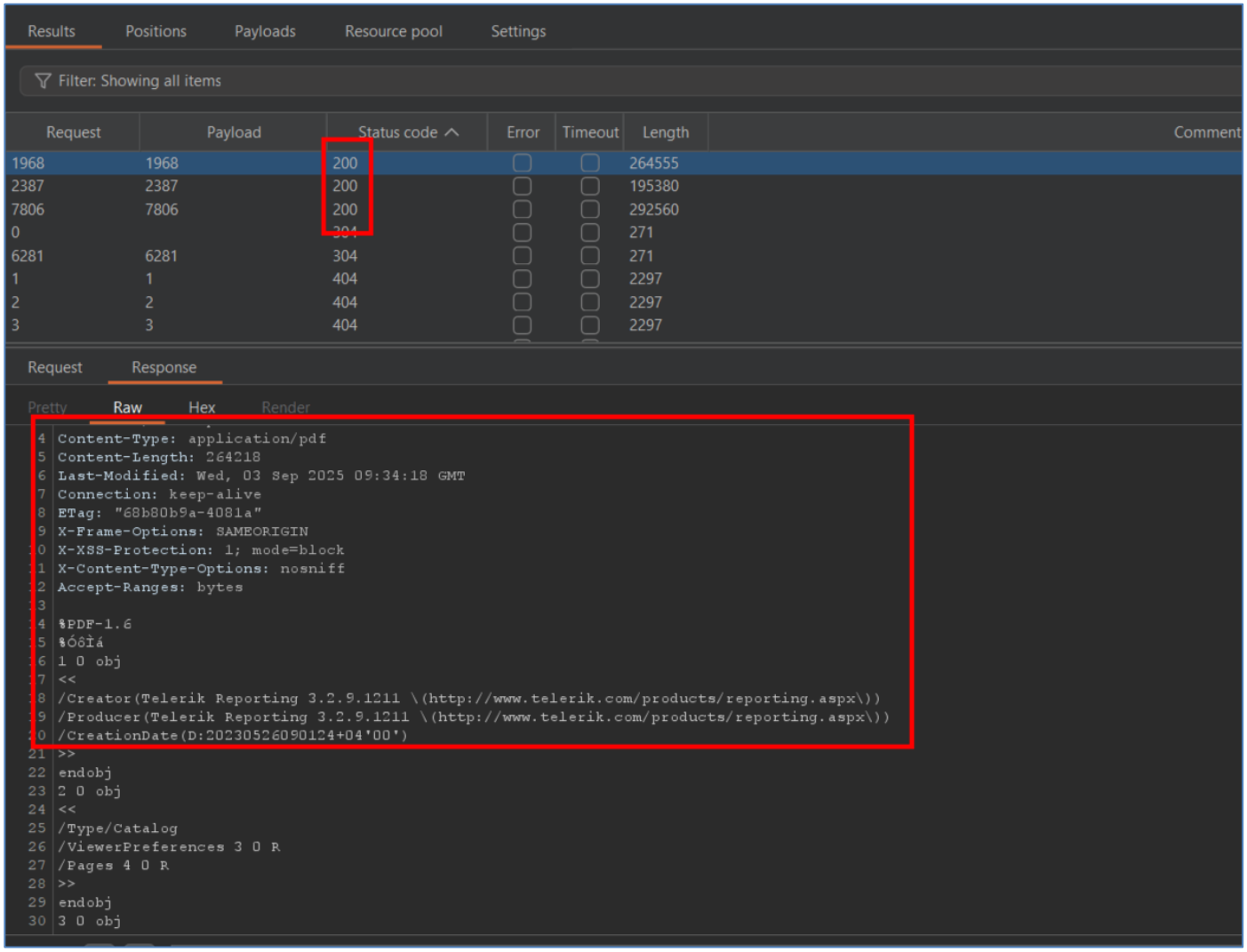
Scenario 2

1. The title deed in the clous storage is assigned a numeric id.



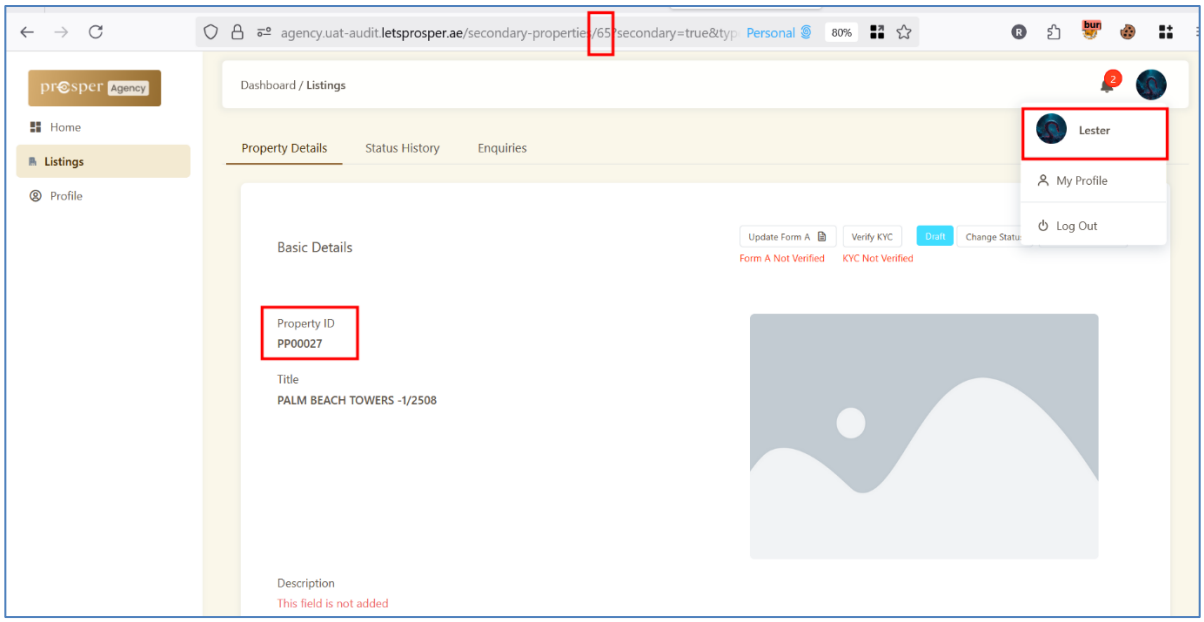
2. Add the current title deed's last 4 digit number into a brute-forcing tool. Upon bruteforcing, various other title deeds become accessible.



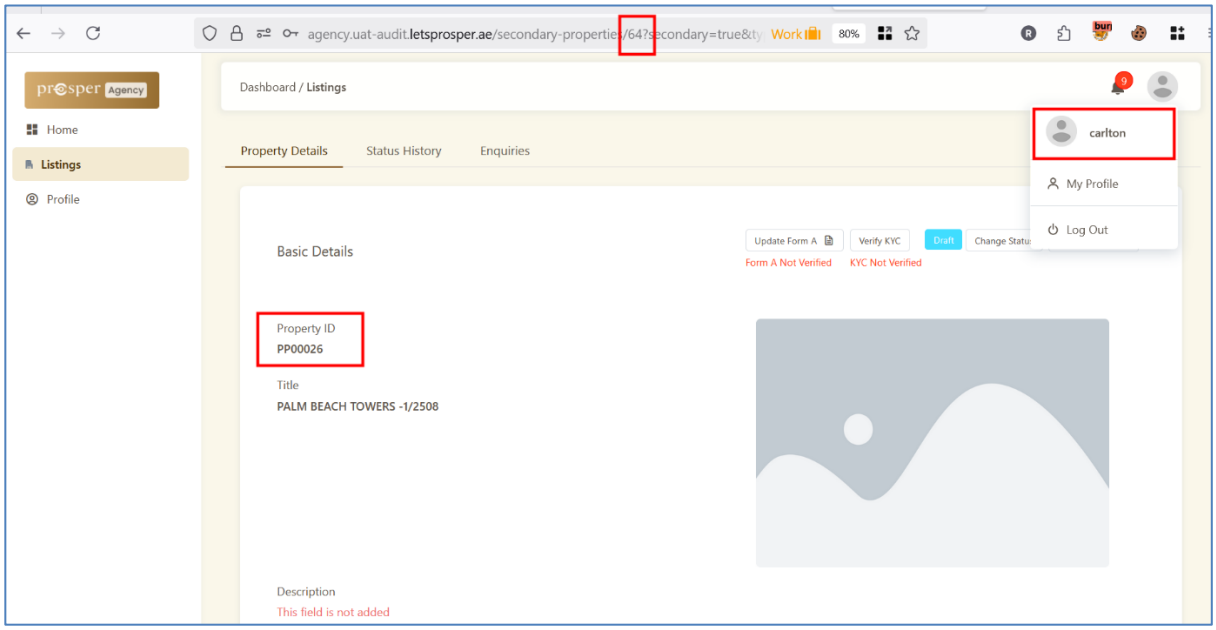


Scenario 3

1. The property with property ID PP00027 is assigned to Secondary Admin “Lester”.



2. Login into Carlton Account and the same property is accessible.



2.7 Broken Access Control - Unrestricted Direct File Access

Severity: High

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: The application stores sensitive files such as **passports, title deeds and other important documents** in cloud storage. However, these files are directly accessible via their URLs without any access control or authentication checks.
Any user who obtains or guesses the URL of these files can access sensitive documents without restriction.

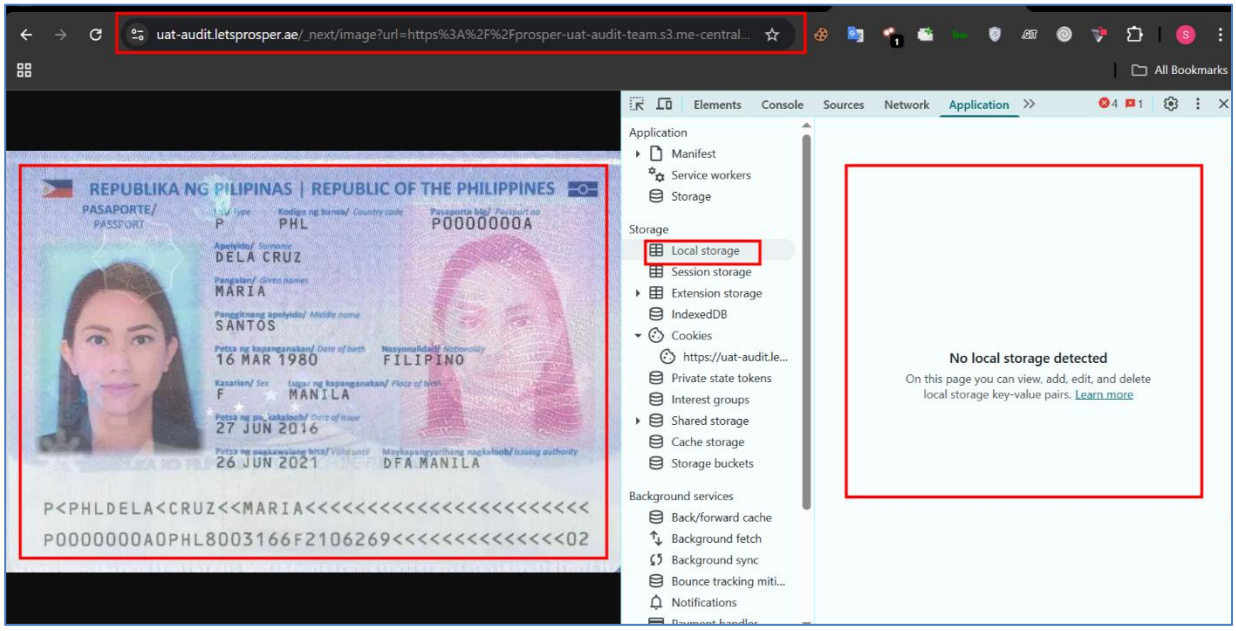
Analysis: Sensitive documents such as passports, title deeds, and contracts are publicly accessible via direct URLs without any authentication. This exposes personal and business data to unauthorized users, risks identity theft, fraud, regulatory violations, and potential public indexing

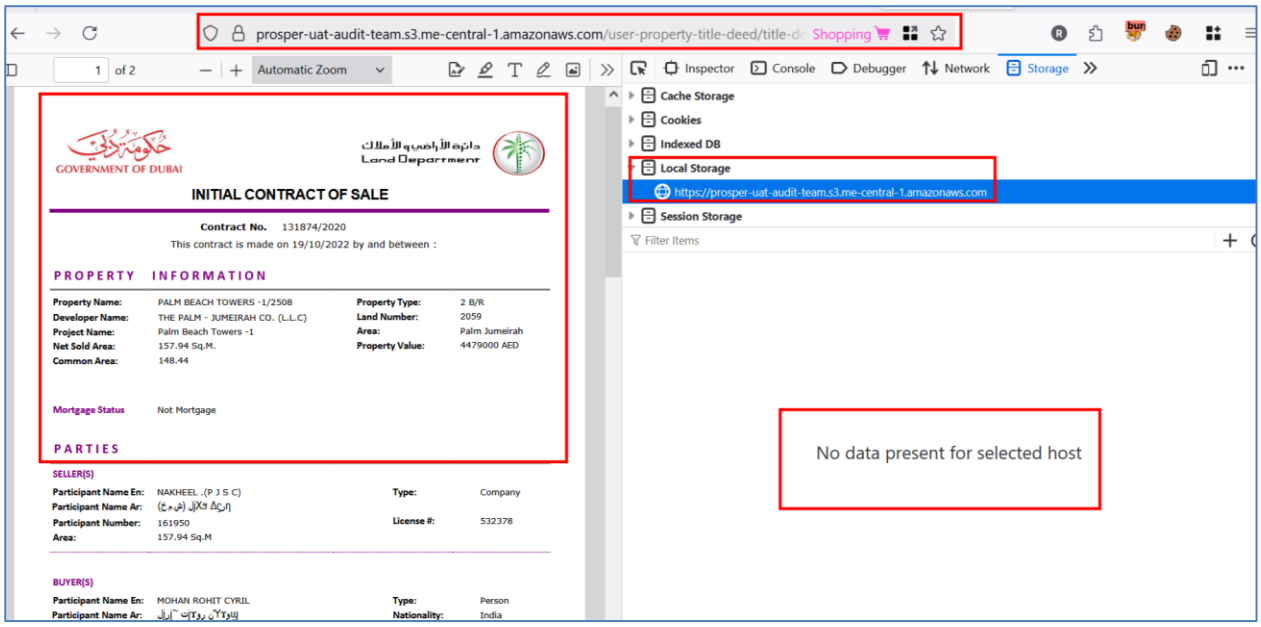
(e.g., Wayback Machine), leading to serious privacy and security breaches.

Recommendation: It is recommended to implement strict access controls by serving files through authenticated application endpoints. Use signed, time-limited URLs for file access and ensure cloud storage buckets are private with no public read permissions.

Proof Of Concept:

1. The passport and title deed documents are directly accessible via URL without any local storage token.





Note:

The above proof of concept describes sample scenarios where the sensitive documents are directly accessible via URL without any access control. However, similar issues can occur across multiple functionalities where file storage is available.

It is therefore recommended to apply the same remediation consistently across all file storage features to ensure comprehensive protection.

Some of the affected endpoints include, but may not be limited to:

- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/market-insight-report/{file_name}
- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/agency-user-profile-images/{file_name}
- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/developer-logos/{file_name}
- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/project-phase-noc-documents/{file_name}
- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/agency-logos/{file_name}

- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/agency-noc-files/{file_name}
- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/feedback-attachments/file/{file_name}
- https://prosper-uat-audit-team.s3.me-central-1.amazonaws.com/banner-images/{file_name}

2.8 Improper Authorization - Auth Token Misuse Across Roles

Severity: High

Issue Related to: Agency Portal

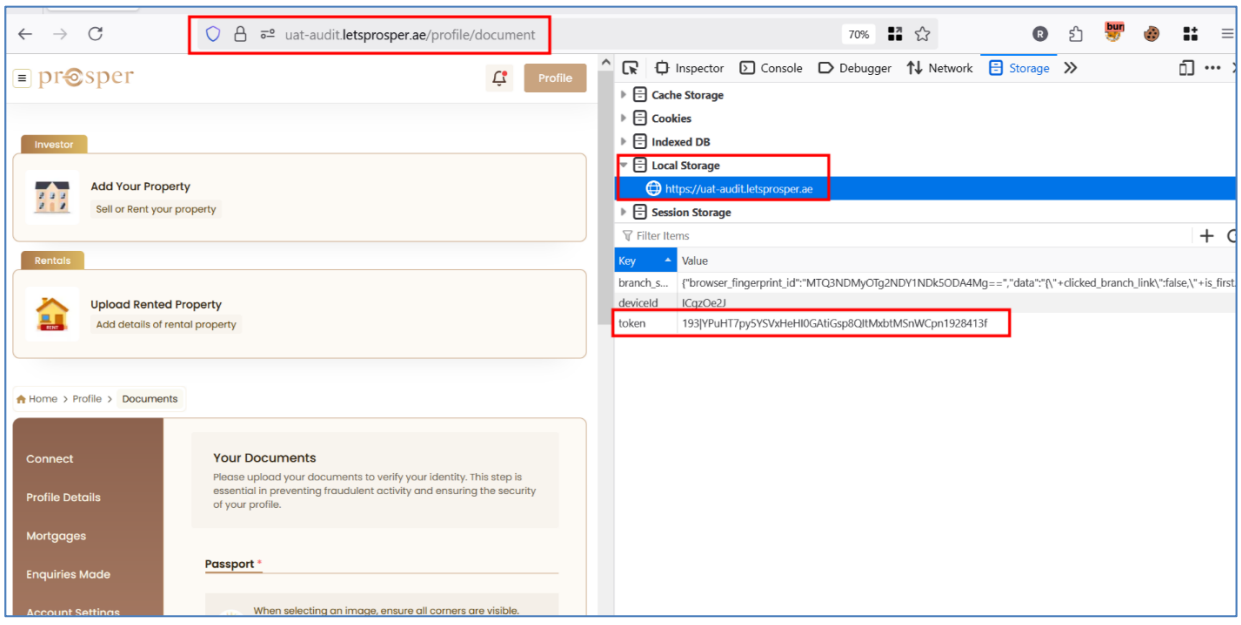
Issue Definition: In the User Portal, bearer tokens are used for authentication. However, a normal user's auth token can be used to perform actions reserved for agency super admins. By replacing the auth token in the request with a normal user's token, the system still processes sensitive agency admin operations without validating the user's role or privileges. This indicates a failure in proper authorization checks.

Analysis: An attacker can escalate their privileges by using a regular user's token to perform administrative operations, such as editing a developer project details. This leads to unauthorized access, data manipulation, and potentially full system compromise, severely impacting the application's security and trustworthiness.

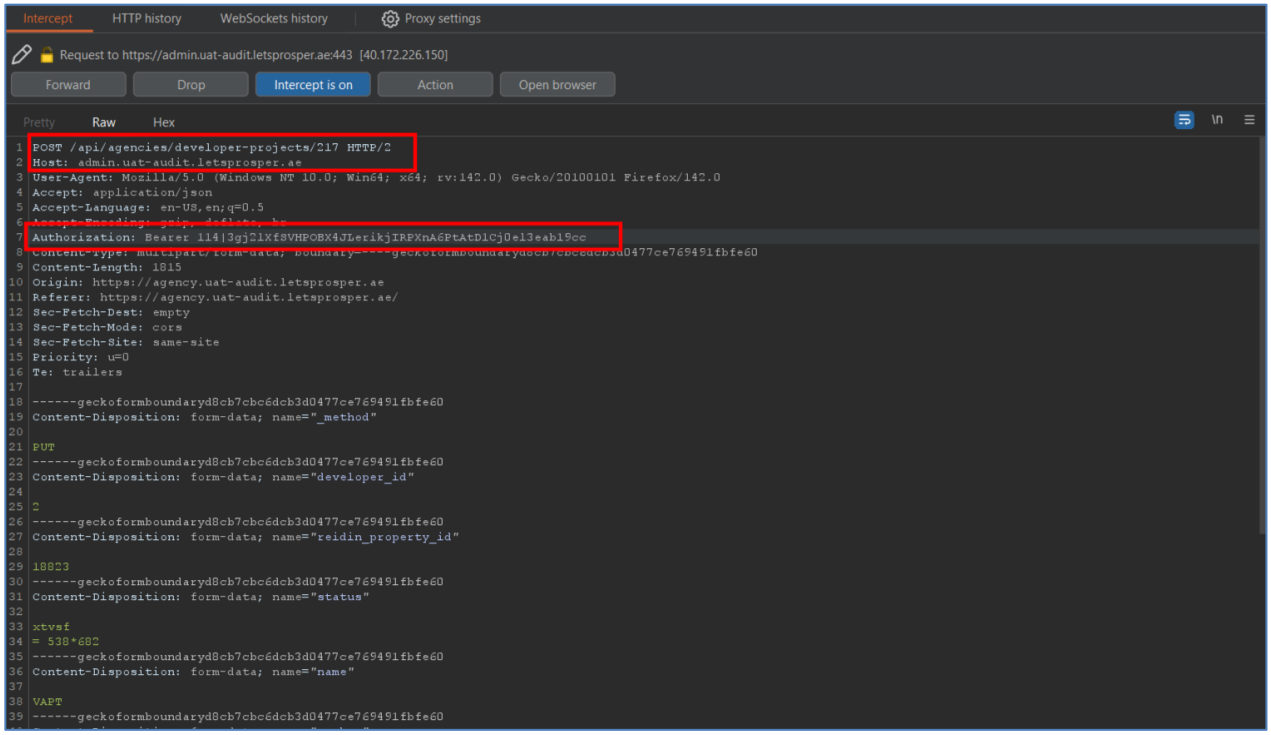
Remediation: It is recommended to enforce strict server-side authorization checks to validate that the authenticated user has the required role or permissions before performing any privileged operation. Ensure that each API endpoint verifies the user's role from the server-side session or token payload, and reject any unauthorized requests, regardless of the token used.

Proof Of Concept:

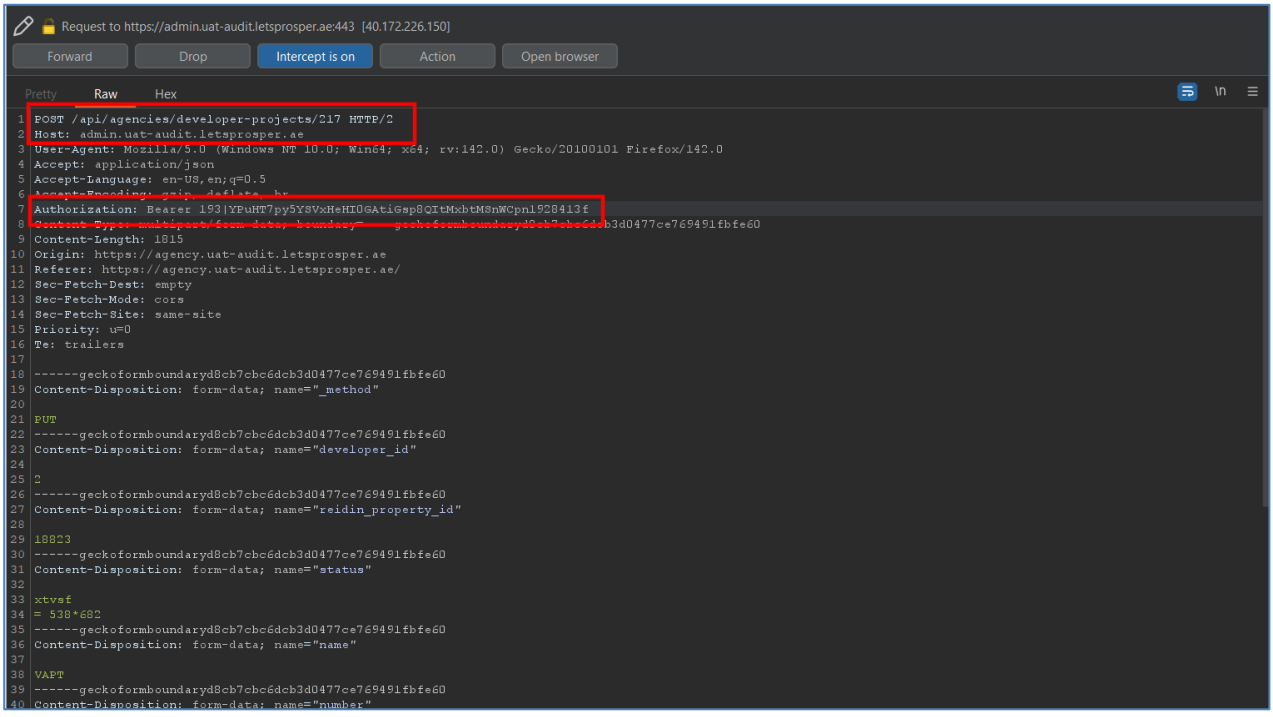
1. Login into any user account and copy the bearer token from the local storage.



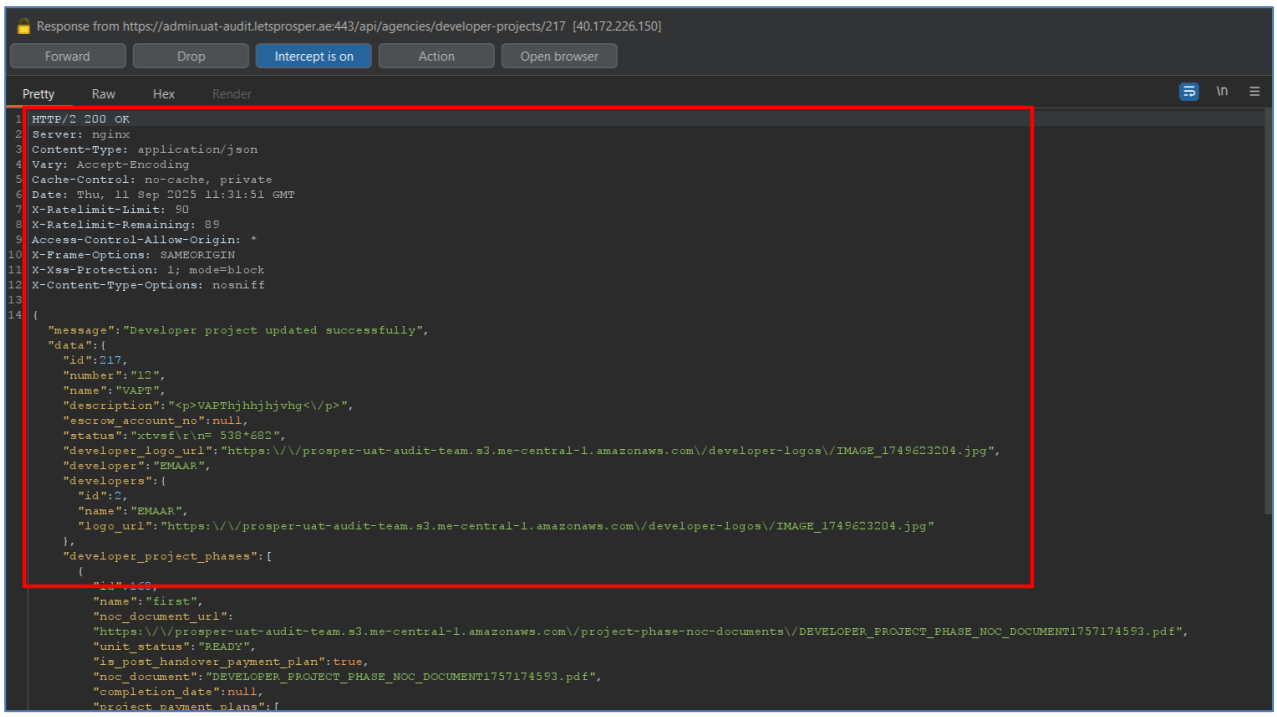
2. Login to the agency admin account and perform any action, in this case, editing any developer project.



