**25 Q4 SandboxPeopleHub Manual Audit**

### Introduction:

As part of the security assessment, a focused manual review was conducted to evaluate the application’s authentication mechanisms and associated trust boundaries. The objective of this assessment was to identify potential weaknesses in the authentication flow, including improper handling of identity provider interactions, insufficient validation of authentication parameters, susceptibility to authentication bypass, and exposure to token or session-related attacks. Testing concentrated on critical authentication endpoints, redirect and callback handling, parameter integrity, and enforcement of security controls such as issuer validation, authentication flow consistency, and protection against replay and cross-site request forgery (CSRF). The assessment was performed in a controlled test environment using a combination of manual inspection and targeted request manipulation to validate the robustness of the authentication implementation against common real-world attack scenarios.

Table of Contents

[Introduction: 2](#_Toc898)

[25Q4 Sandbox PeopleHub Manual Audit 4](#_Toc6234)

[Authorization Code Reuse / Replay 4](#_Toc15878)

[Expired Authorization Code 5](#_Toc2010)

[Missing CSRF Check 6](#_Toc906)

[Issuer Tampering 6](#_Toc10214)

[AuthType Tampering 7](#_Toc19389)

[Issuer Omission 8](#_Toc26596)

[Missing Authentication Type 8](#_Toc3355)

[Code Bound to Client Check 10](#_Toc15974)

[Cacahed callback Response 14](#_Toc24038)

[Verification of User-to-Resource Binding Post Authentication 14](#_Toc25267)

### 25Q4 Sandbox PeopleHub Manual Audit

**Description**

The application is manually testeed for any security vulnerabilities

**Steps**

* Access the application sandboxpeoplehub.zalaris.com
* Capture the request using Burp proxy
* During the login call-back is invoked, the following URL is called

|  |
| --- |
| https://sandboxpeoplehub.zalaris.com/login/callback?authType=ias&code=716d3e10e5e54e81acf9600349edd003&iss=https%3A%2F%2Fahekfr4s6.accounts.cloud.sap |

The code is received directly in the URL Passing authentication parameters such as authorization codes and issuer identifiers via URL query strings introduces risks including credential leakage, replay attacks, issuer spoofing, CSRF, and cross-tenant authentication confusion

### Authorization Code Reuse / Replay

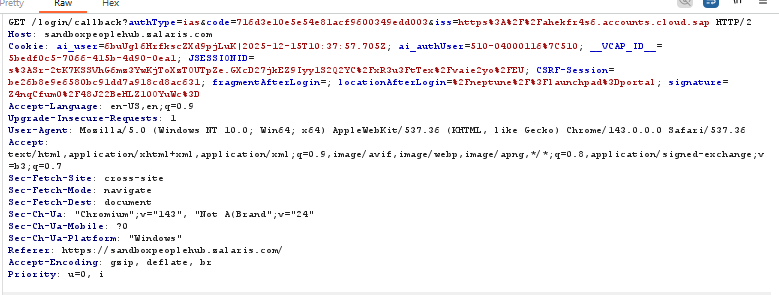
**Description**

Check whether the auth code is vulnerable to authorization code re-use or replay

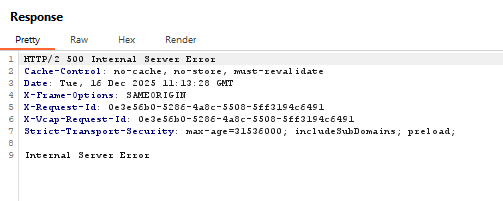
**Steps**

* Complete a valid login and capture code from URL using Burp
* Replay same callback URL again
* The Server responds with 500 Internal Server error.
* Refer Screenshot attached for evdidence

**Request**



**Response**



This is the expected safe behavior

**Issue Found :** None

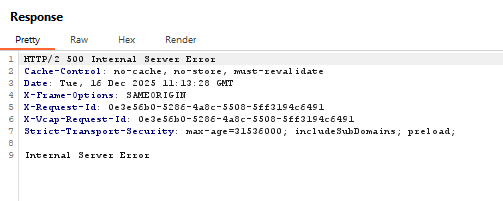
### Expired Authorization Code

**Description**

Check whether the application verified for authorization code expiration

**Steps**

* Complete a valid login and capture code from URL using Burp
* Wait for a period so that the code expired
* Replay same callback URL again
* The code is rejected due to expiration
* Refer Screenshot attached for evdidence



Code is rejected due to expiration. This is the expected behavior

**Issue Found :** None

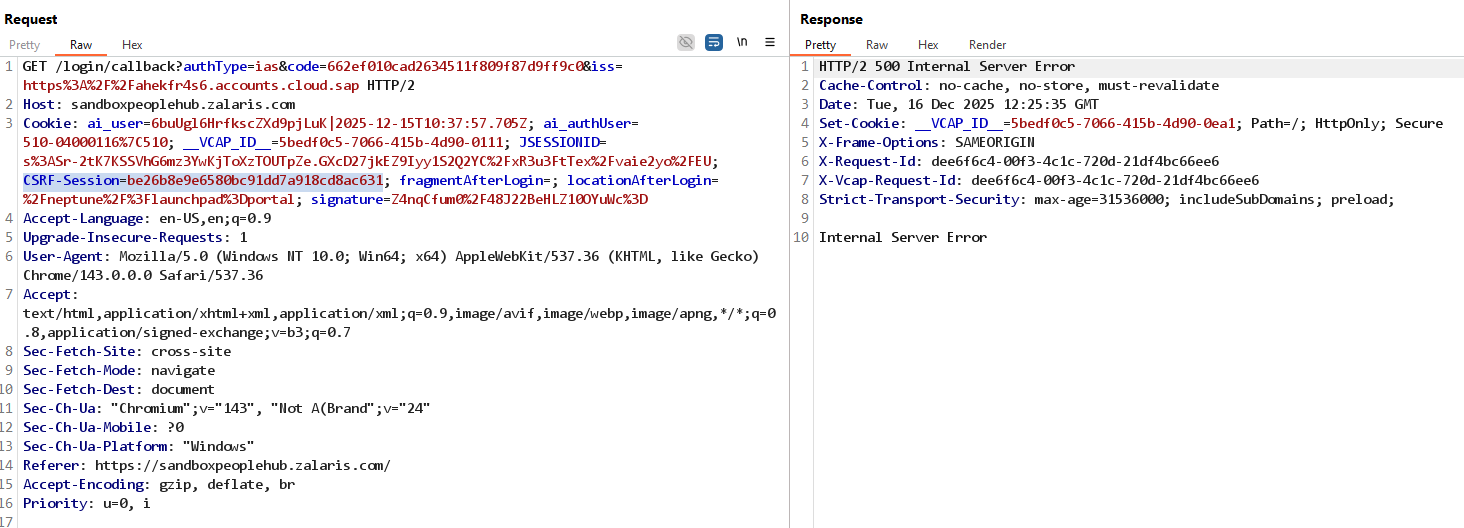
### Missing CSRF Check

**Description**

Check whether the auth request is vulnerable to CSRF attack

**Steps**

* Complete a valid login and capture code from URL using Burp
* The captured request is not having valid anti CSRF token
* The CSRF session is found in the Cookie Value
* Refer screenshot attached for evidence



* The authentication code captured cannot be replayed or resent since the code expires after a single use
* The CSRF token is not needed at this endpoint

**Issue Found :** None

### Issuer Tampering

**Description**

Verify whether the application is vulnerable to issuer tampering

* Complete a valid login and capture code from URL using Burp
* Alter the issuer value to a random value : <http://192.168.200.195:5000>
* The server accepts the request and give 302 Found



* The server accepts the request and gives 302 Found response
* The tampered Issuer value must not be accepted
* The value that is injected is reflected in the referrer header alone and not in some other area such as response header or request header
* The response confirms that the iss query parameter is complete;y ignored and the application is using the server side configuration issuer/ idP

**Issue Found:** None . The issuer parameter need not be forwarded to the server the code parameter alone is enough

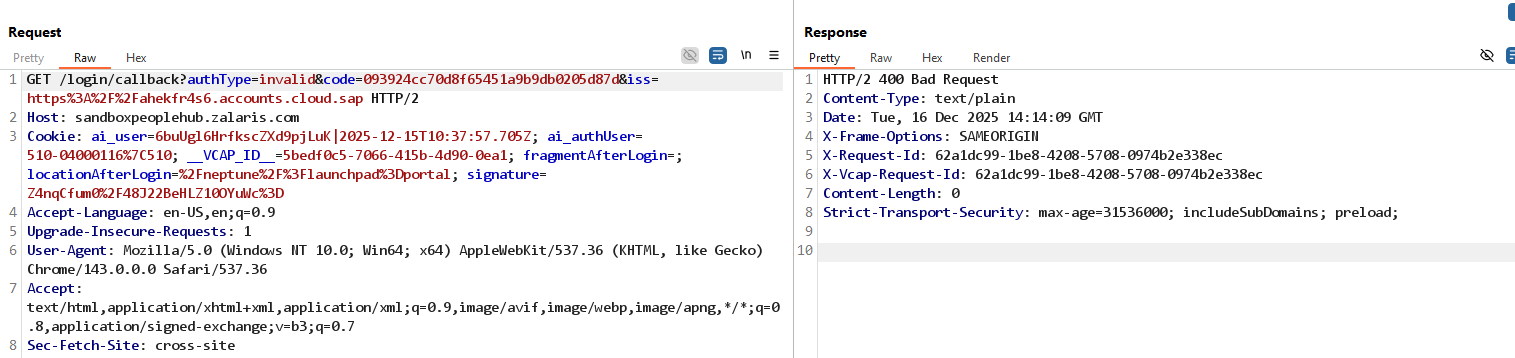
### AuthType Tampering

**Description**

Check whether the application accepted the random value for AuthType

**Steps**

* Complete a valid login and capture code from URL using Burp
* Change the AuthType value to the following - test , invalid etc
* The server returns 400 bad request and the authentication is not successful
* Refer the screenshot attached below for evidence



Test whether the following values are accepted by authType parameter

* authType=local
* authType=basic
* authType=password
* authType=sso
* authType=oidc
* authType=saml

The server is not accepting different values for AuthType and returns 400 Bad Request



**Issue Found:** None

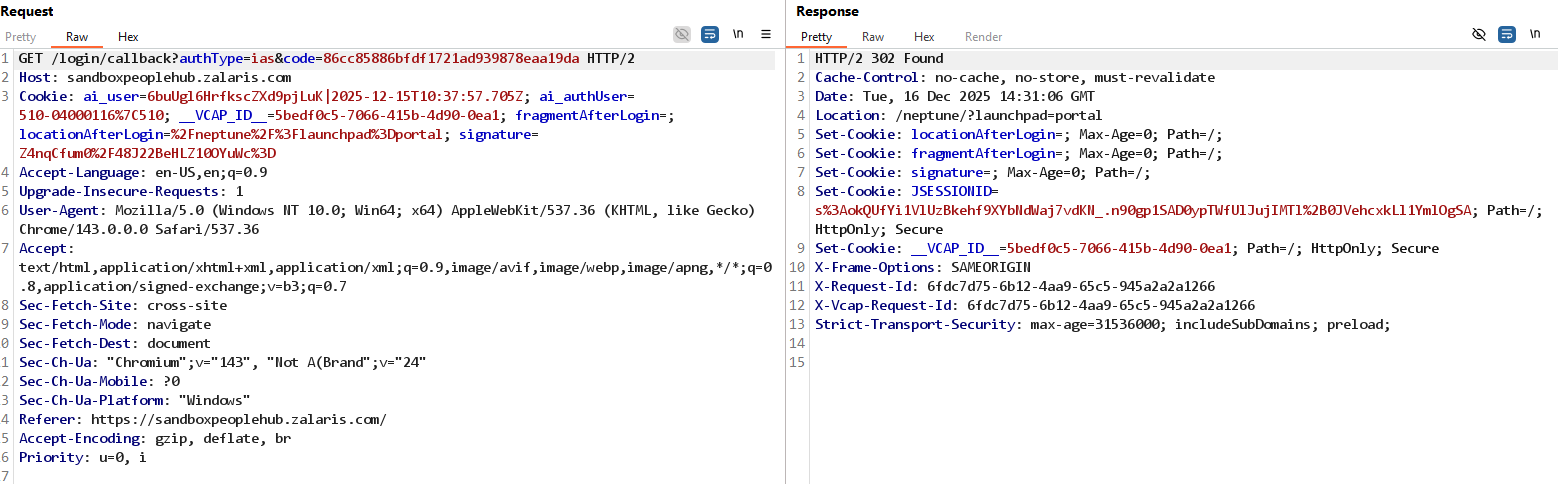
### Issuer Omission

**Description**

Check whether the application accepts the request without the issuer parameter

**Steps**

* Complete a valid login and capture code from URL using Burp
* Remove the issuer and forward the request
* The server is accepting the request and is giving 302 found response



