API VAPT CHECKLIST FOR DEVSECOPS TEAMS

SCORING SYSTEM

Status	Points	What it means
Pass	**2**	Control tested and confirmed effective. Evidence documented.
Risk Accepted	**1**	Vulnerability confirmed but formally accepted with documented sign-off and review date.
Fail	**0**	Vulnerability found and unresolved, or control absent. Requires remediation and retest.
N/A	**—**	Not applicable to this engagement scope. Excluded from total score calculation.

READINESS TIERS AT A GLANCE

Score	Tier	What it signals
0 – 40	**CRITICAL RISK**	Fundamental security controls are absent or broken. Exploitable vulnerabilities likely exist in authentication, authorization, and input validation. APIs in this state represent an active data breach risk.
41 – 55	**VULNERABLE**	Basic perimeter controls exist but significant gaps remain. Authorization and business logic flaws are likely present. Automated scanning is probably in place but manual depth testing has not been conducted or acted upon.
56 – 70	**DEVELOPING**	Core security controls are in place but inconsistently applied. Some endpoints or API types are well-tested while others have not been reviewed. DevSecOps integration is partial — automated scanning exists but coverage is not mapped to OWASP categories.
71 – 85	**MANAGED**	Solid security foundation with documented processes, consistent authorization enforcement, and CI/CD integration. Gaps are typically in advanced testing categories (race conditions, complex business logic, third-party API surface) rather than core controls.
86 – 100	**RESILIENT**	Mature API security posture with comprehensive test coverage, strong DevSecOps integration, and a functioning feedback loop from VAPT findings into development standards. Security is a continuous practice, not a point-in-time exercise.

Organisation: ____ **Scope:** ____

Environment: Staging / Production / Both **Tester:** ______________________________ **Date:** ____________________

SECTION 1 — PRE-ENGAGEMENT SETUP

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
1.1	**Define Testing Objective**	Scoping call with security lead + engineering. Document in writing: Is this compliance-driven (PCI DSS / ISO 27001 / HIPAA) or security posture-driven? The answer changes methodology depth.	No signed objective = no defined success criteria. If the team cannot articulate what a successful finding looks like — stop here.	NIST 800-115 §2	☐
1.2	**Identify API Type(s) in Scope**	Enumerate all API types: REST, GraphQL, SOAP, WebSocket, gRPC. Confirm with engineering — do not rely on documentation alone. Check API gateway logs for actual traffic patterns.	Missing API types = missed attack surface. GraphQL introspection and WebSocket message handling need separate tooling and test cases.	OWASP API9	☐
1.3	**Own API (VAPT) vs Third-Party API (Pentest)**	Split all APIs into: (a) APIs your team built and controls — full VAPT scope; (b) APIs consumed from vendors — pentest scope only, focused on integration surface.	Testing a third-party API with full VAPT methodology produces findings you cannot remediate. Scope mismatch wastes the engagement.	PTES Pre-Engagement	☐
1.4	**Confirm Testing Environment**	Staging preferred for destructive tests (fuzzing, injection). Production requires: written approval, rollback procedure, maintenance window, rate-limited test account, and monitoring alerts active.	Production testing without safeguards risks live data exposure and audit complications. Verify staging reflects production auth config — many do not.	NIST 800-115 §4	☐
1.5	**Obtain Written Scope Sign-Off**	Get written confirmation from: security lead, application owner, legal / compliance (if production is in scope). Include an explicit out-of-scope list.	No authorisation = legal exposure. Out-of-scope list prevents scope creep mid-engagement.	PTES Pre-Engagement	☐
1.6	**Collect & Audit API Documentation**	Request OpenAPI / Swagger spec, Postman collections, API gateway configs. Compare documented endpoints against actual traffic in gateway logs or APM (Datadog, New Relic).	Discrepancy between docs and live traffic = shadow endpoints. Every undocumented endpoint is a potential OWASP API9 finding before testing begins.	OWASP API9	☐
				Section 1 Score → ____ / 12	