3. Replace the admin token with the user token and forward the request.



4. The 200 Ok response is received from the server.



2.9 HTML Injection in Input Fields

Severity: Medium

Issue Related to: User Portal, Agency Portal, Admin Portal

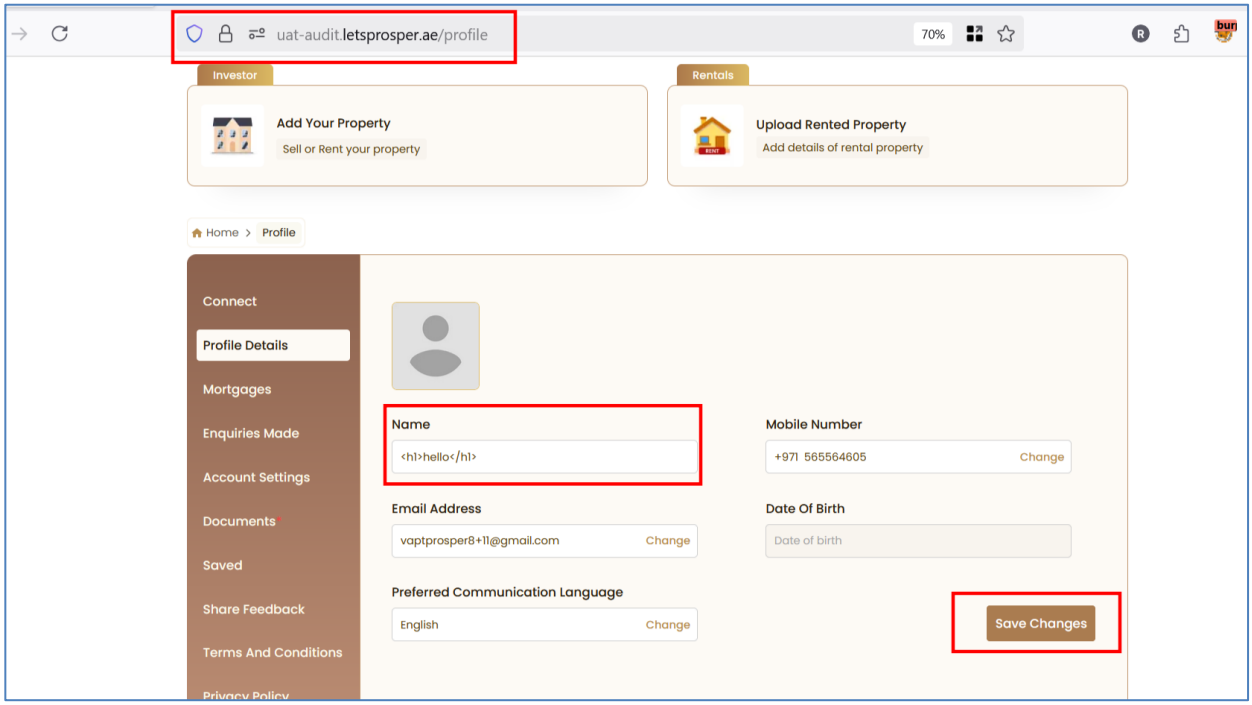
Issue Definition: The application's input fields do not properly sanitize user input, allowing HTML code to be injected. When such input is displayed later (e.g., in profile pages, comments, or descriptions), the injected HTML is rendered as part of the page, leading to an HTML injection vulnerability.

Analysis: Injected HTML can alter the page structure, inject unwanted content, or include malicious links, which can be used for phishing or UI manipulation. Although this is not full XSS (as scripts may not execute), it can confuse users, misrepresent content, and damage the application's integrity and trustworthiness.

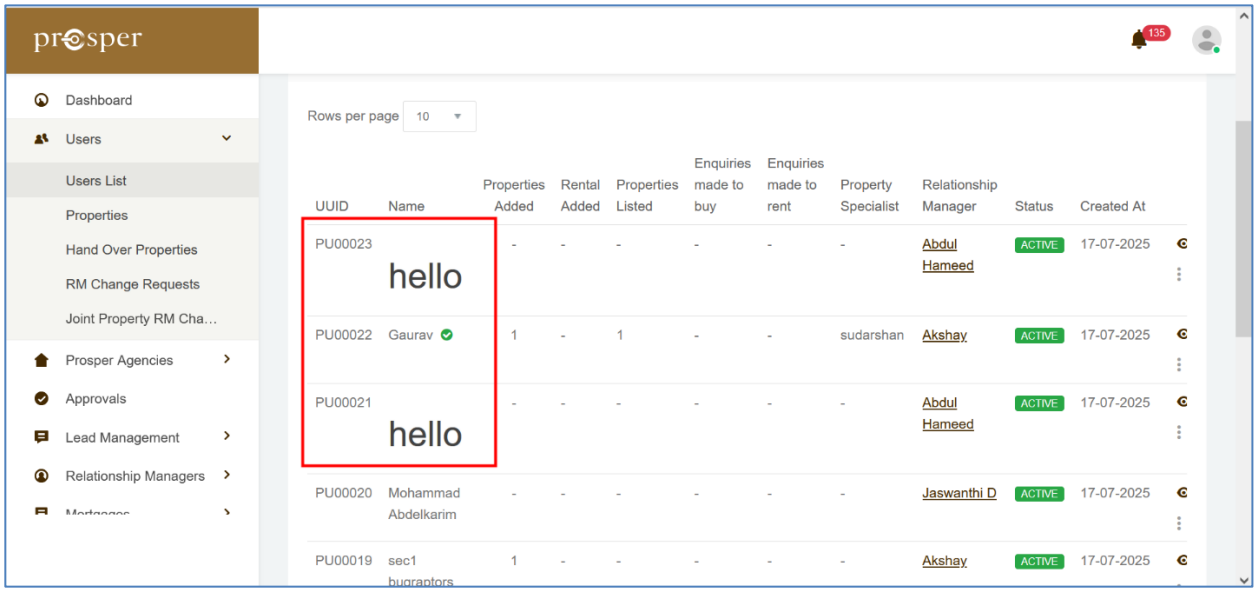
Recommendation: It is recommended to ensure that all user inputs are properly validated, sanitized, and encoded before being displayed on the application. Restrict input to only allow expected data formats and disallow any HTML or special characters unless explicitly required. Proper output encoding should be applied to prevent injected HTML from being rendered.

Proof Of Concept:

1. The user can submit an HTML code into the username parameter.



2. The username when rendered is displayed in the html tags.



Note:

The above proof of concept describes sample scenario where the input is rendered as HTML. However, similar issues can occur across multiple input fields.

It is therefore recommended to apply the same remediation consistently across the entirety of the application. Some of the affected endpoints include, but may not be limited to:

- https://admin.uat-audit.letsprosper.ae/api/agencies/developer-properties/{id}
- https://admin.uat-audit.letsprosper.ae/api/agencies/agency-users/{id}
- https://admin.uat-audit.letsprosper.ae/admin/relationship-managers/{id}
- https://admin.uat-audit.letsprosper.ae/api/users
- https://admin.uat-audit.letsprosper.ae/admin/share-feedbacks

2.10 Insecure Direct Object Manipulation – Missing OTP Verification for Email Update

Severity: Medium

Issue Related to: User Portal

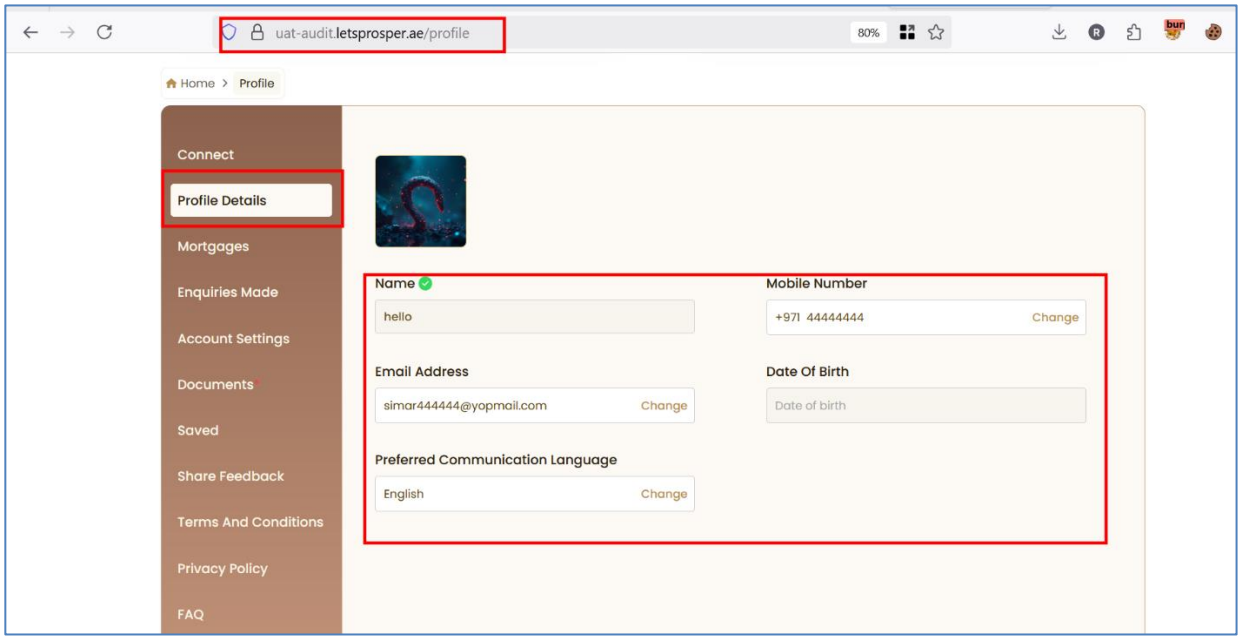
Issue Definition: In the user portal, OTP verification is only required when using the dedicated email and phone number change process. However, other profile fields (such as name, preferred communication language) can be updated without OTP. An attacker can intercept a regular profile update request and modify the email address or phone number parameter before it reaches the server. As a result, the attacker can change these parameters without any OTP verification, bypassing the intended security control.

Analysis: A logged-in user can change their own email address and phone number without proper OTP verification by manipulating the request. This weakens the integrity of account security, as the user could set an unauthorized email or phone number without validating ownership.

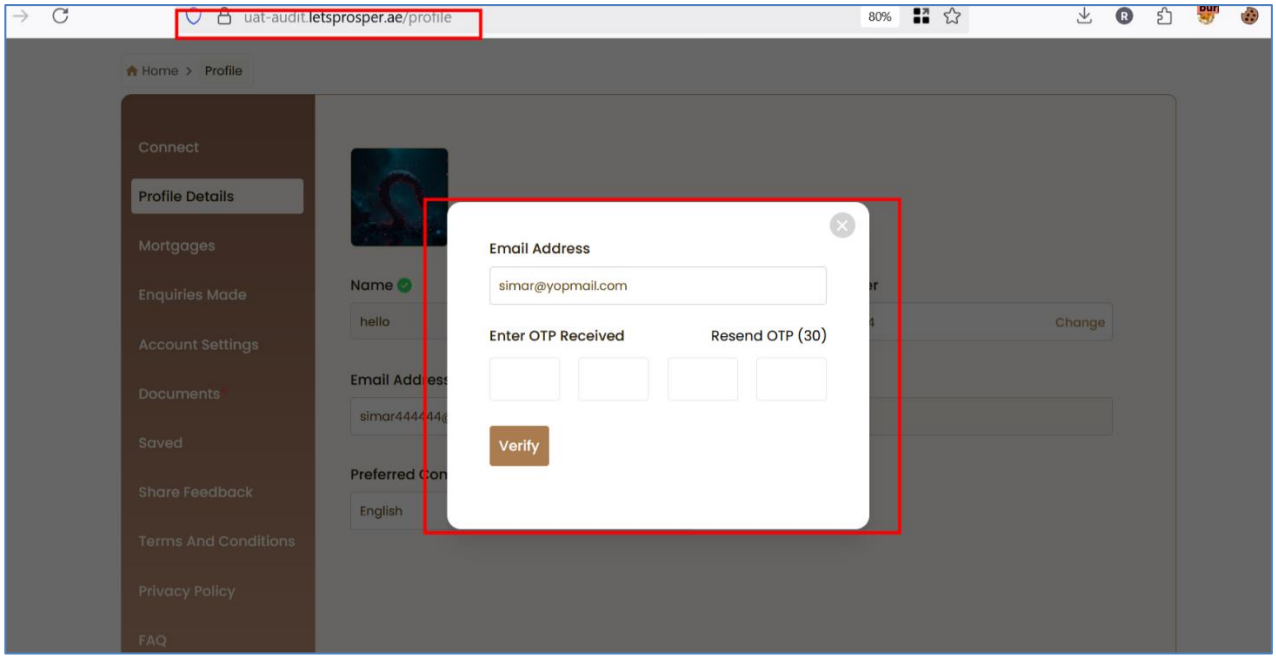
Recommendation: It is recommended to apply OTP verification in the same request where the email or phone number is being changed, and ensure the server validates the OTP before applying the update. Reject any profile update request that attempts to modify the email or phone number without providing a valid OTP, enforcing this check strictly on the server side.

Proof Of Concept:

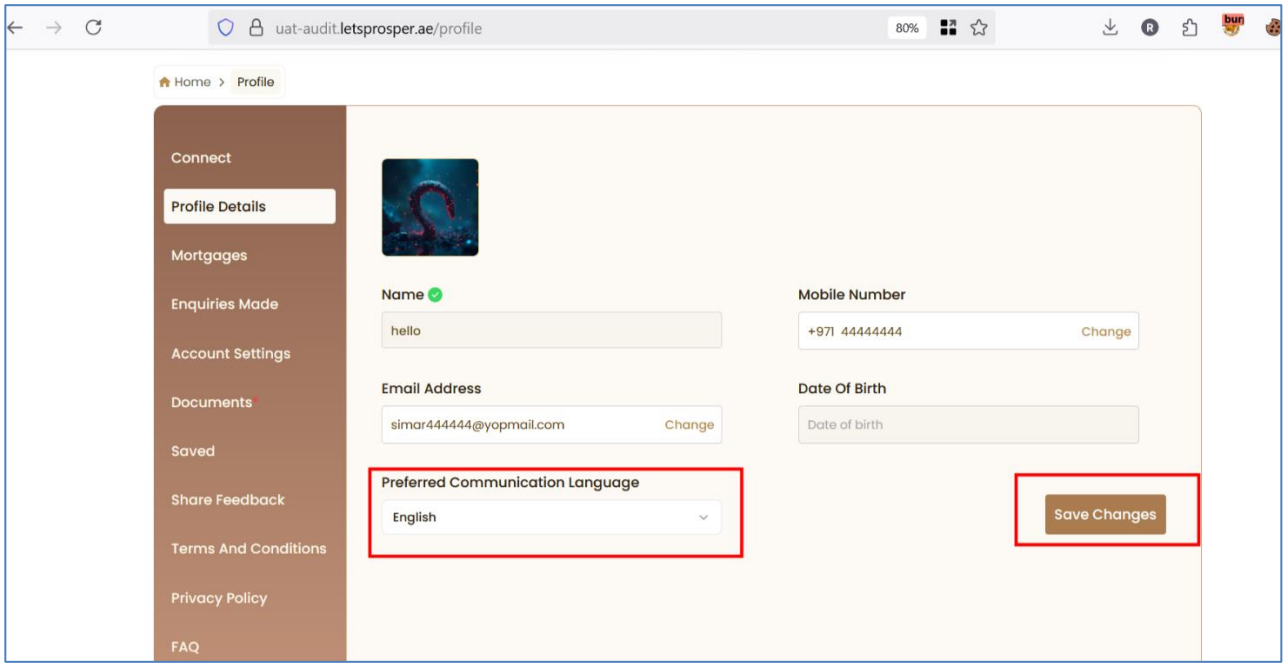
1. Try to change the email/phone number from the profile detail section.



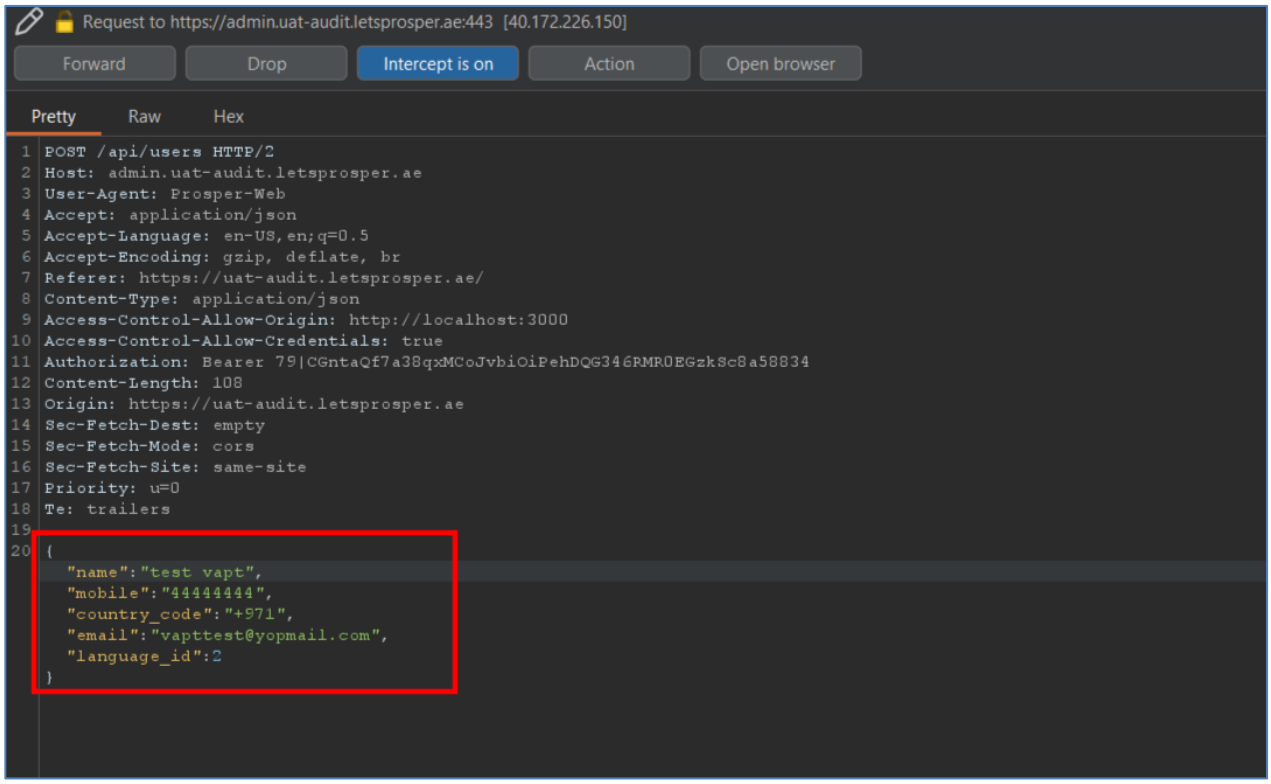
2. The OTP verification mechanism is enforced.



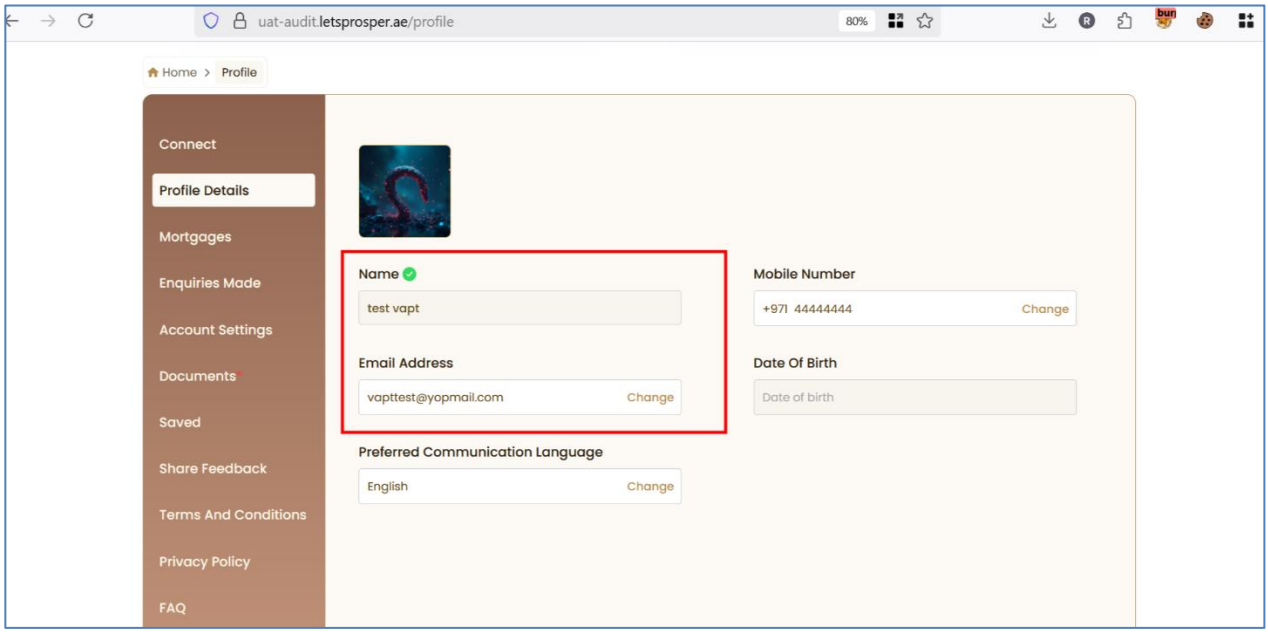
3. Make any other change in the profile and save the changes.



4. Intercept the request and change the email.



5. The email is changed without the verification of OTP. Similar process can be followed for Mobile number change mechanism.



2.11 Using Components with Known Vulnerabilities

Severity: Medium

Issue Related to: Admin Portal

Issue Definition: The application uses outdated versions of jQuery-ui (1.12.1), bootstrap (4.0.0-beta.2), jquery(3.2.1), underscore.js (1.9.1), jquery.datatables version 1.10.16, jquery-validation version 1.19.0 all of which have known security vulnerabilities. These vulnerabilities can be exploited by attackers to compromise the security of the application.

Analysis: Using outdated and vulnerable components exposes the application to a range of potential attacks, including Cross-Site Scripting (XSS) and other injection attacks

Remediation: It is recommended to update all outdated third-party libraries and dependencies to their latest stable versions. Regularly monitor and manage components using tools like npm audit or OWASP Dependency-Check to prevent known vulnerabilities. Avoid using deprecated or beta versions in production.

Proof Of Concept:

Request

```
GET /developers/global/vendor/bootstrap/bootstrap.js HTTP/2
Host: admin.uat-audit.letsprosper.ae
Cookie: XSRF-
TOKEN=eyJpdiI6ImRyZGh3blhkUFZiTTU1WXJtN3N4MVE9PSIsInZhbHVlIjoia0txZ3ByVnFSd0x4SDRjQmVmSTFpV29sRi9vSXNQOGRkQWhRekRqOGZKb0p2
VVZlSFF5dy9xWG5RcGhMTFQ3amg1RFJzVnYxMmdVUzRGN3RITUErNE15RFFYbXp2VDl3ZHh5aWJwNS9pWm8zNnN1VGRxbGlidk9QZVJGaUh5MGYiLCJt
YWMiOiJiOTBkZDMzOWQwZjAwYzFmMzM4YjM1MzY1YzNkM2M4ZGQxMTFlMzc2ZWFkMDljMzE5NGM2YmQxNjE4MjcwNGRkIiwidGFnIjoiIn0%3D;
prosper_session=eyJpdiI6IkZpZUFTVUNwUjhRcTNpeW9kL3N1VVE9PSIsInZhbHVlIjoiU01ibnVZVGUxQkwyazdhMGgzckpWTnNTRDJKUHJuT2g2ekZFSUpWO
GdXRTd3WmgxOEJYTkMzMVNVZU1nSXNPZnE2YWRzZmVTdVJiQkhDeCtaNUsrWjJxN1hMYVV4czBsUk9iWEV3blJJWUZ4bm42UzVET09rSGdybXR2T2Zub
HciLCJtYWMiOiI5NmZhYTM0MTY3MGNlMzI3OWRmYThjNDUxZTM3NjgwNTYwOWZmZTJiNWNkZjg4YmVlYmJmNThkOWY0MzY2YjQwliwidGFnIjoiIn0%3D
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:142.0) Gecko/20100101 Firefox/142.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://admin.uat-audit.letsprosper.ae/reset-password/KZO0QojNXdvG9alicvRRyYnjzKrQWO9THr1Cp6Ri1PkyamvRVfxitUdeCgQup1Su
Sec-Fetch-Dest: script
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
Te: trailers
```

Response

```
HTTP/2 200 OK
Server: nginx
Date: Sat, 06 Sep 2025 05:53:57 GMT
Content-Type: application/javascript; charset=utf-8
Last-Modified: Tue, 22 Jul 2025 12:02:00 GMT
Vary: Accept-Encoding
Etag: W/"687f7db8-1b3fa"
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff

/*!
* Bootstrap v4.0.0-beta.2 (https://getbootstrap.com)
* Copyright 2011-2017 The Bootstrap Authors (https://github.com/twbs/bootstrap/graphs/contributors)
* Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/L
...[SNIP]...
```

Security Audit Report

```
GET /admin/profile HTTP/2
Host: admin.uat-audit.letsprosper.ae
Cookie: XSRF-
TOKEN=eyJpdiI6IkVyY1IKSTQ4RXBuNUFJSkRxRGFWNnc9PSIsInZhbHVlIjoiQnFTR2NNMzZiUE03SzJMMzhmelM5dmtSalBVcklNdjJxUHo2c2tUaTFtdUI0T
3Zidy9BdzdHdTRld0c4ck55dVVuZm5TSFJaQVhBRDV5eEE0OUorMjFvbXpTRk95TkhOU0VlbWlyWlJrMHJhRFcxa0N2K0tDWDVjWFZ6NnVWOXkiLCJtYW
MiOiI2NjAzM2JkM2NhYWY1NzExYzcwMDM3N2VlNDcxZGUwMTkxNzY5NjJkMzZjZDVhODIzMzU5Mzc4MTNhNTllOGFjIiwidGFnIjoiIn0%3D;
prosper_session=eyJpdiI6InBwbVgzck9ZdzAyTUJmb3ptWitXWXc9PSIsInZhbHVlIjoieFBKbVBnV0g5elFqK0ZDNWIrdjVsd2tiU3ZmVEd3NWMwNGZPTTRlV0
xkOG1EMjlGdzluaDI1NFRrNjU2WjZQ0JCRW5VRnhoZjZ6VklsZUhibGJISG8rTXBnR2NWZXZLVkR0dVh5NlBYSmE1VFpxQXlzV0xZSjBJTkVicTlHV1QiLCJt
YWMiOiI1ZDlkYjc3ZDkyMTM1MTc0NWE4ZmJiOWU0MTY0ZDI5Y2FjNzBkYzNkNzUwMmZmMzk4YzEwYjFjMjM0ZTcxZmNkIiwidGFnIjoiIn0%3D
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:142.0) Gecko/20100101 Firefox/142.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://admin.uat-audit.letsprosper.ae/admin/dashboard
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
```

Response

```
expires=Sat, 06 Sep 2025 13:35:48 GMT; Max-Age=7200; path=/; secure; samesite=lax
Set-Cookie:
prosper_session=eyJpdiI6IkxGN1U1VTFJdyt1ekZjQWo2RTMwWmc9PSIsInZhbHVlIjoiS244RzlKWU4zSDE2Y2JuR1hJRERXTkR3QUtXWDVJWWkyMnFNM
2lrTkJZSmVvM0dNVkFPa0V5VnBucE0xVVQ2a3JXK3R0UktxeXo5TEdkV2Y3WUM1UkJjTXpKNmQzM1QwZ2VEcCtrZEtqdTBVbDFmNDJ0a3psQy8yVitubld
PTGkiLCJtYWMiOiJjMjRiMmJlZGIxNTAzNzUxMTAwNzg0ZDM0ZjhiOGNiMmUwZjE3NDEwNmMwNTJlYTM1NDc2NWY5OWY4NmE0N2ZkliwidGFnIjoiIn0%3
D; expires=Sat, 06 Sep 2025 13:35:48 GMT; Max-Age=7200; path=/; secure; httponly; samesite=lax
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff

<!DOCTYPE html>
<html class="no-js css-menubar" lang="en">

<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-
...[SNIP]...
</script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
  [SNIP]
```

Request

```
GET /developers/global/vendor/jquery/jquery.js HTTP/2
Host: admin.uat-audit.letsprosper.ae
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response

```
HTTP/2 200 OK
Server: nginx
Date: Sat, 06 Sep 2025 12:17:36 GMT
Content-Type: application/javascript; charset=utf-8
Last-Modified: Tue, 22 Jul 2025 12:02:00 GMT
Vary: Accept-Encoding
Etag: W/"687f7db8-41707"
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff

/*!
* jQuery JavaScript Library v3.2.1
* https://jquery.com/
*
* Includes Sizzle.js
* https://sizzlejs.com/
*
* Copyright JS Foundation and other contributors
* Released under the MIT license
```

Security Audit Report

Request

```
GET /phpmyadmin/doc/html/_static/underscore.js HTTP/2
Host: admin.uat-audit.letsprosper.ae
Cookie: pma_lang_https=en; phpMyAdmin_https=23obpvdvdmi3simvk04f0akapk; XSRF-
TOKEN=eyJpdiI6IjM1aWFTdUcxd3kvekwzc0N3aXN4bHc9PSIsInZhbHVlIjoiNCtaa1pORzBBSHh4VWpnUnJVOWFhOG8wY1E4OThud2dWQjM5aWd1VVZV
ak9tZlUvTWlrN1RFYTlianPT1FySS80YkhOMWZJeE9uR0lERUR2OFQ5NTRGa3Q2VCsyelZRRGxEYnlrWVJidXZkVHRMTHZlQWVyT055Y1V6TVRTbW4iL
CJtYWMiOiJiMmU4ZWJmNWQ2OTllNTQyM2Q0YTNmMzZmY2U0OWFmYzhmN2U1M2VhNTllNDQxMDJiMzU4NjM0YzNhNTkxOWJkliwidGFnIjoiIn0%3D;
prosper_session=eyJpdiI6IlIteFhWYzBBUnFRUnhhRHdnRmF6c1E9PSIsInZhbHVlIjoicTZVam9FU05HMXVwaWJtVEtBSmsveVpBbWc1ZGNtckFTaEJQbWF
XQllNOEU3RGJQeXR4UzUwbUl5L2h5WjFGZW8xZGhBTjZMUTJYaGduQW5icmxXRTdHcC95TmdMT1VTc2c0cHEvRURlSzRucVVRaGlKRTVsVEx5SHkyR
mNBQTQiLCJtYWMiOiIzN2U3OGIyZDMwYTg0MTcwOTUwZjQyNmUxYzEyZWNkNjAyNGMyZTNhOTg1YzMzMjQ3MTNkNGU3N2MyODVhOTImIiwidGFnIjoi
In0%3D
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:142.0) Gecko/20100101 Firefox/142.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://admin.uat-audit.letsprosper.ae/phpmyadmin/doc/html/index.html
Sec-Fetch-Dest: script
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
Priority: u=2
Te: trailers
```

Response