* The issuer value is not varified during the callback process
* This behavior can be accepted is the server is not validating the issuer from the client side but only from the idP provider and the server side configuration

**Issue found:** None

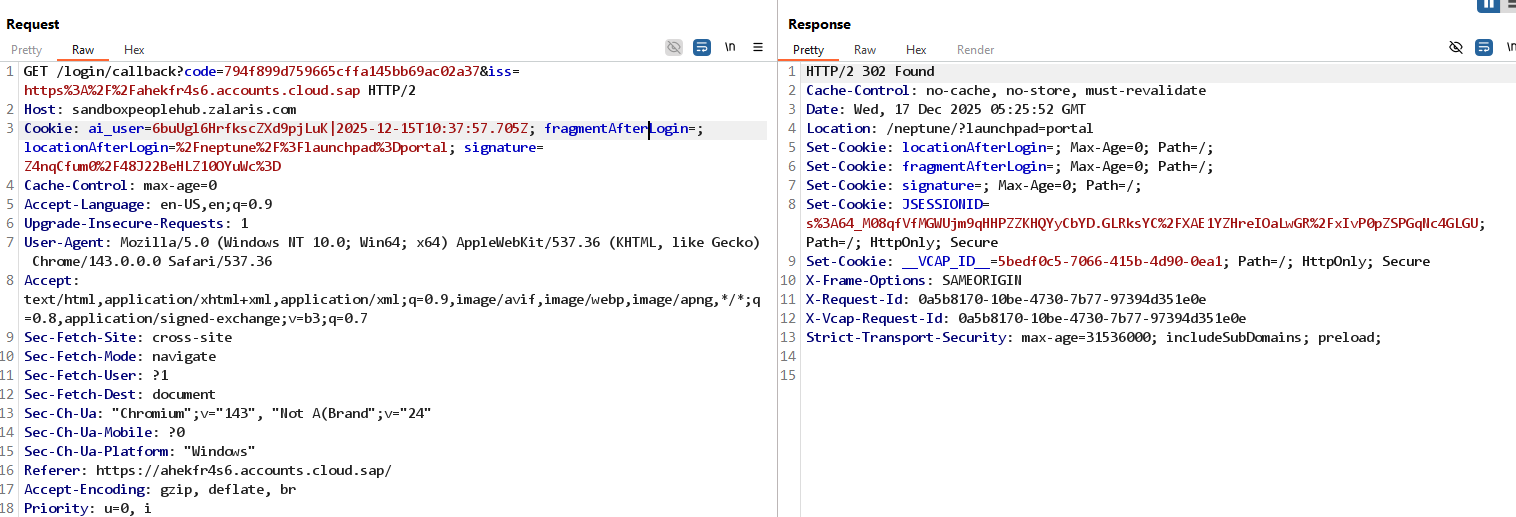
### Missing Authentication Type

**Description**

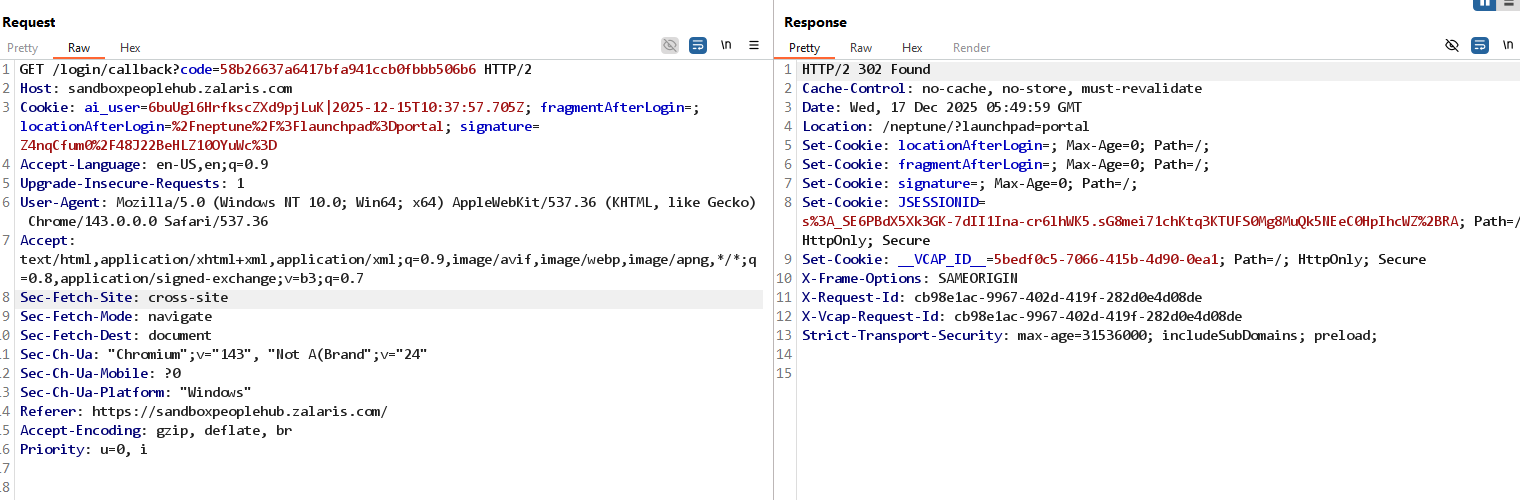
Verify whether the application checks for authentication types

**Steps**

* Complete a valid login and capture code from URL using Burp.
* Remove the auth-Type parameter from the request and forward it.
* The server accepts the request and sends 302 Found response.
* Refer Screenshot attached below for evidence



The application is only validating the code parameter and no other parameters are checked by the server during callback request



**Issue Found:** None [since the client side auth-Type and iss are not accepted ]

**Rate-limit Check**: set on call back since the callback will not be accepted without valid code parameter

**Logging of sensitive information**: The code value can be found in the request parameter , but can be used only once and expires within short span of time so cannot be used even if retrieved from browser

### Code Bound to Client Check

**Classification**

* **OWASP Top 10:** A2 – Broken Authentication
* **OAuth 2.0 RFC 6749:** Section 10.10 – Authorization Code Substitution
* **Severity:** **Critical**
* **Attack Complexity:** Low

**Privileges Required:** None (post-auth code interception)

**CVSS Vector** : CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N

**CVSS Score** : 9.6 Critical

**Description**

Verify whether the code is bound to particular client or particular User

During authentication testing, authorization codes issued for different authenticated users were intercepted and substituted using an interception proxy during the callback request. The application successfully authenticated each session using the substituted authorization code, resulting in User A being logged in as User B and vice versa. This behavior demonstrates that authorization codes are not bound to the original authentication request or browser session, allowing authentication context confusion and account takeover without requiring client-side manipulation.

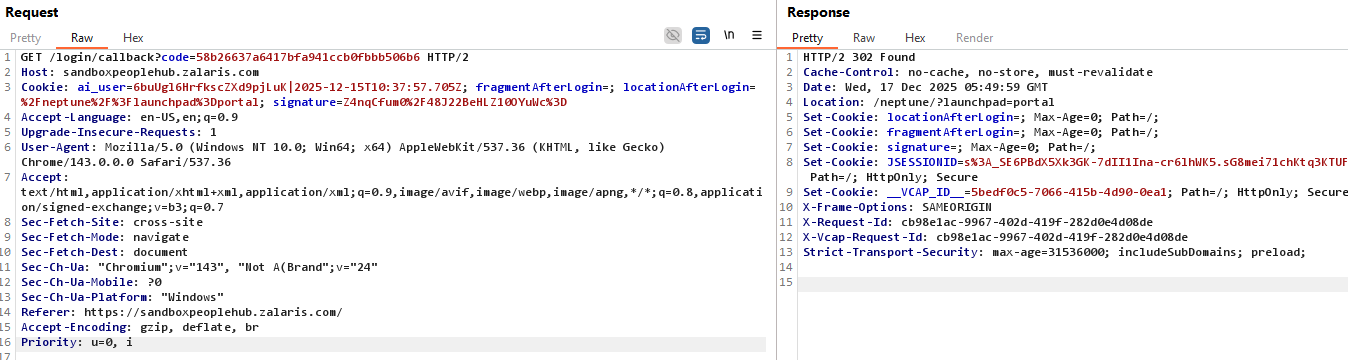
**Scenario:** whether ‘User A’ code can be Used by ‘User B’

**Steps**

* Complete a valid login till User A gets successfully authenticated from the idP
* Capture the callback request from idP to sandboxpeoplehub.zalaris.com
* The request confirms that code parameter is passed onto the server for validation
* Refer screenshot attached below for evidence

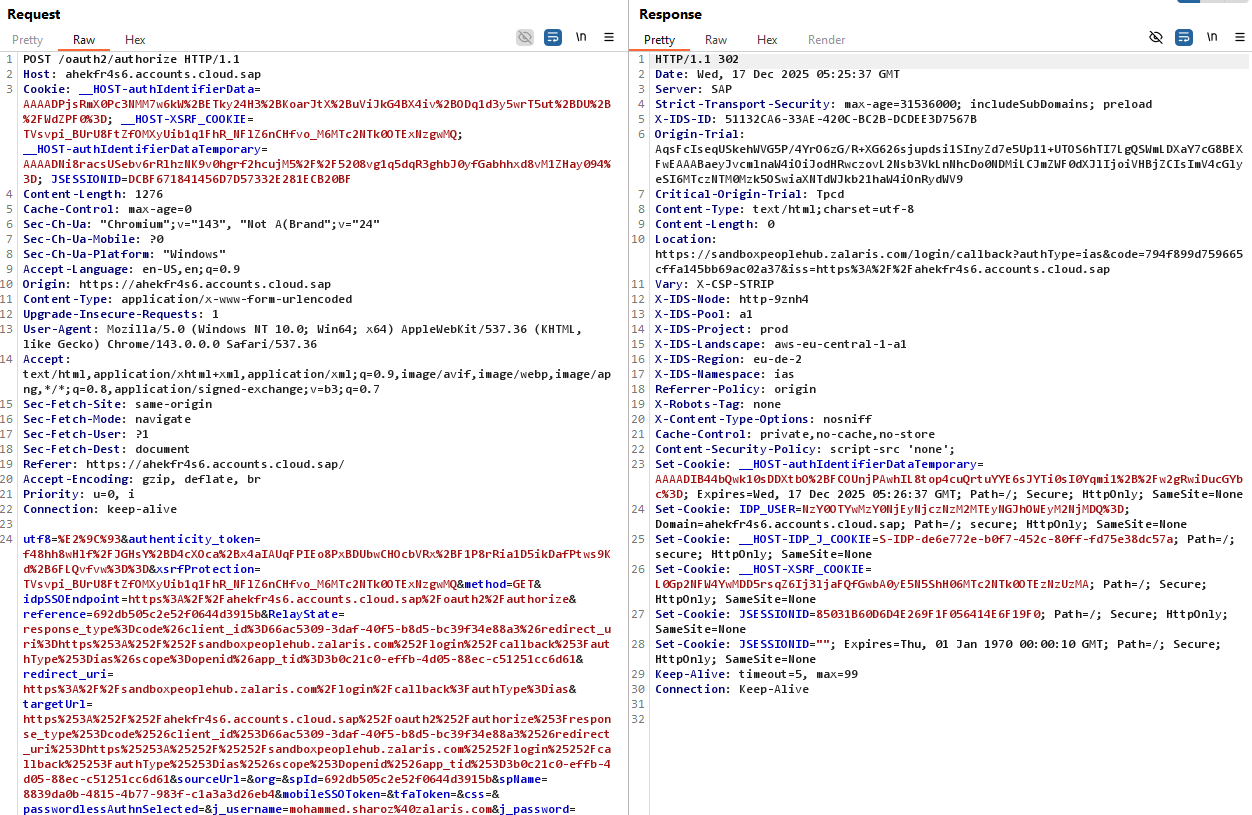


* Repeat the above steps and capture the ‘code’ for User B
* Refer screenshot attached below for evidence

