

[REDACTED CLIENT]

Security Assessment Report

TARGET

PERIOD

AUTHOR

VERSION

[Redacted Client]

2026-09-14 —
2026-09-14

Haxact

1.0

TABLE OF CONTENTS

Executive Summary	3
Scope & Methodology	6
Scope	6
Scope Exclusions	6
Methodology	6
Disclaimer	7
Severity Classification	9
Severity Levels	9
Exploitation Complexity	10
Exposure	10
Findings	11
Overview	11
Critical Severity (2 findings)	13
High Severity (9 findings)	17
Medium Severity (4 findings)	31
Low Severity (5 findings)	37
Informational Severity (5 findings)	43
Coverage Matrix	48
Glossary	49
Appendix A: Test Accounts	51
Appendix B: HTTP Evidence	52

EXECUTIVE SUMMARY

Headline findings

- CRITICAL** Account takeover of a bank officer: PUT /api/Auth/UpdateUser/{id} lets a low-privilege introducer rewrite a victim's email, then reset their password and log in as them
- CRITICAL** Privilege escalation to Admin via roles mass-assignment on PUT /api/Auth/UpdateUser/{id}
- HIGH** Cross-tenant BOLA: POST /api/workflow/{id}/submit transitions another org's application with no ownership check
- HIGH** Cross-tenant IDOR exposing another org's KYC application document meta-data

Recommended priorities

- 1. Add object- and function-level authz to /api/Auth/UpdateUser/{id} and drop client-supplied roles/approvalStatus from the model**
One endpoint drives both criticals and two highs (ea5329d1, 5d9ee1ee, e5d1c244): any introducer can rewrite another user's email/roles/approval status, enabling bank-officer account takeover and self-promotion to Admin.
- 2. Enforce application ownership on all /api/workflow/{id}/* routes, matching the 403 already returned by the readiness/timeline endpoints**
The workflow controller has no object-level authz, so an introducer can submit or start another tenant's application (2af4f02d, 7ef115fc); app ids are small sequential integers and trivially enumerable.
- 3. Scope all application/document/directory read endpoints to the caller's org and role; treat leaked GUIDs as non-authorization**
Cross-tenant reads expose KYC application metadata and user PII across orgs and roles (958f82bb, f5c6837e, a6d943bf, f1c325b9); GetAllUsers also leaks the GUIDs that enable the AT0 and 2FA-abuse chains.
- 4. Validate uploaded file type/content and serve user documents from a sandboxed origin with a non-renderable Content-Type**
Stored XSS via document upload (311c6b56) and unauthenticated stored injection into shared CMS pages (63da3ea3) let attacker-controlled markup execute in the SPA origin where KYC data is handled.
- 5. Gate all /api/admin/* endpoints (rulesets publish, Entity tree, security settings) behind an explicit Admin role check**
Introducer accounts reach admin-only functions (3a21b086, f489aef0) and can publish rulesets; combined with the roles mass-assignment this is a direct path to platform-wide configuration control.

Coverage notes

Well-tested surfaces

- Cross-organization and cross-role IDOR / BOLA across applications, documents, users and groups (attacks driven from the least-privileged OwnOnly introducer accounts; Org A vs Org B)
- Vertical privilege escalation and broken function-level authorization (introducer -> admin / officer functions)

- Auth & session: JWT handling, logout revocation, password-reset flow, the 2FA state machine, and user enumeration
- File upload: type/size validation, stored XSS, path traversal, and cross-tenant write
- Onboarding-workflow integrity: step-skipping, readiness gates, and decision/approval tampering
- Web cache deception, race conditions/TOCTOU, and SignalR hub authorization

No verified findings in

- SQL injection (EF Core parameterization throughout)
- CSRF (SameSite=Strict plus Origin/Referer allow-list)
- Path traversal / LFI (blocked at Cloudflare, IIS and the input filter)
- Insecure deserialization
- Subdomain takeover (candidate origin identified but not shown to be claimable)

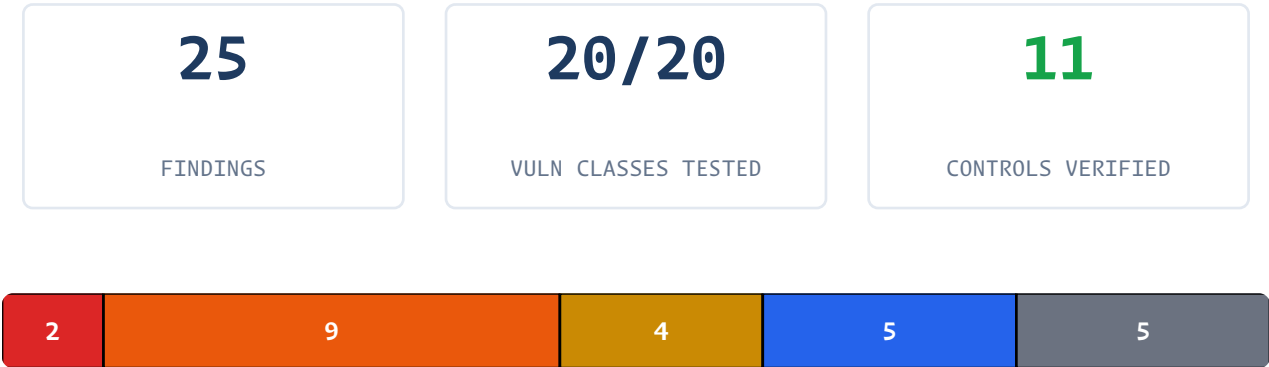
Gaps and limited testing

- The '.full' (Full org-visibility) introducer accounts were exercised only as IDOR targets, not as attacker sessions; every access-control finding was proven from the more-restrictive OwnOnly accounts, and a dedicated Full-role differential pass was not performed. Because all verified issues are cross-organization or vertical, a Full account would violate them identically.
- The second bank officer (Unit 1 was driven for a differential crawl; Unit 2 / [redacted-user] was not driven as an acting session), so an officer-to-officer (Unit 1 <-> Unit 2) differential was not specifically exercised.
- No 2FA-enabled test account was available, so the client-side 2FA-bypass hypothesis could not be fully confirmed (the unauthenticated 2FA state-machine abuse was confirmed).
- The DEV environment ([redacted-host]) was only spot-checked; the account-takeover primitive was confirmed to reproduce there but a full DEV pass was not run.
- The subdomain-takeover candidate ([redacted-host]*.example.test) shows an abandoned origin but could not be confirmed claimable (no pre-Cloudflare origin IP recoverable).

CRITICAL

Overall Risk Assessment

2 critical finding(s) identified, including easily exploitable issues.



■ Critical (2) ■ High (9) ■ Medium (4) ■ Low (5) ■ Info (5)

SCOPE & METHODOLOGY

Scope

The following domains were in scope for this assessment. The table below shows the number of validated findings per severity level for each domain.

Domain	Critical	High	Medium	Low	Info	Total
api.example.test	2	9	4	5	5	25
app.example.test	—	—	—	—	—	0
Total	2	9	4	5	5	25

Testing was performed using 6 test accounts.

Scope Exclusions

The following attack categories were explicitly excluded from this assessment:

Denial of Service (DoS/DDoS)

No availability testing, resource exhaustion, or rate-limit abuse was performed.

Social Engineering

No phishing, pretexting, vishing, or other human-targeted attacks were conducted.

Spoofing & Impersonation

No email spoofing, DNS spoofing, or domain impersonation attempts were made.

Physical Security

No physical access testing, tailgating, or hardware-based attacks were performed.

Supply Chain Attacks

Third-party dependencies, build pipelines, and upstream providers were not targeted.

Methodology

This assessment used an AI-driven approach: specialised software agents tested the application against a curated set of web vulnerability classes. Each class – cross-site scripting, insecure direct object reference, injection, CSRF, open redirect, SSRF, business logic, authentication & session handling, subdomain takeover, JWT, local file inclusion, cache poisoning, input validation, and AI agent abuse – was assigned to its own agent operating an authenticated browser session.

Reconnaissance & discovery: before vulnerability testing began, the in-scope

hosts were crawled and the application's authentication flows mapped. Discovered URLs, captured HTTP traffic, and any leaked secrets surfaced during this phase fed into the per-class tests that followed.

Per-class testing: each agent interacted with the application as a real user would – navigating pages, submitting forms, and manipulating requests – while systematically probing for weaknesses in its assigned vulnerability class. Agents had access to captured request/response history and could replay requests with modifications to confirm exploitability.

Verification: every candidate finding was independently re-tested by a separate verifier agent. Only findings successfully reproduced from the recorded evidence are included in this report. The verifier sets the final severity rating independently of the filing agent, mitigating bias toward more dramatic-sounding outcomes.

Where applicable, Android applications in scope were analysed statically: the APK was decompiled and inspected for hardcoded secrets, exported component issues, WebView misconfigurations, and other source-level weaknesses. Findings from this static analysis follow the same verification process as the web findings.

All HTTP traffic between the testing infrastructure and the target was captured and stored. Relevant request/response pairs are included as evidence alongside the findings.

Denial-of-service, social engineering, and physical security testing were excluded from scope. The Scope Exclusions section provides further detail.

Disclaimer

This assessment represents a point-in-time evaluation of the target application's security posture as observed during the engagement period. The results reflect the state of the application at the time of testing and do not account for changes made after the assessment concluded.

