VAPT MANUAL CHECK-LIST

INETSECURITY.IN INITIATIVE

Name:_____

Project Name:_____

INETSECURITY.IN INITIATIVE

companion to Master API VAPT in 10 Hours

VAPT CHECK LIST

A Printable, Hand-Markable Field Companion for API Penetration Testing

Engagement Information — Fill In Before You Begin

Client / Target Name: _____

Tester Name: _____

Engagement Type (VAPT / Bug Bounty / Internal): _____

Start Date: _____

End Date: _____

Authorization Reference (RoE / Scope Doc #): _____

In-Scope Domains / IPs: _____

Out-of-Scope / Excluded: _____

Testing Window / Rate-Limit Constraints: _____

Primary Contact (Client-Side): _____

Test Accounts Provisioned (roles): _____

How to use this document:

☐ Print this document (A4) before starting an engagement, or fill it digitally.

☐ Every checklist item has a ☐ box — tick it only after you have actually performed that check, not just read it.

☐ Every section ends with a RESULT row and a ruled Notes box — use it to jot endpoint names, payloads that worked, IDs to revisit, and anything to remember before writing the final report.

☐ Work top to bottom on your first pass; on a time-boxed engagement, prioritize the vulnerability sections marked HIGH-YIELD.

☐ Use the Master Coverage Tracker at the end to confirm nothing was skipped before you sign off.

SECTION 1

Pre-Engagement Checklist

- ☐ Signed engagement letter / Rules of Engagement (RoE) received and filed.
- ☐ Scope confirmed in writing — domains, subdomains, IP ranges, mobile apps, API versions.
- ☐ Explicitly excluded targets and techniques confirmed (e.g. no DoS, no social engineering, no physical access).
- ☐ Testing window / blackout periods confirmed with client.
- ☐ Emergency contact (client-side) confirmed for critical findings requiring immediate notification.
- ☐ Test accounts created at every privilege tier needed (guest, user, premium, admin, multi-tenant if applicable).
- ☐ Lab/tooling updated: Burp extensions, Nuclei templates (nuclei -update-templates), wordlists refreshed.
- ☐ Legal/compliance considerations noted (data residency, PII handling, NDA terms).

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / FINDINGS / REMINDERS

SECTION 2

Reconnaissance & Inventory Checklist

TOOLS: *subfinder • httpx • Kiterunner • waybackurls / gau • Arjun • MobSF / apktool • TruffleHog / Gitleaks*

Attack Surface Mapping

☐ Full subdomain enumeration completed; API-looking hosts identified (api., api-stage., partner-api., internal.).

☐ Every endpoint (path + method) inventoried — not just what the UI calls.

☐ Current AND legacy/beta/internal API versions identified (v1, v2, beta, internal).

☐ OpenAPI/Swagger spec located and reviewed, if published.

☐ Historical endpoints pulled via waybackurls/gau and checked for live status.

☐ Mobile app (APK/IPA) decompiled; hardcoded base URLs, version strings, and feature flags extracted.

☐ JS bundles swept for hidden endpoints, internal routes, and hardcoded secrets/keys.

☐ GitHub/GitLab dorking performed for leaked Postman collections, SDKs, or config referencing the API.

Classification

☐ Authenticated vs public endpoints clearly separated in notes.

☐ Auth scheme identified per endpoint group (session cookie / JWT / API key / OAuth2 / HMAC).

☐ Multi-tenant boundaries identified, if applicable.

☐ High-value flows shortlisted (auth, payments, admin, account management, data export).

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

ENDPOINTS DISCOVERED / TO REVISIT

SECTION 3

API Understanding Checklist

Authentication & Session

☐ Token storage location identified (header / cookie / query string).

☐ Token refresh mechanism understood; old/refresh token reuse tested.

☐ Multiple roles/tiers identified (free vs paid, user vs admin, tenant A vs tenant B).

☐ Logout / session revocation behavior confirmed.

Authorization Model

☐ Every ID-accepting endpoint listed (userId, orderId, invoiceId, accountId, etc.).

☐ Object-level, function-level, and property-level authorization treated as three SEPARATE tests per endpoint.

☐ Tenant isolation identified as a primary target (if multi-tenant SaaS).

Data & Business Logic

☐ Sensitive business flows mapped (payments, coupons, referrals, password reset, account deletion).