```
HTTP/2 200 OK
Server: nginx
Date: Sat, 06 Sep 2025 10:46:06 GMT
Content-Type: application/javascript; charset=utf-8
Last-Modified: Tue, 07 Feb 2023 10:56:36 GMT
Vary: Accept-Encoding
Etag: W/"63e22e64-e601"
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff

// Underscore.js 1.9.1
// http://underscorejs.org
// (c) 2009-2018 Jeremy Ashkenas, DocumentCloud and Investigative Reporters & Editors
// Underscore may be freely distributed under the MIT license.
```

Request

```
GET /vendor/datatables.net/jquery.dataTables.js HTTP/2
Host: admin.uat-audit.letsprosper.ae
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response

```
Server: nginx
Date: Sat, 06 Sep 2025 12:17:49 GMT
Content-Type: application/javascript; charset=utf-8
Last-Modified: Tue, 22 Jul 2025 12:02:01 GMT
Vary: Accept-Encoding
Etag: W/"687f7db9-6c4f1"
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff

/*! DataTables 1.10.16
* ..2008-2017 SpryMedia Ltd - datatables.net/license
*/

/**
* @summary DataTables
* @description Paginate, search and order HTML tables
* @version 1.10.16
* @file jquery.dataTabl
...[SNIP]...
```

Request

```
GET /vendor/jquery-validation/additional-methods.min.js HTTP/2
Host: admin.uat-audit.letsprosper.ae
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.63 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response

```
HTTP/2 200 OK
Server: nginx
Date: Sat, 06 Sep 2025 12:17:51 GMT
Content-Type: application/javascript; charset=utf-8
Last-Modified: Tue, 22 Jul 2025 12:02:01 GMT
Vary: Accept-Encoding
Etag: W/"687f7db9-5885"
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff

/*! jQuery Validation Plugin - v1.19.0 - 11/28/2018
* https://jqueryvalidation.org/
* Copyright (c) 2018 J..rn Zaefferer; Licensed MIT */
!function(a){"function"==typeof define&&define.amd?define(["jquery","./jquery.validate.min"],a):"ob
...[SNIP]...
```

The team has found 15 instances of this issue:

- https://admin.uat-audit.letsprosper.ae/
- https://admin.uat-audit.letsprosper.ae/admin/dashboard
- https://admin.uat-audit.letsprosper.ae/admin/forgot-password
- https://admin.uat-audit.letsprosper.ae/admin/login
- https://admin.uat-audit.letsprosper.ae/admin/profile
- https://admin.uat-audit.letsprosper.ae/admin/reset-password/{token}
- https://admin.uat-audit.letsprosper.ae/developers/global/vendor/bootstrap/bootstrap.js
- https://admin.uat-audit.letsprosper.ae/developers/global/vendor/jquery/jquery.js
- https://admin.uat-audit.letsprosper.ae/global/vendor/bootstrap/bootstrap.js
- https://admin.uat-audit.letsprosper.ae/global/vendor/jquery/jquery.js
- https://admin.uat-audit.letsprosper.ae/index.php/admin/login
- https://admin.uat-audit.letsprosper.ae/phpmyadmin/doc/html/_static/underscore.js
- https://admin.uat-audit.letsprosper.ae/vendor/datatables.net/jquery.dataTables.js
- https://admin.uat-audit.letsprosper.ae/vendor/jquery-validation/additional-methods.min.js
- https://admin.uat-audit.letsprosper.ae/vendor/jquery-validation/jquery.validate.min.js

2.12 OTP Flooding

Severity: Medium

Issue Related to: User Portal

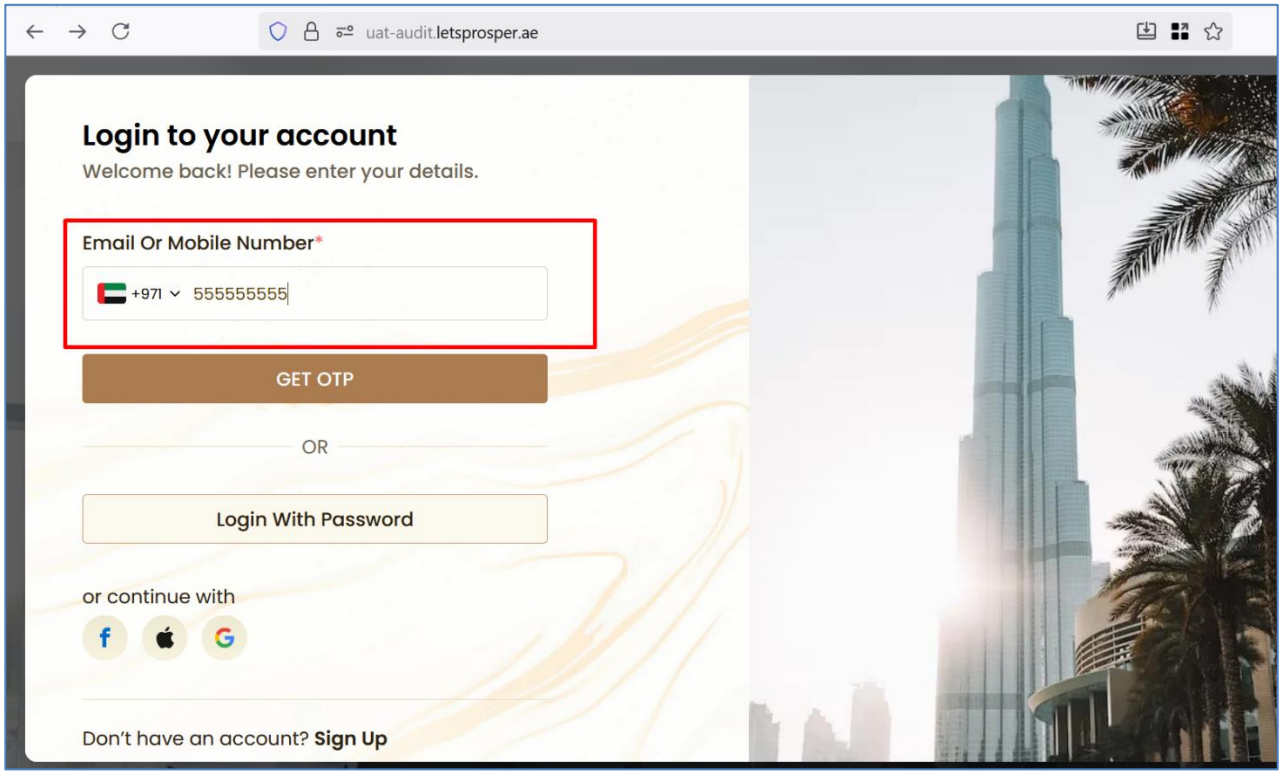
Issue Definition: In the user portal, the application sends OTPs for actions such as login, password reset, or other verification flows. However, there is no limit on the number of OTP requests a user can make. Additionally, there are no mechanisms like CAPTCHA or rate limiting to prevent abuse. This allows an attacker to repeatedly trigger OTPs for the same user, resulting in OTP flooding.

Analysis: The absence of rate limiting for OTP requests allows attackers to repeatedly trigger OTP messages for the same user, potentially overwhelming them and causing denial of service. This can be used to harass users, increase the risk of automated brute-force attacks, and lead to missed or ignored legitimate OTPs. Additionally, frequent OTP generation may result in higher operational costs if SMS or email delivery is charged per message, and can undermine the overall security and reliability of the verification process.

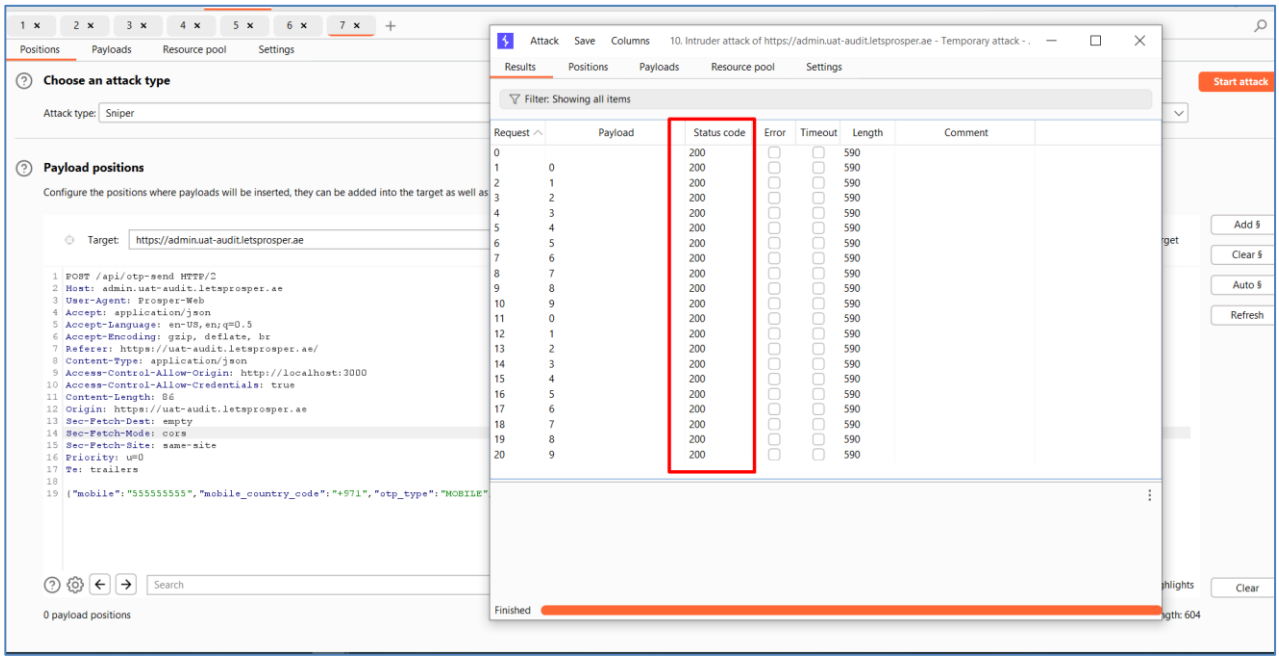
Remediation: It is recommended to implement server-side rate limiting for OTP requests, restricting the number of OTPs that can be generated per user within a defined time window. Introduce additional protections such as CAPTCHA or progressive delays after repeated requests to prevent automated abuse. Monitor and log OTP request patterns to detect suspicious activity.

Proof Of Concept:

1. Fetch an OTP for a user while logging in.



2. Capture the request and replay it number of times. 200OK response is received without any restriction or progressive delays.



2.13 Email Flooding

Severity: Medium

Issue Related to: Agency Portal, Admin Portal

Issue Definition: In both the **agency portal** and the **admin portal**, the forgot password functionality allows users to request password reset emails without any restriction on frequency or number of attempts. There is no rate limiting, CAPTCHA, or other anti-abuse mechanism in place, allowing an attacker to trigger an unlimited number of passwords reset emails for any account.

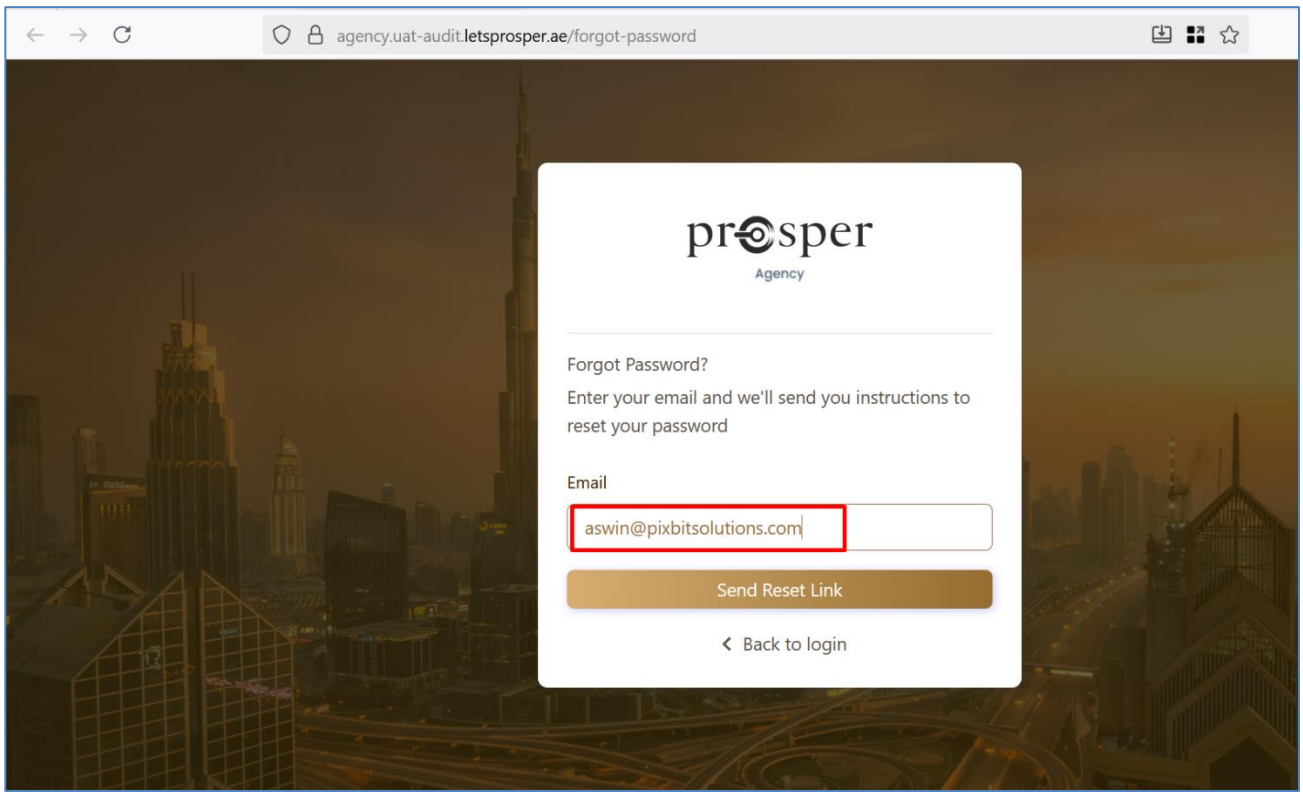
Analysis: The lack of restrictions on password reset requests enables attackers to flood a user's email inbox with an excessive number of password reset emails. This can lead to denial of service by overwhelming the user, causing confusion or preventing legitimate password resets. It may also be exploited to annoy or harass users, and potentially enable social engineering attacks. Additionally, excessive email generation may result in increased operational costs and reputation damage if the system is flagged for sending spam.

Remediation: It is recommended to implement rate limiting on the forgot password functionality to restrict the number of reset email requests per user within a specific time window. Introduce CAPTCHA or other bot protection mechanisms to prevent automated abuse.

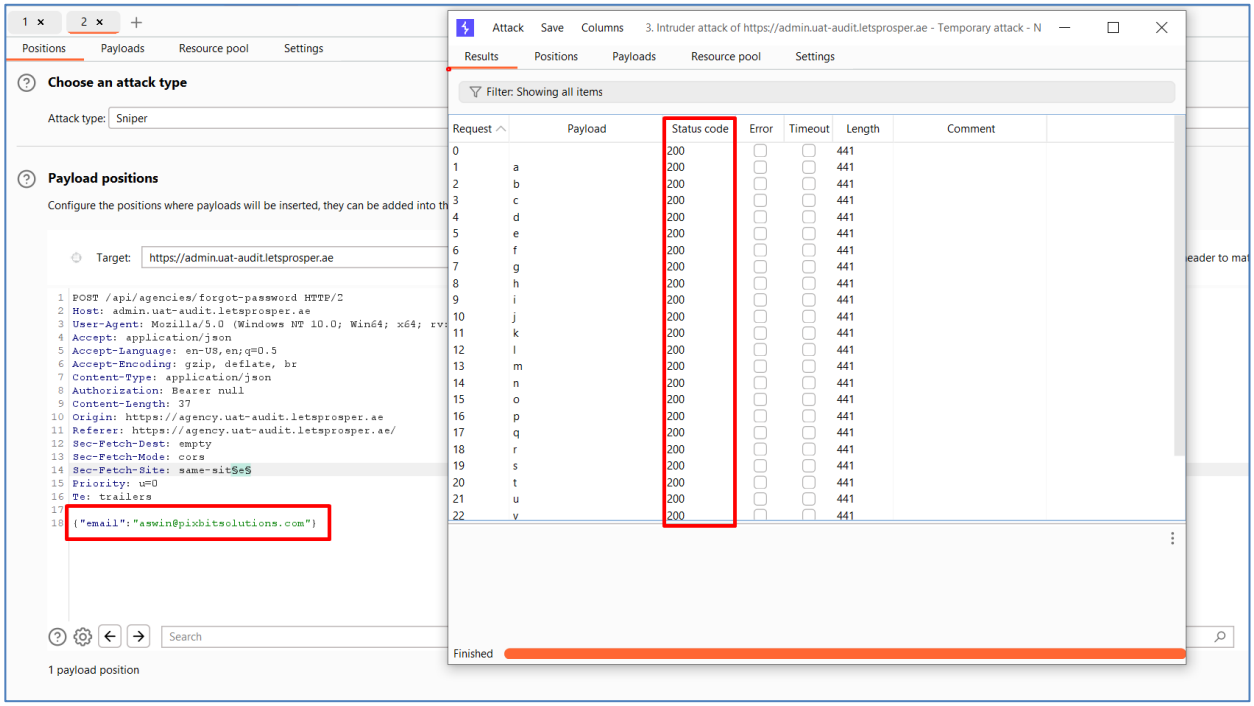
Proof Of Concept:

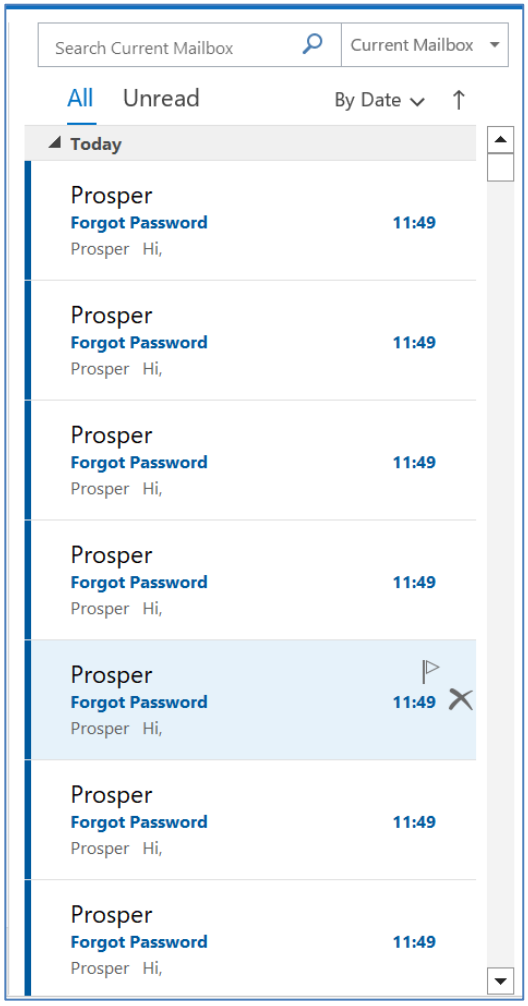
Security Audit Report

1. Fetch a password reset link from the forgot password functionality.



2. Capture the request and replay it multiple times. The emails are received without any restriction.





Note:

The above PoC describes the email flooding issue for the agency portal. The similar issue however is also present in the admin portal.

Affected Endpoints:

https://admin.uat-audit.letsprosper.ae/admin/forgot-password

https://admin.uat-audit.letsprosper.ae/api/agencies/forgot-password

2.14 Account Lockout Not Implemented

Severity: Medium

Issue Related to: User Portal, Agency Portal, Admin Portal

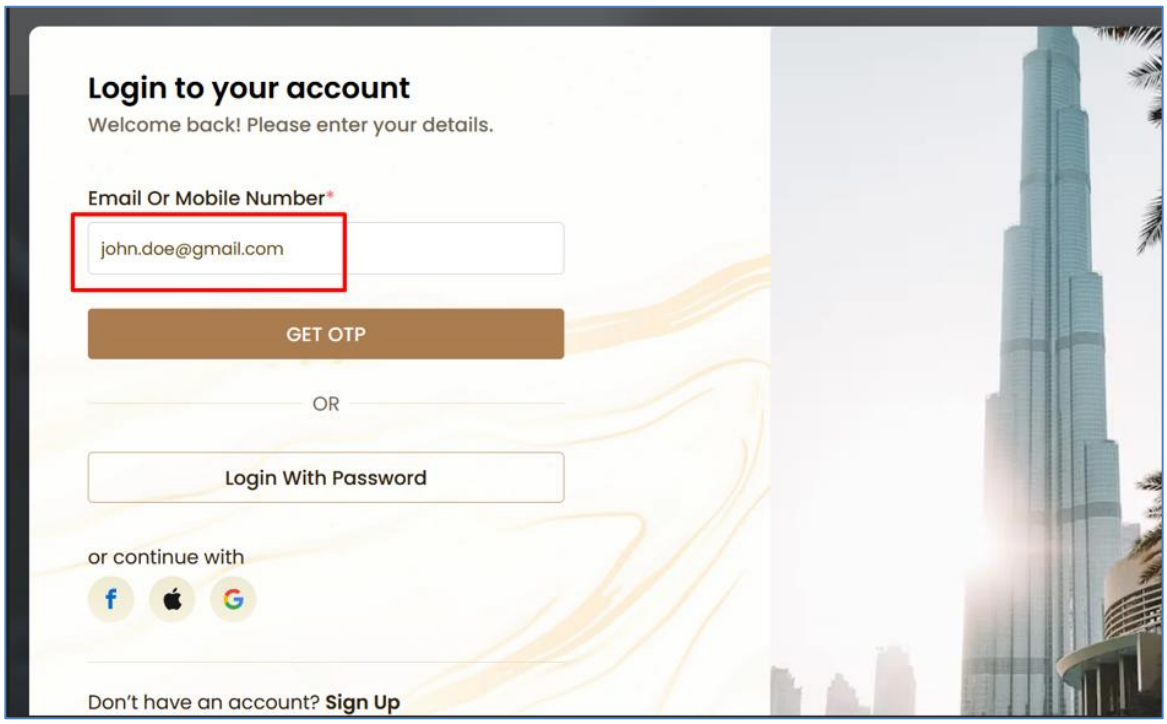
Issue Definition: In all three portals — **User Portal**, **Agency Portal**, and **Admin Portal** — there is no account lockout mechanism in place. Users can attempt unlimited login attempts using passwords (in the agency and admin portals) or password or OTP (in the user portal) without any restriction. Neither temporary account lockout nor IP-based blocking is implemented to prevent brute-force attacks.

Analysis: This weakness allows attackers to perform unlimited brute-force attacks to guess passwords or OTPs without facing any restriction, significantly increasing the risk of account takeover. It exposes user, agency, and admin accounts to unauthorized access, potentially leading to data breaches, privilege escalation, and compromise of sensitive system areas. Lack of IP blocking also allows attackers to continue attempts from the same source without being blocked, further increasing the risk.

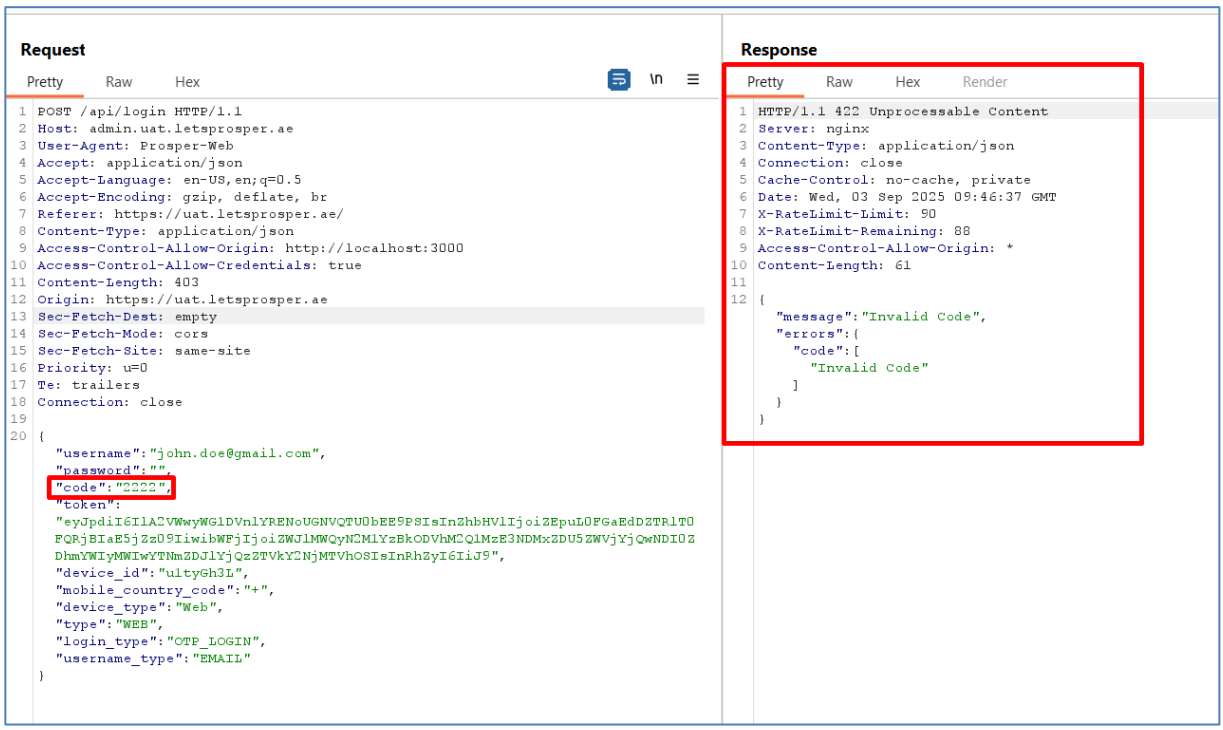
Remediation: It is recommended to implement an account lockout mechanism that temporarily locks the account after a defined number of failed login attempts. Introduce IP-based throttling or blocking after repeated failed attempts. Implement progressive delays or CAPTCHA challenges after multiple failures to prevent automated brute-force attacks. Ensure that lockout and blocking rules are enforced at the server level, not just the client side.

Proof Of Concept:

1. Fetch an OTP for the user to login into the application.



2. Try logging in using an incorrect OTP and observe that 422 error message is received.



Request

Pretty Raw Hex

```
POST /api/login HTTP/1.1
Host: admin.uat.letsprosper.ae
User-Agent: Prosper-Web
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://uat.letsprosper.ae/
Content-Type: application/json
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Credentials: true
Content-Length: 403
Origin: https://uat.letsprosper.ae
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Priority: u=0
Te: trailers
Connection: close