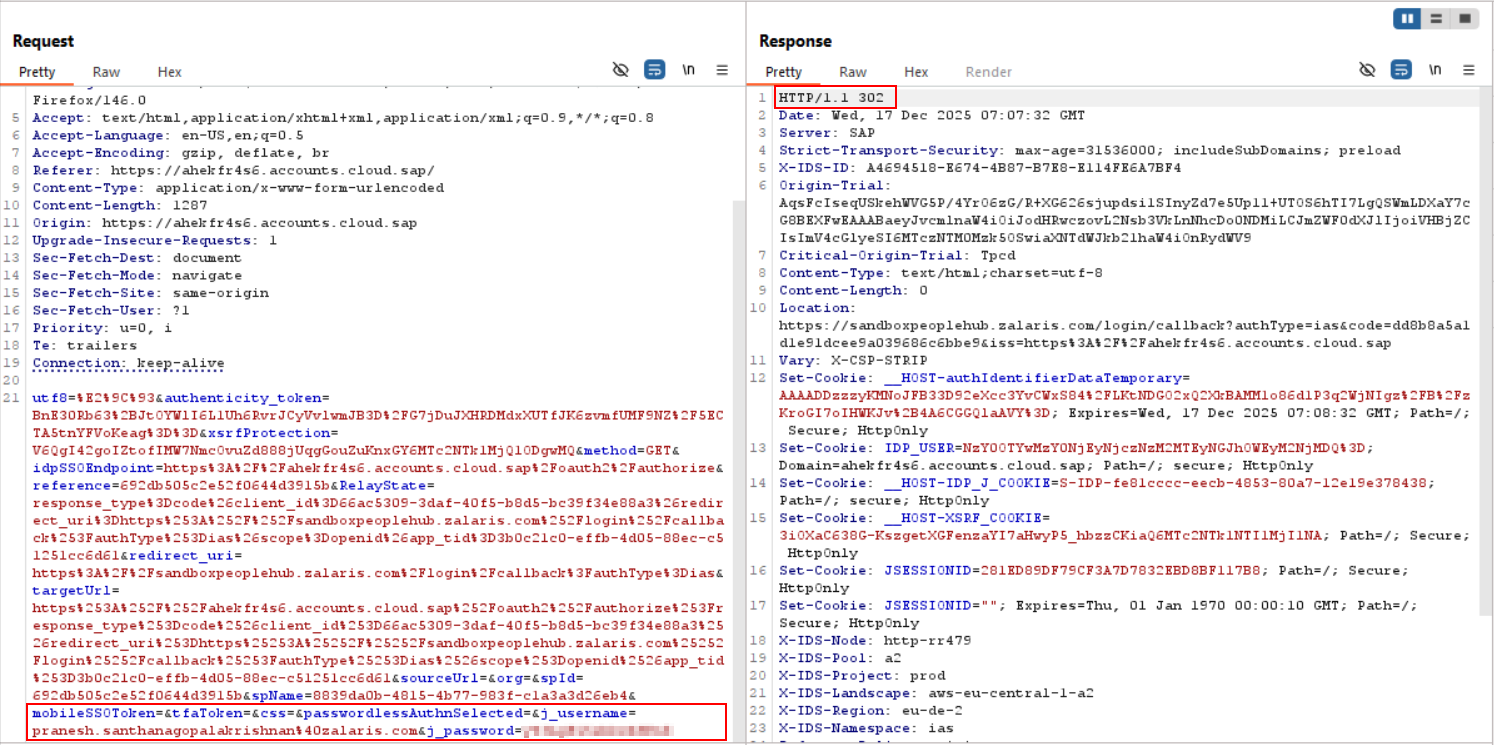


* The authentication process for User A and User B given below

**User A Authentication Process**



**User B Authentication Process**

****

The location header in the server response points to call back request. The request contains the authorization code

Capture the authorization code for different users

Swap User A’s code with User B and forward the request

The authentication process should fail and must validate the “**authentication code to session binding**”

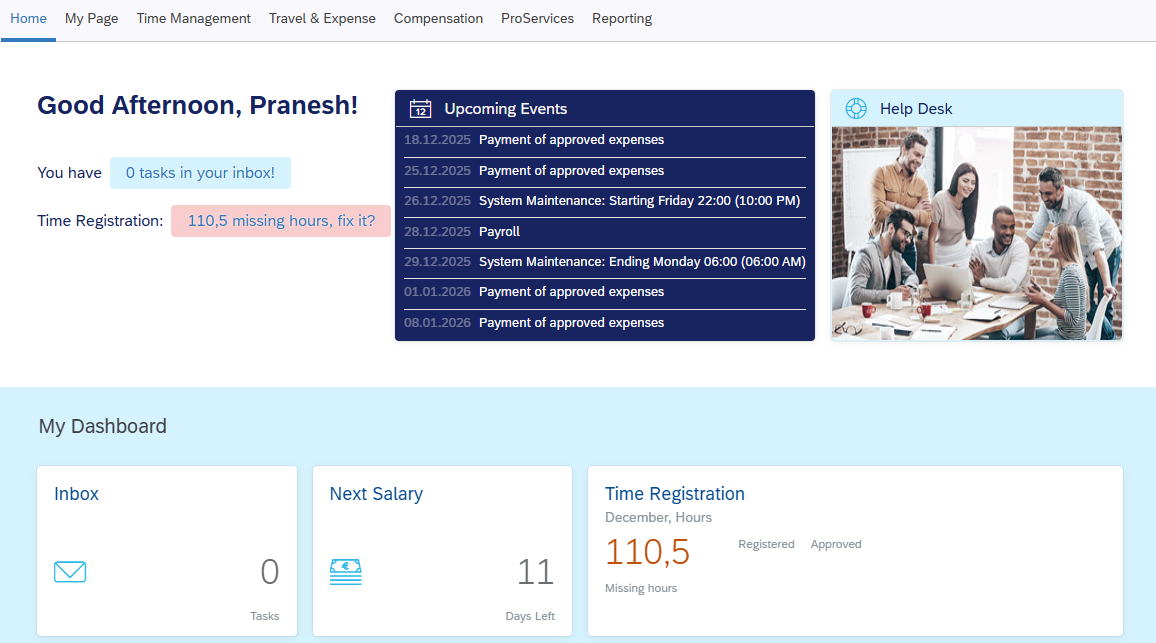
Instead the server responds with 302 Found response which indicates the following

* No abnormal activity has been detected by the server for the current request
* The callback request and path is valid
* Server-side logic processes the particular request
* Redirect continues the flow

The authentication succeeds and “User A” is able to login as “User B” leading to account take over

Refer screenshot attached below for evidence

**Home Page of User B**



The authorization code is sent via the request URL and this can be captured via an MITM attack and once the attacker gets hold of the code can replay it after his authentication process to takeover the victim’s session

**Issue Found:** Authentication code to session binding is broken leading to account take over. This can be treated as Critical as

* Attacker needs **only one stolen code**
* No victim interaction required
* No cookie manipulation required
* Works cross-session, cross-user

This confirms that code is treated as a bearer token with no contextual binding.

**Classification**

* **OWASP Top 10:** A2 – Broken Authentication
* **OAuth 2.0 RFC 6749:** Section 10.10 – Authorization Code Substitution
* **Severity:** **Critical**
* **Attack Complexity:** Low
* **Privileges Required:** None (post-auth code interception)

**One-Line executive impact statement**

An attacker who obtains a valid authorization code can authenticate as another user without their credentials, leading to full account takeover.

Race condition was tested and further test is needed together with Pranesh

**Proceed with the ESS to MSS privilege escalation process**

### Cacahed callback Response

**Description**

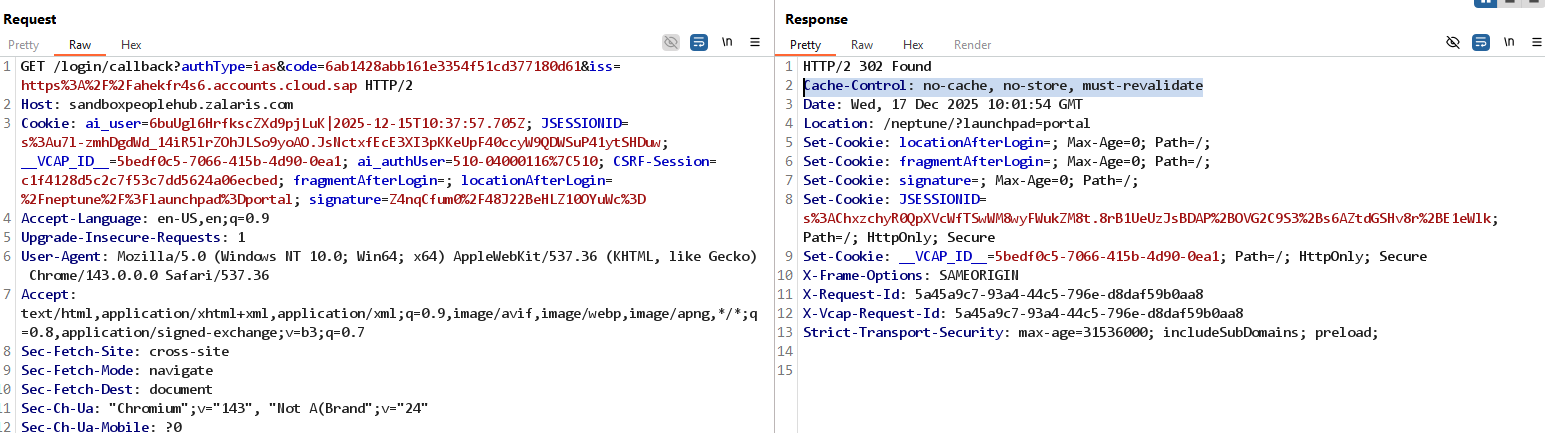
Verify whether the callback response is cached by the browser , proxy or intermediary

The callback responses must never be cache able because they

* Contain authorization codes
* Set authentication Cookies
* Perform Security Critical redirects