☐ Hidden/extra JSON fields identified by comparing GET responses to what the UI actually shows.

☐ Rate limits identified — and whether enforced per-user, per-IP, or not at all.

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / FINDINGS / REMINDERS

SECTION 4

Automated Scanning Checklist

TOOLS: *Nuclei • OWASP ZAP • Nikto • Schemathesis / Restler • Corsy*

☐ Nuclei sweep run with exposure, misconfig, default-login, and CVE tags.

☐ Schema-based fuzzing (Schemathesis/Restler) run against the OpenAPI spec, if available.

☐ CORS scanner (Corsy) run against every authenticated endpoint.

☐ Results triaged — false positives discarded, true positives queued for manual confirmation.

☐ Automated scan results cross-checked against the manual checklist sections below (scanners alone catch a minority of real API logic bugs).

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / FINDINGS / REMINDERS

SECTION 5

Vulnerability Testing Checklists

One checklist per vulnerability class. ★ = high-yield — test these first on time-boxed engagements.

VULN CHECK 01 — CRITICAL ★ HIGH-YIELD

Broken Object Level Authorization (BOLA / IDOR)

TOOLS: *Burp Suite (Repeater + Autorize extension) • Postman with two saved environments (User A / User B tokens) • ffuf for numeric/UUID ID brute-forcing • Python + requests for scripted differential testing*

Understanding Checklist

☐ List every endpoint whose path or body contains an ID-like parameter.

☐ Determine the ID format: sequential integer, UUIDv4, hash, or encoded (base64/ObjectId).

☐ Do you have two accounts of the same privilege tier (User A, User B) to test cross-account access?

☐ Does the same object have multiple access paths (REST path param, query string, JSON body, GraphQL argument)?

☐ Is authorization enforced server-side, or only via UI hiding (buttons/menus not shown)?

Confirmation Checklist

☐ Cross-account replay returns another user's real data with HTTP 200, not 401/403.

☐ The returned data matches known ground truth (you seeded it with User A) — not a cached or generic response.

☐ Repeat with a third, freshly created account to rule out a caching artifact.

☐ Document exact request/response pair as evidence — this is the core of your PoC.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 02 — CRITICAL ★ HIGH-YIELD

Broken Authentication

TOOLS: *Burp Intruder / Turbo Intruder for high-speed brute force • Hydra for classic credential stuffing • jwt_tool for token structure analysis • ffuf for OTP/reset-token brute forcing*

Understanding Checklist

☐ Does /login (or /token, /auth) differentiate 'invalid user' from 'invalid password' in response body, status code, or timing?

☐ Is there rate limiting or progressive delay after failed attempts, and is it per-account or per-IP (or both)?

☐ Are password-reset and OTP codes short, numeric, and not rate-limited?

☐ Do JWTs have a sane expiry, and is the signature algorithm enforced server-side (see Chapter 17)?

☐ Can a session/token be reused after logout or password change (no server-side revocation)?

Confirmation Checklist

☐ You have successfully authenticated with a brute-forced or enumerated credential and received a valid session token.

☐ Or: you have demonstrated N failed attempts with zero lockout/backoff, screenshotted as evidence.

☐ Or: an old/revoked token still succeeds against a protected endpoint after logout — capture both requests as before/after evidence.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 03 — HIGH ★ HIGH-YIELD

Broken Object Property Level Authorization (Mass Assignment & Excessive Data Exposure)

TOOLS: *Burp Suite Repeater • Postman schema inspection • Arjun for hidden parameter discovery • json-diff tools for comparing GET response vs PATCH acceptance*

Understanding Checklist

☐ Does a GET on this object return fields not shown in the UI (isAdmin, isVerified, balance, internal_notes, password_hash)?

☐ Does the same object model back multiple endpoints (public profile GET vs internal admin GET) that might reveal extra fields?

☐ For every write endpoint (POST/PUT/PATCH), have you tried submitting every field seen in any GET response, even undocumented ones?

☐ Are there naming-convention guesses worth trying (isAdmin, is_admin, admin, role, roleId, permissions, verified, balance, price, discount)?

Confirmation Checklist

☐ A field you were never given permission to set (role, isAdmin, balance, price) changes after your request and persists on a follow-up GET.

☐ For excessive exposure: a GET response contains another user's data, credentials, or clearly internal-only fields (e.g. cost_price alongside sell_price).