SECTION 2 — API DISCOVERY & INVENTORY

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
2.1	**Active Endpoint Enumeration**	Run ffuf / feroxbuster against base URL with API wordlists (SecLists). Check JS bundles, mobile app decompiled code, and browser dev tools network tab. Use kiterunner for API-specific route bruteforcing.	Endpoints responding with 200, 301, 403 (not 404) are live. A 403 on an undocumented admin endpoint is a confirmed finding — it exists, it just rejected you this time.	OWASP API9	☐
2.2	**Identify Zombie / Deprecated API Versions**	Check for /v1/, /v2/, /legacy/, /old/, /beta/ path variants on all confirmed endpoints. Test older versions for vulnerabilities patched in the current version.	Deprecated versions frequently run without security controls applied to the current version. Auth bypass on /v1/ when /v2/ is secured = Critical.	OWASP API9	☐
2.3	**GraphQL Introspection Check**	POST /graphql with {"query":"{__schema{types{name}}}"}. Use InQL Burp extension or GraphQL Voyager to map full schema visually.	Introspection enabled in production exposes full API schema to any caller — an attacker's roadmap. Minimum Medium finding.	OWASP API9	☐
2.4	**Auth Mechanism Mapping Per Endpoint**	For each endpoint, document: auth type (JWT / OAuth2 / API key / session / none), token location (header / query / cookie), and whether unauthenticated requests return 401 or data.	Any endpoint returning data on unauthenticated request = immediate Critical. Mixed enforcement (some endpoints unprotected) = BFLA.	OWASP API2	☐
				Section 2 Score → ____ / 8	

SECTION 3 — AUTHENTICATION TESTING

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
3.1	**JWT — Algorithm Confusion**	Decode JWT. Change "alg" to "none", remove signature. Separately: if RS256, change to HS256 and sign with public key as HMAC secret. Tool: python3 jwt_tool.py [token] -X a	Token accepted with alg:none = Critical. HS256 token signed with public key accepted = Critical. Server is not validating the algorithm.	OWASP API2	☐
3.2	**JWT — Expiry & Post-Logout Validity**	Capture a valid JWT. Log user out via /logout or wait for expiry. Replay the original token. Check whether server rejects it or returns data.	Expired or invalidated tokens accepted = High. Stateless JWT without a revocation list is particularly vulnerable.	OWASP API2	☐
3.3	**OAuth 2.0 — Redirect URI Validation**	In the auth request, manipulate redirect_uri: extra path segments, different subdomain, URL encoding tricks (%2F, %40). Observe where the auth code is sent.	Auth code redirected to attacker-controlled URI = Critical. Weak validation (prefix match only) is common and exploitable.	OWASP API2	☐
3.4	**API Key — Transmission & Exposure**	Check: Is the key in URL query params? Search GitHub, public Postman collections, JS bundle, error responses. Test key rotation — does old key immediately stop working?	API key in URL = High (logged in proxies, CDN). Key in public repo = Critical. No rotation on compromise = systemic risk.	OWASP API2	☐
3.5	**SSO / SAML Assertion Tampering**	Intercept SAML response. Attempt XML signature wrapping: duplicate the signed assertion, modify the copy, keep original signature. Use SAML Raider Burp extension.	Server processes modified assertion without signature validation failure = Critical. Many legacy SSO implementations have never been tested for this.	OWASP API2	☐
3.6	**Brute Force & Credential Stuffing Protection**	Send 20–50 rapid auth requests with invalid credentials. Observe: account lockout? CAPTCHA triggered? Progressive delay? Use Burp Intruder or ffuf.	No rate limiting or lockout after repeated failures = High. Error message revealing username vs password mismatch = Medium (user enumeration).	OWASP API4	☐
				Section 3 Score → ____ / 12	

SECTION 4 — AUTHORIZATION TESTING (BOLA / BFLA)

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
4.1	**BOLA — Horizontal Privilege Escalation**	Create two accounts (User A and User B). Authenticate as A. Capture request referencing A's object ID (order, document, profile). Replace with B's ID. Observe response.	B's data returned in A's session = Critical BOLA. Server must validate authenticated user owns the requested object — not just that they are authenticated.	OWASP API1	☐
4.2	**BOLA — ID Predictability & Enumeration**	Observe object IDs returned. If sequential integers or UUID v1 (time-based): iterate IDs in a range. If GUIDs: try to find IDs from other responses, error messages, referral data.	Sequential IDs + no ownership check = trivial mass data exfiltration. Unpredictable IDs are not a security control if ownership is not server-enforced.	OWASP API1	☐
4.3	**BFLA — Admin / Privileged Function Access**	As a standard user, send requests to: /admin/, /internal/, /management/, DELETE on resources, POST to user-creation endpoints. Test with standard user JWT — do not assume 404 means inaccessible.	Standard user reaching admin function = Critical BFLA. Many implementations rely on UI hiding rather than server-side role enforcement.	OWASP API5	☐
4.4	**Mass Assignment — Hidden Field Injection**	On any POST/PUT/PATCH endpoint, add undocumented fields: isAdmin:true, role:"admin", verified:true, balance:99999. Observe whether server silently accepts and processes them.	Server binds user-supplied fields to internal object properties without whitelisting = High. Common in frameworks with auto-binding (Rails, Spring, Node/Express).	OWASP API3	☐
4.5	**Object Property Level Authorization (BOPLA)**	After retrieving an object as User A, send PATCH with User B's object ID but fields only A should control. Test whether the server enforces field-level ownership.	Update applied without field-level ownership verification = BOPLA (OWASP API3). User has access to the object but not all its properties.	OWASP API3	☐
				Section 4 Score → ____ / 10	