**Steps**

* Complete a valid login and capture code from URL using Burp
* Analyze the response headers and look for cache control headers
* Refer screenshot attached below for server side cache control value



* The server response shows that the Cache-Control headers are set with value n-store , no-cache and must-re validate

**Issue Found:** None cache control security header is properly set

### Verification of User-to-Resource Binding Post Authentication

**Objective:** Verify that an authenticated user cannot access files or messages belonging to another user

**Description**

This test verifies that authenticated users are restricted to accessing only resources explicitly associated with their own identity. The objective is to confirm that, after successful authentication, the application correctly enforces authorization controls and prevents a user from accessing files, messages, or other protected resources belonging to a different user account. The test evaluates whether user identity and session context are consistently validated across back-end endpoints and whether resource access decisions are properly bound to the authenticated user

**Targeted functionalities and Requests**

The following application functionalities and HTTP requests were targeted during this test:

**Inbox Tile**

* Test whether the User B’s inbox can be accessed by User A

**eFile**

* Test whether User A is able to access User B’s File

**File download features in the following Tiles**

* Coworkers Tab
* Work Location Over View

**Travel and Expense Report**

* Can User A access User B’s travel and expense report

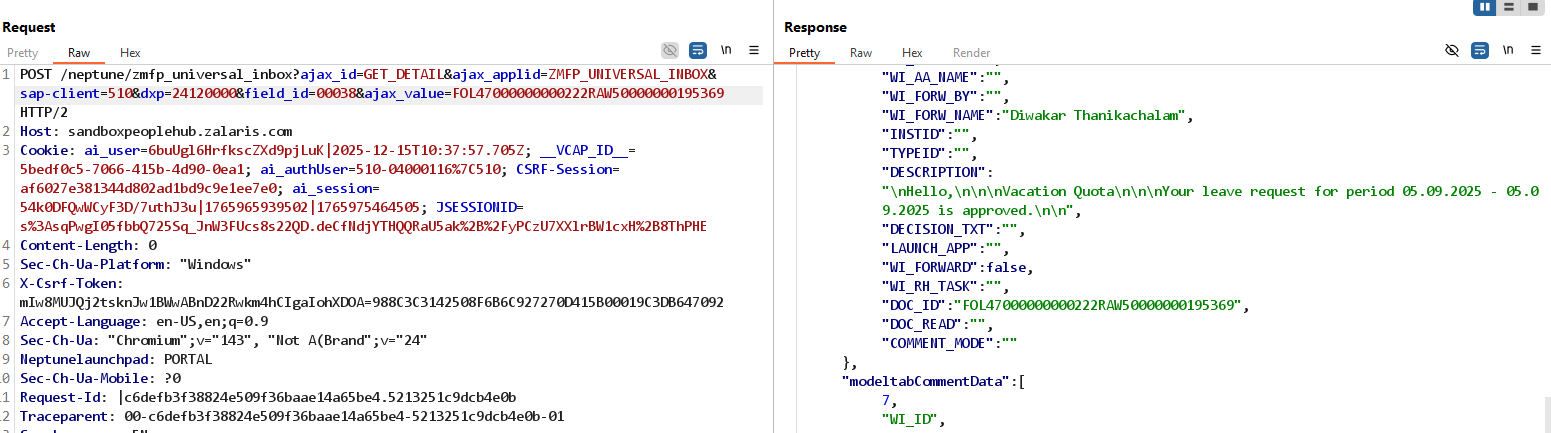
**SAP Password Reset**

* Can user A reset User B’s SAP Password

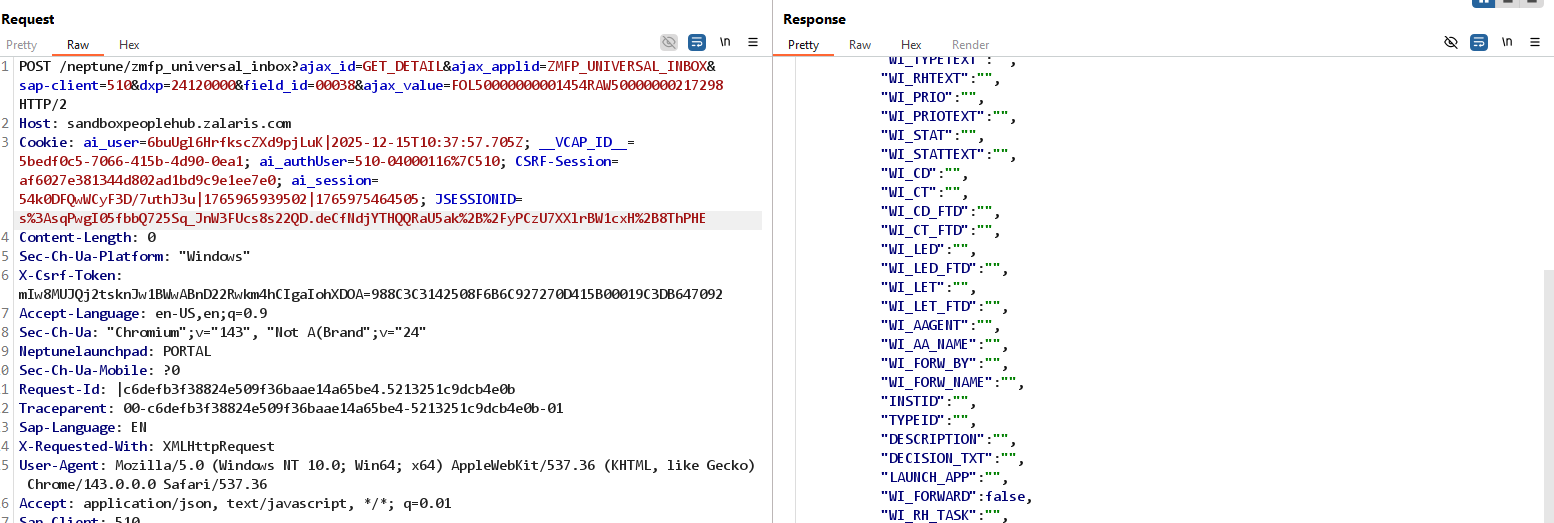
**Test 1 - Inbox Tile**

**Steps**

* Complete a valid login
* Access the inbox tile. Open an inbox message and capture the request
* The file is opened using the ajax\_values id
* Refer screenshot attached for evidence



* Capture the inbox request from User B swap the ajax\_values and check whether User A can read User B’s inbox file
* The server responds with empty data indicating that the file is not returned from the server
* Refer Screenshot attached below for evidence

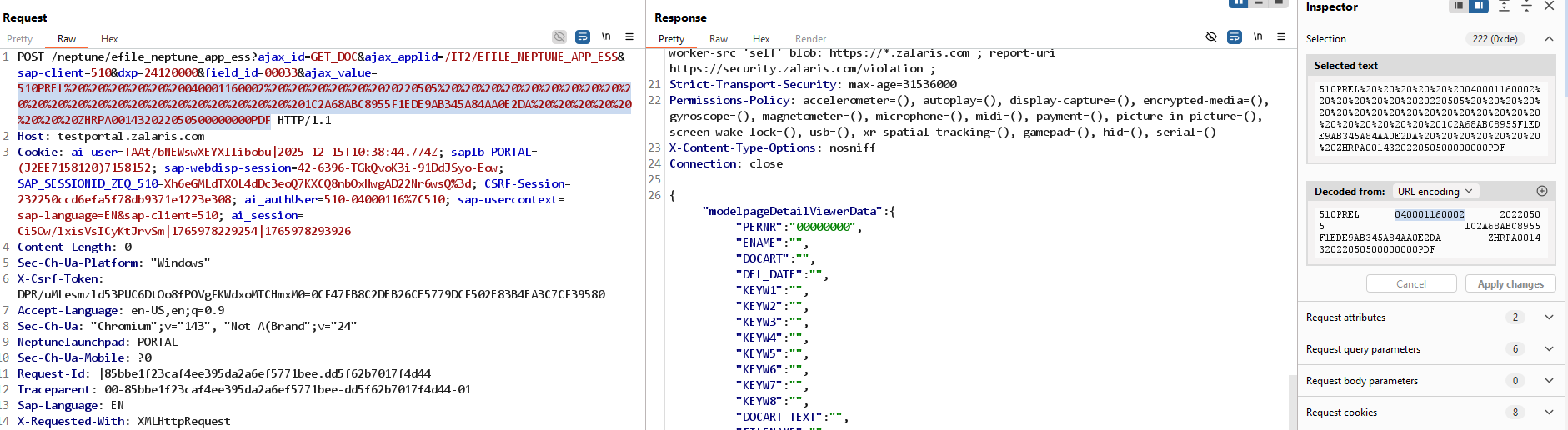


**Issue Found:** None, User A is not allowed to access User B’s inbox file.

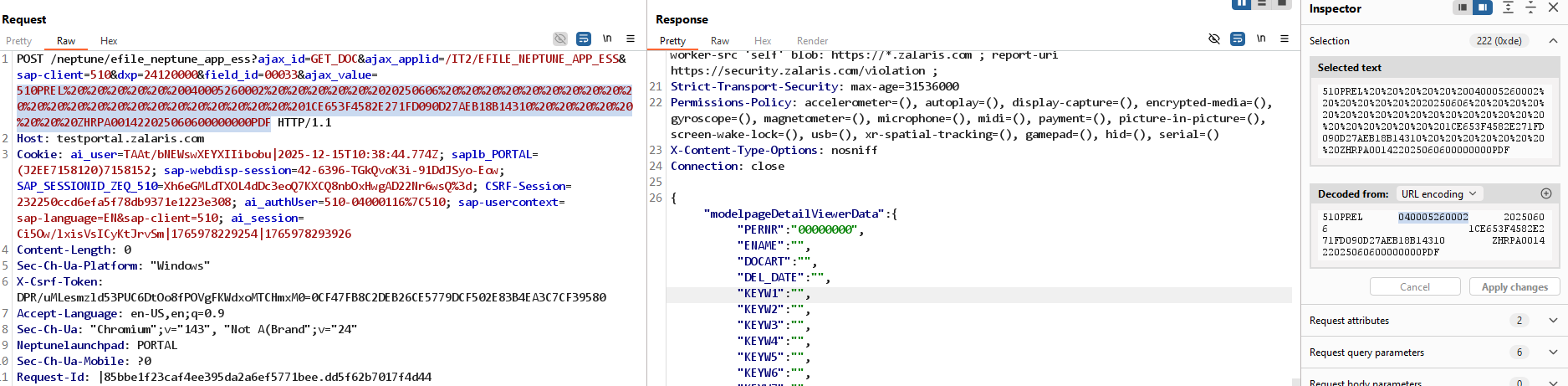
**Test 2 - eFile**

**Steps**

* Complete a valid login
* Access the eFile Tile
* Access any file and capture the request , the file is identified by the server using the ajax\_values
* Refer screenshot attached below for evidence



* The file associated with the user is not fetched successfully
* Replace the ajax\_value of User A with the ajax\_value of User B and forward the request
* The server still responds with the same data
* Refer Screenshot attached below evidence



**Issue Found:** None, the valid process is not completed successfully no files are fetched for normal request sent as well as the altered request sent