The absence of findings in a given vulnerability category does not guarantee the absence of vulnerabilities. Testing coverage is influenced by the complexity of the application, the available attack surface, time constraints, and the scope defined for this engagement. Certain areas of the application may not have been fully exercised during testing.

Denial-of-service, social engineering, physical security, spoofing, and supply chain attacks were explicitly excluded from scope and were not attempted.

We recommend periodic re-testing, particularly after significant application changes, infrastructure updates, or the introduction of new features. Security is an ongoing process, and regular assessments help ensure that new vulnerabilities are identified and addressed promptly.

This report is confidential and intended solely for the named recipient. Distribution to unauthorized parties is not permitted without prior written consent.

SEVERITY CLASSIFICATION

Every finding in this report is classified using a structured severity model based on three factors: how easily the vulnerability can be exploited (exploitation complexity), who can reach the vulnerable endpoint (exposure), and what happens if it is successfully exploited (business impact). These three factors are combined to assign a severity level ranging from Critical to Informational.

This classification helps prioritize remediation efforts. Critical and High severity findings should be addressed immediately, as they represent the greatest risk to the organization. Medium findings warrant attention during the next development cycle. Low and Informational findings are improvement opportunities that strengthen the overall security posture but do not represent immediate threats.

Severity Levels

Level	Business Impact	Technical Description
CRITICAL	Could devastate core business operations, cause major financial loss, mass data breach, or trigger regulatory action.	Full system compromise, remote code execution, authentication bypass, or mass data exfiltration. Easily exploitable by external attackers with minimal effort.
HIGH	Significant disruption to operations, sensitive data exposure affecting many users, or meaningful reputational damage.	Unauthorized access to sensitive systems or data, privilege escalation, stored XSS, SQL injection with data access. Exploitable with moderate effort.
MEDIUM	Moderate impact on operations, limited data exposure, or conditions that become dangerous when combined with other flaws.	Requires specific conditions, user interaction, or chaining with other vulnerabilities. Reflected XSS, CSRF on important actions, targeted information disclosure.
LOW	Minimal standalone impact. May facilitate discovery of more serious issues or affect individual users under specific conditions.	Requires local access, insider knowledge, or chaining with multiple other vulnerabilities. Self-XSS, CSRF on minor actions, verbose errors.
INFORMATIONAL	No direct security impact. Best-practice deviation or hardening opportunity that strengthens defense-in-depth.	Not exploitable in current configuration. Missing security headers, information disclosure

		with no sensitive data, technology fingerprinting.
--	--	--

Exploitation Complexity

Indicates how easy it is to exploit a vulnerability once discovered.

Level	Description
Trivial	No authentication required, no special tools, single request. Anyone with a web browser can exploit this.
Low	Requires authentication or basic tooling, but exploitation is straightforward once prerequisites are met.
Moderate	Requires chaining multiple steps, specific timing, or particular application state. May need custom tooling.
High	Requires privileged access, race conditions, complex multi-step setup, or deep application-specific knowledge.

Exposure

Indicates who can reach the vulnerable endpoint.

Level	Description
Public	Unauthenticated and internet-facing. Any external attacker can reach the vulnerable endpoint.
Authenticated	Requires a valid user account. Any registered user can reach the vulnerable endpoint.
Privileged	Requires elevated permissions (admin, staff, or specific role). Only users with special access can reach the endpoint.
Chained	Only reachable by first exploiting another vulnerability. Not directly accessible.

FINDINGS

Overview

Finding	Severity
Privilege escalation to platform Admin via roles mass-assignment on user-update	CRITICAL
Account takeover of any user (incl. bank officers) via unauthorized email change and password reset	CRITICAL
Full user-directory disclosure – GetAllUsers returns every user across all organizations	HIGH
Unauthenticated stored XSS via the CMS content endpoint	HIGH
Cross-tenant IDOR exposing other organizations' KYC application data	HIGH
Broken function-level authorization – introducer can invoke admin ruleset and user functions	HIGH
Broken function-level authorization on officer-only KYC/AML compliance decisions	HIGH
Stored XSS via KYC document upload (XSLT-in-XML) executing in a reviewer's session	HIGH
Cross-organization user approval and re-rolling via UpdateUserApprovalStatus	HIGH
Cross-tenant BOLA – submit another organization's onboarding application	HIGH
Cross-tenant IDOR – read any user group's roster, including internal bank groups	HIGH
Cross-tenant IDOR – read any user's profile by GUID	MEDIUM
Broken function-level authorization – introducer reads the admin entity/configuration surface	MEDIUM
Missing authorization – an applicant can self-approve their own KYC documents	MEDIUM
Cross-tenant BOLA – start another organization's onboarding workflow	MEDIUM
SignalR hub exposes cross-organization user presence/status	LOW
Unrestricted file upload – no type or content validation on the KYC document endpoint	LOW

Unauthenticated 2FA-code delivery to arbitrary users	LOW
Unauthenticated 2FA attempt-exhaustion / account-lockout denial of service	LOW
User enumeration via password-reset differential response	LOW
SignalR hub returns verbose error detail	INFO
SignalR hub connection survives logout (session not revoked)	INFO
Access JWT is not revoked on logout – token replay valid until expiry	INFO
Web cache deception – authenticated API responses served to anonymous clients	INFO
Missing server-side file-size validation on document upload	INFO

Critical Severity (2 findings)

Privilege escalation to platform Admin via roles mass-assignment on user-update

CRITICAL

Location PUT https://api.example.test/api/Auth/UpdateUser/{id} (roles)

Complexity: low**Exposure:** authenticated

Business Impact: Any external introducer can self-promote to platform Admin, gaining full access to all tenants' KYC applications, PII/documents, and admin configuration/write functions (rulesets, user management, security settings) – complete compromise of the onboarding platform.

Evidence

Mass-assignment privilege escalation: an introducer (External/OwnOnly) can set their own `roles` array to ["Admin"] via PUT /api/Auth/UpdateUser/{own-guid}. The endpoint is the self-service profile-update API (Referer /profile) and accepts a `roles` field that is bound server-side and persisted. After the update, GET /api/Auth/GetUserById returns roles:["Admin"] and roleId=e7f28b88-fc3a-45c7-bdae-97bae8a5a825 (the Admin role), and a re-issued JWT carries role:"Admin". Previously-403 admin-only endpoints (e.g. GET /api/admin/SecuritySettings, /api/Auth/GetAllUsersWithPresence, /api/admin/system-banners, /api/admin/document-triggers) then return 200. The SPA also routes the account to /admin-dashboard. This is a full vertical privilege escalation from a low-privileged external introducer to platform Admin, giving access to all tenants' data and admin config/write functions. Account was restored to its original role after verification (restore req 55f8c856).

Reproduction Steps

1. Login as [redacted-email] (Org A introducer, roles Initiator/External/User/OwnOnly).
 2. GET /api/Auth/GetUserById/c15890ad-c7c8-4146-bf08-6ee37b81dde9 to confirm baseline roles.
 3. PUT /api/Auth/UpdateUser/c15890ad-c7c8-4146-bf08-6ee37b81dde9 with headers Origin: https://app.example.test, Referer: https://app.example.test/profile, Content-Type: application/json and body {"id":"c15890ad-c7c8-4146-bf08-6ee37b81dde9","firstName":"Org A","lastName":"User1","userName":"[redacted-email]","email":"[redacted-email]","statusId":64,"origin":"External","roles":["Admin"]} -> 200.
 4. Re-login (or wait for token refresh) and GET /api/Auth/GetUserById -> roles: ["Admin"]; navigate to /admin-dashboard.
 5. GET /api/admin/SecuritySettings -> 200 (previously 403).
- Restore: PUT same endpoint with roles:["Initiator","External","User","OwnOnly"] and roleId ceec5c88-5a3d-473c-ade0-23cd5aa3f5f1.

Screenshots

[Screenshot Redacted]

Recommendation

Remove ``roles`/`roleId`` from the `UpdateUser` bind model; assign roles only through an admin-gated endpoint with server-side authorization. Never trust a client-supplied role array.

Account takeover of any user (incl. bank officers) via unauthorized email change and password reset

CRITICAL

Location PUT `https://api.example.test/api/Auth/UpdateUser/{id}` (email)

Complexity: low**Exposure:** authenticated

Business Impact: Any authenticated external introducer can take over the account of any other user on the platform, including bank officers (Unit 1/ Unit 2) and platform administrators, by redirecting the victim's email to an attacker-controlled mailbox and completing the password-reset flow. This grants the attacker the victim's role and internal permissions (KYC/AML intake, IDV & sanctions, risk scoring, credit check, compliance review, account setup) – full compromise of the KYC/onboarding platform's privileged users. GUIDs are enumerable via GetAllUsers, and the same flaw exists on the DEV environment.