SECTION 5 — INPUT VALIDATION & INJECTION

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
5.1	**SQL Injection**	Inject on all string params: single quote ('), UNION SELECT, time-based blind payloads (SLEEP(5), WAITFOR DELAY). Use SQLMap --level=3 --risk=2 only after manual confirmation.	Database error in response = injection surface confirmed. Time delay = blind SQLi. Never rely on SQLMap alone — manually confirm before automated exploitation.	OWASP API8	☐
5.2	**NoSQL Injection (MongoDB)**	Replace string values with operator objects: {"$gt":""}, {"$ne":null}. Test login with: username={"$ne":null}&password={"$ne":null}. Use Content-Type: application/json.	Auth bypass via $ne operator = Critical. Common in Node.js + MongoDB stacks where input is deserialized directly into query objects.	OWASP API8	☐
5.3	**SSRF — URL & Hostname Parameters**	Inject into URL-accepting params: http://169.254.169.254/latest/meta-data/ (AWS metadata), http://localhost:8080/admin, http://internal-redis:6379. Use Burp Collaborator for blind SSRF.	Cloud metadata in response = Critical. Blind SSRF (Collaborator ping) = High. Especially dangerous in microservices — SSRF on one service can pivot to internal network.	OWASP API7	☐
5.4	**Parameter Tampering & Data Type Confusion**	Change types: strings to ints, arrays to strings, booleans to ints. Exceed boundaries: INT_MAX, negative numbers, empty strings, null, very long strings (>10,000 chars). Use Burp Repeater for targeted tests.	Stack trace in response = internal info leakage (High). Unexpected behaviour on negative/zero = logic flaw entry point.	OWASP API8	☐
5.5	**Fuzzing — Body, Headers & Path Segments**	Fuzz: request body fields, URL path segments, HTTP headers (Content-Type, Accept, X-Forwarded-For), query string values. Tools: FFUF, Wfuzz. Monitor: 500 errors, response time spikes, unique response sizes.	Application crash (500) on specific input = unhandled exception, likely deeper vulnerability. Response time >3x = potential DoS vector.	OWASP API8	☐
5.6	**XXE & JSON Deserialization**	XML endpoints: inject <!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>. JSON: test prototype pollution (__proto__), deeply nested arrays to trigger stack overflow.	File content in response = Critical XXE. OOM / 500 on deeply nested JSON = resource consumption vulnerability. Prototype pollution in Node.js can lead to RCE.	OWASP API8	☐
				Section 5 Score → ____ / 12	

SECTION 6 — RATE LIMITING & RESOURCE CONSUMPTION

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
6.1	**Rate Limit Enforcement**	Send 100+ rapid requests to: auth endpoints, search/query endpoints, export endpoints. Test with valid and invalid credentials. Use Burp Intruder (null payload loop) or curl loop.	No throttling after sustained requests = High. Rate limiting on unauthenticated but not authenticated requests = common misconfiguration.	OWASP API4	☐
6.2	**Rate Limit Bypass Techniques**	Attempt bypass: rotate X-Forwarded-For header values, vary User-Agent strings, distribute across multiple test accounts, use null bytes or case variation in headers.	Rate limit tied to IP only = trivially bypassed with header rotation. If bypass succeeds, rate limiting provides false assurance.	OWASP API4	☐
6.3	**Batch / Bulk Operation Abuse**	Test endpoints accepting arrays: send 10,000-item array, request 1,000,000 records, chain multiple expensive GraphQL queries in a single request.	Unbounded batch processing = resource exhaustion DoS risk. GraphQL nested queries multiplying DB calls = N+1 attack. Response time linear with input = no processing cap.	OWASP API4	☐
6.4	**File Upload — Size & Type Validation**	Upload: 1GB file, .exe renamed to .jpg, polyglot file (valid image that executes as PHP). Check if content is validated or just extension / MIME type.	Extension-only validation = High. No max file size = DoS vector. Executable upload + web-accessible path = RCE.	OWASP API8	☐
				Section 6 Score → ____ / 8	