**Test 4 - SAP Password Reset**

**CVSS 7.1 High [**if password is not changed**]**

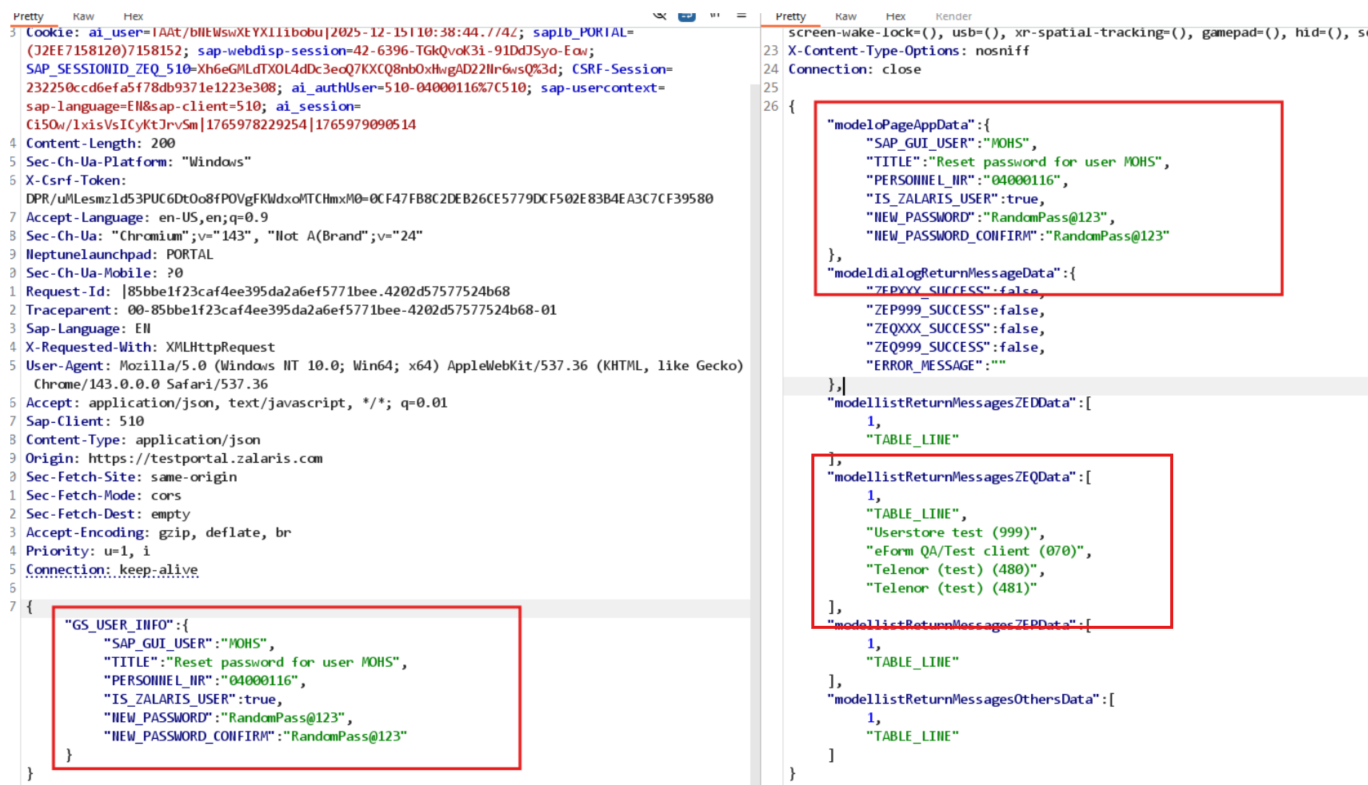
**CVSS 8.8 High [**If password changes and can lead to acoount takeover**]**

**Description**

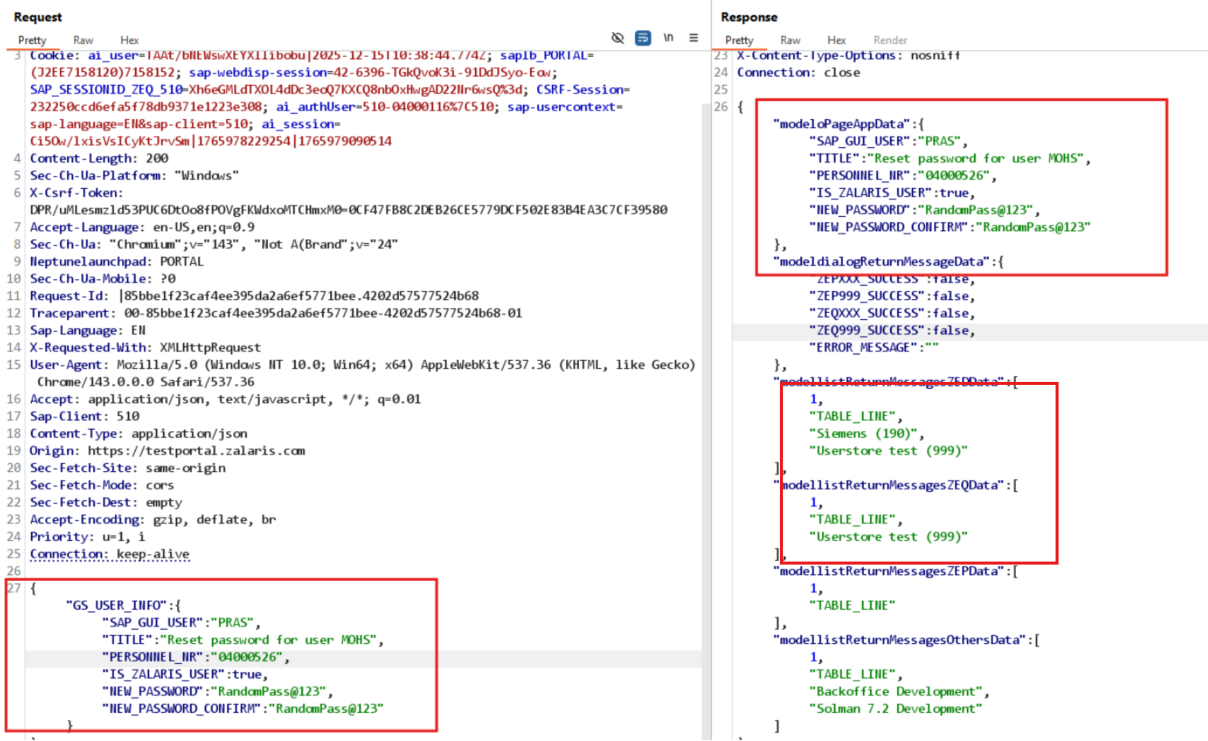
The password reset functionality intended for the currently authenticated user fails to enforce object-level authorization. By manipulating the username parameter, an authenticated user can retrieve another user’s client assignments and active access status. This indicates that the backend relies on client-supplied identity parameters rather than the authenticated session context. The issue constitutes a Broken Access Control (IDOR) vulnerability and results in sensitive authorization data exposure. Depending on backend behavior, this flaw may also enable unauthorized password resets and account takeover

**Steps**

* Complete a valid login
* Access - access management tab
* Proceed with SAP password reset for user MOHS and Capture the request
* Refer screenshot attached below for evidence

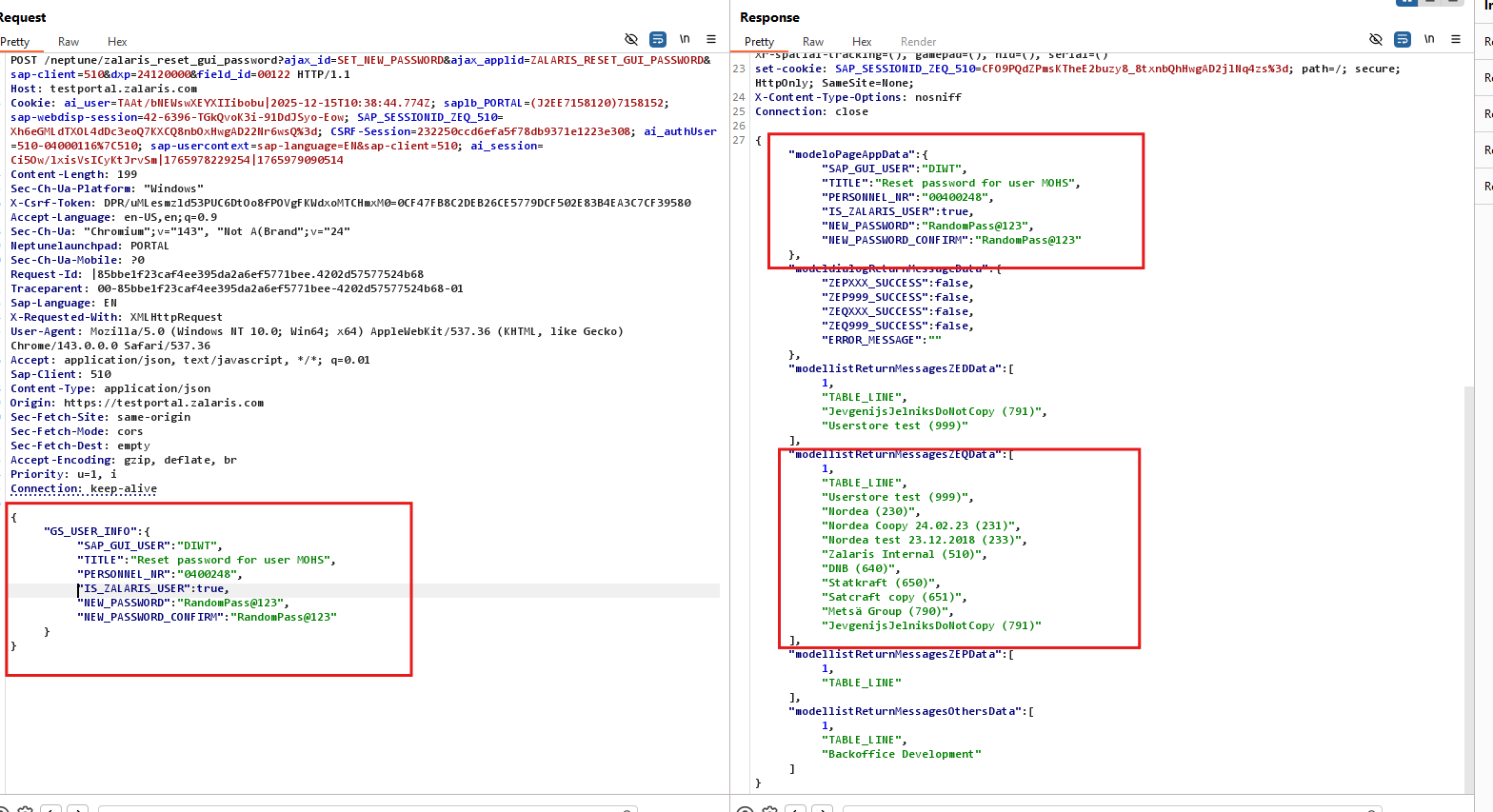


* Alter the following parameters to point to User B
  + SAP GUI USER
  + PERSONNEL IR
* Forward the request
* The server returns with 200 OK response
* Refer screenshot attached for evidence



The request is successful and also fetches the test user details for the target user

Refer screenshot attached below for evidence



**Issue Found:**

* User A can reset password for User B
* The server gives information on the test user client for User B to User A when password reset was prompted