☐ Reproduce twice to rule out a race condition or cached response.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 04 — MEDIUM

Unrestricted Resource Consumption (Rate Limiting & DoS)

TOOLS: *Burp Intruder for concurrent request flooding (lab-authorized only) • Apache Bench (ab) / hey for load testing • wrk for HTTP benchmarking • Custom Python asyncio scripts for concurrency tests*

Understanding Checklist

☐ Is there a documented or observable rate limit (check response headers: X-RateLimit-Limit, Retry-After)?

☐ Are pagination parameters (limit, pageSize, per_page) bounded server-side, or can you request limit=999999?

☐ Are file upload endpoints capped in size and count per time window?

☐ Are expensive operations (search, export, report generation, email sending) throttled independently from simple GETs?

☐ Is rate limiting applied per-account/per-API-key, or only per-IP (trivially bypassed via proxies/VPN rotation)?

Confirmation Checklist

☐ Response time or error rate degrades measurably as request volume/parameter size increases, with no throttling response (429) ever returned.

☐ An oversized parameter (e.g. limit=100000) is accepted and returns proportionally more data / takes proportionally longer, unbounded.

☐ Header manipulation (X-Forwarded-For rotation) resets a per-IP counter, proving the limiter is trivially bypassable.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 05 — CRITICAL ★ HIGH-YIELD

Broken Function Level Authorization (Privilege Escalation)

TOOLS: *Burp Suite (manual endpoint discovery + Repeater) • Autorize / AuthMatrix (Burp extensions) for automated role-matrix testing • Kiterunner / ffuf to discover hidden /admin, /internal, /staff routes • JS bundle analysis (LinkFinder, SourceMapper) for admin routes referenced client-side*

Understanding Checklist

☐ Have you fully mapped which endpoints exist for EACH role (admin, staff, vendor, regular user, guest)?

☐ For every admin/privileged endpoint you discover, do you have a low-privileged token ready to test it?

☐ Have you checked JS bundles, Swagger docs, and mobile APK strings for admin-only route names?

☐ Does the API version matter — is an old v1 admin route still live and unprotected while v2 is fixed?

☐ Are HTTP verbs consistently protected (GET /admin/users blocked, but PUT /admin/users/{id} open)?

Confirmation Checklist

☐ A low-privileged (or unauthenticated) token successfully performs a privileged action — HTTP 200/201/204 with real effect (verify by checking the object state changed).

☐ You have ruled out that the response is a misleading 200 with no actual server-side change (always verify via a follow-up GET).

☐ Reproduce on at least one more privileged function to demonstrate a pattern, not a one-off.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 06 — HIGH

Unrestricted Access to Sensitive Business Flows

TOOLS: *Custom Python/requests automation scripts • Burp Turbo Intruder for concurrency testing • Multiple test accounts / disposable email automation for scale testing • Browser automation (Playwright/Selenium) to test flows that require full session context*

Understanding Checklist

☐ Which flows are business-critical and abuse-sensitive: purchases, referrals, coupon codes, account creation, review posting, voting/rating?

☐ Is there CAPTCHA, device fingerprinting, or behavioral analysis on these flows, and can the API be called directly bypassing them?

☐ Is there a per-account or per-device limit, and is it enforced server-side?

☐ Can the flow be automated end-to-end via the API alone, without ever touching the web UI?

Confirmation Checklist

☐ The full flow completes successfully via direct API calls, with a blank/invalid/reused CAPTCHA or anti-bot token.

☐ Business-limited actions (e.g. one coupon per user) succeed multiple times across accounts you control, demonstrating the limit is not enforced.

☐ Document with a proof video/script showing volume achieved in a short window, which is the strongest evidence for business stakeholders.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 07 — HIGH ★ HIGH-YIELD

Server-Side Request Forgery (SSRF)

TOOLS: *Burp Collaborator / Interactsh for out-of-band (blind) SSRF detection • curl for direct manual testing • SSRFmap for automated SSRF exploitation chains • gopherus for crafting Gopher-protocol payloads against internal services*

Understanding Checklist

☐ Which endpoints accept a URL, hostname, IP, or file path as input (webhooks, avatar-by-URL, PDF/screenshot generation, 'import from link', SSO metadata URL)?