SECTION 7 — BUSINESS LOGIC TESTING

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
7.1	**Multi-Step Workflow Bypass**	Map the intended flow (e.g., cart → payment → confirm). Jump directly to step 3 without completing steps 1–2. Replay step 2 after step 3. Send requests out of sequence.	Server processes step 3 without validating prior steps = High. Confirmation without payment = Critical. Session state not enforced server-side is the root cause.	OWASP API6	☐
7.2	**Negative Value & Boundary Abuse**	Submit negative quantities (-1, -9999) on quantity/transfer/discount fields. Submit zero-values on required numeric fields. Test currency with extreme decimal precision (0.000001).	Negative quantity accepted = potential credit fraud. Zero-value transaction processed = payment bypass. All monetary values must be positive within business-defined ranges.	OWASP API6	☐
7.3	**Race Condition Testing**	Burp Repeater "Send group in parallel" (Last-Byte sync technique). Target: balance deduction, coupon redemption, one-time token validation, trial activation. Send 10–20 concurrent identical requests.	Multiple successful responses to a single-use action = race condition. Classic: redeeming a discount code twice simultaneously before the first transaction locks the record.	OWASP API6	☐
7.4	**Sensitive Data Over-Exposure in Responses**	Review every API response vs what the UI renders. Look for: internal IDs, PII not shown to user, system flags (is_admin), hashed passwords, API keys, tokens hidden in response fields.	Fields in response not rendered in UI = data over-exposure (OWASP API3). Any response with password hashes or internal tokens = High minimum. UI hiding is not a security control.	OWASP API3	☐
7.5	**Account / Resource State Manipulation**	Test: can deactivated/suspended/deleted accounts still authenticate? Are soft-deleted records accessible via direct object reference? Can suspended accounts be reactivated via API?	Deleted/suspended account retains API access = Critical. Soft-deleted records accessible via ID = BOLA variant. Common where "delete" sets a flag rather than removing the record.	OWASP API1	☐
				Section 7 Score → ____ / 10	

SECTION 8 — SECURITY CONFIGURATION & HEADERS

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
8.1	**CORS Policy Validation**	Send requests with Origin: https://attacker.com, Origin: null, Origin: https://trusted.evil.com. Observe Access-Control-Allow-Origin header and whether credentials are included in cross-origin requests.	ACAO returns attacker origin + ACAC: true = Critical. Null origin accepted = exploitable from sandboxed iframes. ACAO: * without ACAC = Low (non-credentialed).	OWASP API8	☐
8.2	**Security Headers Audit**	Check responses for: Strict-Transport-Security, Content-Security-Policy, X-Content-Type-Options, X-Frame-Options, Cache-Control (especially on authenticated responses), Referrer-Policy.	Missing HSTS = SSL stripping risk. No Cache-Control: no-store on sensitive responses = credentials may be cached. Multiple missing headers cluster to Medium.	OWASP API8	☐
8.3	**Error Response Verbosity**	Deliberately trigger errors: invalid IDs, malformed JSON, SQL payloads, incorrect content types. Check: does the error reveal stack traces, framework names, DB types, file paths, internal IPs?	Stack trace in response = Medium minimum, High if it reveals exploitable internal detail. Framework version disclosure = Low. All verbose errors must be suppressed in production.	OWASP API8	☐
8.4	**TLS Configuration**	Run testssl.sh or ssllabs.com. Check: TLS version (1.0/1.1 deprecated), weak ciphers (RC4, 3DES, NULL), certificate expiry, wildcard scope, HSTS preloading.	TLS 1.0/1.1 enabled = Medium (POODLE/BEAST). Weak cipher accepted = Medium. Expired cert = High. Self-signed cert in production = High.	OWASP API8	☐
				Section 8 Score → ____ / 8	