Evidence

```
CONFIRMED end-to-end account takeover of a bank officer by a low-privilege external
introducer (strengthens prior finding 46f141cd, which could not observe reset-email
delivery). An authenticated Org A introducer ([redacted-user], roles
[Initiator,External,User,OwnOnly]) PUT /api/Auth/UpdateUser/{guid} for a Unit 1 bank
officer in a different org/role ([redacted-user], ec6ecb9a-69dc-4e8a-a040-
588c3cdad71c) and changed the officer's email to an attacker-controlled address with
no ownership/role check -> 200 {isSuccess:true}. The change persists. POST /api/Auth/
ForgotPassword with the attacker address returns 200 and the reset email IS actually
delivered to the stored email (verified in a controlled mailbox: "[Redacted Client]
Onboarding - Password Reset Request" containing https://app.example.test/reset-
password?email=<attacker>&code=<token>). POST /api/Auth/ResetPassword
{Email,ResetCode,NewPassword} with the emailed token returns 200 and sets the
password. POST /api/Auth/Login with the attacker email + new password returns 200
authenticated as the OFFICER (userId ec6ecb9a..., roles [OrgUnit,Initiator,User],
internalPermissions [Intake validation,Idv & Sanctions,Risk Scoring,Credit Check,Com
pliance Review,Account Setup,Document Preparation,Welcome & Activation]); the
officer's original password then returns 401. Login identifier is the email field, so
the attacker fully owns the account. Victim restored (email + password + org unit).
Same primitive also confirmed on [redacted-host] (separate DB) with the UAT creds.
GUIDs enumerable via GET /api/Auth/GetAllUsers (finding f1c325b9), so any user incl.
admins can be targeted.
```

Reproduction Steps

1. login(app.example.test, [redacted-email] / [redacted]) (Org A external introducer).
2. GET /api/Auth/GetAllUsers -> pick any victim GUID (e.g. bank officer ec6ecb9a-69dc-4e8a-a040-588c3cdad71c).
3. PUT /api/Auth/UpdateUser/{victim-guid} (Origin/Referer https://app.example.test) body {"id":"<guid>","firstName":"...","lastName":"...","userName":"<victim-username>","email":"<attacker-mail>","statusId":64,"origin":"External","roleId":"ceec5c88-5a3d-473c-ade0-23cd5aa3f5f1","roles":

```
[{"OrgUnit","Initiator","User"},"organizationalUnitId":<orig>} -> 200.
```

4. POST /api/Auth/ForgotPassword {"email": "<attacker-mail>"} -> 200; read the reset email at the attacker mailbox.

5. POST /api/Auth/ResetPassword {"Email": "<attacker-mail>", "ResetCode": "<code from email>", "NewPassword": "<new>"} -> 200.

6. POST /api/Auth/Login {"userName": "<attacker-mail>", "password": "<new>"} -> 200 authenticated as the victim.

7. Restore: repeat 4-5 to set the original password, then PUT UpdateUser with the original email + organizationalUnitId.

Screenshots

[Screenshot Redacted]

Recommendation

Enforce object- and function-level authorization on PUT /api/Auth/UpdateUser/{id} (a user may edit only their own record; email/role/approval changes require elevated authz). Require re-verification (confirm-email) before an email change takes effect, and invalidate outstanding reset tokens on email change.

High Severity (9 findings)

Full user-directory disclosure – GetAllUsers returns every user across all organizations

HIGH

Location GET https://api.example.test/api/Auth/GetAllUsers (none)

Complexity: low**Exposure:** authenticated

Business Impact: Cross-org disclosure of the full user directory (bank staff + all introducer/associate accounts) to any authenticated external introducer. Exposes PII (names, corporate emails, phones), internal role/permission structure, org-unit hierarchy and stable user GUIDs usable to pivot into other IDOR endpoints. In a KYC/AML onboarding platform this is a serious privacy and reconnaissance exposure.

Evidence

Authenticated low-privilege introducer ([redacted-user], roles Initiator/External/User/OwnOnly, org Org A) can call GET /api/Auth/GetAllUsers and receives the ENTIRE platform user directory: 87 users across all orgs (Org A, Org B, [Redacted Client], ClearServe, Quick/New Serve, Direct-C2) including bank staff PII (names, emails, phone numbers, country), role assignments, organizationalUnitId/Name, entityName, approvalStatus, reportToUserId, roleId and directoryObjectGuid. This is a global admin-style directory a self-service external introducer should never reach. Unauth returns 401, so it is auth-gated but NOT role-gated (BFLA). 73,970 bytes / 87 records. Enables cross-org user enumeration and harvest of GUIDs used as object identifiers elsewhere (GetUserById/{guid}, GetUserEntityAssignment/{guid}).

Reproduction Steps

1. Login as [redacted-email] / [redacted] (introducer, org Org A). 2. GET https://api.example.test/api/Auth/GetAllUsers with the session cookie. 3. Observe 200 with 87 user records spanning all organizations. 4. Repeat without cookie -> 401.

Recommendation

Restrict GetAllUsers to admin roles and scope results to the caller's organization; paginate and omit sensitive fields. Treat user GUIDs as non-secret but never as authorization.

Unauthenticated stored XSS via the CMS content endpoint

HIGH

Location PUT https://api.example.test/api/content (contents)

Complexity: moderate**Exposure:** authenticated

Business Impact: Unauthenticated attacker can persistently inject JavaScript into shared CMS pages (legal terms, privacy policy, consent screens) served to all users of the bank's KYC/onboarding portal. When a logged-in user opens the page and interacts, the attacker's script runs in the portal origin and can call authenticated APIs, read/exfiltrate applicant PII and KYC documents, or perform actions as the victim.

Evidence

Stored XSS in the CMS content store, reachable UNAUTHENTICATED. POST /api/content (create) and PUT /api/content (update) require no session (no cookie sent -> 200 isSuccess:true). Content is rendered on the SPA route /content-view?locator=... via Angular bypassSecurityTrustHtml (getSafeHtml in app-content-view component), i.e. no sanitization. A server-side blacklist filter ("Property 'request.Contents' contains potentially dangerous content: Cross-Site Scripting (XSS)") blocks literal <script/<iframe/<object, "javascript:", "onerror"/"onload" and generic on\w+= event handlers, but is bypassable by HTML-entity-encoding the "s" in javascript: (javascript:) - the browser decodes it in the href value to javascript: while the filter's substring check misses it. Confirmed execution: alert('XR3F_XSS') fired on https://app.example.test/content-view?locator=cxp_content_displaysetting_individualtermofservice (dialog log id e84). Impact: any unauthenticated attacker can poison shared legal/consent pages (individual-term-of-service, privacy-policy, etc.) so that any logged-in user who opens them executes attacker JS in the app origin (same-origin -> can call authenticated APIs, exfiltrate KYC/PII). Note: the unauth write itself is also a broken-access-control issue but is filed here as the XSS delivery vector.

Reproduction Steps

1. UNAUTH: PUT https://api.example.test/api/content with header Content-Type: application/json, Referer: https://app.example.test/ and body {"id": 1, "pageCode": "individual-term-of-service", "categoryId": 673, "title": "Individual Term of Service", "summary": "xR3f", "contents": "XR3F_CLICK", "isActive": true, "displaySettingId": 677, "isDraft": false} -> 200 {"isSuccess": true} (req e2e8aaad). (POST /api/content also works unauth for create, req 9290975d.)
2. Verify persistence unauth: GET https://api.example.test/api/content/1 -> contents contains the payload (req e060da2c).
3. Log in as any account (e.g. [redacted-user]) and open https://app.example.test/content-view?locator=cxp_content_displaysetting_individualtermofservice
4. Inspect DOM: XR3F_CLICK (entity decoded by browser). Click it -> alert('XR3F_XSS') fires (dialog log id e84).

Screenshots

[Screenshot Redacted]

Recommendation

Require authentication and an editor role on POST/PUT /api/content; server-side sanitize/allow-list HTML (strip javascript: URIs and event handlers); apply output encoding and a strict CSP on rendered CMS pages.

Cross-tenant IDOR exposing other organizations' KYC application data

HIGH

Location GET `https://api.example.test/api/applications/{id}/documents/expected-materialized(id(path))`

Complexity: moderate

Exposure: authenticated

Business Impact: Cross-tenant KYC data disclosure: an introducer of one organization can read another organization's onboarding applications – applicant/entity name, application reference, document requirements, and internal review/screening/risk metadata – by enumerating the sequential integer application id. Breaks the Org A/Org B tenant isolation the platform relies on.

Evidence

Application-scoped endpoints lack the ownership check their siblings enforce, allowing an introducer in org Org A to read another org's (Org B) application data by integer id. App ids are sequential and enumerable. Confirmed cross-org reads: GET `/api/applications/37/documents/expected-materialized?applicationType=Business` returns the Org B business application's full expected-document list including the applicant entity name "[redacted] Technologies" and its recipientPublicId; GET `/api/applications/335/documents/expected-materialized` and `/documents/expected` return Org B individual app 335's document requirements; GET `/api/applications/335/reviewer-summary?applicationType=Individual` returns the Org B app's review summary (reference I-00000335, initiatorName "Org B User1", channel, completeness, screening/risk counts). Same pattern on `/api/applications/{id}/CurrentSteps`, `/compliance/risk-profile-with-effective`, `/section-exemptions/board`, `/checkpoint-verifications`. Meanwhile the correctly-gated siblings (`/timeline`, `/submission-readiness`, `/compliance/conditions`, `/Message/application/{id}/messages`) return 403 "You do not have access to this application." – proving the missing check is an oversight, not intended.