☐ Is there any allow-list or validation on the target host, and can it be bypassed (redirect chains, DNS rebinding, alternate IP encodings)?

☐ Is the server running in a cloud environment with a metadata service (169.254.169.254 for AWS/GCP/Azure)?

☐ Does the response ever reflect any part of the fetched content back to you (visible SSRF), or is it fully blind (requires OOB)?

Confirmation Checklist

☐ An Interactsh/Collaborator DNS or HTTP interaction is logged after triggering the vulnerable feature — proof the server made an outbound request you controlled.

☐ For visible SSRF, the response body contains content only obtainable by reaching an internal resource (metadata credentials, internal service banner, localhost admin page).

☐ Confirm impact by successfully fetching a genuinely internal-only resource, not just an external domain (which is expected/benign).

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 08 — MEDIUM

Security Misconfiguration

TOOLS: *Nuclei (misconfiguration and exposure template packs) • Nikto for classic web server misconfig scanning • Burp Suite manual verb/path testing • Shodan/Censys for exposed staging or debug endpoints at scale*

Understanding Checklist

☐ Do error responses (400/500) leak stack traces, internal file paths, SQL queries, or framework version banners?

☐ Are common debug/admin/monitoring paths present and unauthenticated (/actuator, /debug, /swagger-ui, /.env, /admin, /console)?

☐ Are security headers present and correct (CORS, CSP where relevant, X-Content-Type-Options, Strict-Transport-Security)?

☐ Does the server respond to unusual HTTP methods (TRACE, TRACK, CONNECT) with information disclosure?

☐ Are default or sample credentials still active on any exposed admin panel?

Confirmation Checklist

☐ A sensitive path returns real internal data (env vars, config, source code, credentials) with HTTP 200.

☐ An error response includes a full stack trace revealing internal file structure, ORM queries, or library versions.

☐ A default credential pair successfully authenticates to an exposed admin/monitoring interface.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 09 — MEDIUM

Improper Inventory Management (Shadow & Zombie APIs)

TOOLS: *waybackurls / gau for historical endpoint discovery • Kiterunner for OpenAPI/Swagger-informed route brute-forcing • GitHub dorking + Gitleaks/TruffleHog for leaked internal API references • Mobile APK/IPA static analysis (apktool, MobSF) for hardcoded legacy endpoints*

Understanding Checklist

☐ Have you checked web.archive.org and waybackurls for any historically indexed API paths?

☐ Have you decompiled the mobile app (APK/IPA) to search for hardcoded base URLs, version strings, or feature-flagged endpoints?

☐ Have you tried common version prefixes against every known endpoint (v1, v2, v3, beta, internal, legacy, old)?

☐ Have you searched GitHub/GitLab (including forks and old commits) for the company's API client code or Postman collections?

☐ Does DNS recon reveal additional API subdomains (api-old, api-staging, api-internal, partner-api)?

Confirmation Checklist

☐ A discovered legacy/undocumented endpoint responds with real data (HTTP 200) and is not present in the current documented API surface.

☐ You demonstrate the legacy endpoint lacks a security control that the current version has (e.g. no rate limit, weaker auth, unpatched BOLA).

☐ Evidence includes both the discovery method (wayback/APK string) and the live confirmation request/response.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 10 — MEDIUM

Unsafe Consumption of Third-Party APIs

TOOLS: *Burp Suite Repeater for webhook forgery testing • Postman for scripting signature-bypass attempts • OpenSSL for HMAC/signature crafting when testing your own authorized webhook configuration • mitmproxy for intercepting outbound third-party calls the app makes*

Understanding Checklist

☐ Does the application accept inbound webhooks from third parties, and are those requests cryptographically signature-verified?

☐ Does the application follow redirects or fetch content from third-party-supplied URLs without validation (ties into SSRF, Chapter 12)?

☐ Is data received from a third-party API (even a 'trusted' one) sanitized before being stored or rendered, or trusted implicitly?

☐ Are API keys/secrets used to call third-party services scoped minimally, or do they carry excess privilege that would matter if leaked?

Confirmation Checklist

☐ A forged webhook, sent without a valid provider signature, is accepted and changes real application state (e.g. order marked paid).

☐ If a signature check exists but is broken, you can demonstrate a request with a tampered/blank signature still succeeding.

☐ Document the state change with a before/after screenshot of the affected object (order status, account balance, etc.).