{
  "username":"john.doe@gmail.com",
  "password":"",
  "code":"2222",
  "token":
  "eyJpdiI6IlA2VWwyWGlDVnlYRENoUGNVQTU0bEE9PSIsInZhbHVlIjoiZEpuL0FGaEdDZTRlT0
  FQRjBIaE5jZz09IiwibWFjIjoiZWJlMWQyN2MlYzBkODVhM2QlMzE3NDMxZDU5ZWVjYjQwNDI0Z
  DhmYWIyMWIwYTNmZDJlYjQzZTVkY2NjMTVhOSIsInRhZyI6IiJ9",
  "device_id":"ultyGh3L",
  "mobile_country_code":"+",
  "device_type":"Web",
  "type":"WEB",
  "login_type":"OTP_LOGIN",
  "username_type":"EMAIL"
}
```

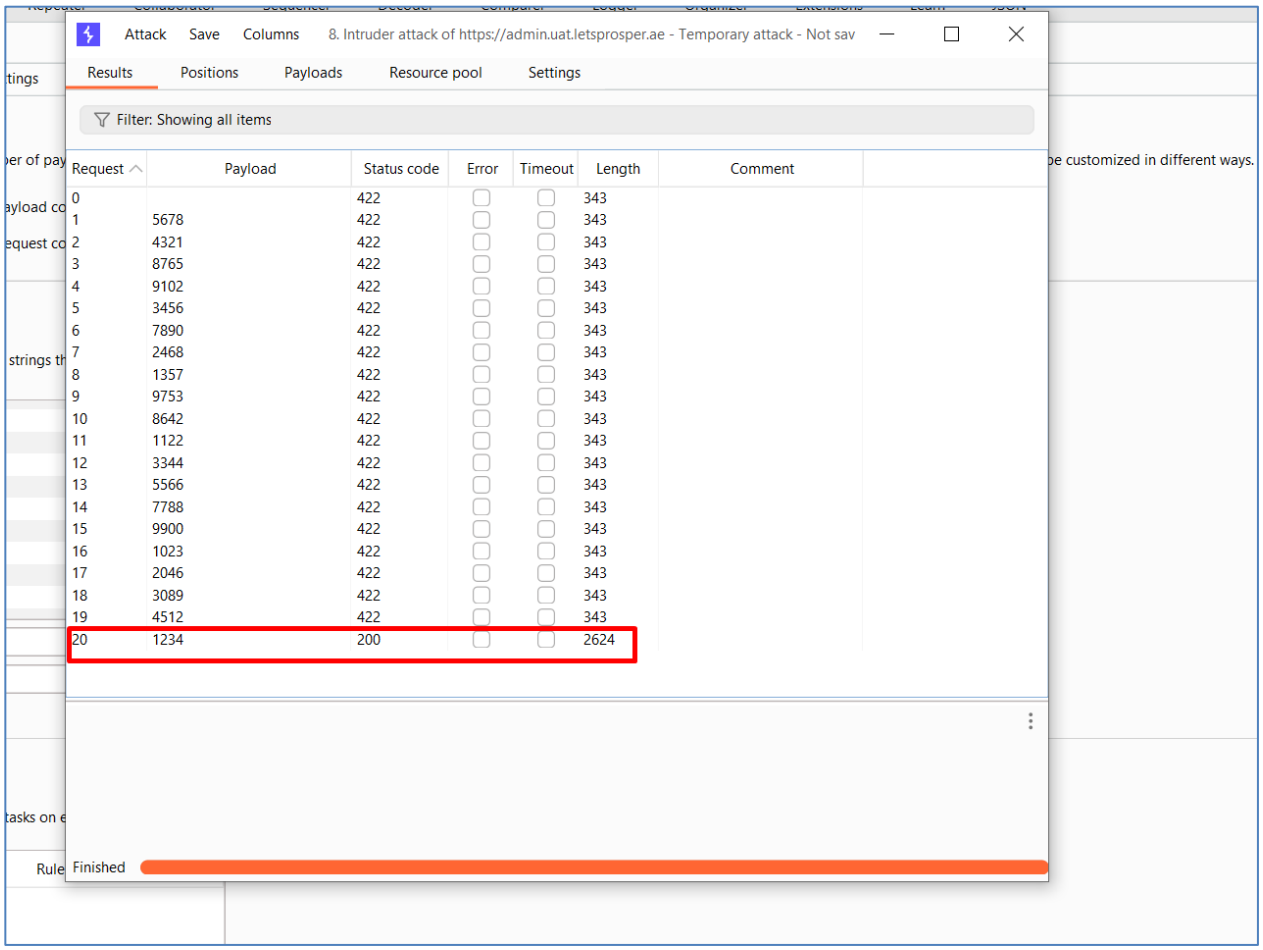
Response

Pretty Raw Hex Render

```
HTTP/1.1 422 Unprocessable Content
Server: nginx
Content-Type: application/json
Connection: close
Cache-Control: no-cache, private
Date: Wed, 03 Sep 2025 09:46:37 GMT
X-RateLimit-Limit: 90
X-RateLimit-Remaining: 88
Access-Control-Allow-Origin: *
Content-Length: 61