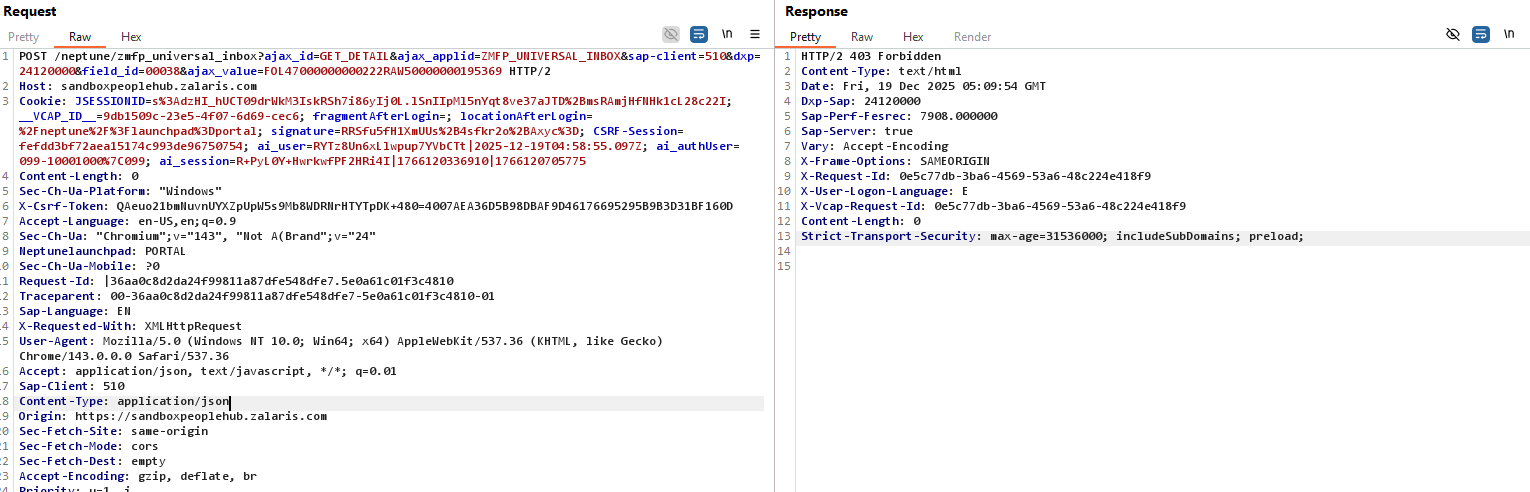
### User 099- try to access MOHS account details

Access Inbox

099 try to access MOHS inbox

**Steps**

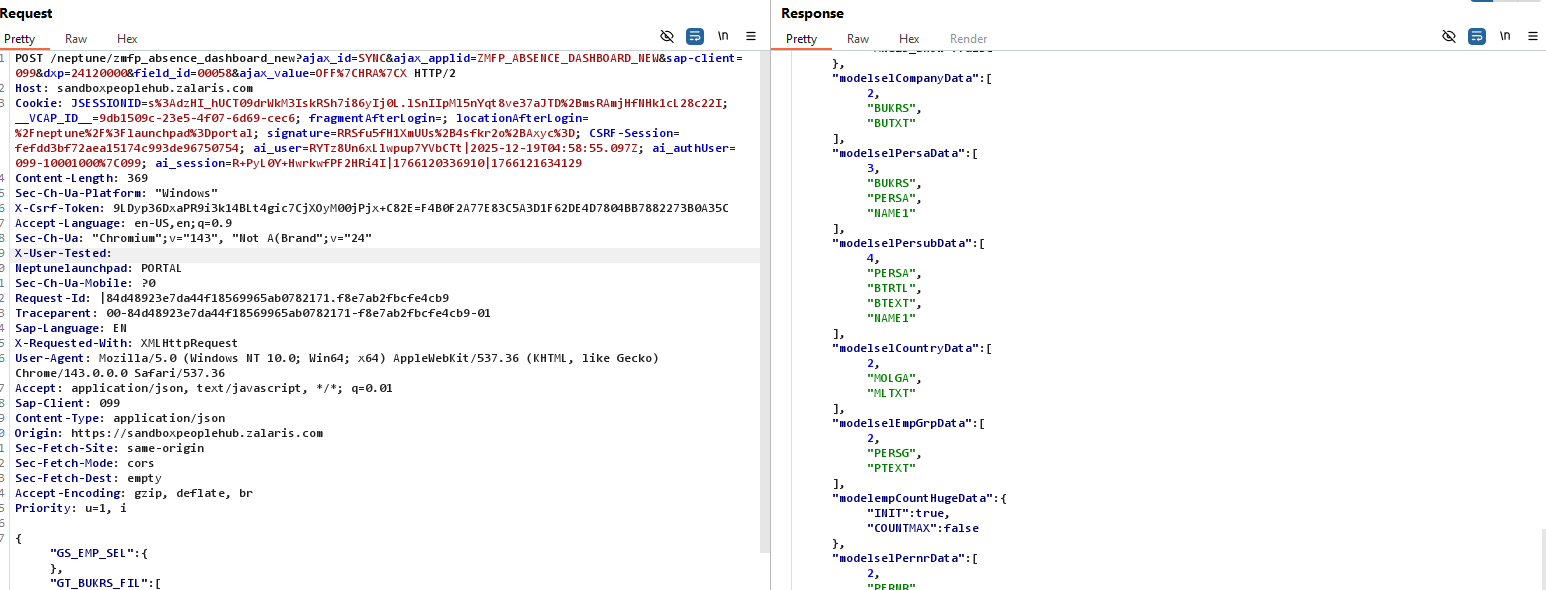
* Complete a valid login
* Access the inbox tile. Open an inbox message and capture the request
* The file is opened using the ajax\_values id
* Try access the message of user MOHS by using 099 User
* User 099 receives 403 Forbidden error confirming that cross user inbox access is forbidden
* Refer Screenshot attached below for evidence



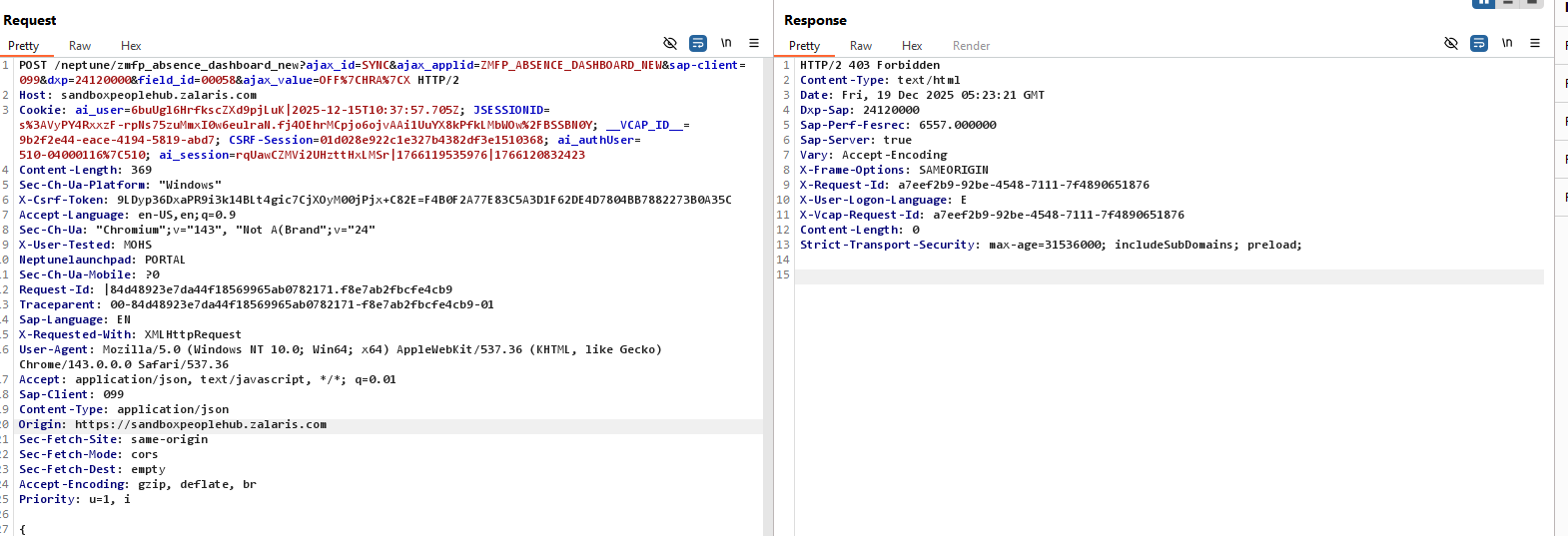
Access absence overview

**Steps**

* Complete a valid login
* Access the inbox tile. Open an inbox message and capture the request
* The file is opened using the ajax\_values id
* Access the absence overview for user 10001000
* The absence overview access is success
* Refer screenshot attached below for evidence



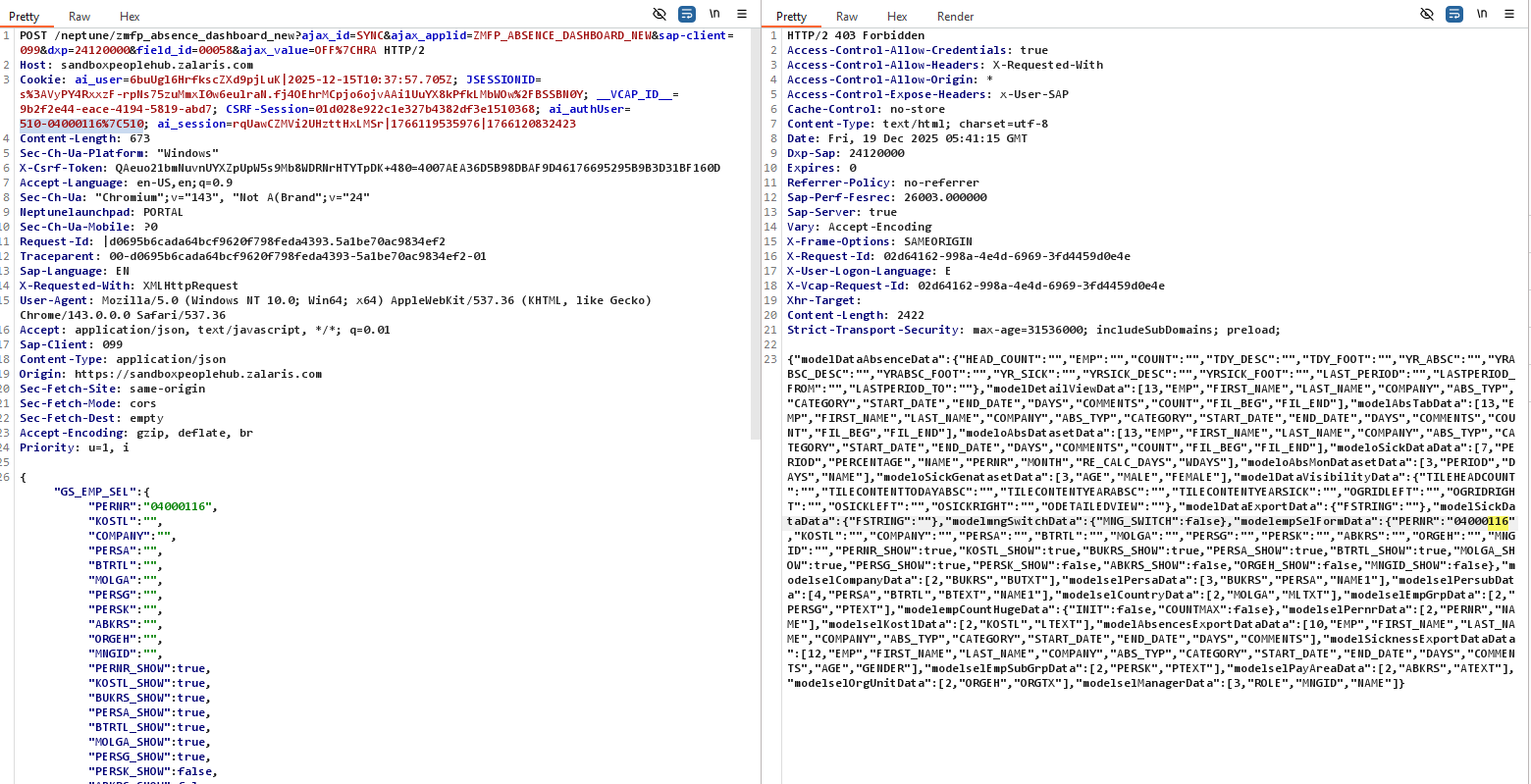
* Access the absence overview for user 10001000 user the cookie for User 04000116
* The absence overview access is forbidden
* This is expected behavior
* Refer screenshot attached below for evidence