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 11 — CRITICAL ★ HIGH-YIELD

SQL & NoSQL Injection

TOOLS: *sqlmap for automated SQL injection detection/exploitation • NoSQLMap for MongoDB-style NoSQL injection • Burp Suite Intruder for manual payload sweeping • Interactsh/Burp Collaborator for blind/out-of-band SQLi confirmation*

Understanding Checklist

☐ Which parameters (path, query string, JSON body fields, headers like X-Forwarded-For) reach a database query?

☐ Is the backend SQL (Postgres/MySQL/MSSQL) or NoSQL (MongoDB, DynamoDB, Elasticsearch)?

☐ Do error responses differ between valid and syntactically-broken input (a strong signal for error-based/blind injection)?

☐ For JSON APIs: does the endpoint accept nested objects/arrays where a string is expected, and does it reject or silently accept them?

Confirmation Checklist

☐ sqlmap successfully extracts a database/table name or confirms an injectable parameter with a specific technique and DBMS fingerprint.

☐ A NoSQL operator payload authenticates you as a target account without knowing its real password.

☐ Time-based payloads produce a measurable, repeatable delay proportional to the injected sleep value (rule out network jitter by repeating 3x).

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 12 — HIGH

XML External Entity (XXE) Injection

TOOLS: *Burp Suite Repeater for manual XXE payload crafting • XXEinjector for automated exploitation • Interactsh/Burp Collaborator for blind XXE via OOB entity resolution • docx/xlsx/svg file crafters for XXE-via-file-upload testing*

Understanding Checklist

☐ Does any endpoint accept Content-Type: application/xml, text/xml, or SOAP envelopes?

☐ Do file-upload endpoints accept formats that are secretly XML-based (SVG, DOCX, XLSX, PPTX are all ZIP-wrapped XML)?

☐ Does the response ever reflect any part of the parsed XML content back to you?

☐ Is the target parser one with a history of XXE-by-default (older libxml2, older Java XML parsers, .NET XmlDocument without safe settings)?

Confirmation Checklist

☐ The response body directly contains contents of a local file you referenced (e.g. root:x:0:0 from /etc/passwd).

☐ For blind XXE, your Interactsh/Collaborator server logs an inbound request triggered by the parser resolving the external DTD.

☐ For file-upload-based XXE, the server-side processing step (thumbnail, text extraction) errors or behaves differently when the malicious file is uploaded versus a benign one.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 13 — CRITICAL

Command Injection

TOOLS: *Burp Suite Repeater for manual payload injection • Interactsh/Burp Collaborator for blind command injection confirmation (out-of-band DNS/HTTP) • Commix for automated command injection detection • curl for direct scripted testing*

Understanding Checklist

☐ Does any endpoint's purpose imply shelling out to an OS utility (ping, traceroute, image/video conversion, PDF rendering, file compression, DNS lookup)?

☐ Are file names or paths user-controlled and passed to a shell command (e.g. a filename used in an unzip or convert call)?

☐ Does the response reflect command output directly, or is it fully blind (no visible feedback)?

☐ What OS is the backend running (Linux vs Windows) — this changes payload syntax and available OOB techniques.

Confirmation Checklist

☐ Command output (username, file contents, hostname) appears in the API response where only ping results were expected.

☐ A time-based payload (sleep 10) produces a consistent ~10 second delay across repeated tests, ruling out network noise.

☐ An OOB payload triggers a logged DNS/HTTP interaction on your Interactsh/Collaborator server, confirming blind execution.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 14 — CRITICAL ★ HIGH-YIELD

JWT & Token Vulnerabilities

TOOLS: *jwt_tool for comprehensive JWT attack automation • Burp Suite JSON Web Tokens extension • hashcat / John the Ripper for offline HMAC secret cracking • CyberChef for manual JWT decode/encode/tamper workflows*

Understanding Checklist

☐ What algorithm does the token use (HS256, RS256, ES256, none)?

☐ Is the algorithm enforced server-side as a fixed expectation, or does the server trust whatever 'alg' the token header claims?

☐ Are expiry (exp), not-before (nbf), and issuer (iss) claims actually validated, or just present?

☐ If HS256, is the secret plausibly weak enough to brute-force offline?

☐ Are tokens revocable server-side (logout, password change), or valid until natural expiry regardless of account state changes?