{
  "message":"Invalid Code",
  "errors":{
    "code":[
      "Invalid Code"
    ]
  }
}
```

3. Try to Bruteforce the OTP and observe that even after a number of unsuccessful attempts, 200 OK response is received after a valid OTP.



Attack Save Columns 8. Intruder attack of https://admin.uat.letsprosper.ae - Temporary attack - Not sav

Results Positions Payloads Resource pool Settings

Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
0		422			343	
1	5678	422			343	
2	4321	422			343	
3	8765	422			343	
4	9102	422			343	
5	3456	422			343	
6	7890	422			343	
7	2468	422			343	
8	1357	422			343	
9	9753	422			343	
10	8642	422			343	
11	1122	422			343	
12	3344	422			343	
13	5566	422			343	
14	7788	422			343	
15	9900	422			343	
16	1023	422			343	
17	2046	422			343	
18	3089	422			343	
19	4512	422			343	
20	1234	200			2624	

Finished

Note:

The above PoC describes the account lockout issue for the user portal. The similar issue however is also present in the agency and admin portal.

Affected Endpoints:

https://uat-audit.letsprosper.ae/login

https://agency.uat-audit.letsprosper.ae/login

https://admin.uat-audit.letsprosper.ae/login

2.15 Password Reset Link Reusable After First Use

Severity: Medium

Issue Related to: Agency Portal, Admin Portal

Issue Definition: In both the **Agency Portal** and the **Admin Portal**, the password reset functionality is insecurely implemented. The password reset link remains valid even after it has been used once. Moreover, when a new reset link is generated, the previously issued link does not expire and continues to work. Although the reset link expires after a certain time period, within its validity window, the same link can be reused multiple times to change the password repeatedly.

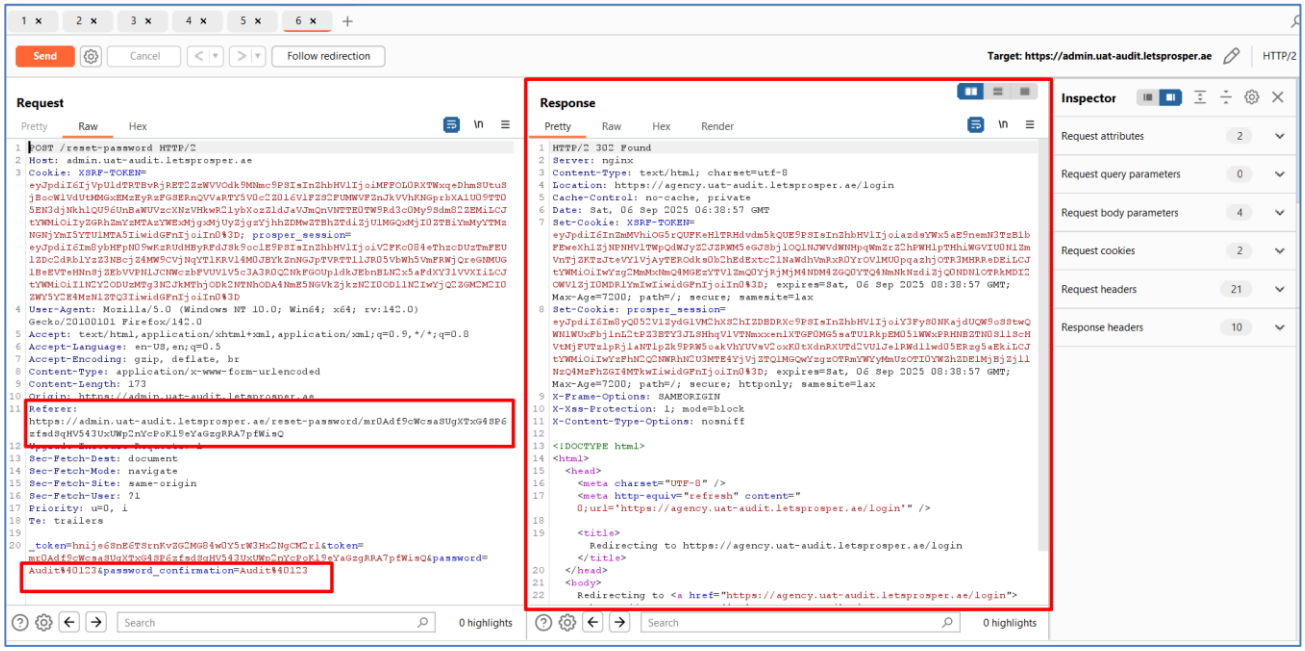
Analysis: This flaw allows attackers or malicious users to repeatedly use the same password reset link to change an account's password without further verification. As a result, an attacker who intercepts or obtains a valid reset link can lock out the legitimate user by continuously changing the password or gain persistent unauthorized access.

Remediation: It is recommended to implement single-use, time-limited password reset tokens that expire immediately after being used. Ensure that requesting a new password reset link invalidates all previously issued links. Tokens should be securely generated, unpredictable, and verified on every reset attempt, with proper server-side checks to prevent reuse or abuse.

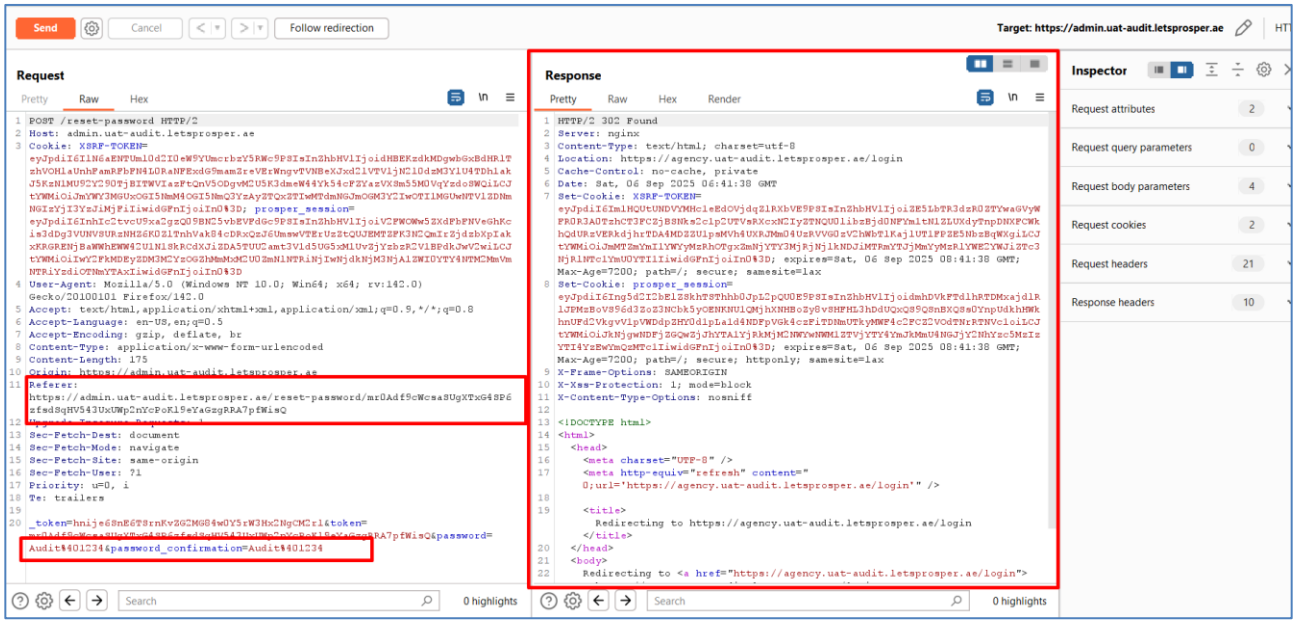
Proof Of Concept:

1. Use the reset link first time (here shown in the referrer header) and change the password successfully.

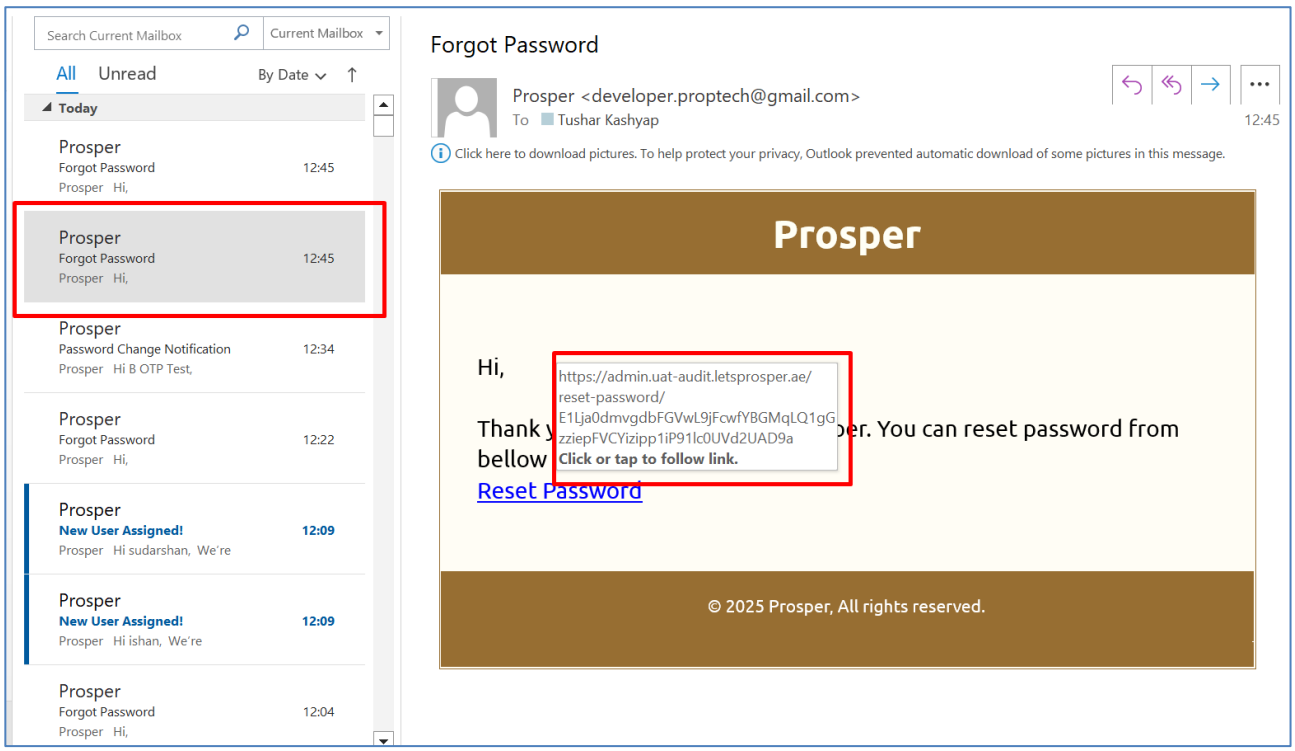
Security Audit Report



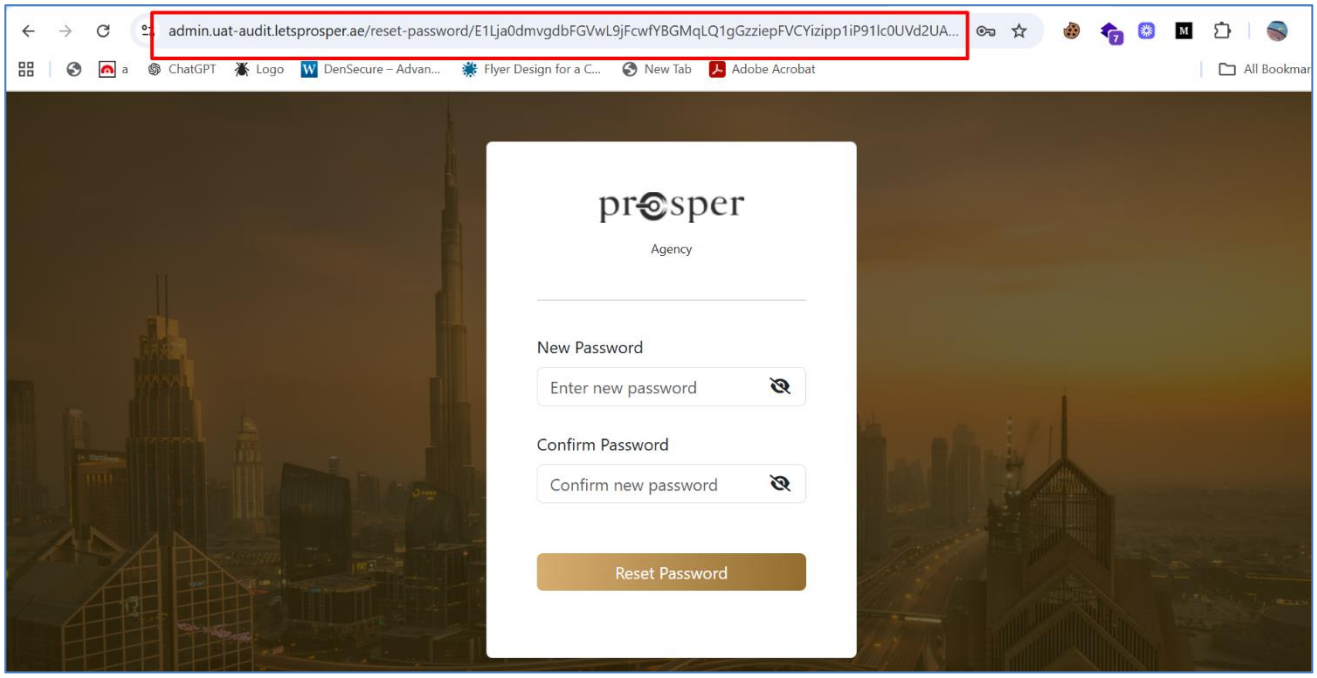
2. Reuse the same reset link and reset the password successfully again indicating that the link did not expire.



3. As visible, 2 reset links have been fetched for a mail id. Copy the older link and try reset the password.



4. As visible, the link was not expired upon fetching a new link.



2.16 Insecure Design – Replayable Publish Property Request

Severity: Medium

Issue Related to: Admin Portal

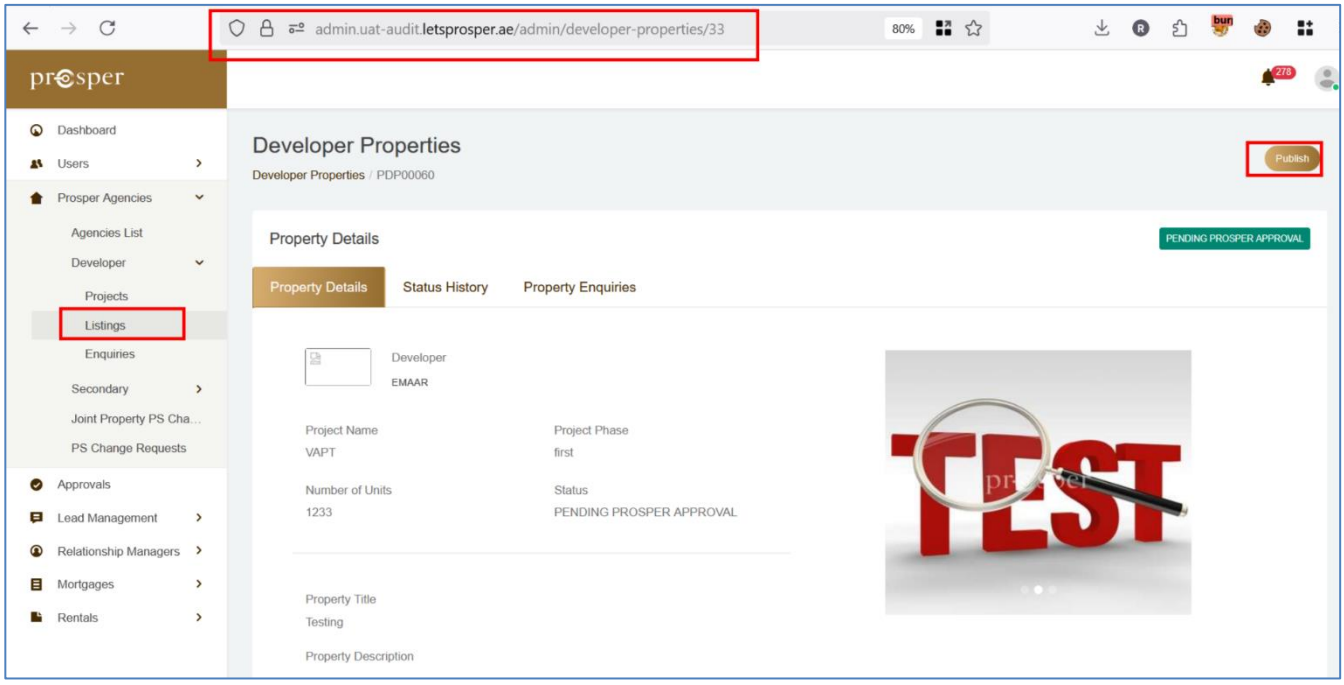
Issue Definition: In the Prosper Admin Portal, the admin has the authority to publish a property. The publish property request can be captured and replayed multiple times, leading to the same property being published repeatedly. The application does not implement any mechanism (such as nonces or unique request tokens) to prevent replay of the request, resulting in duplicate entries.

Analysis: Replay of the publish property request leads to the same property being published multiple times, causing data duplication, cluttered property listings, and inconsistent system behavior. This can confuse end users, and complicate property management.

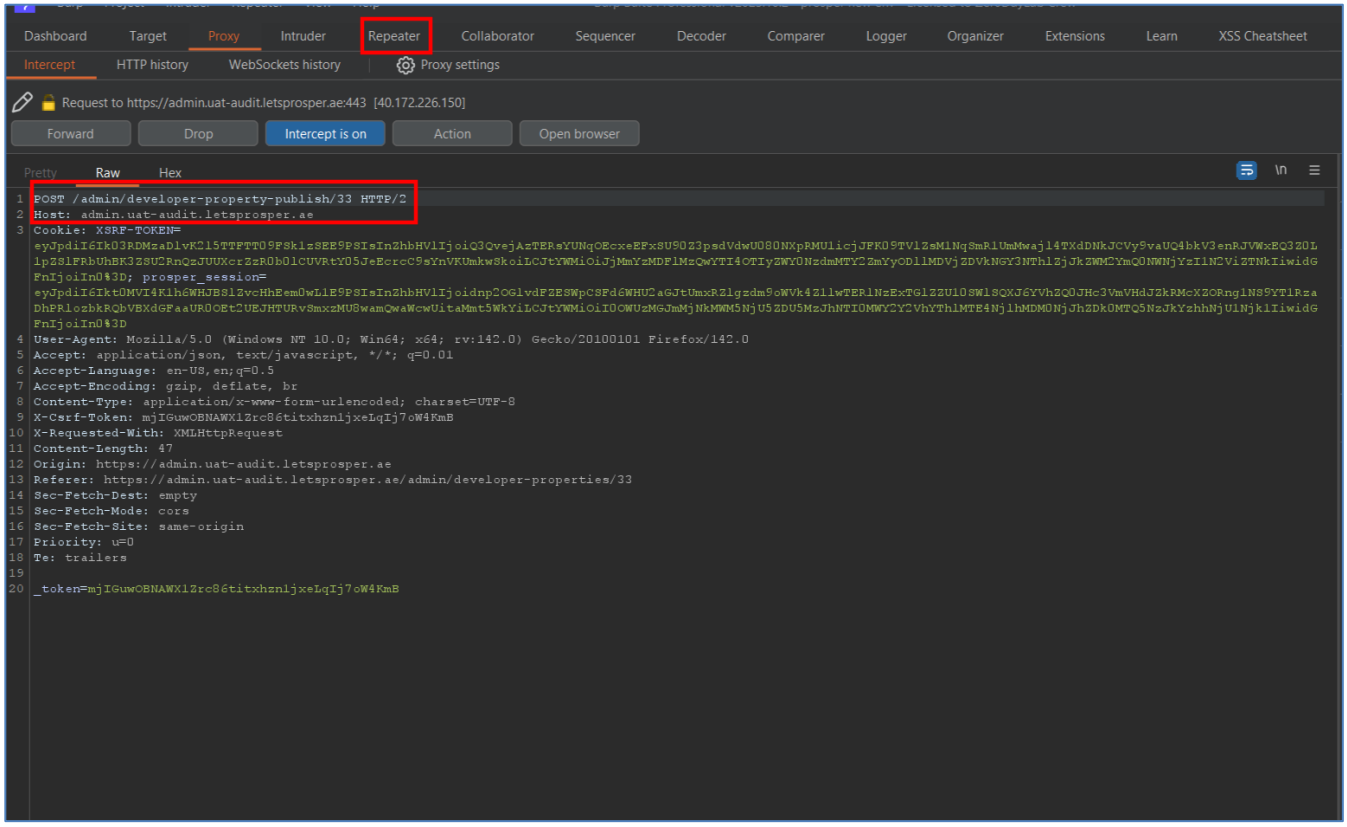
Remediation: It is recommended to ensure the system checks whether the property is already published before processing the publish request. Implement server-side validation to verify the property's status and prevent duplicate publishing by rejecting any request where the property is already listed as published.

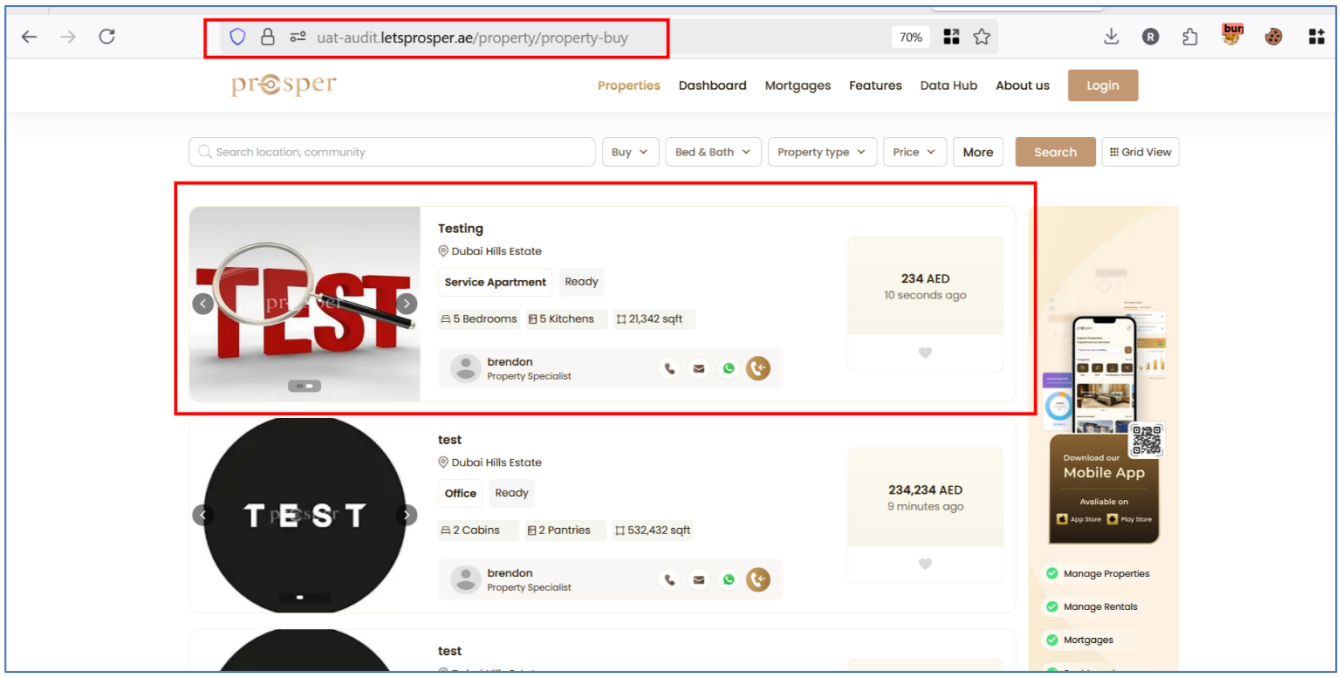
Proof Of Concept:

1. In the prosper admin portal, publish any property in the prosper market, capture and copy the request. The property gets published.

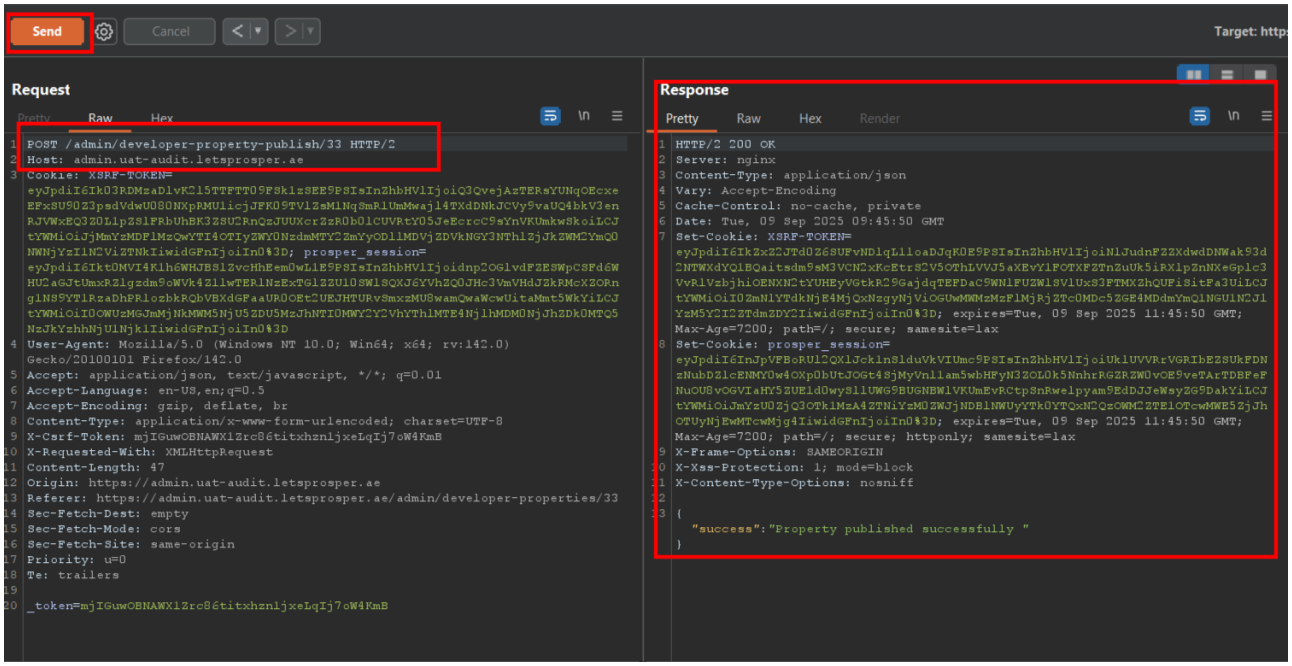


Security Audit Report

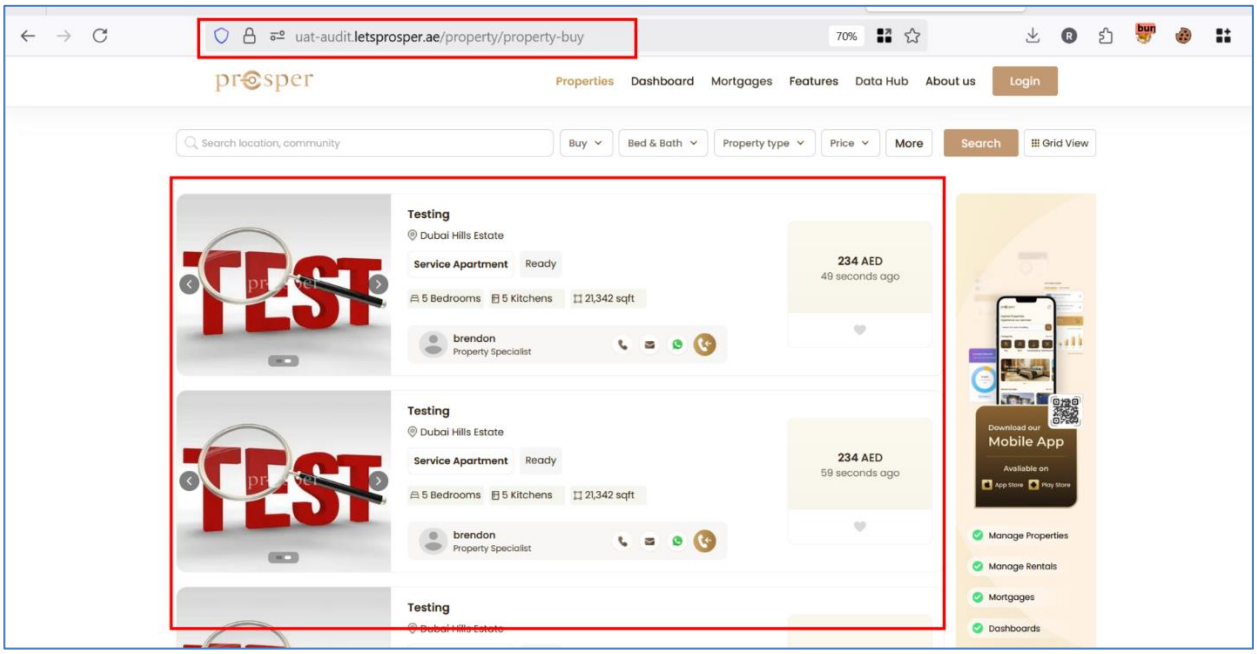




2. Replay the captured request again. 200 Ok response is received.



3. The same property gets published again.



2.17 Email / Phone Number Enumeration

Severity: Medium

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: In all three portals — User Portal, Agency Portal, and Admin Portal — the application reveals whether an email or phone number is registered during login and forgot password flows.

- In the **User Portal**, this occurs on both login and forgot password pages, for **email and phone number** fields.

- In the **Agency Portal**, it occurs on login and forgot password pages for the **email** field.