Alter both the cookie and the X-Csrf-token and forward the request

The server now gives back data with the response model together with the value 04000116 reflected in the response

Refer screenshot attached below for evidence

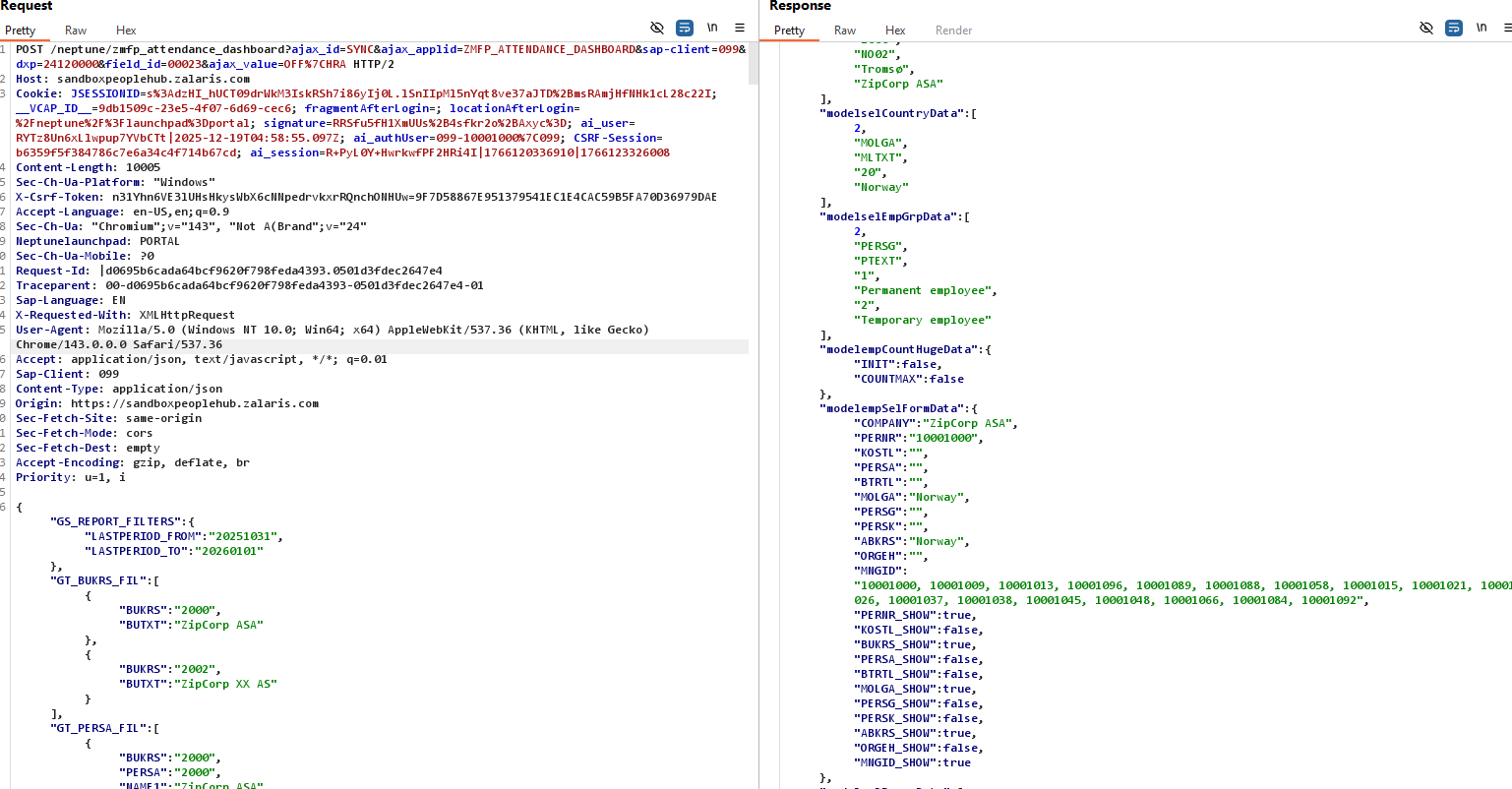


**Issue Found** : The server returns data from the server even though it has 403 Forbidden response

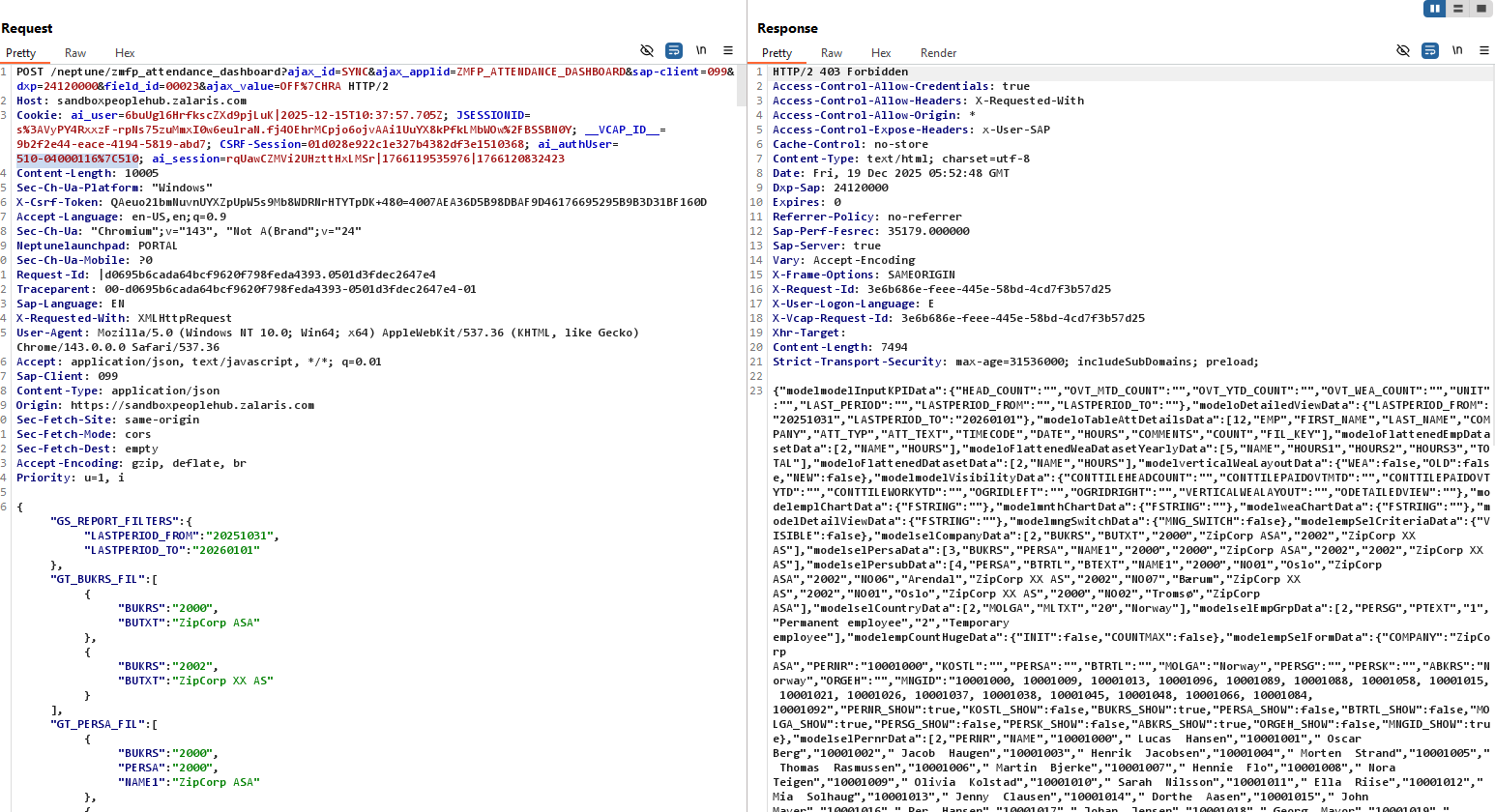
**Overtime overview**

**Steps**

* Complete a valid login
* Access the overtime overview tile using user 10001000 and capture the request
* Refer screenshot attached below for evidence



* Now alter the authentication cookie of user 10001000 with the cookie for user 04000116 and forward the request
* The request is successful even though the user received 403 forbidden error from the server still the data for user 10001000 are received by user 04000116
* Refer screenshot attached below for evidemce

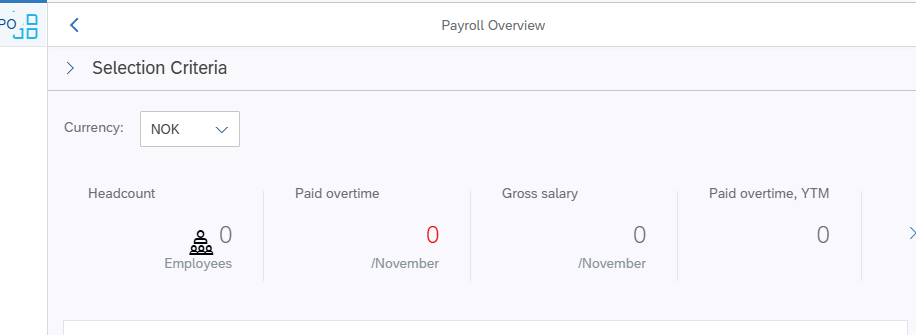


**Issue found :** user 04000116 is able to access user 099-10001000’s data without authentication or cookies needed for user 099-10001000 and this must not be accepted