Reproduction Steps

1. login(app.example.test, [redacted-email]) (Org A).
2. GET `/api/applications/37/documents/expected-materialized?applicationType=Business` -> 200 with Org B application 37 data (recipientName "[redacted] Technologies").
3. GET `/api/applications/335/reviewer-summary?applicationType=Individual` -> 200 with Org B app 335 review summary. Org B app ids were enumerated by logging in as [redacted-user] and calling POST `/api/Applications/Filter`.

Recommendation

Scope every `/api/applications/{id}/*` read to the caller's organization and role; return 403 for foreign application ids exactly as the readiness endpoints already do.

Broken function-level authorization – introducer can invoke admin ruleset and user functions

HIGH

Location POST https://api.example.test/api/admin/rulesets/{id}/publish (id)

Complexity: low**Exposure:** authenticated

Business Impact: An external, self-service introducer account can invoke administrative configuration write operations on the bank's KYC/onboarding platform – publishing global form-validation rulesets, bulk-updating initiator status, and creating/updating application roles. This is a privilege-escalation / integrity break: an external party can alter the platform's global onboarding rules and role definitions that govern how all applications (including other organisations') are validated and approved. A malicious introducer could weaken mandatory-field/document rules or manipulate role/permission definitions to bypass onboarding controls.

Evidence

Broken function-level authorization (OWASP API5): a low-privilege EXTERNAL introducer ([redacted-user], roles Initiator/External/User/OwnOnly, org Org A) can execute privileged /api/admin/* WRITE operations. Most damning: POST /api/admin/rulesets/1/publish returned 200 and the response body proves a real state change performed by the introducer – "updatedBy":"[redacted-email]","updatedAt":"2026-09-14T10:25:35.3702582Z" (the ruleset's global form-validation config was re-published by the external user). Other admin write handlers also execute for the introducer (they reach business logic instead of returning 403): POST /api/admin/Initiator/BulkUpdateStatus -> 200 {"data":false,"isSuccess":true}; PUT /api/admin/ApplicationRole/CreateRole and /UpdateRole -> 400 "Application role with ID 0 not found."; POST /api/admin/document-rulesets/1/approve -> 400 "DocumentRuleSet 1 must be in PendingApproval status to approve. Current: 'Published'.". All are auth-gated (unauth -> 401) but NOT role-gated. Gating is inconsistent: sibling destructive admin writes /api/admin/CacheManagement/clear and /api/admin/DataManagement/wipe-applications correctly return 403, proving the admin role check exists elsewhere but was omitted on these routes.

Reproduction Steps

1. Login as [redacted-email] / [redacted] (external introducer, org Org A).
2. POST https://api.example.test/api/admin/rulesets/1/publish with header Referer: https://app.example.test/ and body {"id":1} using the session cookie.
3. Observe 200 and response field "updatedBy":"[redacted-email]" (the introducer performed the admin write).
4. Repeat POST /api/admin/Initiator/BulkUpdateStatus -> 200 isSuccess:true; PUT /api/admin/ApplicationRole/CreateRole -> 400 (handler reached, not 403).
5. Repeat with no cookie -> 401.

Screenshots

[Screenshot Redacted]

Recommendation

Gate all /api/admin/* endpoints behind an explicit Admin role check (fail closed with 403) rather than authentication alone.

Broken function-level authorization on officer-only KYC/AML compliance decisions

HIGH

Location POST `https://api.example.test/api/applications/{id}/compliance/risk-override (id)`

Complexity: low

Exposure: authenticated

Business Impact: An external introducer can invoke the bank's officer-only KYC/AML compliance decision functions – risk-rating override, PEP override, RegTek screening re-run, override-seal commit-and-advance, QA decision and conditional acceptance – and read the officer-only risk profile and screening results. These are the controls that determine whether a customer is accepted or rejected for onboarding; a client-side actor reaching them undermines the segregation between the introducer (data-entry) role and the bank's compliance/decision role, enabling an introducer to attempt to self-approve or manipulate the risk decision on applications.

Evidence

Broken function-level authorization (OWASP API5): a low-privilege EXTERNAL introducer ([redacted-user], roles Initiator/External/User/OwnOnly) can invoke officer-only KYC/AML compliance decision functions. These endpoints are used exclusively by bank officers (per the officer-role crawl) and should return 403 to an introducer, but instead the requests reach the business logic – the introducer receives domain-specific responses rather than an authorization denial: POST `/api/applications/121/compliance/risk-override` -> 409 `{"errorCode":"ASSESSMENT_NOT_RUN"}` (the override handler executed and checked assessment state), pep-override -> 409 `ASSESSMENT_NOT_RUN`, rerun-regtek-screening -> 400, override-seal/commit-and-advance -> 500, assessment-record/qa-decision -> 400, conditional-acceptance -> 400. The introducer can also READ officer-only data: GET `.../compliance/risk-profile-with-effective` -> 200 (full risk profile with KPIs/conditions) and GET `.../compliance/screening/results-with-adjudications` -> 200. No role check is enforced on this compliance controller for the introducer role.

Reproduction Steps

1. Login as [redacted-email] / [redacted] (external introducer, org Org A).
2. POST `https://api.example.test/api/applications/121/compliance/risk-override` with Referer: `https://app.example.test/` and body `{"riskLevel":"Low","justification":"test"}` -> 409 `ASSESSMENT_NOT_RUN` (the officer-only handler ran; a properly role-gated endpoint would return 403).
3. POST `.../pep-override` -> 409; POST `.../rerun-regtek-screening` -> 400; POST `.../override-seal/commit-and-advance` -> 500.
4. GET `.../compliance/risk-profile-with-effective?applicationType=Business` -> 200 risk profile; GET `.../compliance/screening/results-with-adjudications?applicationType=Business` -> 200.

Screenshots

[Screenshot Redacted]

Recommendation

Add a role check to the compliance controller so only officer/checker roles can read or invoke risk/PEP/screening/QA decision endpoints; reject introducer sessions with 403.

Stored XSS via KYC document upload (XSLT-in-XML) executing in a reviewer's session

HIGH

Location POST https://api.example.test/api/ApplicantDocuments/upload

Complexity: low

Exposure: authenticated

Evidence

The document-upload endpoint performs NO file-type validation (extension, content-type, or magic bytes). An authenticated introducer can upload an XML file containing an embedded XSLT stylesheet (via an ``<?xml-stylesheet?>`` processing instruction with a `data:` URI). The file is stored with its attacker-chosen extension and served by /api/ApplicantDocuments/download/{fileName}?inline=true with `Content-Type: application/xml` and `Content-Disposition: inline`. When the app's own document viewer renders the document (it does `downloadFile(fileName,true)` -> Blob -> `window.open(blobUrl,"\_blank")`); the doc-viewer component also renders non-image/non-pdf docs in an ``<iframe [src]=safeUrl>``, the browser applies the XSLT and executes attacker JavaScript in the app.example.test origin. Confirmed end-to-end: uploaded xss2.xml (applicantDocumentId 522), fetched it back, rendered the blob in an iframe AND via window.open exactly as the app does - `window.\_\_xss` was set to 42 and the XSLT-transformed body "XSLT-EXEC" rendered (screenshot). Because the blob inherits the SPA origin, the JS can make same-site credentialed API calls (the JWT cookie is httpOnly on api.example.test but sent automatically) -> full account/data compromise of any user who views the document, including bank officers.

Reproduction Steps

1. Login as [redacted-email] / [redacted] (or any introducer).
2. POST /api/ApplicantDocuments/upload (multipart, session cookie) with fields ApplicantId=<applicant public id, e.g. 1be07d2d-803c-f111-8603-d1c237308e24>, ApplicationId=150, ApplicationTypeId=1, DocumentTypeId=101, ExpectedDocumentId=2861, and a file part named Document with filename xss2.xml, content-type application/xml, body = the payload above. -> 200 {"data":522}.
3. GET /api/ApplicantDocuments/application/150?applicationType=Individual to read the stored fileName (e.g. 5ce649bb-...-03f4075f794a.xml).
4. In the app (app.example.test origin), run: `fetch('https://api.example.test/api/ApplicantDocuments/download/<fileName>?inline=true',{credentials:'include'}).then(r=>r.blob()).then(b=>{const u=URL.createObjectURL(b);window.open(u,'_blank')})` - or simply open the document via the app's own "view" (eye) action. The XSLT executes: `window.__xss===42`.
5. Alternatively GET the download URL directly in a browser context that has the cookie; the response is application/xml inline.

Screenshots

[Screenshot Redacted]

Recommendation

Validate uploaded file type by content (magic bytes) against an allow-list; store and serve user documents from a sandboxed origin with Content-Disposition: attachment and a non-renderable Content-Type; never serve XML/HTML/SVG inline from the app origin.

Cross-organization user approval and re-rolling via UpdateUserApprovalStatus

HIGH

Location PUT https://api.example.test/api/Auth/UpdateUserApprovalStatus (userId)

Complexity: low**Exposure:** authenticated

Business Impact: An introducer in one org can approve and re-role any user in any org, including bank staff and admins – enabling cross-tenant account takeover / privilege manipulation and bypass of the account-approval control that gates portal access.