- In the **Admin Portal**, it occurs in the forgot password flow for the **email** field.

When an invalid or unregistered email/phone is entered, the application returns a different error message compared to a valid one, allowing an attacker to determine valid user identifiers.

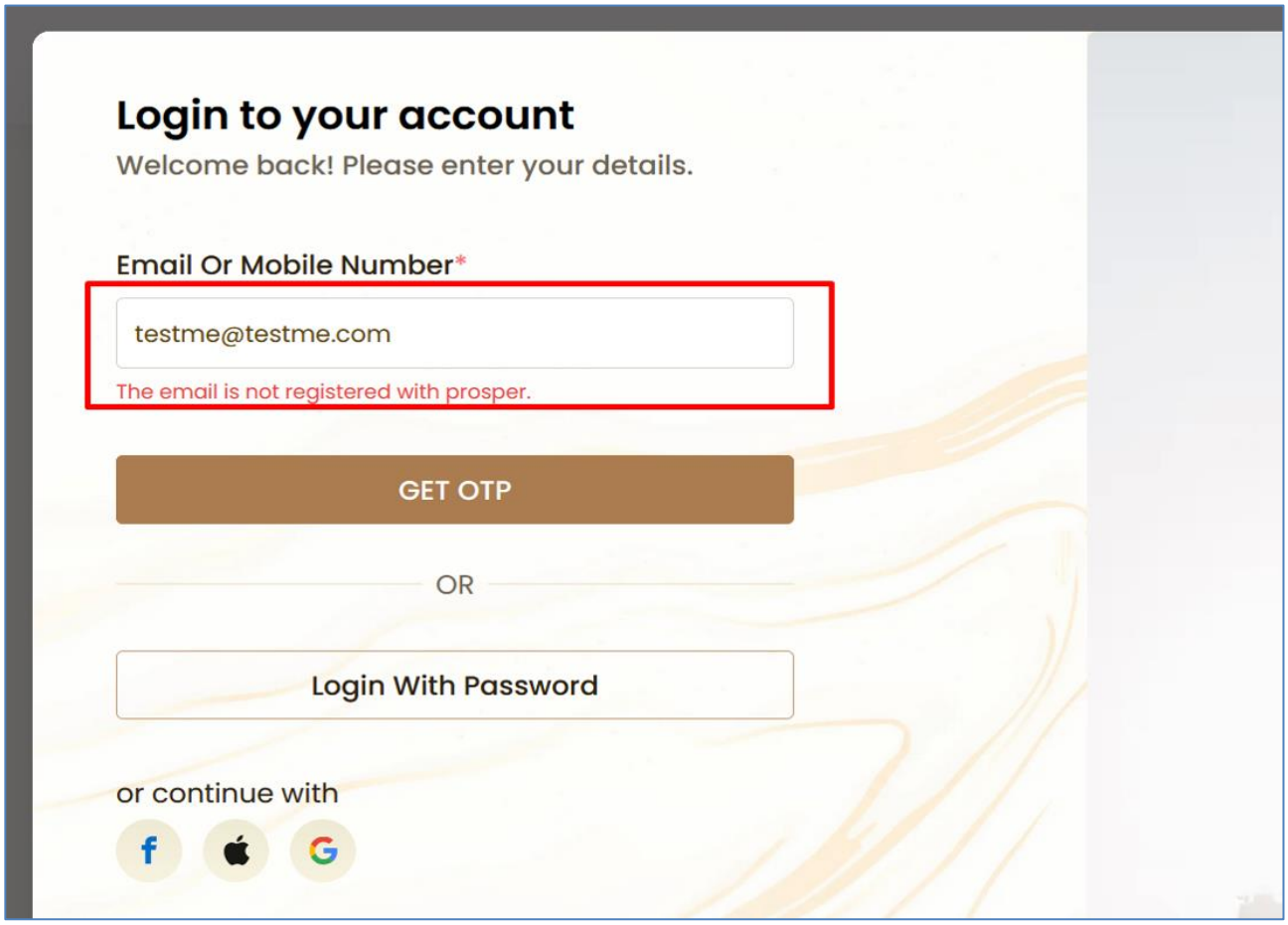
Analysis: This allows attackers to systematically enumerate valid email addresses and phone numbers by observing the application's responses. Enumerating valid accounts makes targeted attacks like credential stuffing, phishing, or brute-force attacks easier, increasing the risk of account compromise and unauthorized access.

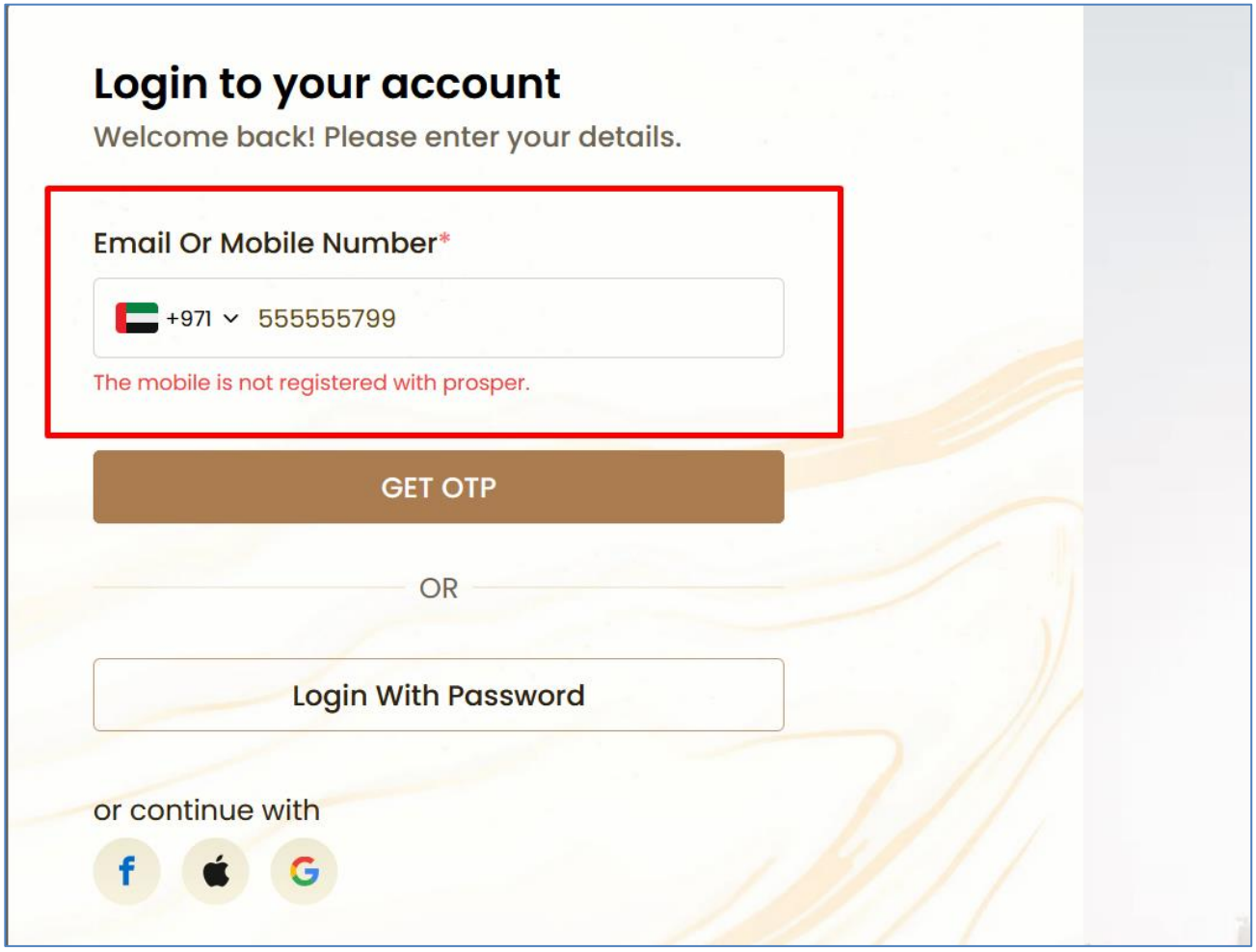
Remediation: It is recommended to standardize error messages for login and forgot password flows, providing generic responses (e.g., "If the provided credentials are correct, you will receive an email") regardless of whether the email or phone number exists in the system. Implement consistent response times and avoid disclosing the existence of user accounts in any part of the application.

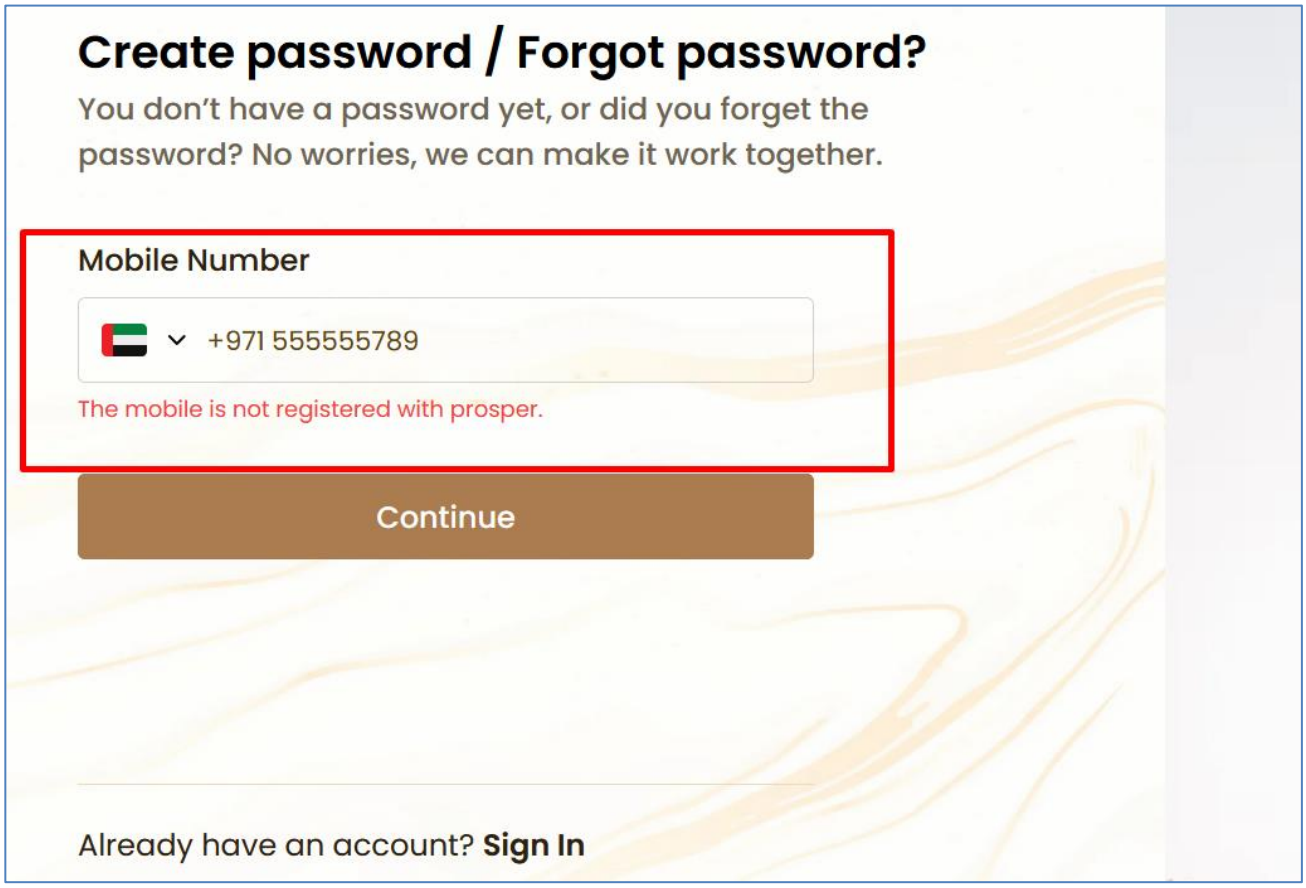
Proof Of Concept:

1. The below screenshots highlight the specific error messages received when the email or phone number is not registered in the prosper system which in turn leads to their enumeration.

   User Portal:

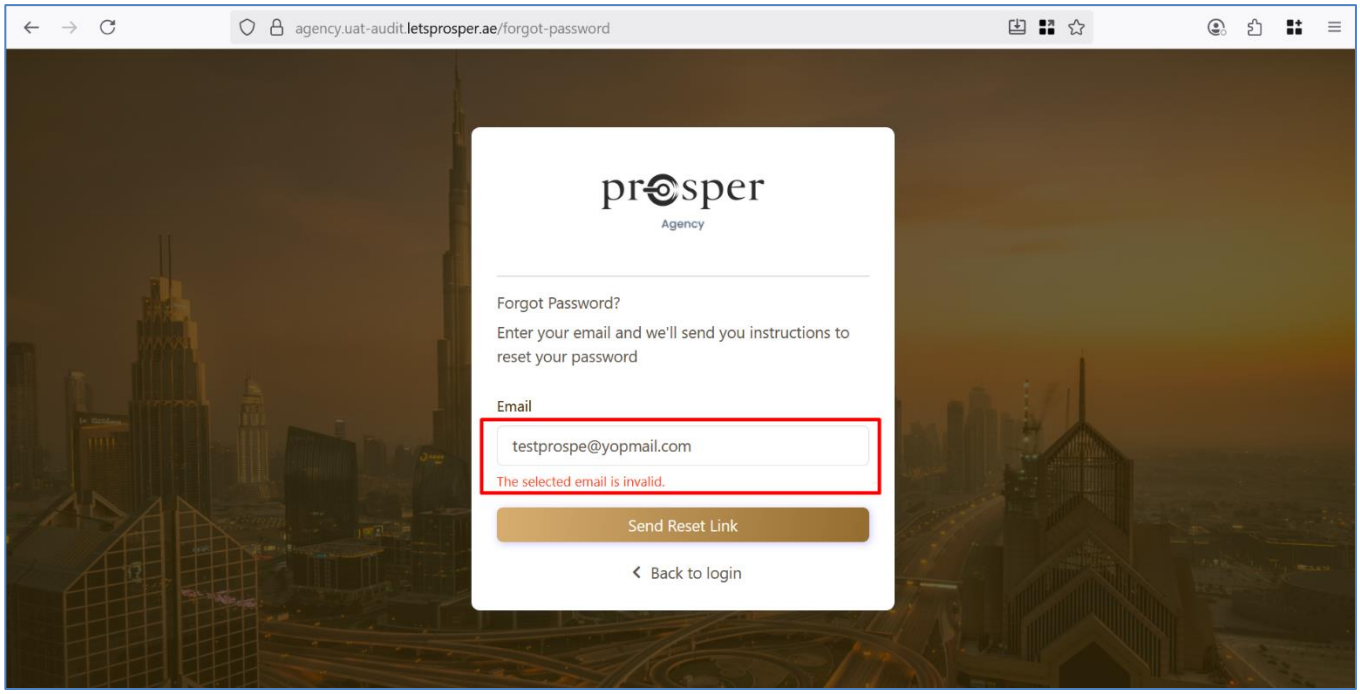


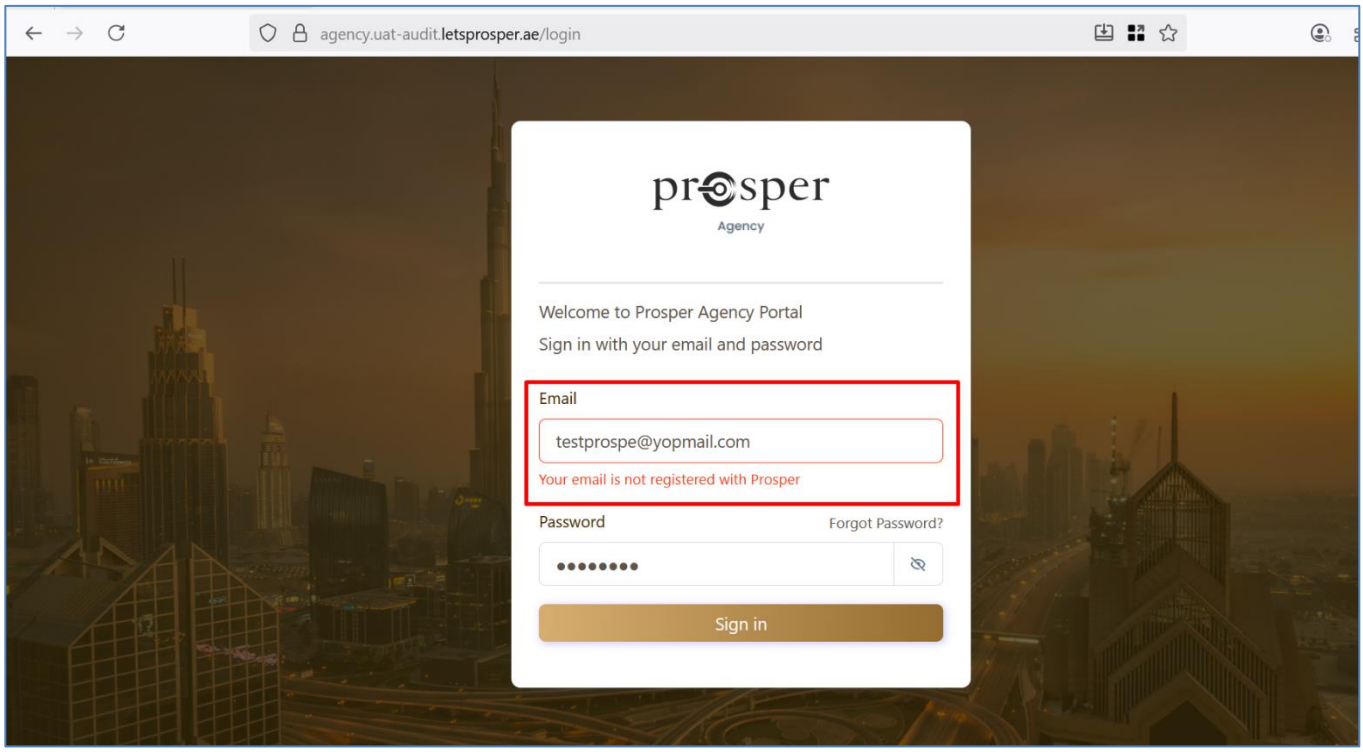

Login to your account
Welcome back! Please enter your details.
Email Or Mobile Number*
+971
555555799
The mobile is not registered with prosper.
GET OTP
OR
Login With Password
or continue with


Create password / Forgot password?
You don't have a password yet, or did you forget the password? No worries, we can make it work together.
Mobile Number
+971 555555789
The mobile is not registered with prosper.
Continue
Already have an account? Sign In

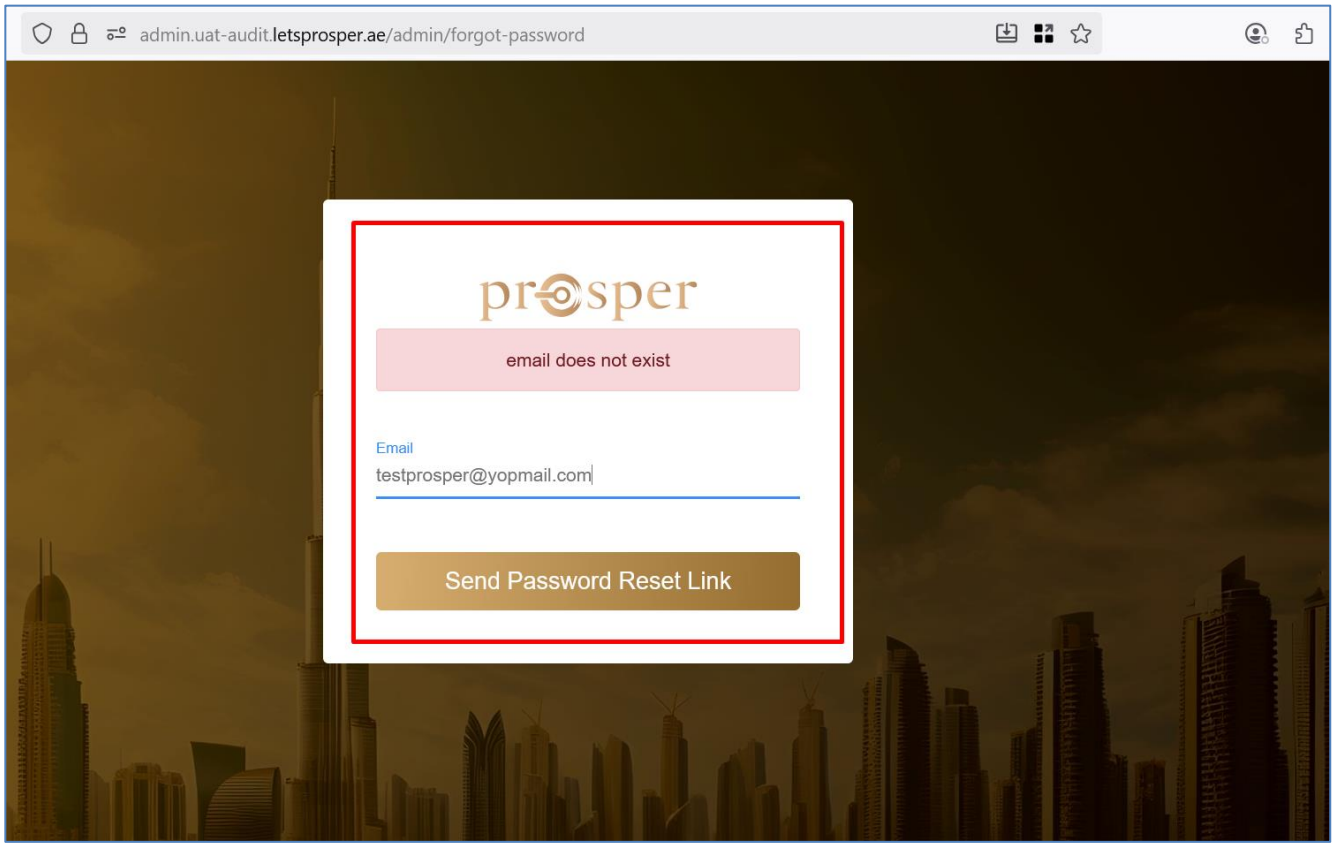
Security Audit Report

Agency Portal:





Admin Portal:



2.18 Clickjacking Attack

Severity: Medium

Issue Related to: User Portal, Agency Portal

Issue Definition: The User and Agency portals do not implement the X-Frame-Options header, making them vulnerable to clickjacking attacks. This allows an attacker to embed the application within an iframe on a malicious site and trick users into performing unintended actions.

Analysis: Without proper frame options, an attacker can craft malicious pages that embed the portals in iframes and overlay deceptive UI elements. This can lead to unintentional user actions such as changing account settings, submitting forms, or performing transactions without the user's explicit consent, impacting application integrity and user trust.

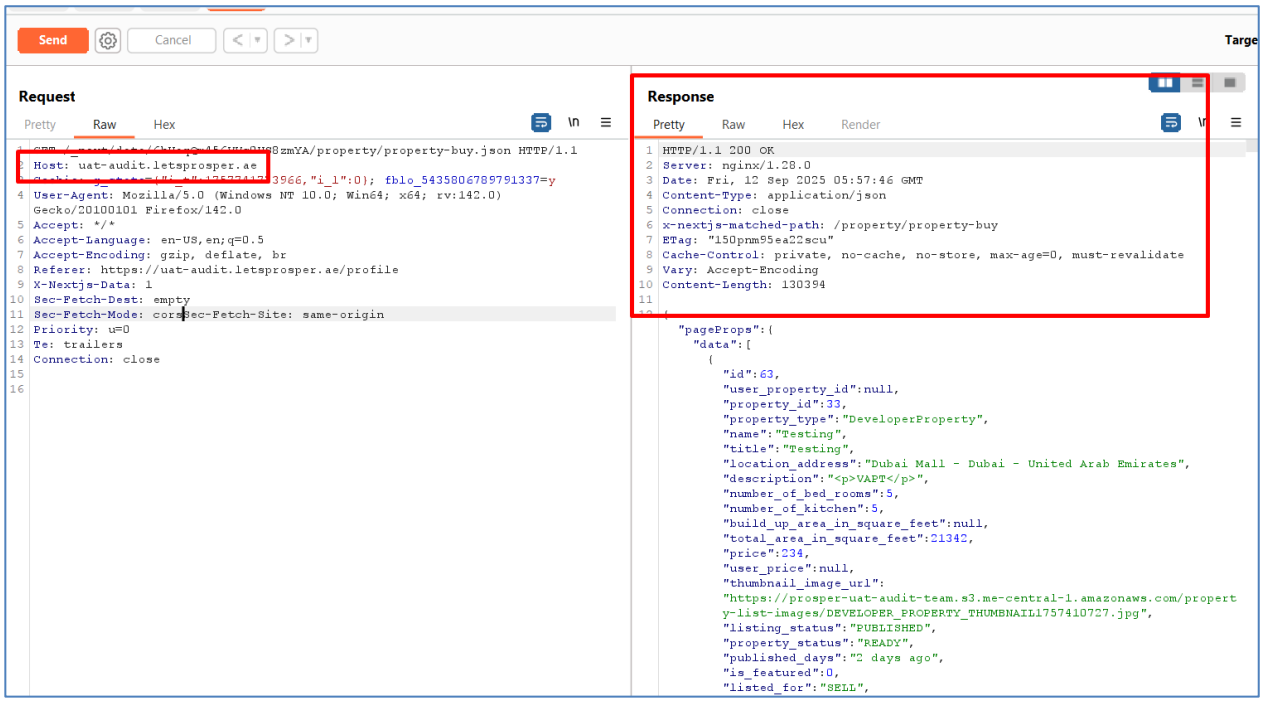
Remediation: It is recommended to implement security header **X-Frame-Options: DENY or SAME-ORIGIN** to prevent the application from being embedded in iframes. This mitigates the risk of clickjacking by enforcing strict framing policies.

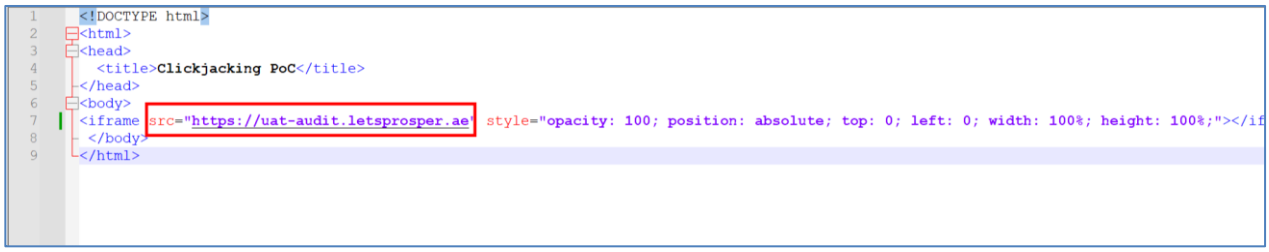
Security Audit Report

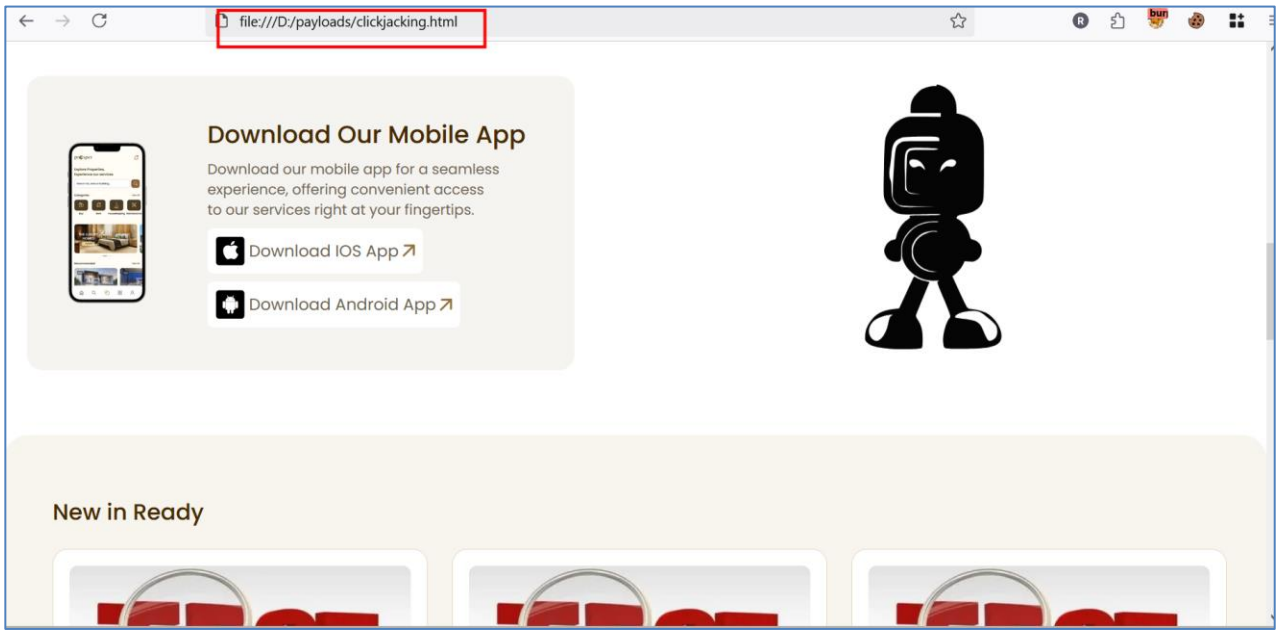
Proof Of Concept:

User Portal

1. X-Frame-Options header is not set. Therefore, the user portal is can be loaded in an iframe.



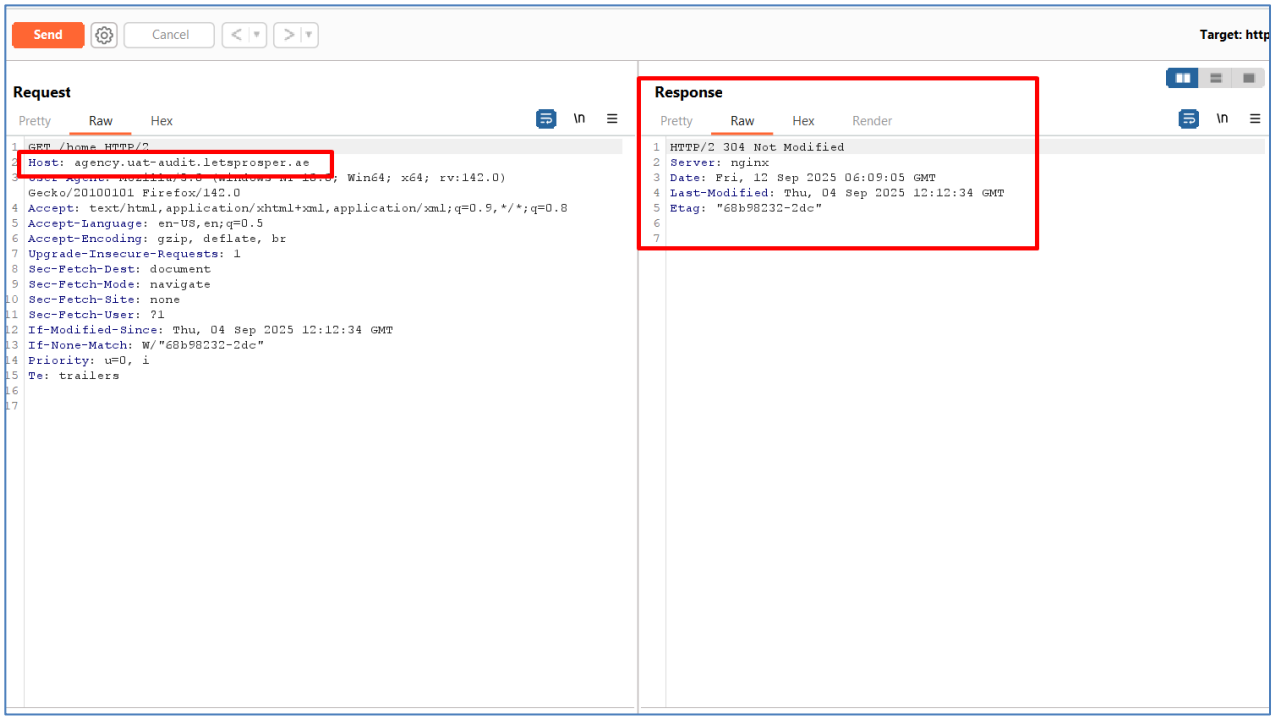


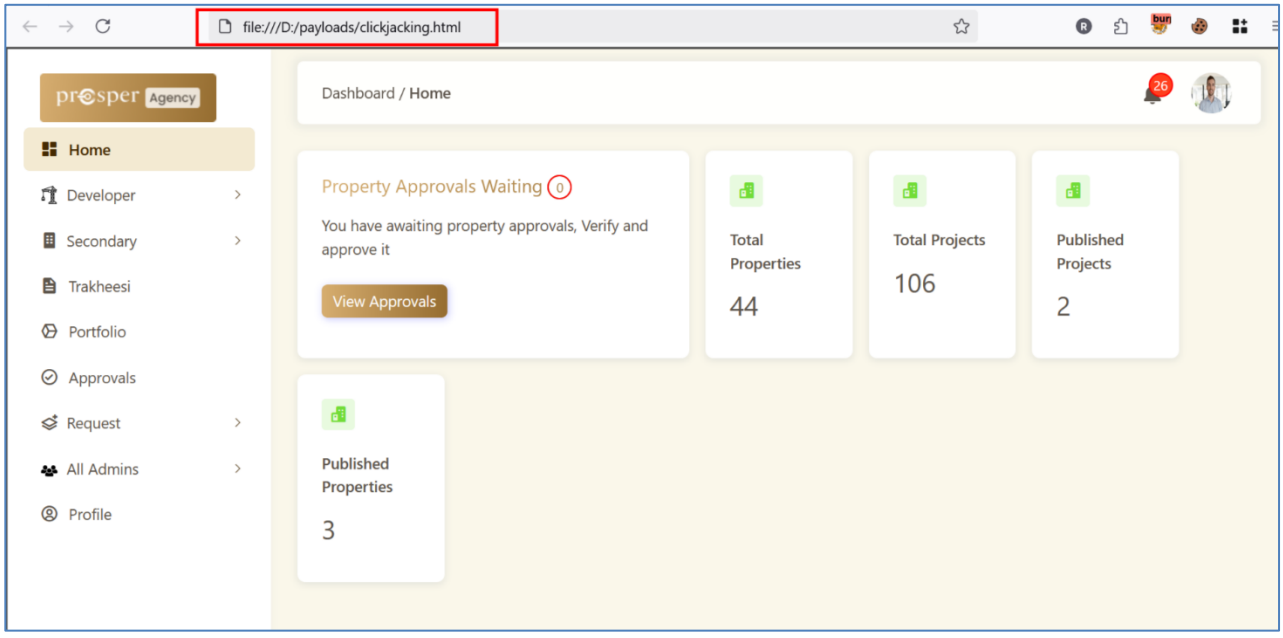


Security Audit Report

Agency Portal

1. X-Frame-Options header is not set. Therefore, the user portal is can be loaded in an iframe.





2.19 Insecure Session Management

Severity: Medium

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: Upon changing the password of an account in any of the portals (User, Admin, Agency), the existing active sessions are not invalidated. This allows the user to remain logged in on other devices or browsers without reauthentication.

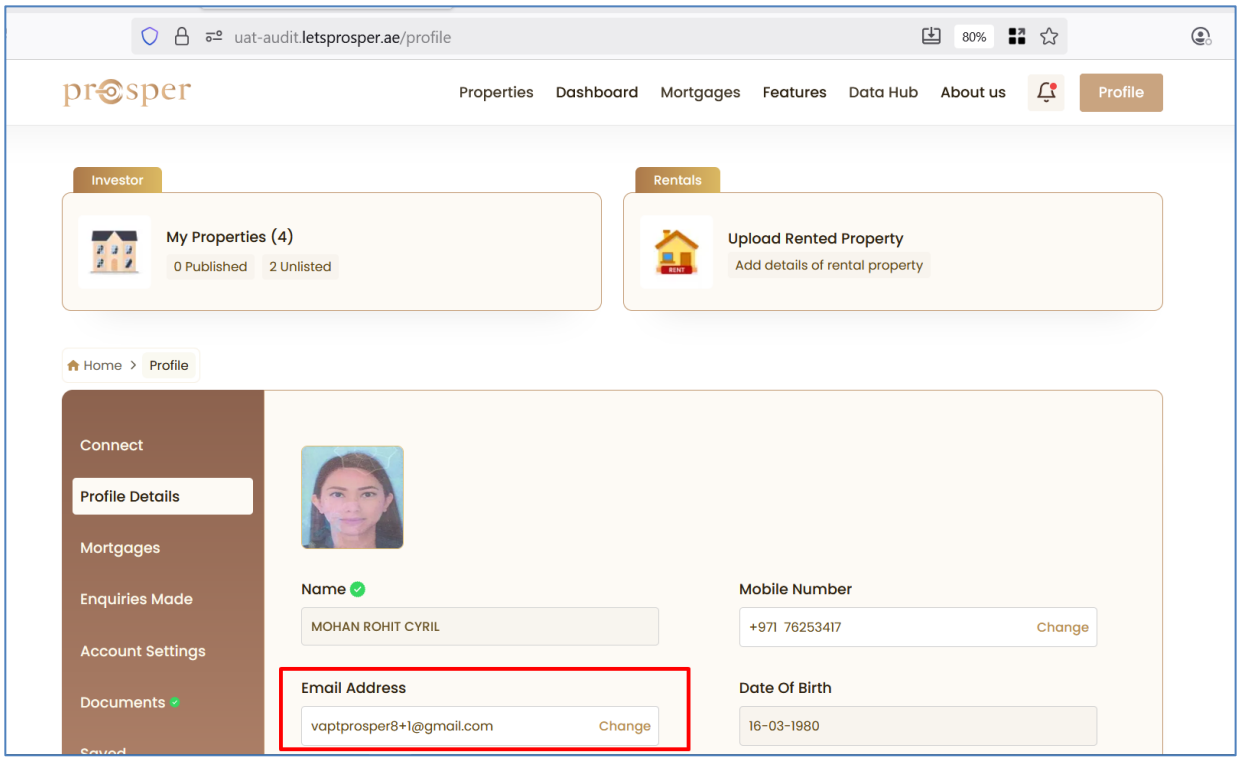
Analysis: This is a security design flaw in session management. If an attacker has an active session, they can continue accessing the account even after the password has been changed, defeating the purpose of the password update for securing the account. It increases the risk of unauthorized access, especially in scenarios where the password change was initiated to mitigate a suspected compromise.

Remediation: It is recommended to invalidate all active sessions immediately after a password change. The user should be required to re-login on all devices using the new password. This ensures that any potentially compromised session is terminated and enhances overall account security.

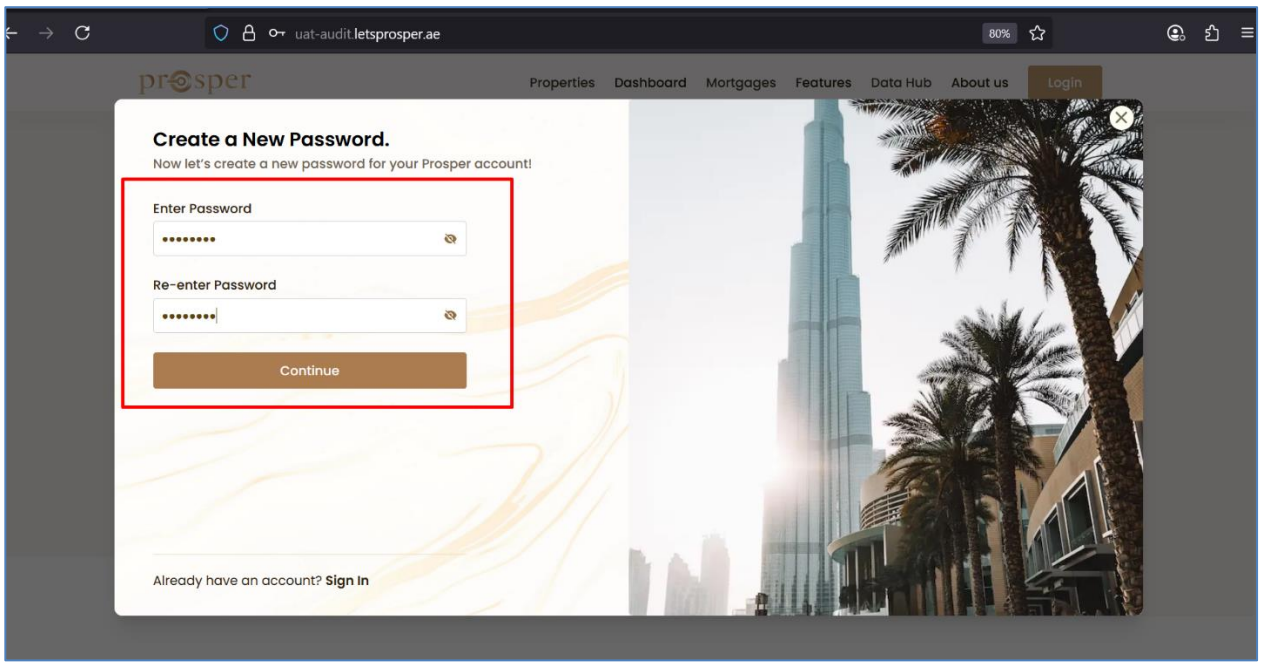
Proof Of Concept:

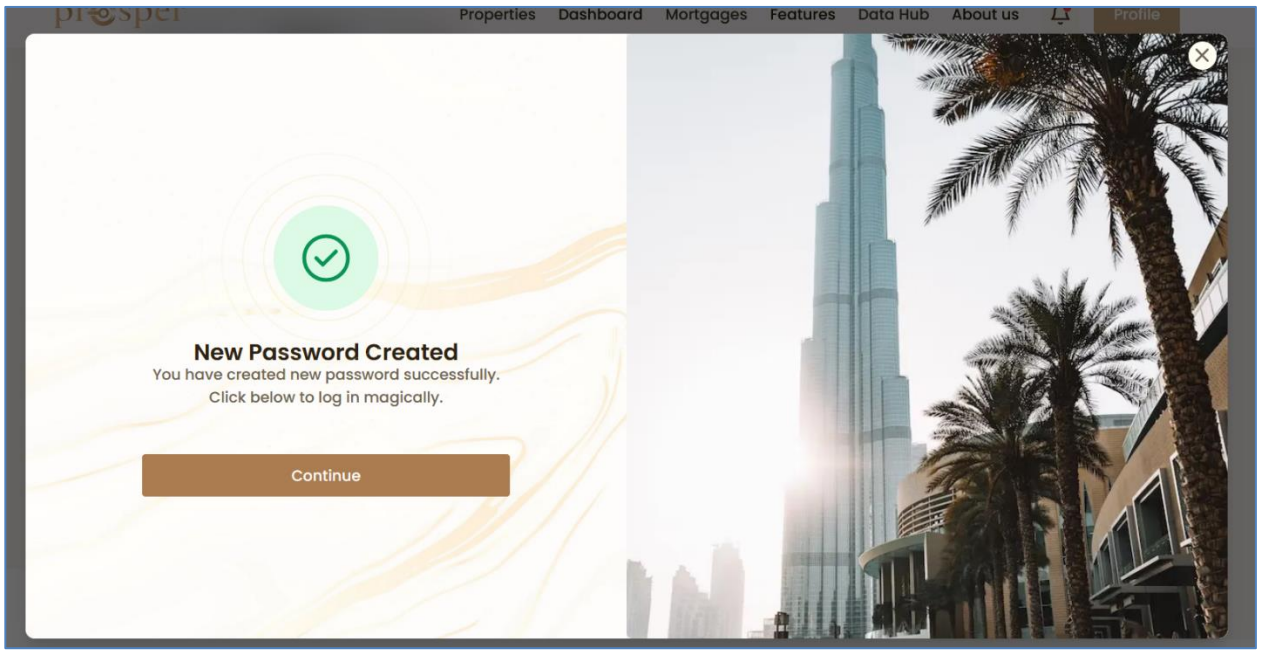
User Portal:

1. Login to a User account.

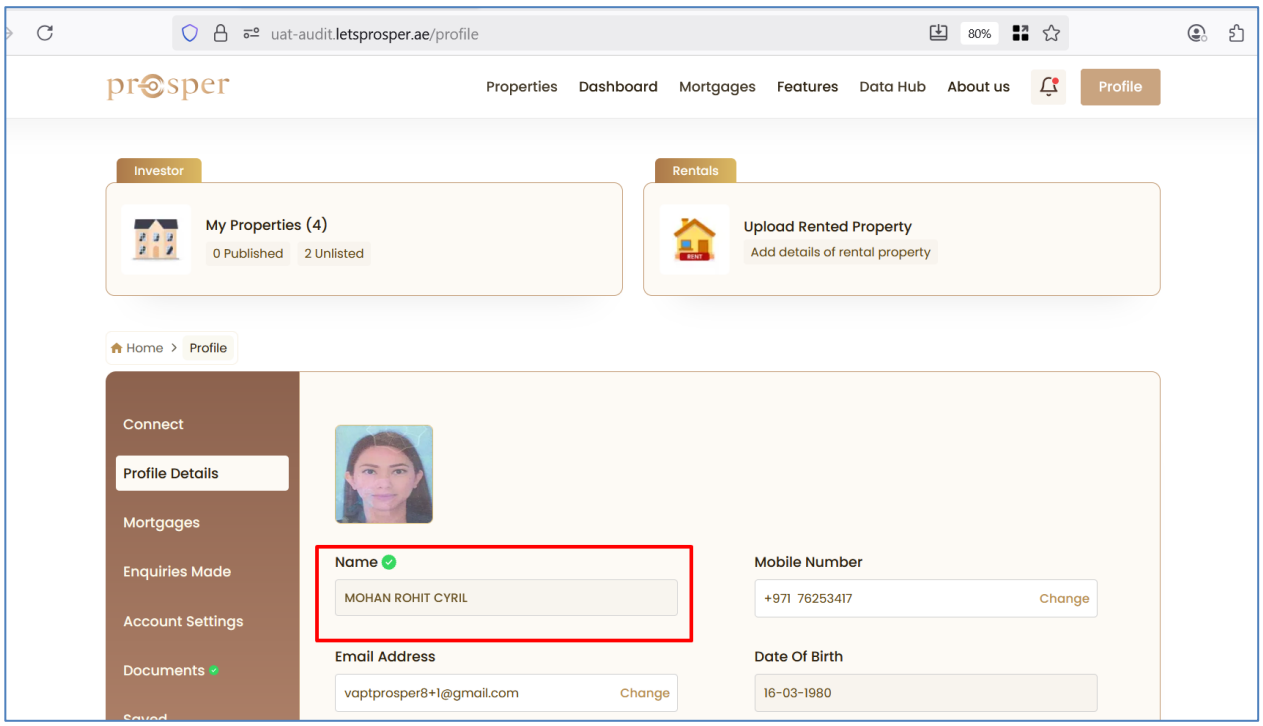


2. In another browser or a private window, change the password for this account.



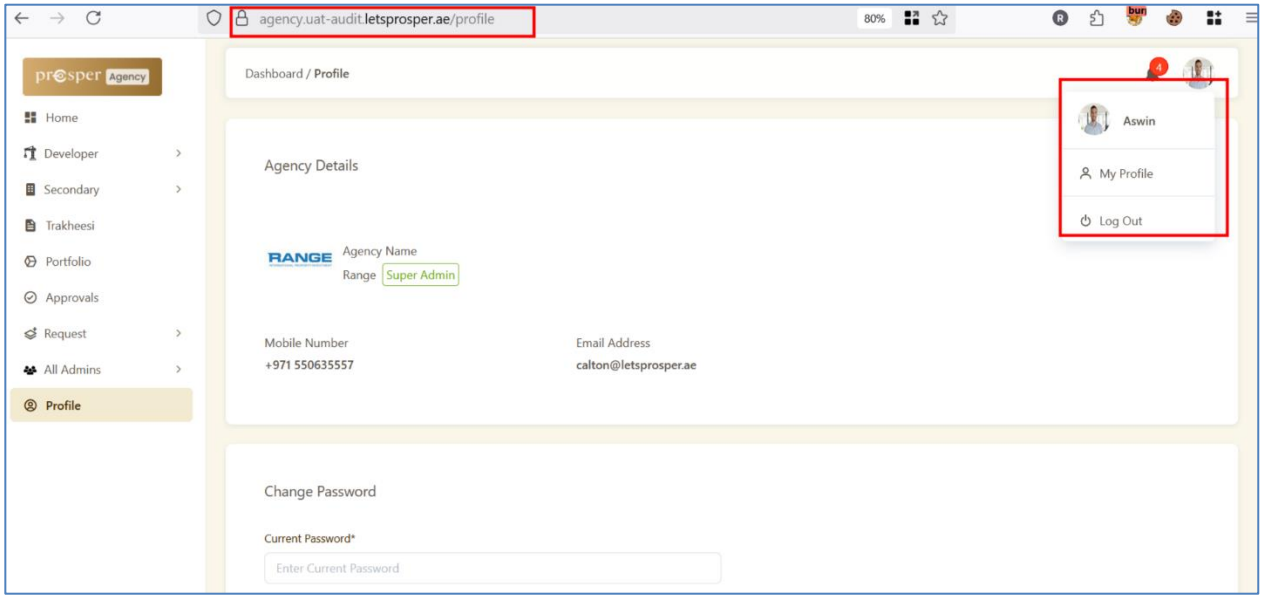


3. The user still remains logged in the application.

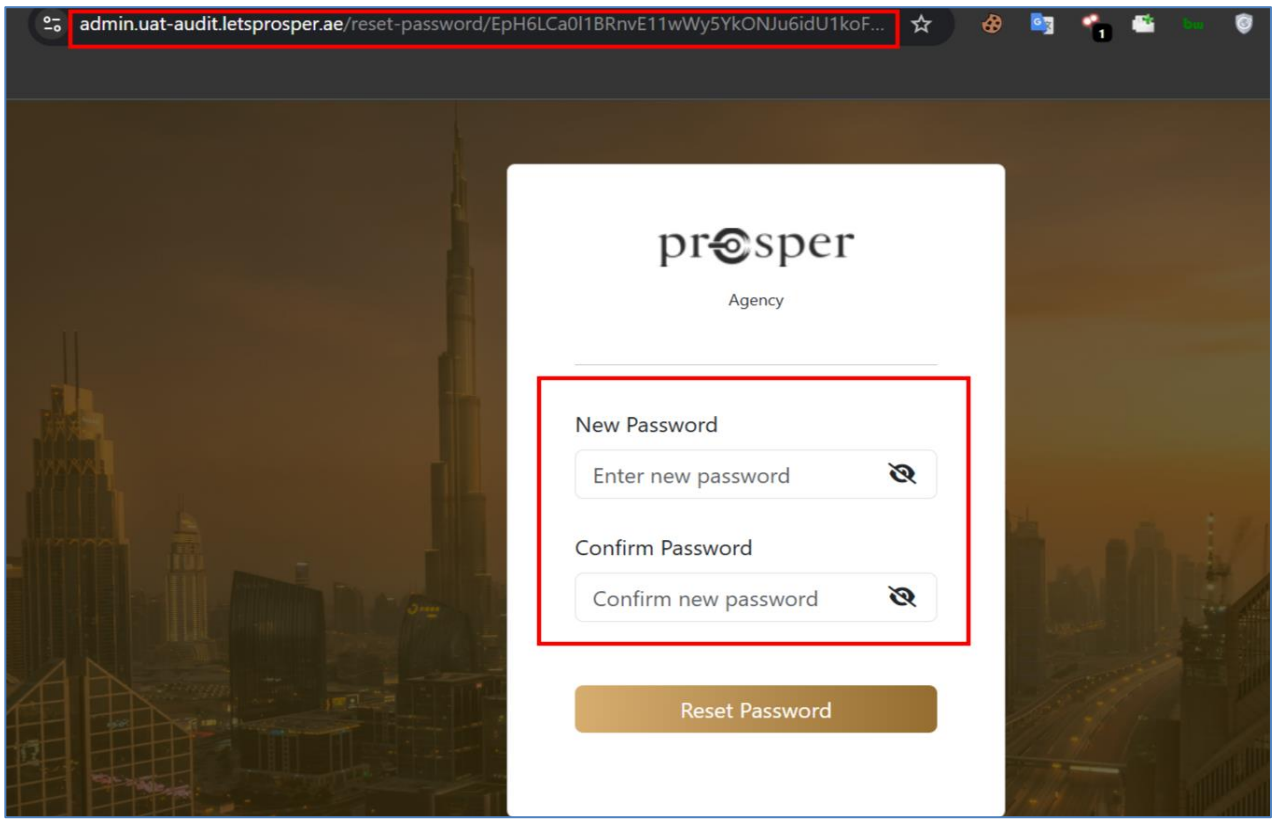


Agency Portal

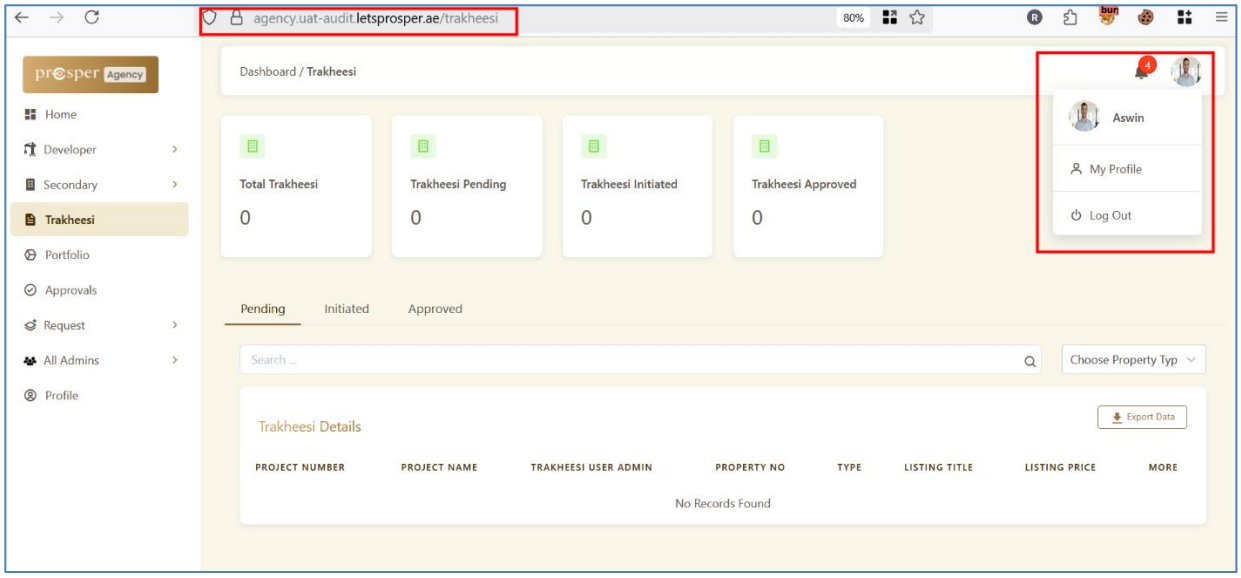
1. Login into the agency portal with an account.



2. In another browser or a private window, change the password for this account.