Confirmation Checklist

☐ A tampered token (elevated role, changed user ID, or 'none' algorithm) is accepted by the API and grants access matching the tampered claim.

☐ An offline-cracked or algorithm-confused re-signed token successfully authenticates.

☐ A revoked/expired token is still accepted past its expected invalid point, evidenced by a before/after comparison.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 15 — HIGH

GraphQL-Specific Vulnerabilities

TOOLS: *InQL (Burp extension) for GraphQL schema mapping and vulnerability scanning • GraphQL Voyager / Altair GraphQL Client for schema visualization • graphql-cop for automated GraphQL-specific misconfiguration checks • Burp Suite Repeater for manual query/mutation crafting*

Understanding Checklist

☐ Is introspection enabled on the production GraphQL endpoint (test directly, don't trust the client app's behavior)?

☐ Does the schema expose fields/mutations/queries not used anywhere in the actual application?

☐ Are deeply nested queries (self-referencing types) capped in depth or complexity server-side?

☐ Does the endpoint accept batched queries in a single request, and is rate limiting applied per-operation or per-HTTP-request?

☐ Is authorization checked at the object level only, or independently on every sensitive field resolver?

Confirmation Checklist

☐ The introspection query returns the full schema despite the client application not using introspection normally.

☐ An undocumented query/mutation discovered via the schema executes successfully and returns or modifies real data beyond your authorization level.

☐ A deeply nested or batched query produces a measurable performance impact or bypasses an otherwise-observed rate limit.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 16 — MEDIUM

CORS Misconfiguration

TOOLS: *Burp Suite Repeater (manual Origin header manipulation) • Corsy for automated CORS misconfiguration scanning • A simple hosted HTML PoC page for live confirmation • curl for scripted header testing*

Understanding Checklist

☐ Does the API set Access-Control-Allow-Origin at all, and if so, is it a fixed value or dynamically reflected?

☐ Is Access-Control-Allow-Credentials set to true alongside a reflected or wildcard-adjacent origin?

☐ Does the server accept null as a valid origin (exploitable via sandboxed iframes)?

☐ Does the server use a weak origin-matching regex (e.g. matching *.target.com but also matching targetcom.evil.com via naive substring checks)?

Confirmation Checklist

☐ The server reflects an arbitrary attacker-controlled Origin back in Access-Control-Allow-Origin AND sets Allow-Credentials: true simultaneously.

☐ A hosted PoC HTML page successfully performs a credentialed cross-origin fetch and displays the victim's private data in the browser console/page, captured on video/screenshot.

☐ Confirm the affected endpoint actually returns sensitive data (a misconfiguration on a public, non-sensitive endpoint has minimal real impact).

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 17 — HIGH

API Key, Secret & Credential Leakage

TOOLS: *TruffleHog / Gitleaks for repository and filesystem secret scanning • LinkFinder + a custom regex sweep for JS-bundle secret extraction • GitHub/GitLab dorking (site:github.com + company name + api_key) • Shodan/Censys for exposed .env or config files at scale*

Understanding Checklist

☐ Have you downloaded and grepped every JS file the target application loads for key-like patterns (AKIA, sk_live_, xox[bp]-, AIza, ey J for JWTs used as static secrets)?

☐ Have you checked the company's public GitHub organization, including forks, old commits, and deleted-but-cached commits via GitHub's search?

☐ Have you tried requesting common config/env file paths directly (/.env, /config.json, /firebase-config.js)?

☐ Have you checked mobile app binaries (APK/IPA) for hardcoded keys using static string extraction?

☐ For any key found, have you determined its actual scope/permissions before assuming maximum impact?

Confirmation Checklist

☐ A found key successfully authenticates against its provider's API (verified via a read-only, non-destructive call like get-caller-identity, or a scoped 'whoami' equivalent).

☐ You have documented the exact permission scope the key grants (e.g. full S3 read/write on a specific bucket, or a Stripe secret key with charge-creation ability).

☐ The key is confirmed to originate from the in-scope target, not a third-party library or unrelated public sample.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 18 — HIGH ★ HIGH-YIELD

Business Logic Vulnerabilities (Price, Quantity & Race Conditions)

TOOLS: *Burp Suite Repeater and Intruder (for race-condition timing) • Turbo Intruder for precise concurrent request race conditions • Postman for scripted multi-step business flow replay • A spreadsheet/notebook to model expected vs actual state at each step*

Understanding Checklist

☐ At every step of a multi-step flow (cart -> checkout -> payment -> confirmation), which values does the SERVER recalculate versus TRUST from the client?