Evidence

Missing authorization on the user-approval/role-assignment endpoint: an introducer (External/OwnOnly, org Org A) can call PUT /api/Auth/UpdateUserApprovalStatus for a user in a DIFFERENT org (Org B) and it returns 200 and persists the change. The endpoint is the admin function that approves pending accounts and assigns their role. After the call, the target Org B user's approvalStatus flipped to "Approved" and its roles/roleId changed (["Initiator", "External", "User", "OwnOnly"]/ceec5c88 -> ["External", "OwnOnly"]/98329923). The sibling PUT /api/Auth/UpdateUser/{foreign-guid} likewise accepts a `roles` array for a foreign user and returns 200 (req 328c24d6), so an introducer can re-role arbitrary users across tenants. No ownership/role check is enforced. Account was restored after verification.

Reproduction Steps

1. Login as [redacted-email] (Org A introducer).
2. GET /api/Auth/GetUserById/7530c756-5d03-47ee-bcac-015102173a0f (Org B user) to record baseline roles/approvalStatus.
3. PUT /api/Auth/UpdateUserApprovalStatus with Origin/Referer = https://app.example.test and body {"userId":"7530c756-5d03-47ee-bcac-015102173a0f", "approvalStatus":"Approved", "roleId":"ceec5c88-5a3d-473c-ade0-23cd5aa3f5f1"} -> 200.
4. GET the same user again -> roles/approvalStatus changed (cross-org write confirmed).
5. Restore: PUT /api/Auth/UpdateUser/7530c756-... with roles ["Initiator", "External", "User", "OwnOnly"] and roleId ceec5c88-5a3d-473c-ade0-23cd5aa3f5f1.

Recommendation

Require an admin/approver role and enforce same-organization scope on UpdateUserApprovalStatus and the UpdateUser role array; reject cross-org targets with 403.

Cross-tenant BOLA – submit another organization's onboarding application

HIGH

Location POST `https://api.example.test/api/workflow/{id}/submit` (id)

Complexity: low**Exposure:** authenticated

Business Impact: Any authenticated user (including a low-privilege external introducer) can submit / push ANY application into the bank's KYC review workflow by enumerating sequential integer application ids, regardless of ownership or organisation, even for applications they are explicitly denied read access to. This lets an attacker force other introducers'/orgs' applications into the bank review queue (queue flooding / DoS of the bank's onboarding pipeline), trigger premature submission of applications the owner had not yet submitted, and manipulate the workflow state of other tenants' KYC records. It also demonstrates the submit endpoint lacks the object-level authorization that its sibling read endpoints enforce, i.e. the whole `/api/workflow/{id}/*` family should be re-checked for the same omission.

Evidence

Broken object-level authorization (OWASP API1/BOLA) on the workflow submission endpoint. POST `/api/workflow/{id}/submit` enforces the submission-readiness gate but performs NO ownership/access check on the application id. A low-privilege EXTERNAL introducer ([redacted-user], org Org A, roles [Initiator,External,User,OwnOnly], user id `c15890ad-c7c8-4146-bf08-6ee37b81dde9`) is explicitly DENIED read access to application 552 (owned by a different user/org – the Unit 1 bank officer [redacted-user], entity [Redacted Client]): GET `/api/applications/552/submission-readiness` -> 403 "You do not have access to this application." and GET `/api/applications/552/timeline` -> 403. Yet the SAME introducer session POSTs `/api/workflow/552/submit` -> 200 `{"data":{"workflowInstanceId":331,"transitionLogId":2307},"isSuccess":true}`, driving that foreign application from DRAFT into the RM_SCREENING_INTAKE workflow activity. The resulting timeline event (read back as the owning officer) records the introducer as the actor: WORKFLOW_TRANSITION actorUserId=c15890ad-... actorName="Org A User1", fromActivityCode DRAFT -> toActivityCode RM_SCREENING_INTAKE, decisionCode SUBMIT. The access check is simply absent on the submit path (contrast the readiness/timeline endpoints which correctly return 403). Application ids are small sequential integers (552, 27, 121, 335, ...) so any application is trivially enumerable.

Reproduction Steps

1. Login to `https://app.example.test` as [redacted-email] / [redacted] (external introducer, org Org A).
2. Confirm no access: GET `https://api.example.test/api/applications/552/submission-readiness?applicationType=Individual` -> 403 "You do not have access to this application."
3. Submit the foreign application: POST `https://api.example.test/api/workflow/552/submit` with Referer: `https://app.example.test/` and body `{"applicationType":1,"decisionCode":"SUBMIT","idempotencyKey":"<any-uuid>","comments":"x"}` -> 200 `{"data":{"workflowInstanceId":331,...},"isSuccess":true}`.
4. Login as the owning officer [redacted-email] and GET `/api/applications/552/`

timeline?applicationType=Individual -> the new WORKFLOW_TRANSITION event shows actorUserId = the introducer (c15890ad-...), DRAFT -> RM_SCREENING_INTAKE.

Recommendation

Enforce application-ownership authorization on all /api/workflow/{id}/* routes; return 403 for applications the caller does not own, matching the submission-readiness endpoint.

Cross-tenant IDOR – read any user group's roster, including internal bank groups

HIGH

Location GET https://api.example.test/api/UserGroup/{id} (id (path))

Complexity: moderate**Exposure:** authenticated

Business Impact: Cross-tenant user directory disclosure: any authenticated introducer can enumerate user-group ids and dump the full member roster (name, email, roles, organization) of internal bank groups (BankAdmin, BA Checker) and of every other tenant's group. Enables targeted phishing/social-engineering against bank staff and full user enumeration across all organizations.

Evidence

GET /api/UserGroup/{id} performs no authorization check on the requested group: an introducer in org Org A can read the member roster of any group, including internal bank groups they do not belong to. Group ids are small sequential integers (enumerable). Confirmed reading group 2 "BankAdmin" (bank staff [redacted-email], [redacted-email], entity [Redacted Client]) and group 9 "BA Checker" (10 members spanning ClearServe Ltd, FastServe Ltd, Org A, Org B, Holy/Cash/Flexi/New/Quick Serve Ltd). Each member record exposes full name, email, roles, org unit and entity. Same data also reachable via GET /api/UserGroup/by-user/{any-guid}.

Reproduction Steps

1. login(app.example.test, [redacted-email]). 2. GET /api/UserGroup/2 -> 200 BankAdmin members. 3. GET /api/UserGroup/9 -> 200 BA Checker members (10 users across 8 orgs). 4. Enumerate ids 1..N (404 for absent).

Recommendation

Authorize UserGroup reads against the caller's group memberships and organization; do not expose internal bank group rosters to external introducers. Use non-enumerable identifiers.

Medium Severity (4 findings)

Cross-tenant IDOR – read any user's profile by GUID

MEDIUM

Location GET https://api.example.test/api/Auth/GetUserById/{id} (id (path))

Complexity: moderate**Exposure:** authenticated

Business Impact: Cross-tenant PII disclosure: any introducer can enumerate and read every user's profile (name, email, roles, org) across all organizations on the bank KYC platform. Combined with GetAllUsers this is a full directory dump.

Evidence

Introducer account [redacted-user] (org Org A, roles Initiator/External/User/OwnOnly) can read ANY other user's profile across tenant boundaries by GUID. No ownership/org check is performed. GUIDs are trivially harvested from the over-permissive GET /api/Auth/GetAllUsers (finding f1c325b9), which returns all 87 users across every org. Confirmed reading Org B users [redacted-user] (7530c756-5d03-47ee-bcac-015102173a0f) and [redacted-user] (ecfb7984-17a2-4226-91a3-420bdc93f667): full profile incl. email, userName, roles, org unit, entity name. Sibling endpoints leak further: GET /api/Auth/GetUserEntityAssignment/{id} returns the target's entity/org-unit assignment; GET /api/UserGroup/by-user/{id} returns the target's group memberships. All 200 with the Org A session.

Reproduction Steps

1. login(app.example.test, [redacted-email]). 2. GET /api/Auth/GetAllUsers -> harvest any user GUID. 3. GET /api/Auth/GetUserById/<foreign-guid> with the Org A session -> 200 full profile of a Org B user. Repeat for GetUserEntityAssignment/{guid} and UserGroup/by-user/{guid}.

Recommendation

Scope GetUserById (and GetUserEntityAssignment / UserGroup/by-user) to the caller's organization and role; do not return foreign users' profiles.

Broken function-level authorization – introducer reads the admin entity/configuration surface

MEDIUM

Location GET https://api.example.test/api/admin/Entity/Tree

Complexity: low**Exposure:** authenticated

Business Impact: Any authenticated external introducer can read the bank's administrative configuration: the complete organisation/entity hierarchy across all tenants (with relationship-manager GUIDs and contact emails), all initiator accounts with their privilege flags, platform audit statistics, and the full document-type/inclusion-rule configuration. This exposes internal structure, user identifiers and security-relevant configuration to an untrusted external party and aids further targeted attacks.

Evidence

Broken function-level authorization (OWASP API5): a low-privilege EXTERNAL introducer ([redacted-user], roles Initiator/External/User/OwnOnly, org Org A) can read the platform's administrative configuration surface under /api/admin/*. Endpoints that return 200 to the introducer include: GET /api/admin/Entity/Tree (full entity/org tree for all organisations incl Org A/Org B/[Redacted Client] with relationshipManagerId GUIDs and contact emails), GET /api/admin/Entity/hierarchy, GET /api/admin/Entity/statistics, POST /api/admin/Entity/Filter (all 14 entities), POST /api/admin/Initiator/Filter (all initiators with userIds and canApproveWorkflows/canManageUsers flags), GET /api/admin/audit/stats, POST /api/admin/audit/events, GET /api/admin/document-type-definitions (52KB), GET /api/admin/document-inclusion-rules (53KB). All are auth-gated (unauth -> 401) but NOT role-gated – an external introducer should have no access to admin configuration. Sibling admin endpoints are correctly 403 (SecuritySettings, system-banners, acknowledgement-policies, document-triggers), confirming the role check was omitted on these routes.