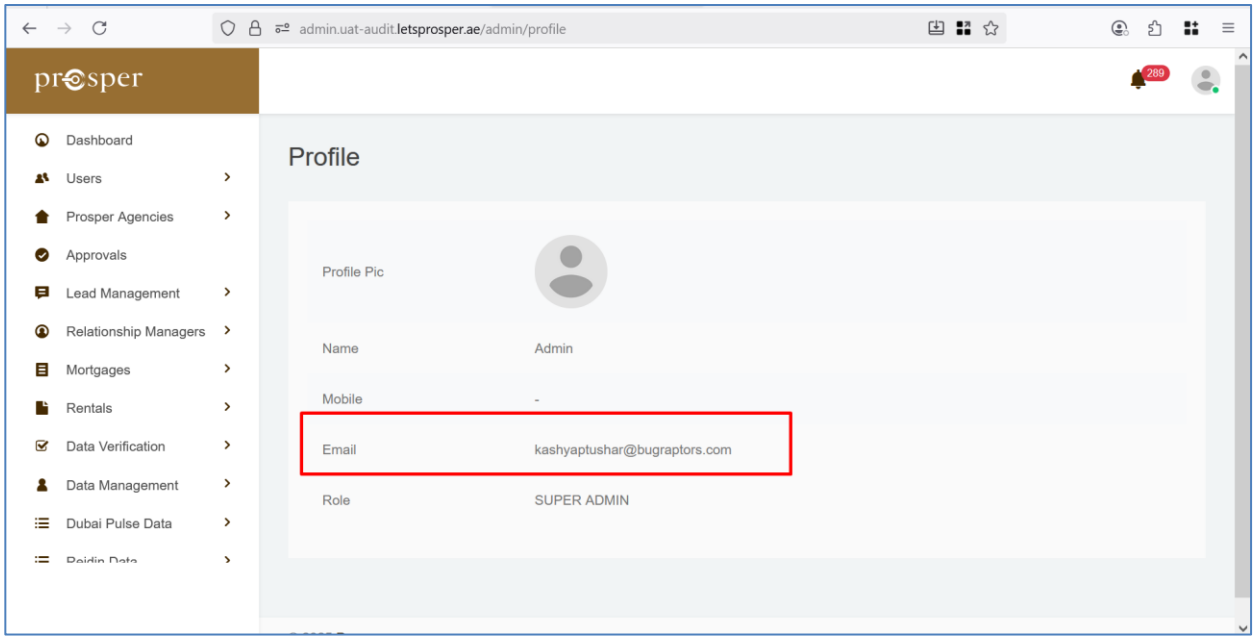
3. Even after the password change, the user still remains logged in.



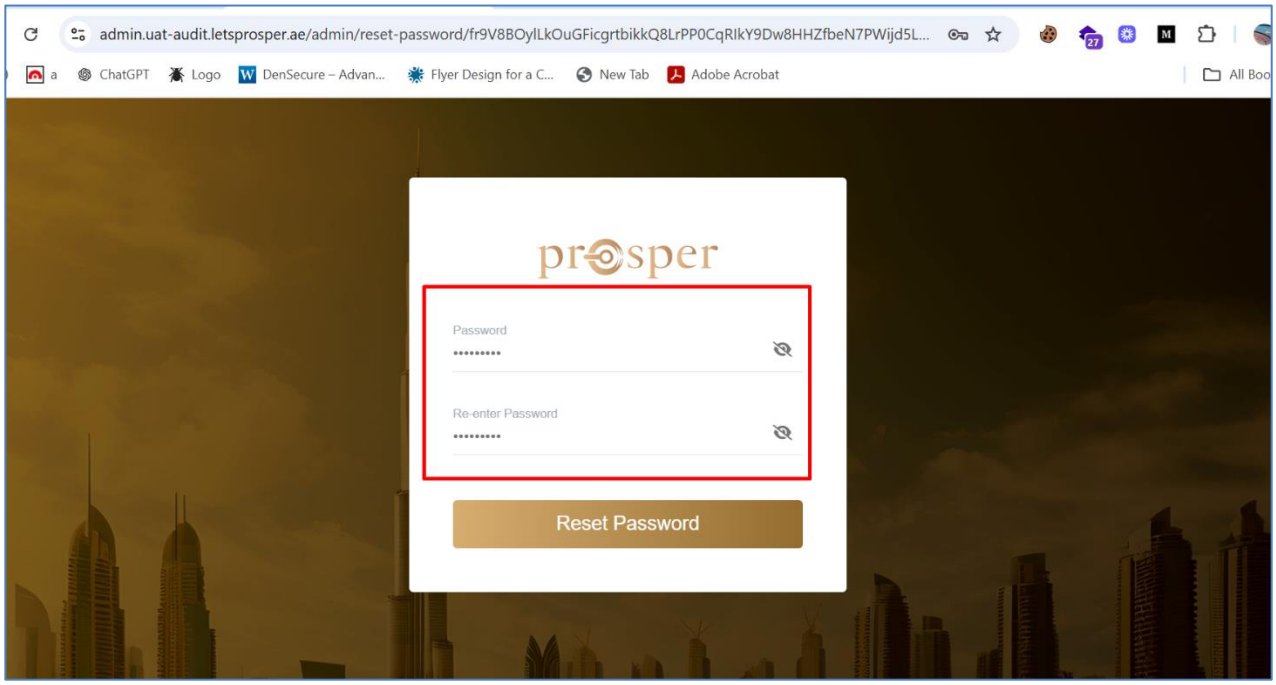
Security Audit Report

Admin Portal:

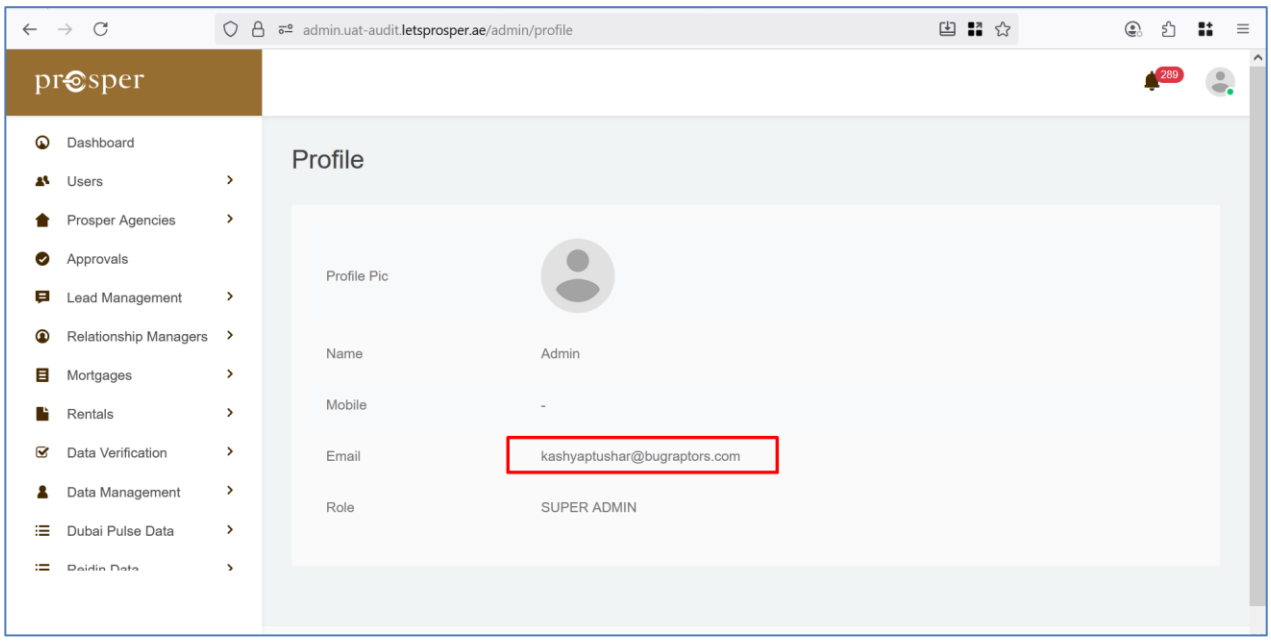
1. Login into the admin portal with an account.



2. In another browser or private window, change the password for this account.



3. The account is not logged out even after the password change.



2.20 Absence of Web Application Firewall

Severity: Medium

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: The current infrastructure lacks a Web Application Firewall (WAF), leaving the web application unprotected against common web threats and vulnerabilities. This omission exposes the system to various attacks, such as SQL injection, cross-site scripting (XSS), and other exploitable vulnerabilities.

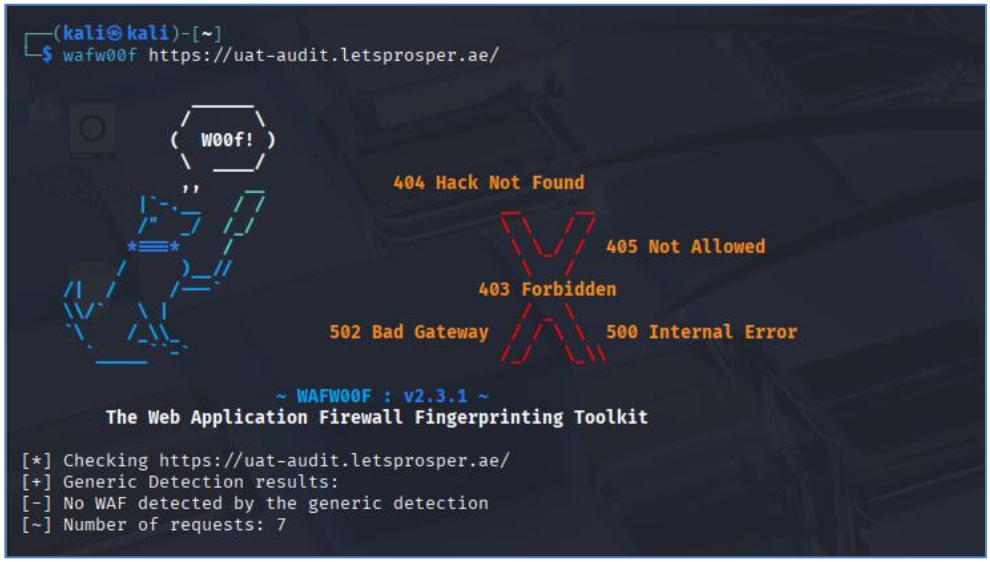
Analysis: Without a WAF, the application is more susceptible to a wide array of attacks that could compromise data integrity, confidentiality, and availability.

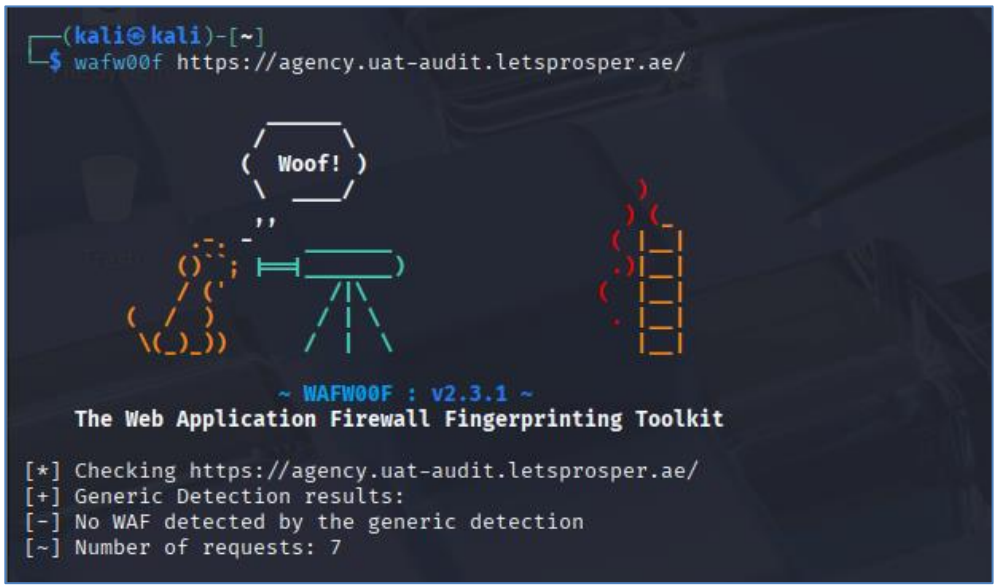
Remediation: It is recommended to deploy a robust Web Application Firewall that can inspect incoming traffic to identify and block malicious requests based on established security rules and patterns.

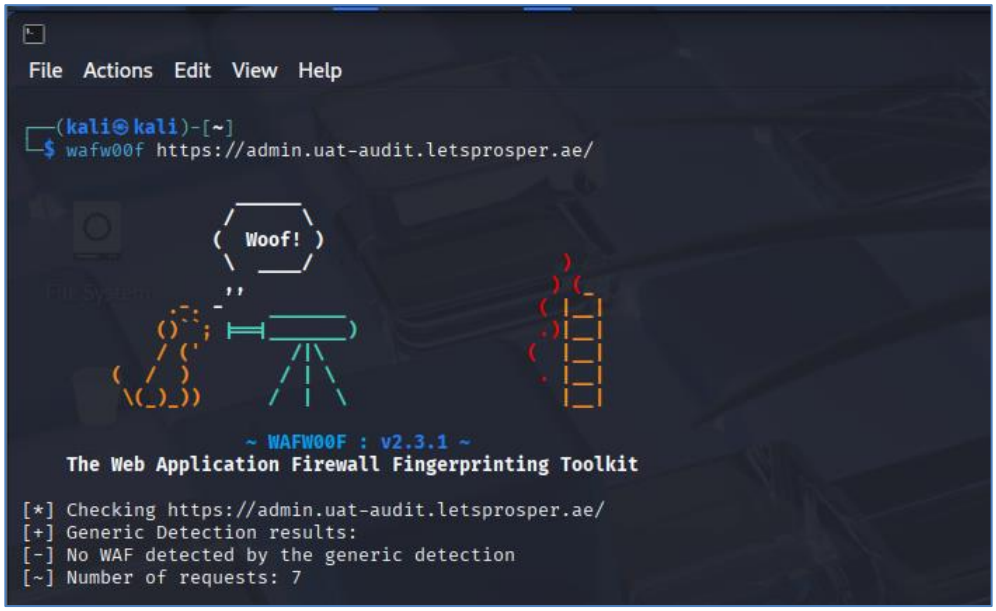
Proof of Concept:

1. WAF is not present.

Security Audit Report







2.21 Sensitive Information Exposure via Referrer Header

Severity: Low

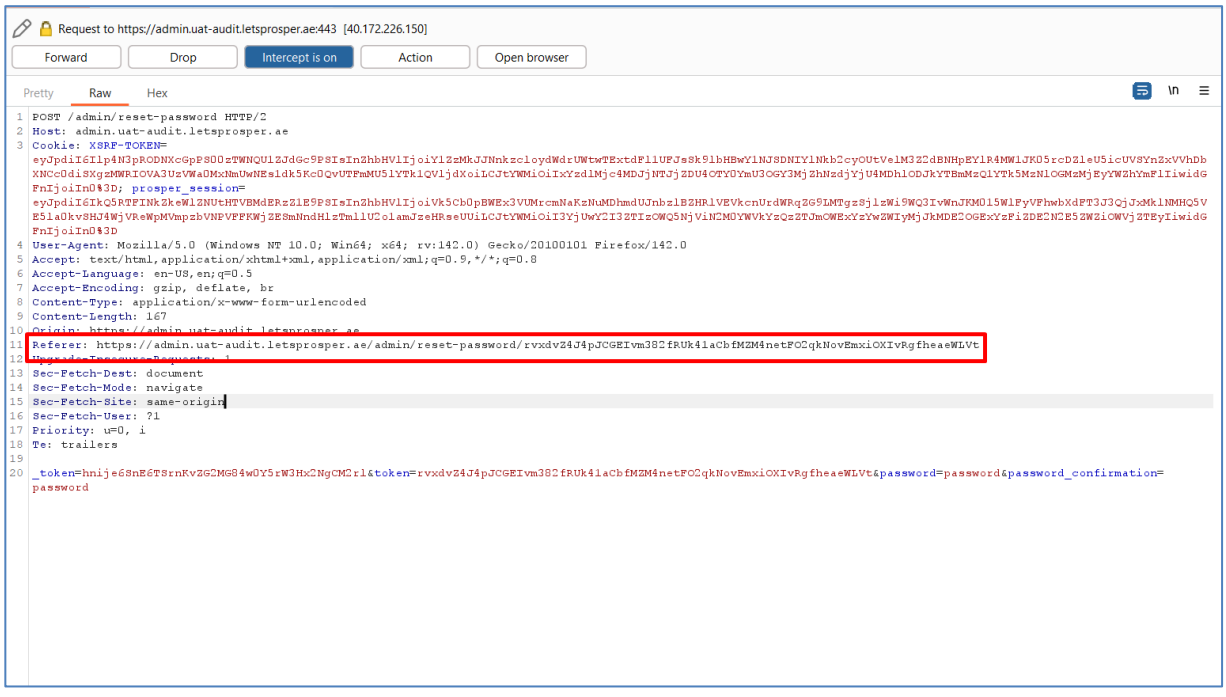
Issue Related to: Agency Portal, Admin Portal

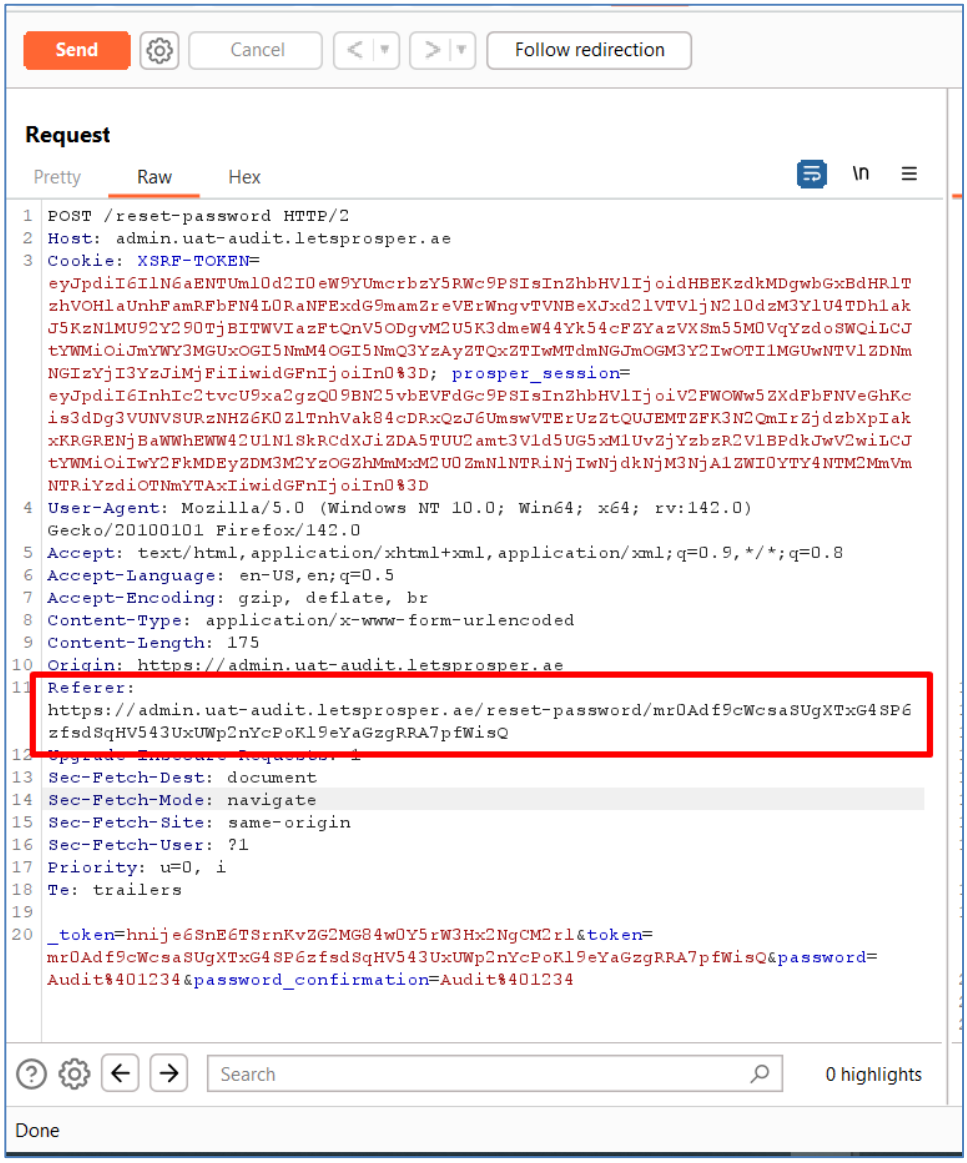
Issue Definition: When using the password reset link received via email, the user opens the link and proceeds to set a new password. The request to the password update (forward password) endpoint includes the full reset URL (which contains the sensitive reset token) in the **Referrer header**. Currently, this header is sent to the same application domain, so no external leakage occurs today. However, since the Referrer header exposes the full URL with the token, it poses a future risk.

Analysis: Although the referrer header is currently sent only to the same application domain, if in the future the password reset page includes external links (e.g., social media icons or third-party resources), the full reset URL containing the sensitive token could be sent to external domains. This would expose the reset token, allowing unauthorized users to hijack the password reset process and take over accounts, compromising user security and system integrity.

Remediation: It is recommended to set a strict **Referrer-Policy** header, such as no-referrer or strict-origin, to prevent sensitive URL parameters from being sent in the Referrer header.

Proof Of Concept:





2.22 Autocomplete Enabled for Password Fields

Severity: Low

Issue Related to: User Portal, Agency Portal, Admin Portal

Issue Definition: In the **User Portal**, **Agency Portal**, and **Admin Portal**, the password field in the login form does not explicitly set autocomplete="off". As a result, autocomplete is enabled by default, allowing browsers to store and autofill the password field in future sessions.

Analysis: With autocomplete enabled by default, sensitive credentials can be automatically filled by the browser on shared or public devices, increasing the risk of unauthorized access if the device is accessed by other users. This may lead to accidental credential exposure or account compromise.

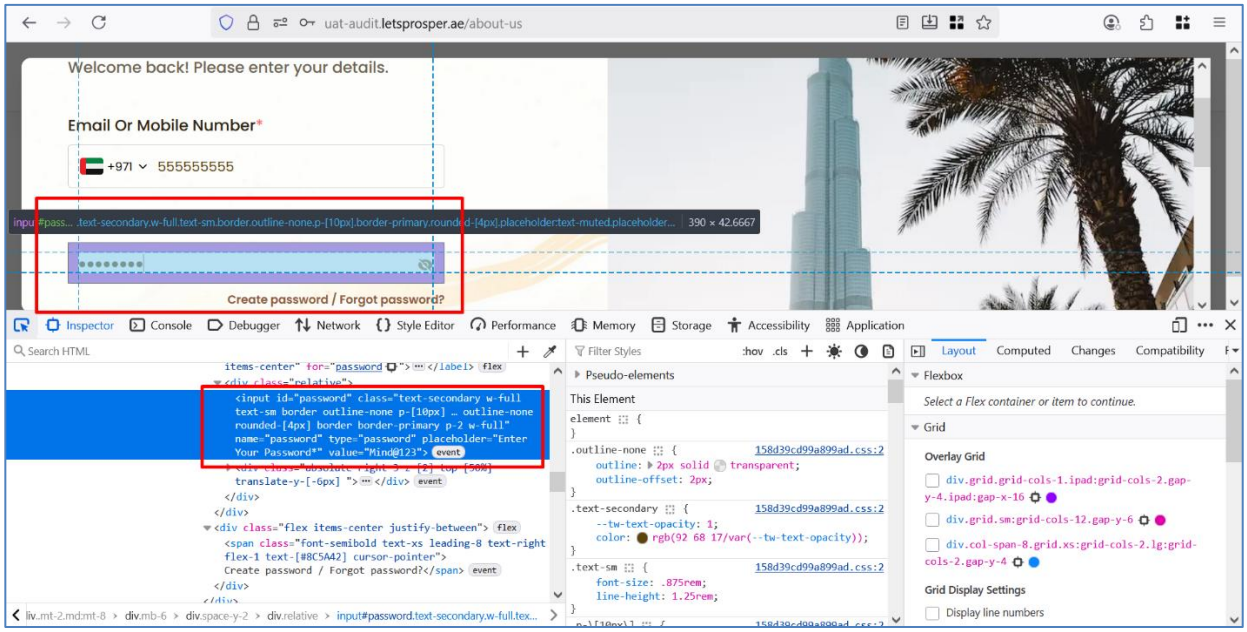
Remediation: It is recommended to explicitly set autocomplete="off" on password fields in login forms to prevent the browser from storing or autofilling sensitive information. This ensures that users are required to manually enter their credentials for every login attempt.

Security Audit Report

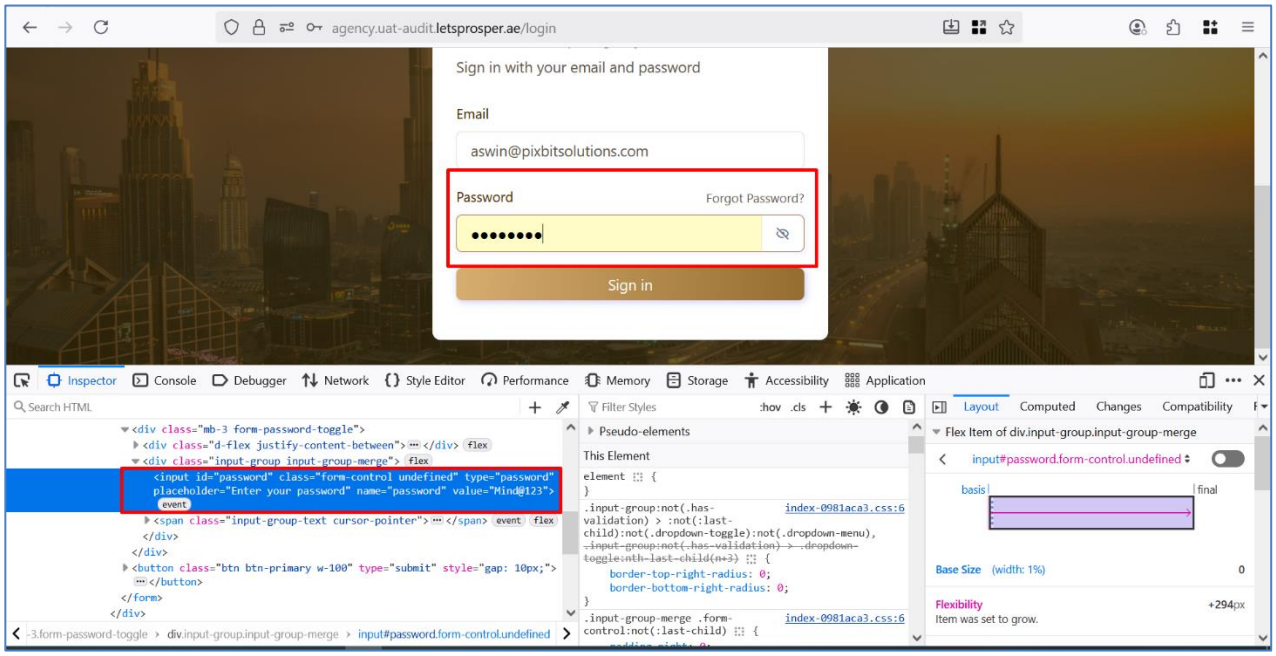
Proof Of Concept:

1. Autocomplete has not explicitly been disabled for all the 3 web portal login pages.

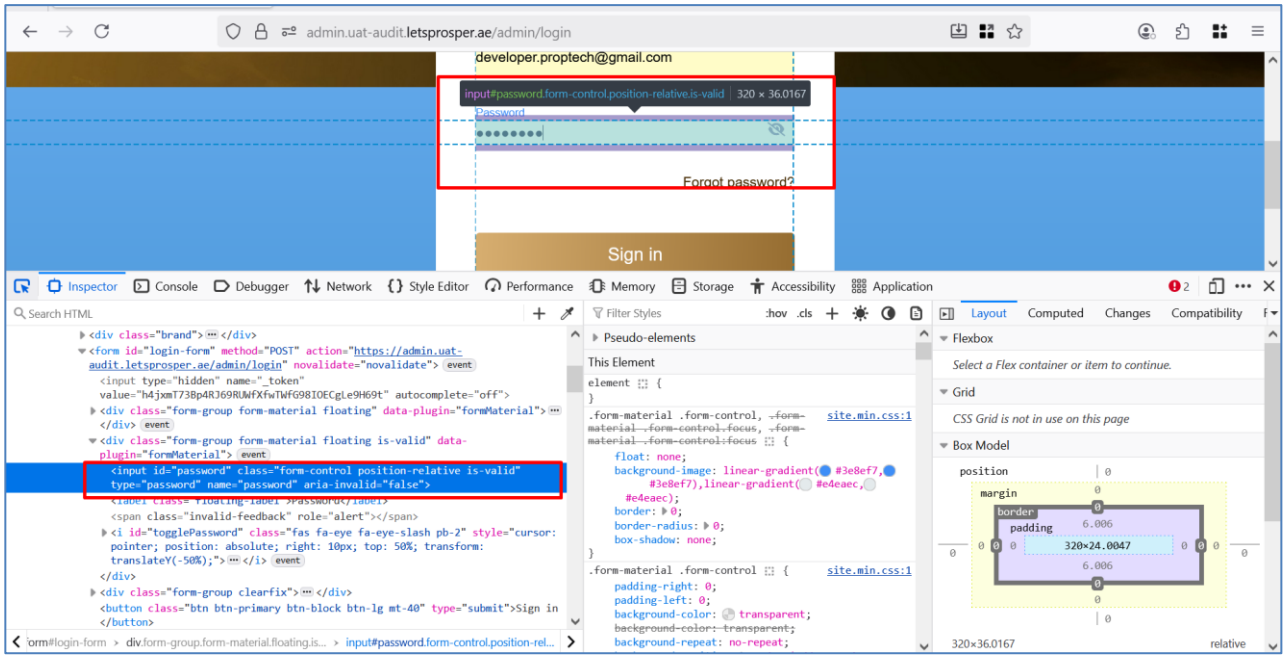
User Portal:



Agency Portal:



Admin Portal:



2.23 Exposure of phpMyAdmin Interface and Documentation

Severity: Low

Issue Related to: Admin Portal

Issue Definition: The phpMyAdmin login interface and its documentation pages are publicly accessible via predictable URLs in the admin subdomain of the application, specifically:

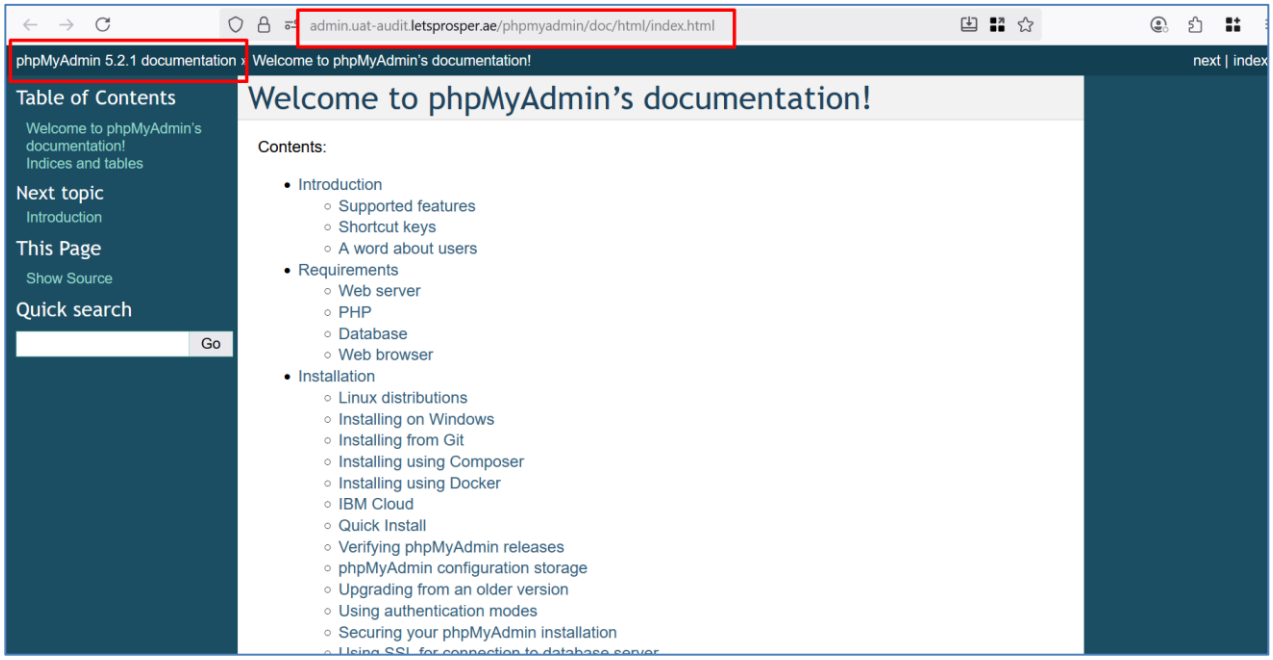
- /phpmyadmin/
- /phpmyadmin/doc/html/index.html

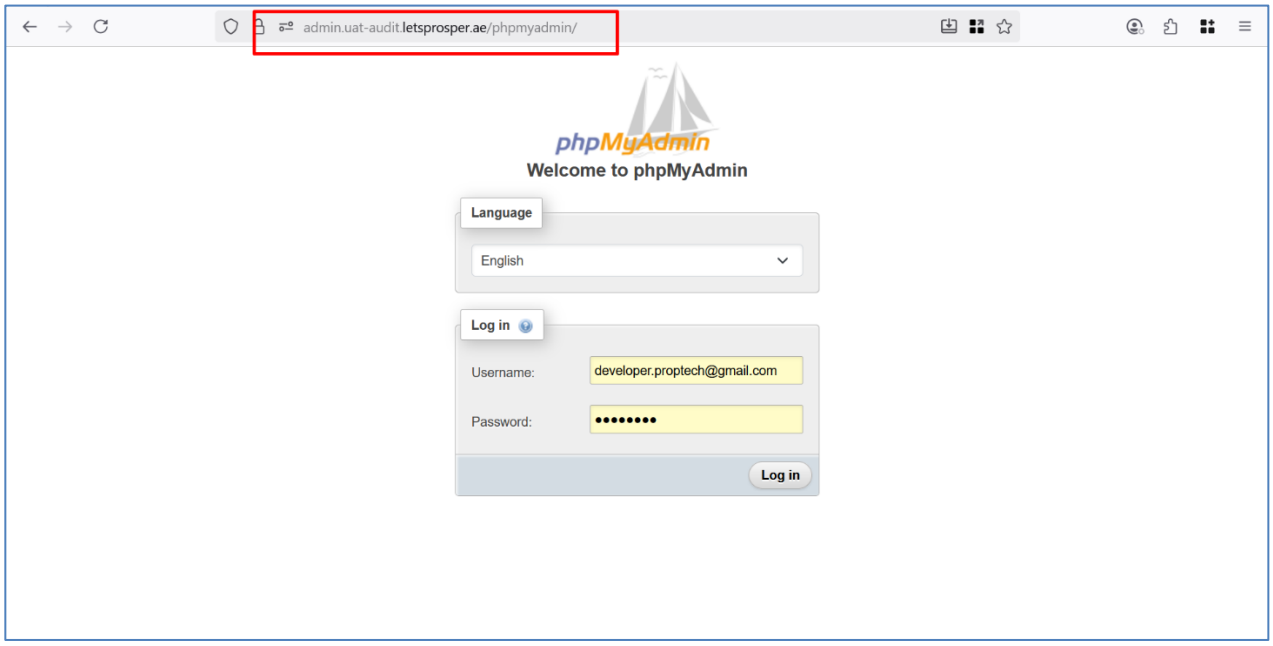
These pages do not require any authentication or additional access control, exposing sensitive administrative tools and documentation that can aid attackers in understanding and exploiting the system.

Analysis: Exposing the phpMyAdmin interface and documentation publicly increases the risk of unauthorized database access attempts and system compromise. Attackers can use the publicly accessible interface to attempt brute-force logins or exploit known vulnerabilities in phpMyAdmin.

Remediation: It is recommended to restrict access to phpMyAdmin interfaces and documentation by removing them from the public domain or securing them with strong authentication mechanisms. Ideally, restrict access via IP whitelisting or VPN access only.