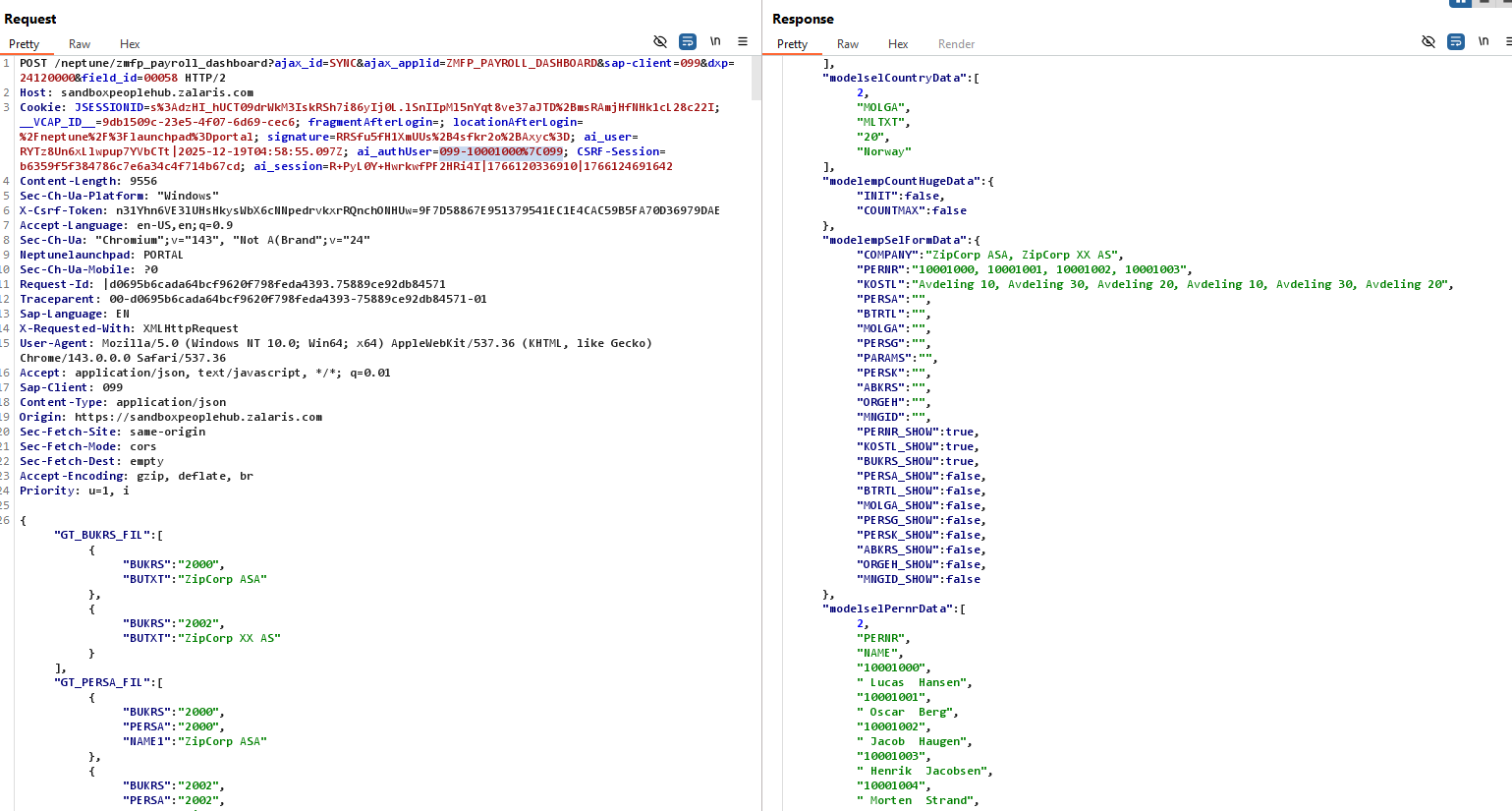
**Payroll Dashboard**

**Steps**

* Complete a valid login
* Access the payroll dashboard tile using user 10001000 and capture the request
* Refer screenshot attached below for evidence



* The request and the valid response are present in the screenshot attached below



**Issue Found:** User 510-04000116 is able to access the data for user 099-10001000in the payroll dashboard

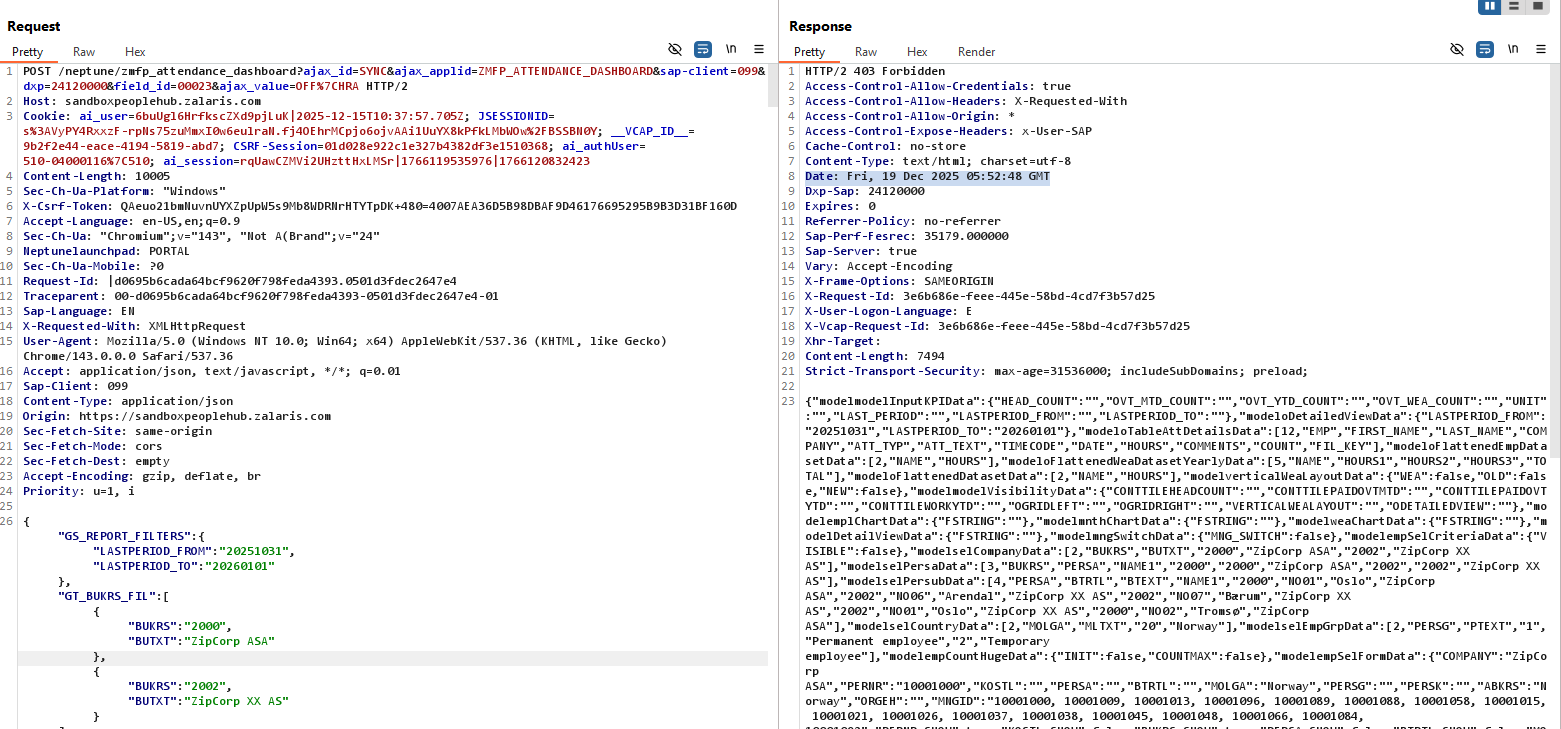
Lets try the other way

Whether user 099-10001000 is able to access the data for user 510-04000116

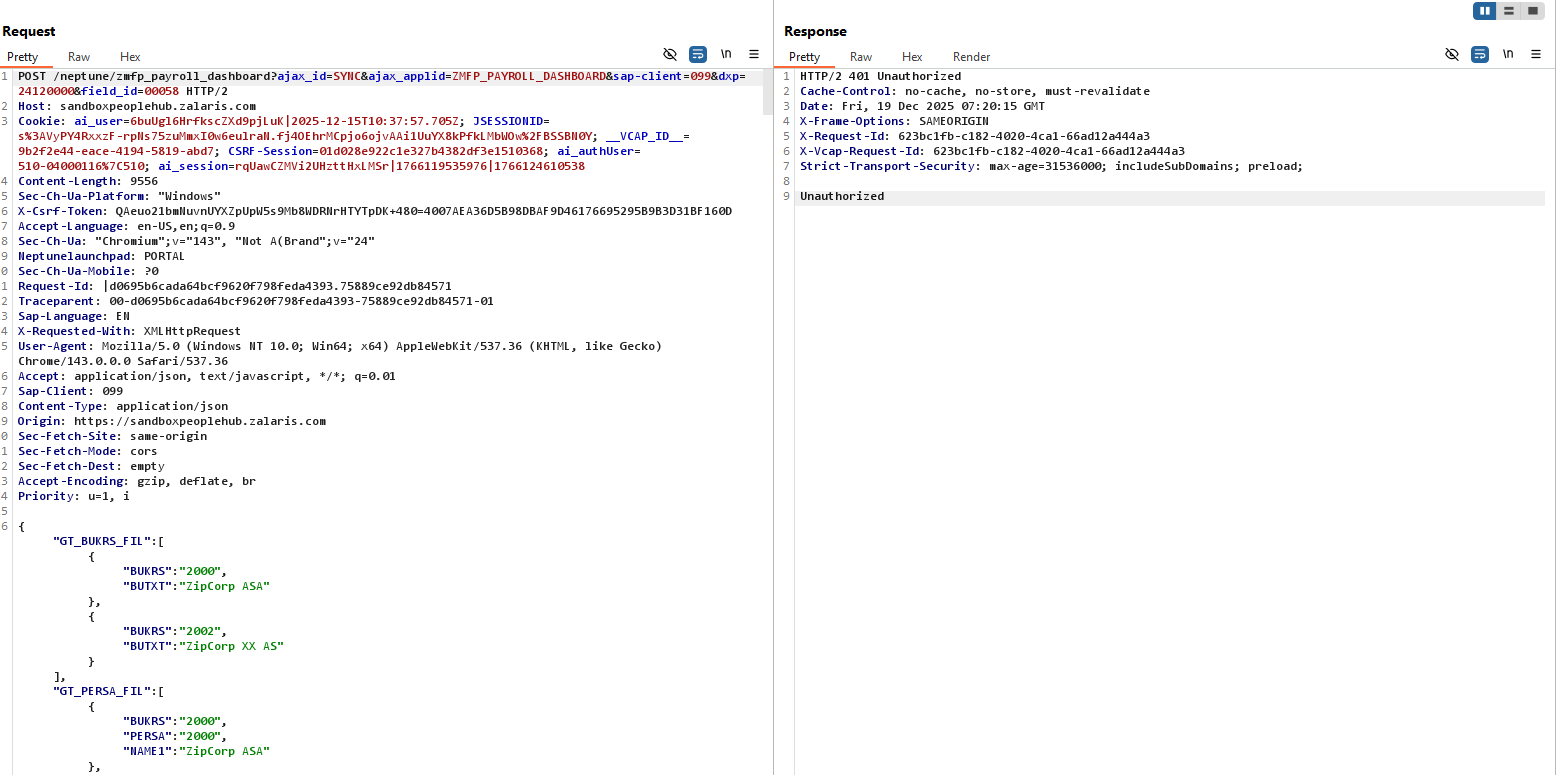
**Test Result:** The server gave 403 Forbidden Error and then it gave 401 Unauthorized

**Note:** server started to behave differently from 12:00 PM IST and started to return data as 401 Un-Authorized rather than the noted last response where the server gives 403 Forbidden and does not give out

Screenshot below gives the GMT Time and Date when the data is returned from the server



**Time stamp when the server started to throw 401 unauthorized error**



**Cookie Check**

Cookies at hand :

* VCAP ID
* JSESSION ID

| **Attribute** | **Expected** | **Why** |
| --- | --- | --- |
| Secure | ✅ Enabled | Prevents MITM |
| HttpOnly | ✅ Enabled | Prevents XSS theft |
| SameSite | Lax or Strict | CSRF defense |
| Path | Restricted | Scope control |
| Domain | Not overly broad | Cross-app abuse |

VCAP ID :

Value : 9db1509c-23e5-4f07-6d69-cec6

HTTP Only:Yes

Secure :Yes

Expires on : session

Samesite:

JSESSIONID: s%3Ak0js828y0K9o8qjCGmloGDYqTGXZ7TEw.FkWaUrgRrdV%2Bai32abgSNFZfz1kEzsP6VaXXGf7680k