☐ Can any numeric field (price, quantity, discount percentage, points balance) be set to zero, negative, or an unexpectedly large value?

☐ Are single-use resources (coupon codes, referral bonuses, gift cards, free trial flags) protected against being redeemed twice via simultaneous requests?

☐ Does changing the ORDER of API calls (e.g. applying a discount after payment confirmation instead of before) produce an inconsistent state?

Confirmation Checklist

☐ A tampered numeric field is accepted and reflected in the final, persisted business record (order total, balance, discount) rather than being recalculated server-side.

☐ A single-use resource (coupon, referral, gift card) is successfully redeemed more than once, confirmed by checking the account's real balance/history afterward.

☐ Out-of-sequence API calls produce a state the intended flow should have prevented (e.g. a shipped order that was never actually marked paid).

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 19 — HIGH

File Upload Vulnerabilities

TOOLS: *Burp Suite Repeater (manual header/filename/magic-byte manipulation) • ffuf for extension/bypass wordlist fuzzing • exiftool for crafting files with malicious embedded metadata • A local web shell payload set (for authorized engagements only) matched to the identified backend language*

Understanding Checklist

☐ Is file type validated by Content-Type header, file extension, magic bytes/content, or a combination — and which of these are only client-side?

☐ Is the upload destination web-accessible, and is script execution possible in that directory (or a related one via path traversal)?

☐ Is the uploaded filename sanitized, or can path traversal sequences (../../) or absolute paths be embedded in it?

☐ Is there a file-size limit, and does the server validate file content matches its claimed type (e.g. an actual image parser, not just a header check)?

☐ Are uploaded files later processed by another vulnerable component (image resizer, PDF renderer, ZIP extractor) that has its own attack surface (ties to Chapters 17 and 9)?

Confirmation Checklist

☐ The uploaded file is stored with its executable extension intact (or a bypassed variant) in a location reachable via direct URL.

☐ Requesting the uploaded file directly executes server-side code (confirmed via a minimal, non-destructive command's output appearing in the response).

☐ For stored-XSS-via-upload cases, the file is served with a content type that triggers browser script execution (text/html or image/svg+xml) rather than being forced to a safe download.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

VULN CHECK 20 — MEDIUM

HTTP Request Smuggling & Parameter Pollution

TOOLS: *Burp Suite (built-in HTTP Request Smuggler extension / native desync detection) • Burp Repeater with 'Update Content-Length' disabled for precise raw request crafting • curl --http1.0/1.1 with raw socket-level control for manual confirmation • Custom Python sockets for parameter pollution differential testing across layers*

Understanding Checklist

☐ Is the API served behind a reverse proxy, load balancer, WAF, or CDN in front of the actual application server (check via Server headers, response timing differences, or known provider fingerprints)?

☐ Does the stack mix HTTP/1.0, 1.1, and 2 handling anywhere (common in CDN-to-origin translation)?

☐ Do any endpoints accept the same parameter name multiple times (?id=1&id=2, or duplicate JSON-adjacent form fields), and if so, which value 'wins' at each layer?

☐ Are validation checks (e.g. an authorization check) performed at the proxy/WAF layer separately from the business logic at the app layer, creating an opportunity for a parameter/parsing mismatch between the two?

Confirmation Checklist

☐ Automated desync detection (Burp) flags a confirmed smuggling primitive with a specific technique classification (CL.TE, TE.CL, TE.TE).

☐ A crafted smuggled request measurably affects a subsequent request on the same connection in a controlled test (e.g. response mismatch), confirmed WITHOUT impacting real user traffic — this class of testing carries real risk and should be minimized and tightly scoped.

☐ For parameter pollution, you demonstrate a concrete case where a validation layer and the application layer disagree on which duplicate parameter value is authoritative, resulting in a bypass of an intended restriction.

Endpoint(s) / Parameter(s) Tested: _____

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / PAYLOADS THAT WORKED / REVISIT LATER

SECTION 6

Chaining & Business Logic Review

Run this after individual vulnerability testing — this is where Low/Medium findings become Critical.