Reproduction Steps

1. Login as [redacted-email] / [redacted] (external introducer).
2. GET https://api.example.test/api/admin/Entity/Tree with the session cookie -> 200 full org/entity tree.
3. POST /api/admin/Initiator/Filter (Referer set) -> 200 all initiators with userIds/permissions.
4. GET /api/admin/audit/stats, /api/admin/document-type-definitions, /api/admin/document-inclusion-rules -> all 200.
5. Repeat any with no cookie -> 401.

Screenshots

[Screenshot Redacted]

Recommendation

Add role-based authorization to the `/api/admin/Entity` and audit endpoints so external introducers cannot read the platform entity tree, initiator list, or audit configuration.

Missing authorization – an applicant can self-approve their own KYC documents

MEDIUM

Location POST `https://api.example.test/api/ApplicantDocuments/{id}/mark-as-approved (id)`

Complexity: moderate

Exposure: authenticated

Evidence

An introducer (role "Initiator/External/User/OwnOnly", no checker/officer role) can self-approve their own uploaded KYC document, bypassing the checker/officer document-review workflow. POST `/api/ApplicantDocuments/522/mark-as-approved` returned 200 `{"isSuccess":true,"successMessage":"Document status updated to 'Approved' successfully"}` and the document's status flipped from 659 "Uploaded" to 661 "Approved". The document (id 522, Passport/ID Card, app 150) was uploaded by the same introducer (createdByUserId=c15890ad-c7c8-4146-bf08-6ee37b81dde9). No role/permission check is enforced server-side; the "Approve" action is a review decision that should be reserved for the bank officer/checker. This lets an applicant mark their own identity/proof-of-address documents as verified, defeating KYC document verification.

Reproduction Steps

1. Login as [redacted-email] / [redacted] (introducer, OwnOnly).
2. Upload a document to an application (e.g. POST `/api/ApplicantDocuments/upload`) or use an existing uploaded doc id (e.g. 522).
3. POST `/api/ApplicantDocuments/522/mark-as-approved` with the session cookie and body `{}`.
4. Observe 200 success; re-GET `/api/ApplicantDocuments/application/150?applicationType=Individual` and confirm status is now "Approved".

Recommendation

Restrict document approval to officer/checker roles; an applicant must never be able to approve their own or others' documents.

Cross-tenant BOLA – start another organization's onboarding workflow

MEDIUM

Location POST <https://api.example.test/api/workflow/{id}/start> (id)

Complexity: low**Exposure:** authenticated

Business Impact: Any authenticated user (including a low-privilege external introducer) can start a workflow instance on ANY application by enumerating sequential integer ids, regardless of ownership or organisation, and without the application being submission-ready. This lets an attacker create workflow instances / transition records on other tenants' KYC applications (state pollution, audit-trail injection, potential queue/processing disruption) and demonstrates that the `/api/workflow/{id}/*` family lacks the object-level authorization enforced by its sibling read endpoints.

Evidence

Broken object-level authorization (BOLA) on POST `/api/workflow/{id}/start`. Like `/api/workflow/{id}/submit`, the `/start` endpoint performs NO ownership/access check on the application id. A low-privilege EXTERNAL introducer ([redacted-user], org Org A, user id `c15890ad-c7c8-4146-bf08-6ee37b81dde9`) is denied read access to application 8 (owned by the Unit 1 bank officer [redacted-user], createdById `ec6ecb9a-69dc-4e8a-a040-588c3cdaa71c`) – GET `/api/applications/8/submission-readiness` and `/api/applications/8/timeline` both return 403 "You do not have access to this application." – yet the same session POSTs `/api/workflow/8/start` -> 200 and a workflow instance (id 332) is created on that foreign application with `createdBy` = the introducer's GUID. The officer's timeline for app 8 records the introducer as the actor of a `WORKFLOW_TRANSITION`. Notably `/start` does NOT enforce the submission-readiness gate either (app 8 is not ready – its `/submit` returns `SUBMIT_SIGNATORY_MANDATE_UNCONFIRMED` etc.), so this endpoint both crosses the tenant boundary and skips the readiness control that `/submit` enforces. Application ids are small sequential integers, so any application is enumerable.

Reproduction Steps

1. Login to <https://app.example.test> as [redacted-email] / [redacted] (external introducer, org Org A).
2. Confirm no access: GET <https://api.example.test/api/applications/8/submission-readiness?applicationType=Joint> -> 403 "You do not have access to this application."
3. Start a workflow instance on the foreign app: POST <https://api.example.test/api/workflow/8/start> with Referer: <https://app.example.test/> and body `{"applicationType":3,"decisionCode":"SUBMIT","idempotencyKey":"<uuid>","comments":"x"}` -> 200 `{"data":{"id":332,"applicationId":8,"createdBy":"c15890ad-...", "isSuccess":true}}`.
4. Login as the owning officer [redacted-email] and GET `/api/applications/8/timeline?applicationType=Joint` -> the `WORKFLOW_TRANSITION` event `actorUserId` is the introducer.

Recommendation

Enforce application-ownership authorization on POST /api/workflow/{id}/start, as for /submit.

Low Severity (5 findings)

SignalR hub exposes cross-organization user presence/status

LOW

Location POST https://api.example.test/messageHub

Complexity: moderate

Exposure: authenticated

Business Impact: Any authenticated introducer (external, low-privilege) can enumerate the global online roster and read the email address + online status + last-seen timestamp of users in OTHER organisations (including bank officers and other introducers) in real time, with no object-level authorization. Combined with the GetAllUsers GUID dump this yields a cross-tenant presence/PII map of the bank's onboarding platform.

Evidence

SignalR hub /messageHub exposes cross-org presence + email with no per-object authorization. (a) The server pushes UserOnline(userId, EMAIL, ts) to EVERY connected client regardless of org/role: a live connection authenticated as [redacted-user] (org Org B) received [redacted-user]'s (org Org A) UserOnline broadcast containing its email, and vice-versa. (b) Hub method GetUserStatus(userId) accepts ANY user GUID and returns that user's real online status + last-seen timestamp across orgs (Org A introducer read Org B user 7530c756 → lastSeen 08:30:47, and Unit 2 officer acbb7602 → 08:32:34; unknown GUID returns current time, proving stored per-user data). (c) Hub method GetOnlineUsers() returns the GLOBAL OnlineUsersList of all online user GUIDs across all orgs. GUIDs are enumerable via the already-confirmed /api/Auth/GetAllUsers PII dump, so this becomes a full cross-org presence/email map. Hub connection itself is auth-gated (unauth negotiate → 401), but individual methods/broadcasts are not org-scoped.

Reproduction Steps

1) Authenticate as [redacted-user] (Org A). 2) POST /messageHub/negotiate?negotiateVersion=1 (cookie) → connectionToken. 3) Open wss://api.example.test/messageHub?id=<token>, send {"protocol":"json","version":1}\x1e. 4) Send {"target":"GetUserStatus","arguments":["acbb7602-4a32-435c-92da-c79ff5189fc8"],"invocationId":"1","type":1}\x1e (a Unit 2 officer GUID) → server returns that user's online status + last-seen. 5) Send {"target":"GetOnlineUsers","arguments":[],"invocationId":"2","type":1}\x1e → global list of all online user GUIDs. 6) To observe the broadcast: open a second connection as [redacted-user] (Org B) via ?access_token=<its JWT> and have [redacted-user] invoke NotifyUserOnline; the Org B socket receives UserOnline with [redacted-user]'s email.

Recommendation

Authorize SignalR hub method calls and group membership server-side; scope presence/status to the caller's organization; do not trust client-supplied user ids on hub methods.

Unrestricted file upload – no type or content validation on the KYC document endpoint

LOW

Location POST https://api.example.test/api/ApplicantDocuments/upload (Document (multipart file part) / filename / Content-Type)

Complexity: low

Exposure: authenticated

Evidence

The KYC document upload endpoint performs NO file-type validation (no extension allow-list, no Content-Type check, no magic-byte sniff). Any authenticated introducer can upload arbitrary file types – server-executable (.php), HTML, SVG, XHTML, JS, XML – as a "Passport / ID Card" document. The attacker-chosen extension is preserved in the stored filename (GUID + original extension), and the file is retrievable via /api/ApplicantDocuments/download/{fileName}. Confirmed: uploaded xss.svg (200, id 505), xss.html (506), shell.php (507), shell.php.png (508), ../../../../pwn.svg (509, traversal stripped), pwn.php (510), t.xml (514), t.xhtml (517), t.js (518) – all accepted with isSuccess=true. Retrieval returns the exact uploaded bytes with the original extension (e.g. download/166aeled-...php returns the PHP source). Impact: arbitrary attacker-controlled files are stored server-side on the bank's KYC platform; the download endpoint derives Content-Type from the extension, so .xml is served as application/xml inline (XSLT/XSS-capable) while .html/.svg/.php map to application/octet-stream + nosniff (which limits direct stored-XSS/RCE). A malicious document (e.g. a .pdf/.docx with client-side exploit) can also be planted for a bank officer to open. No static/web-executable path for the stored files was found (upload dir not under wwwroot), so direct webshell execution is not confirmed.