Proof Of Concept:





2.24 Server Banner Information Disclosure

Severity: Low

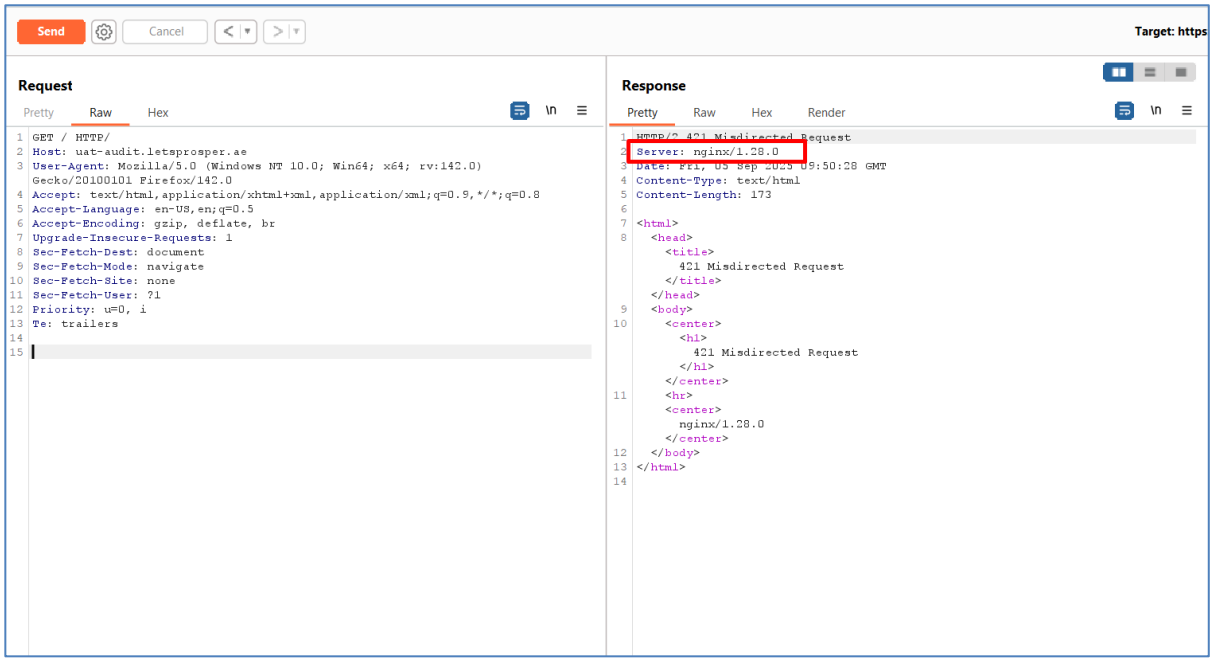
Issue Related to: User Portal

Issue Definition: The application's HTTP response headers reveal detailed server banner information, specifically: Server: nginx/1.28.0
This exposes the exact web server software and version being used.

Analysis: Exposing server banner information provides attackers with knowledge of the server software and its version. This information can be used to identify known vulnerabilities and tailor attacks specific to the version of the server in use, increasing the risk of successful exploitation.

Remediation: It is recommended to configure the web server to suppress or hide version details in response headers. For nginx, set server_tokens off; in the configuration file to prevent the server version from being disclosed in HTTP headers.

Proof Of Concept:



2.25 Cleartext Submission of Password

Severity: Low

Issue Related to: User Portal, Agency Portal, Admin Portal

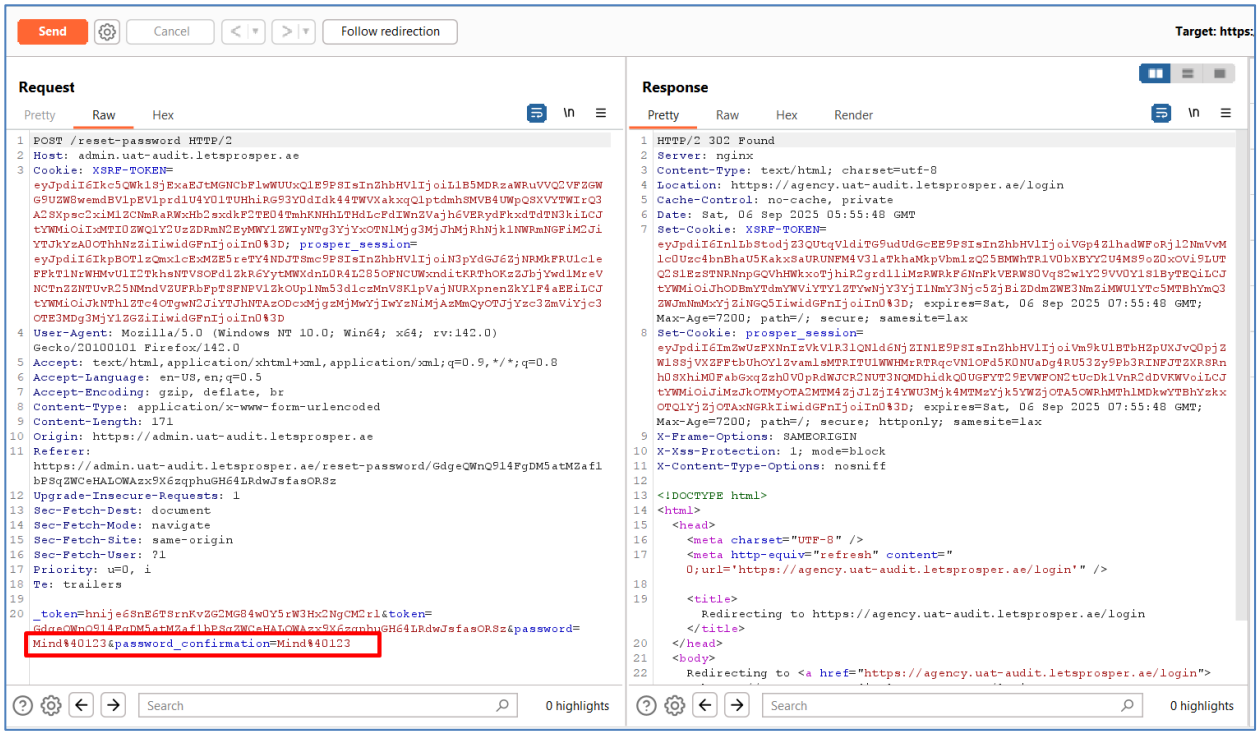
Issue Definition: A security concern was identified during our security assessment regarding password submission in the login or the forgot/reset/create password process. The issue lies in the transmission of passwords in an unencrypted cleartext format.

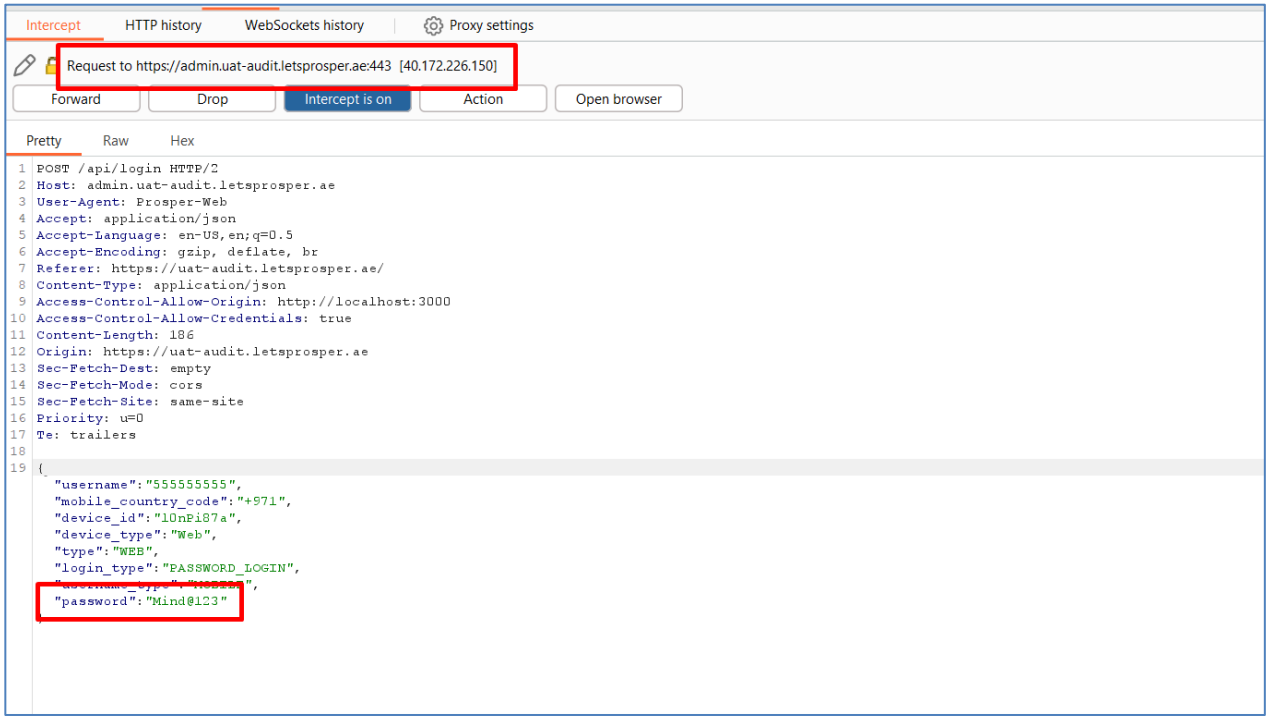
Analysis: Cleartext password submission, despite TLS/SSL usage, poses a security risk during transmission, potentially allowing unauthorized access to user credentials. However, it's important to note that if local encryption of passwords before submission is implemented, even during periods when the TLS/SSL certificate is not operational, the risk of compromise by a Man-in-the-Middle (MITM) attack is mitigated.

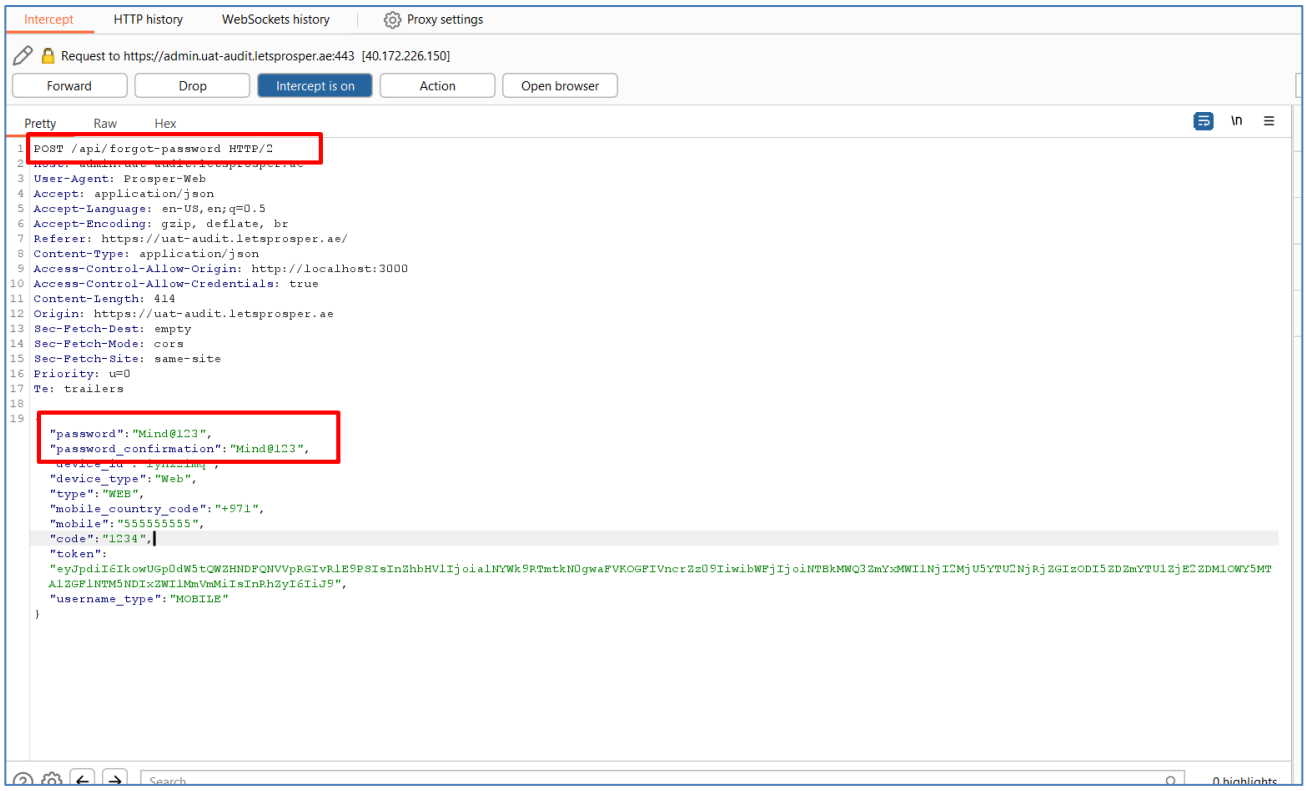
Remediation: It is recommended to implement local encryption for passwords before submission to improve overall security. This measure ensures an additional layer of protection, mitigating risks associated with potential interception.

Security Audit Report

Proof of Concept:







2.26 Improper Error Handling

Severity: Informational

Issue Related to: Admin Portal

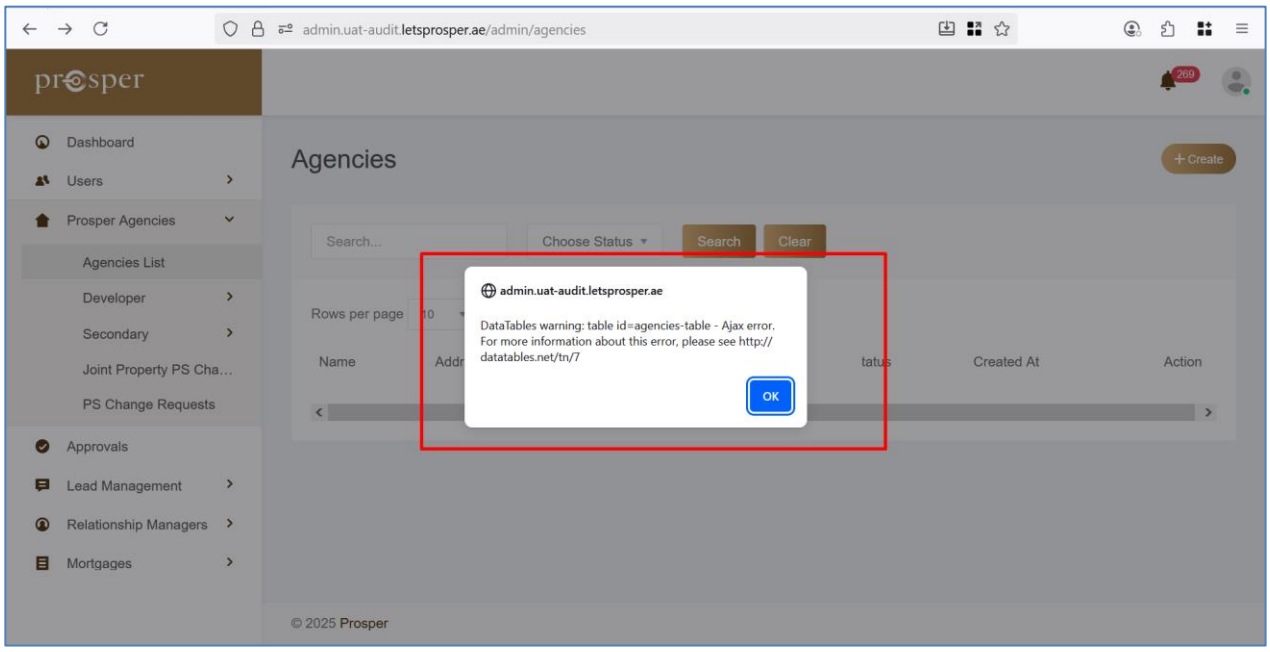
Issue Definition: When a user logs out of the Admin Portal and uses the browser back button to navigate to a previously visited page (e.g., Agencies List), a DataTables warning appears: DataTables warning: table id=agencies-table - Ajax error. This happens because the session is no longer valid, and the subsequent AJAX request to load the table data fails.

Analysis: This leads to a poor user experience and may confuse legitimate users. While it does not expose sensitive data or result in a security vulnerability directly, it indicates missing proper session validation and error handling mechanisms.

Remediation: It is recommended to implement proper session validation for all AJAX endpoints. If the session has expired or is invalid, return a structured error response (e.g., HTTP 401 Unauthorized), and redirect the user to the login page.

Proof of Concept:

1. Logout from the admin portal and click on the browser back button. This error appears.



2.27 Session and XSRF Token Regeneration on Every Request

Severity: Informational

Issue Related to: Admin Portal

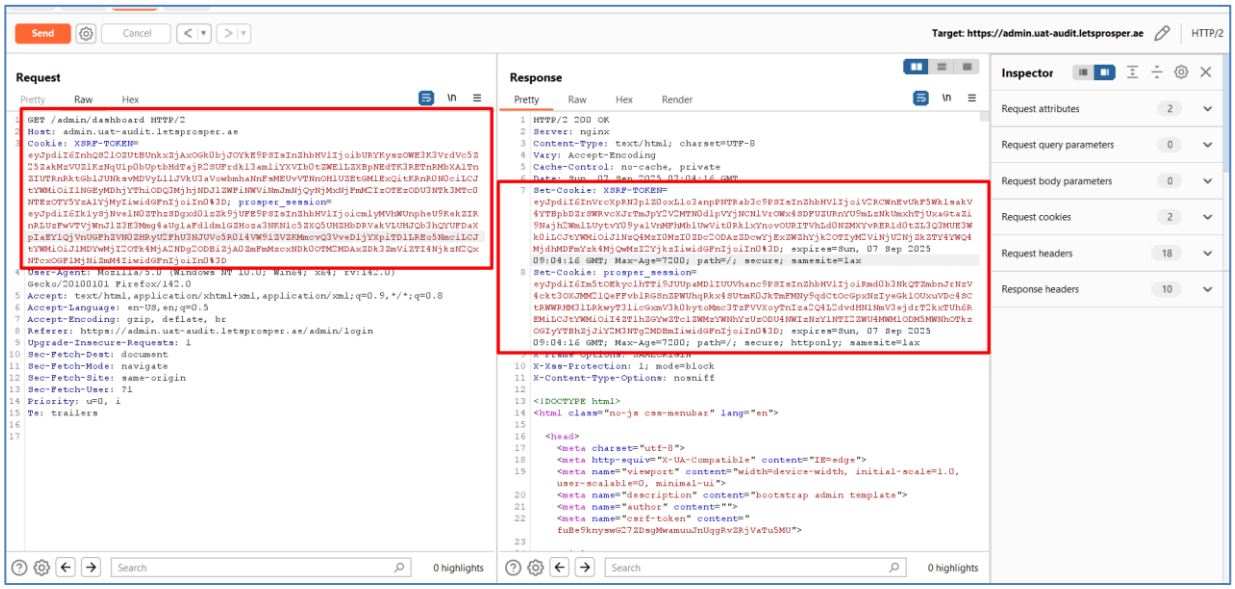
Issue Definition: In the admin portal, after signing in, the server regenerates the session cookie and XSRF token on every request (GET or POST). Each response sets new values for these cookies, while all previously issued tokens remain valid until the user logs out.

Analysis: While this behavior does not directly lead to a security vulnerability, it results in multiple valid session and CSRF token values being maintained simultaneously for a single user. This increases server complexity and resource usage, and can make debugging or session management more cumbersome. However, authentication and token validity remain intact, and tokens become invalid after logout.

Remediation: It is recommended to maintain the same session and XSRF token values for the duration of the user's session, from login until logout. Tokens should only be regenerated on login or logout events, or for specific sensitive actions, to reduce server overhead while maintaining secure session handling.

Proof of Concept:

1. In every authenticated request, the response header contain a Set-Cookie header which sets new values for both the prosper_session and XSRF-Token.



3 Conclusion

Conclusively, this web app security audit report has outlined vulnerabilities within the Prosper web applications. This holistic assessment not only identifies vulnerabilities but also provides actionable insights and recommendations for enhancing the overall security of the application. As we move forward, the report serves as a roadmap for implementing robust security measures and fortifying the system against potential threats, ensuring a resilient and secure environment for the Prosper System.