SECTION 6

Chaining Review Checklist

- ☐ For every confirmed finding, asked: 'What does this let me reach that I couldn't reach before?'
- ☐ Re-tested every OTHER checklist section against any NEW access level or endpoint gained.
- ☐ Checked whether any discovered leaked credential/key/token unlocks a different vulnerability class.
- ☐ Checked whether a low-severity data leak (BOLA/exposure) feeds a business-logic or account-takeover chain.
- ☐ Built a simple map/diagram of every credential, token, role, and internal path discovered so far.
- ☐ Attempted at least one full end-to-end PoC connecting 2+ findings into a single narrative.

CHAIN MAP / FINDINGS THAT CONNECT

SECTION 6B

Business Logic & Race Condition Checklist

☐ Identified which values are server-recalculated vs client-trusted at every step of a multi-step flow.

☐ Tested negative, zero, and oversized values on every numeric field (price, quantity, discount, points).

☐ Tested single-use resources (coupons, referrals, gift cards) for double-redemption via race conditions (Turbo Intruder single-packet attack).

☐ Tested out-of-order execution of multi-step flows (skip a step, call steps in reverse).

☐ Verified impact against REAL persisted state (order record, balance) — not just a 200 OK response.

RESULT: ☐ Not Tested ☐ Tested – No Issue ☐ Tested – Vulnerable ☐ Not Applicable

NOTES / FINDINGS / REMINDERS

SECTION 7

Reporting & Sign-Off

Confirm every finding is reproducible and every section above has been addressed before delivering the report.

SECTION 7

Reporting Checklist

☐ Every finding has: title, CVSS score + full vector string, summary, affected endpoint(s), steps to reproduce, PoC, impact, remediation, and reference (OWASP/CWE).

☐ Steps to reproduce tested by re-following them exactly, as if a stranger with no context was reading them.

☐ Screenshots/PoC evidence attached and redacted of any real customer PII before sharing.

☐ Chained findings written up as a single connected narrative, not disconnected bullet points.

☐ Executive summary opens with business risk in plain language, not technical jargon.

☐ Findings-by-severity summary table included at the top of the report.

☐ CVSS scores double-checked against the official calculator (first.org) for consistency.

☐ Report reviewed for accuracy and tone before delivery — no inflated severities, no unverified claims.

FINAL REMINDERS BEFORE SUBMISSION / DELIVERY

MASTER TRACKER

Coverage Tracker

Tick one outcome box per row before you consider the engagement complete.

COVERAGE

Master Vulnerability Coverage Tracker

Use this as your final pass — every row below should have exactly one outcome ticked before sign-off.

#	Vulnerability Class	Not Tested	No Issue	Vulnerable	N/A
1	Broken Object Level Authorization (BOLA / IDOR)	☐	☐	☐	☐
2	Broken Authentication	☐	☐	☐	☐
3	Broken Object Property Level Authorization (Mass Assignment & Excessive Data Exposure)	☐	☐	☐	☐
4	Unrestricted Resource Consumption (Rate Limiting & DoS)	☐	☐	☐	☐
5	Broken Function Level Authorization (Privilege Escalation)	☐	☐	☐	☐
6	Unrestricted Access to Sensitive Business Flows	☐	☐	☐	☐
7	Server-Side Request Forgery (SSRF)	☐	☐	☐	☐
8	Security Misconfiguration	☐	☐	☐	☐
9	Improper Inventory Management (Shadow & Zombie APIs)	☐	☐	☐	☐
10	Unsafe Consumption of Third-Party APIs	☐	☐	☐	☐
11	SQL & NoSQL Injection	☐	☐	☐	☐
12	XML External Entity (XXE) Injection	☐	☐	☐	☐
13	Command Injection	☐	☐	☐	☐
14	JWT & Token Vulnerabilities	☐	☐	☐	☐
15	GraphQL-Specific Vulnerabilities	☐	☐	☐	☐
16	CORS Misconfiguration	☐	☐	☐	☐
17	API Key, Secret & Credential Leakage	☐	☐	☐	☐
18	Business Logic Vulnerabilities (Price, Quantity & Race Conditions)	☐	☐	☐	☐
19	File Upload Vulnerabilities	☐	☐	☐	☐
20	HTTP Request Smuggling & Parameter Pollution	☐	☐	☐	☐

Tester Signature: _____

Date Completed: _____

Reviewed By: _____