Reproduction Steps

1) Authenticate as [redacted-email] (introducer, org Org A). 2) POST multipart to https://api.example.test/api/ApplicantDocuments/upload with Referer https://app.example.test/ and fields ApplicantId=1be07d2d-803c-f111-8603-d1c237308e24, ApplicationId=150, ApplicationTypeId=1, DocumentTypeId=101, ExpectedDocumentId=2861, and a file part name="Document" filename="shell.php" content_type="application/x-php" body=<?php echo "PWNEED_".(7*7); ?>. 3) Response is 200 {"data":<newId>,"isSuccess":true}. 4) GET /api/ApplicantDocuments/application/150?applicationType=Individual to read the stored fileName (GUID.php). 5) GET /api/ApplicantDocuments/download/{fileName} returns the PHP source verbatim.

Recommendation

Validate uploaded file type/content server-side against an allow-list; reject executable/markup types on the KYC document endpoint.

Unauthenticated 2FA-code delivery to arbitrary users

LOW

Location POST `https://api.example.test/api/Auth/SendTwoFactorCode`

Complexity: low**Exposure:** public

Evidence

Unauthenticated 2FA code generation for an arbitrary user. POST `/api/Auth/SendTwoFactorCode` accepts a raw client-supplied `userId` and, with NO authentication, triggers a 2FA code email to that user's address. Verified anonymously (no cookies/session) for two different users: [redacted-user] (`c15890ad-c7c8-4146-bf08-6ee37b81dde9`) -> 200 "Two-factor authentication code has been sent to your email." (obs `c1cb9338`), and a DIFFERENT user [redacted-user] (`7530c756-5d03-47ee-bcac-015102173a0f`) -> 200 same message (obs `e0800665`). The endpoint trusts `userId` instead of binding to a server-side login challenge, so any anonymous caller can spam 2FA codes to any account's inbox (email bombing / phishing pretext) and use it as a 2FA-enabled-account oracle. User GUIDs are enumerable via `GetAllUsers` (finding `f1c325b9`).

Reproduction Steps

1) Obtain any user's GUID (`GetAllUsers` leaks all user GUIDs). 2) POST `https://api.example.test/api/Auth/SendTwoFactorCode` with header `Origin: https://app.example.test` and body `{"userId": "<victim-guid>", "deviceFingerprint": null, "purpose": "Login"}` – no session/cookie. 3) Server returns 200 and emails a 2FA code to the victim. Repeat every ~10s (per-user resend cooldown) or across many users.

Recommendation

Bind 2FA-code issuance to an authenticated login challenge/session, not a client-supplied `userId`; rate-limit per account and per source.

Unauthenticated 2FA attempt-exhaustion / account-lockout denial of service

LOW

Location POST https://api.example.test/api/Auth/VerifyTwoFactorCode

Complexity: low**Exposure:** public

Evidence

Unauthenticated attacker can exhaust any user's 2FA verification attempts and lock the account's 2FA for 30 minutes (login DoS), and read the victim's 2FA attempt state. POST /api/Auth/VerifyTwoFactorCode accepts a raw client-supplied userId with NO authentication and decrements a per-user counter. Verified anonymously against [redacted-user] (c15890ad-c7c8-4146-bf08-6ee37b81dde9): attempts remaining went 4->3->2->1 (obs fb6a0eaf, cdc87a03, dd3434ee, 7045db22) then the 5th attempt returned 400 "Maximum verification attempts exceeded. You have used all 5 attempts. Please try again after 30 minutes." (obs 54419e0e). GET /api/Auth/GetTwoFactorAttempts?UserId=<guid> is likewise unauthenticated and discloses {"remainingAttempts": N, "maxAttempts": 5, "usedAttempts": M} for any user (obs 3c0b46d2, 59b32c74); the purpose parameter does not isolate the counter (Login vs ResetPassword share it, obs a4bf3268/ab5dcd4d). The SPA treats remainingAttempts===0 as a terminated session, so a locked-out victim cannot complete 2FA login.

Reproduction Steps

1) Get a victim GUID. 2) POST /api/Auth/VerifyTwoFactorCode (Origin: https://app.example.test) body {"userId":"<victim-guid>","code":"000000","purpose":"Login"} with no session – repeat 5 times; each returns "Attempts remaining: N" and the 5th returns "Maximum verification attempts exceeded... try again after 30 minutes". 3) GET /api/Auth/GetTwoFactorAttempts?UserId=<victim-guid> (no session) to read remainingAttempts/maxAttempts/usedAttempts.

Recommendation

Bind 2FA verification and attempt-count lookup to a server-side session; do not expose or decrement attempt state for an arbitrary userId; isolate counters per purpose.

User enumeration via password-reset differential response

LOW

Location POST https://api.example.test/api/Auth/ForgotPassword

Complexity: low**Exposure:** public

Evidence

User enumeration via differential response on the password-reset request. POST /api/Auth/ForgotPassword returns 200 {"data":"Password reset link has been sent to your email.", "isSuccess":true} for a registered email ([redacted-email], obs e0e5d971) but 400 {"errorMessages":["User not found"], "isSuccess":false} for an unregistered one (nonexistent@example.com, obs 442e6dce). An unauthenticated attacker can therefore confirm whether any email address has an account on the platform. Rate limit is 3/hour (global).

Reproduction Steps

POST https://api.example.test/api/Auth/ForgotPassword with Origin: https://app.example.test and body {"email":"<candidate>"}. A registered email yields 200 isSuccess:true; an unknown email yields 400 "User not found".

Recommendation

Return an identical generic response for known and unknown emails on ForgotPassword; apply per-account and global rate limiting.

Informational Severity (5 findings)

SignalR hub returns verbose error detail

INFO

Location POST https://api.example.test/messageHub

Complexity: moderate

Exposure: authenticated

Business Impact: Internal .NET type names, framework internals and server configuration details (AddHubOptions buffer size) are disclosed to any authenticated client, aiding further exploitation and fingerprinting. Low direct impact but a clear information-disclosure weakness.

Evidence

The SignalR hub is configured with `EnableDetailedErrors=true`, so instead of the default generic error the server returns raw .NET exception messages to the client. Malformed hub invocations leak internal exception type names and framework internals: ``HubException: Method does not exist.``, ``InvalidDataException: Invocation provides 1 argument(s) but target expects 0.``, ``InvalidDataException: Error binding arguments. Make sure that the types of the provided values match the types of the hub method being invoked.``, and for an oversized message ``InvalidDataException: The maximum message size of 32768B was exceeded. The message size can be configured in AddHubOptions.`` (leaks the exact configured buffer limit and the configuration API).

Reproduction Steps

1) Authenticate and open `wss://api.example.test/messageHub?id=<token>`. 2) Handshake `{"protocol":"json","version":1}\x1e`. 3) Send `{"target":"GetOnlineUsers","arguments":["x"],"invocationId":"1","type":1}\x1e` → observe the raw .NET exception text in the `type:3 Completion "error" field`. 4) Send a `>32KB` argument to observe the `AddHubOptions` config leak.

Recommendation

Disable detailed SignalR errors (`EnableDetailedErrors=false`) in non-development environments.

SignalR hub connection survives logout (session not revoked)

INFO

Location POST `https://api.example.test/messageHub`

Complexity: moderate

Exposure: authenticated

Business Impact: A stolen or leaked access token (or an XSS/CSWSH-obtained session) remains usable over the already-open SignalR connection for the full token lifetime (up to 1h) even after the victim logs out, defeating logout as a session-termination control for real-time data. The server does not close affected hub connections on logout.

Evidence

SignalR caches the authenticated principal (`Context.User`) at connection time and does not revalidate it for the life of the connection. After a successful POST `/api/Auth/Logout` (which clears the auth cookies and makes normal REST calls return 401), an already-open `/messageHub` WebSocket continues to successfully invoke hub methods. Verified: before logout `GetUserStatus` returned `isOnline=true`; after logout the same open socket's `GetUserStatus` returned a valid `UserStatus` result (`invocationId y2`, `result null`) while a REST call with the same (now-cleared) session returned 401. The server never closes the socket on logout.

Reproduction Steps

1) Log in as [redacted-user]; open `wss://api.example.test/messageHub?id=<negotiate token>` and handshake. 2) Invoke `GetUserStatus` → succeeds. 3) POST `/api/Auth/Logout` (200; cookies cleared) and confirm a normal REST call now returns 401. 4) Without closing the WebSocket, invoke `GetUserStatus` again → still succeeds, proving the hub connection retains its authenticated principal after logout.

Recommendation

Terminate active SignalR connections on logout and re-validate the access token on reconnect; do not rely on a cached `Context.User` for the connection lifetime.

Access JWT is not revoked on logout – token replay valid until expiry

INFO

Location GET https://api.example.test/api/Auth/GetUserById/{id}

Complexity: moderate**Exposure:** authenticated

Business Impact: Stolen/leaked access tokens remain usable for their full 1-hour lifetime after the user logs out; logout does not terminate an attacker's session. On a bank KYC platform this extends the window for cross-org PII/document access using a compromised token.

Evidence

Access JWT (HS256, httpOnly cookie `cxp_access_token`) is NOT revoked server-side on logout. POST /api/Auth/Logout returns 200 and clears the cookies, but replaying the exact same, previously-issued access token afterwards still authenticates (200 + full user object). The token carries a ``sid`` claim, implying session tracking, yet no revocation list/blacklist is enforced – the token stays valid until its ``exp`` (1h lifetime). An attacker who captured the token (e.g. via XSS/logs) keeps authenticated access for up to an hour after the victim logs out, defeating logout as a remediation step.

Reproduction Steps

1. Login as [redacted-email]; capture `cxp_access_token` cookie value T (JWT, exp ~1h).
2. GET https://api.example.test/api/Auth/GetUserById/c15890ad-c7c8-4146-bf08-6ee37b81dde9 with Cookie: `cxp_access_token=T` -> 200.
3. POST https://api.example.test/api/Auth/Logout with Cookie: `cxp_access_token=T` and Referer: https://app.example.test/ -> 200, Set-Cookie clears tokens.
4. Re-issue step 2 with the SAME T -> still 200 (authenticated). No server-side revocation.

Recommendation

Revoke access tokens server-side on logout (maintain a jti/sid revocation list or short-lived tokens with refresh invalidation) so a replayed token no longer authenticates after logout.

Web cache deception – authenticated API responses served to anonymous clients

INFO

Location GET https://api.example.test/api/admin/Entity/Tree.css

Complexity: low**Exposure:** public

Evidence

Web cache deception: Cloudflare caches authenticated API responses on api.example.test and serves them to unauthenticated clients. The origin (IIS/ASP.NET) emits `Cache-Control: max-age=14400` on any response whose URL path ends in a static file extension (e.g. `.css`), and Cloudflare's edge cache honours it. The cache key is only scheme+host+path+query – it does NOT include the session cookie/JWT – so a response generated for an authenticated request is replayed to anonymous requests for the same URL.

Reproduction (no auth needed for step 3):

1. Authenticated (cookie `cxp_access_token`) GET https://api.example.test/api/admin/Entity/Tree.css -> 200, `cache-control: max-age=14400`, `cf-cache-status: MISS`, `x-correlation-id: aa58cce4-...` (obs 8b763aa8).
2. Repeat with session -> 200 `cf-cache-status: HIT`, `age: 16`, same correlation id (obs bf8b2612).
3. WITHOUT any cookie (empty request_headers) GET the same URL -> 200 `cf-cache-status: HIT`, `age: 17`, SAME `x-correlation-id: aa58cce4-...` (obs 780e1c32) – the anonymous client received the response the authenticated request produced.

Same primitive reproduced on a second endpoint: authenticated GET `/api/UserGroup/by-user/{guid}.css` -> 200 `max-age=14400` MISS (obs 6f95a4b1); anonymous GET -> 200 HIT age 174, same correlation id d880f489 (obs ff4b6806).

Scope: any API route whose last path segment is a `{param}` matches with the extension appended (literal routes 404), and the origin marks it cacheable. Confirmed cacheable 200s: `/api/admin/Entity/Tree.css`, `/api/admin/Entity/Filter.css`, `/api/admin/Entity/hierarchy.css`, `/api/UserGroup/by-user/{guid}.css`, `/api/OrganizationUnits/GetFlaggedUnitsForDropdown.css`. In this instance the cached bodies are `{"data":null}` / `{"data":[]}` (the extension corrupts the route parameter), so no PII was demonstrated – but the cache is storing session-scoped API responses and replaying them cross-session, so any endpoint that returns user data at a static-extension URL is exposed. Also note 404/405 error responses carry the same `max-age=14400` and are cached (obs 64718695 HIT), giving a cache-poisoning/DoS primitive.

Recommendation

Do not emit `Cache-Control: max-age` on authenticated API responses; add `Cache-Control: private, no-store` and `Vary: Cookie/Authorization`; configure the CDN to bypass cache for `/api/*` and never key solely on path/extension.

Missing server-side file-size validation on document upload

INFO

Location POST https://api.example.test/api/ApplicantDocuments/upload (Document)

Complexity: low**Exposure:** authenticated

Evidence

Server-side file-size validation is missing on the document upload endpoint. The Angular UI advertises a hard limit ("PDF JPG PNG DOC max 5MB", translation key `cxp_applications_pdfjpgpngdocmaxmb`) and blocks larger files client-side, but the API accepts arbitrarily large files. Uploaded a 6 MiB file (applicantDocument id 523) and a 20 MiB file (id 525) – both returned 200 {"data":<id>,"isSuccess":true}. Only the client enforces the 5 MB limit, so an authenticated introducer can fill server disk / exhaust resources by repeatedly uploading large files (DoS). The same endpoint also stores the client-supplied originalFileName verbatim with no length/character validation.

Reproduction Steps

1) Login as [redacted-user]. 2) GET /api/applications/150/documents/expected-materialized?applicationType=Individual to obtain applicantPublicId (1be07d2d-803c-f111-8603-d1c237308e24) and an expected id (2861, DocumentTypeId 101). 3) POST /api/ApplicantDocuments/upload as multipart/form-data with Origin/Referer https://app.example.test/ and a Document file part >5 MiB (e.g. 20 MiB). 4) Observe HTTP 200 with a new applicantDocument id – no size rejection.

Recommendation

Enforce the documented file-size limit server-side on the upload endpoint (reject oversize uploads) in addition to the client-side check.

COVERAGE MATRIX

The Status column summarises the result for each vulnerability class: **Findings** means one or more verified vulnerabilities were discovered, **Clear** means the class was tested and no exploitable issues were found, **Failed** means the class was attempted but the testing stage did not complete cleanly so coverage is incomplete, and **Not Tested** means the class was not selected for this engagement.

Vulnerability Class	Findings	Status
xss	2	Findings
idor	3	Findings
injection	0	Clear
csrf	0	Clear
open_redirect	0	Clear
ssrf	0	Clear
business_logic	0	Clear
auth_session	4	Findings
subdomain_takeover	0	Clear
jwt	0	Clear
lfi	0	Clear
cache_poisoning	0	Clear
validation	1	Findings
ai_agent	0	Clear
access_control	10	Findings
other	2	Findings
privilege_escalation	1	Findings
race_condition	0	Clear
unsafe_file_upload	1	Findings
user_enumeration	1	Findings

GLOSSARY

CSRF

Cross-Site Request Forgery. An attack that forces authenticated users to submit unintended requests by exploiting the browser's automatic inclusion of cookies.

XSS

Cross-Site Scripting. An injection attack where malicious scripts are injected into trusted websites, executing in victims' browsers.

IDOR

Insecure Direct Object Reference. An access control vulnerability where modifying a resource identifier in a request grants unauthorized access to other users' data.

SameSite

A cookie attribute that controls whether cookies are sent with cross-site requests. Values: Strict (never), Lax (top-level navigations only), None (always).

JWT

JSON Web Token. A compact, URL-safe token format used for authentication and information exchange, consisting of a header, payload, and signature.

SSRF

Server-Side Request Forgery. A vulnerability where an attacker can make the server send requests to unintended locations, potentially accessing internal services.

SQLi

SQL Injection. An attack where malicious SQL code is inserted into application queries, potentially allowing data theft, modification, or deletion.

LFI

Local File Inclusion. A vulnerability where an attacker can read arbitrary files from the server by manipulating file path parameters.

Subdomain Takeover

A vulnerability where an attacker claims control of a subdomain that points to an unclaimed external service (e.g. dangling CNAME), enabling phishing and cookie theft.

Cache Poisoning

An attack that manipulates web cache behavior to serve malicious content to other users by exploiting unkeyed request components.

Open Redirect

A vulnerability where an application redirects users to an attacker-controlled URL, commonly used in phishing attacks to exploit trust in the legitimate domain.

Business Logic

A class of vulnerabilities arising from flaws in application workflows or rules rather than technical implementation errors, such as bypassing payment steps or manipulating quantities.

Session Fixation

An attack where the attacker sets a known session identifier before the victim authenticates, then hijacks the session after login using the pre-set identifier.

RCE

Remote Code Execution. A vulnerability that allows an attacker to run arbitrary code on the target server, typically leading to full system compromise.

Privilege Escalation

An attack where a user gains access to resources or functionality beyond their authorized level, either vertically (user to admin) or horizontally (user A accessing user B's data).

DoS

Denial of Service. An attack aimed at making a system or service unavailable. Excluded from scope in this assessment.

APPENDIX A: TEST ACCOUNTS

The following accounts were created and used during the assessment. These credentials are provided so that the client can log in with the same accounts to verify findings or perform follow-up testing.

Email	Role	Password
[redacted]	Introducer – Org A (OwnOnly)	[redacted]
[redacted]	Introducer – Org A (Full)	[redacted]
[redacted]	Introducer – Org B (OwnOnly)	[redacted]
[redacted]	Introducer – Org B (Full)	[redacted]
[redacted]	Bank officer – Unit 1	[redacted]
[redacted]	Bank officer – Unit 2	[redacted]

APPENDIX B: HTTP EVIDENCE

The full report includes the raw HTTP request/response pairs that prove each finding, cross-referenced to its parent finding by ID. That evidence has been removed from this public sample.

[Appendix Content Redacted]