SECTION 9 — DEVSECOPS & CI/CD INTEGRATION

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	
9.1	**DAST Coverage in Pipeline Validated**	Review CI/CD config. Confirm automated DAST runs on PR merge (staging) and nightly full regression. Map scanner capabilities to OWASP API Top 10 explicitly — not just "a scan runs".	A scan running ≠ meaningful coverage. If scanner only catches injection and not BOLA or business logic, it provides false confidence.	NIST 800-115	☐
9.2	**Secrets in Code / Build Artifacts**	Run trufflehog or gitleaks on the repository. Check .env files, Dockerfiles, Docker image layers, CI environment variables, and build logs for: API keys, JWT secrets, DB credentials, cloud tokens.	Any credential in version control = Critical — treat as compromised immediately. Secrets in Docker image layers = High.	OWASP API2	☐
9.3	**API Specification Drift**	Compare OpenAPI / Swagger spec in repository against actual API behaviour in staging. Use 42Crunch Audit or OWASP ZAP OpenAPI integration to test spec vs reality.	Endpoints in production not in spec = shadow API (OWASP API9). Spec drift is a systemic DevSecOps process failure — not just a documentation issue.	OWASP API9	☐
9.4	**Container / Runtime Security**	Check: does the API container run as root? (docker inspect — look for "User": ""). Are resource limits set (CPU, memory)? Are debug ports or Actuator endpoints exposed externally?	Container running as root = High privilege escalation risk. Spring Boot Actuator endpoints exposed (/actuator/env, /actuator/heapdump) = High data exposure.	OWASP API8	☐
9.5	**Logging & Alerting Validation**	Execute known attack patterns (BOLA test, auth failure burst, injection payloads). Check SIEM: were events captured? Did alerts fire? Confirm sensitive data (tokens, passwords) is NOT being logged.	Attack patterns not generating alerts = detection gap. Sensitive data in logs = High. No structured logging = VAPT findings cannot be correlated with historical incidents.	OWASP API8	☐
				Section 9 Score → ____ / 10	

SECTION 10 — REPORTING & REMEDIATION CLOSURE

#	Test Item	How to Test (Field)	What to Look For	OWASP / Ref	Status
10.1	**Finding Documentation Standard Met**	Every finding must include: affected endpoint, HTTP method, full request + response evidence, reproduction steps runnable by a developer not present, CVSS score, and business impact statement.	Findings without reproduction steps are routinely deprioritised. If a developer cannot reproduce it in 15 min, the finding loses credibility.	CVSS v3.1	☐
10.2	**Severity Classification Pre-Agreed**	Before testing: agree severity definitions with client team. What is Critical vs High vs Medium in their business context? A data exposure finding may be Critical for healthcare, Medium for a public API.	Severity without business context = engineering teams deprioritise everything as "Medium." Pre-agreed definitions prevent post-engagement disputes.	CVSS v3.1	☐
10.3	**Critical & High Findings Retested**	After developer remediation: independently retest every Critical and High finding using exact reproduction steps from the original report. Test for partial fixes (same vuln, different parameter).	Partial fixes are common — specific reproduction case resolved but underlying vulnerability exploitable via another vector. Retest must be independent from the developer who fixed it.	NIST 800-115	☐
10.4	**Risk Acceptance Documented**	For findings not remediated: require written sign-off from Security Lead or CISO acknowledging the specific risk, business rationale for acceptance, and a review date.	Undocumented risk acceptance = audit liability. Risk accepted verbally and never recorded = finding reappears in next engagement with no closure history.	ISO 27001 A.12.6	☐
10.5	**Lessons Learned Fed Back to Dev Standards**	After each engagement: 1-hour post-mortem with engineering team. Map recurring finding categories to root causes in the development process. Update secure coding guidelines accordingly.	BOLA in three consecutive engagements = object-level authorization not enforced at framework level. Fixing in VAPT treats the symptom. Fixing in coding standards prevents it.	DevSecOps Shift-Left	☐
				Section 10 Score → ____ / 10	

API VAPT READINESS DASHBOARD

Complete this page after scoring all sections. Your total score determines your readiness tier and priority action plan.

SECTION-BY-SECTION SCORE SUMMARY

Section	Area	Max Score	Your Score	Notes / Priority Findings
S1	PRE-ENGAGEMENT SETUP	**12**		
S2	API DISCOVERY & INVENTORY	**8**		
S3	AUTHENTICATION TESTING	**12**		
S4	AUTHORIZATION TESTING (BOLA / BFLA)	**10**		
S5	INPUT VALIDATION & INJECTION	**12**		
S6	RATE LIMITING & RESOURCE CONSUMPTION	**8**		
S7	BUSINESS LOGIC TESTING	**10**		
S8	SECURITY CONFIGURATION & HEADERS	**8**		
S9	DEVSECOPS & CI/CD INTEGRATION	**10**		
S10	REPORTING & REMEDIATION CLOSURE	**10**		
				Score ÷ 100 × 100 =

YOUR READINESS TIER & PRIORITY ACTIONS

0 – 40 pts -- CRITICAL RISK *(0–40%)*

Action 1	Halt deployment of new API features until Critical and High findings are remediated.
Action 2	Engage an external security team for an emergency VAPT if not already in progress.
Action 3	Prioritise: authentication hardening, BOLA/BFLA controls, and disabling verbose error responses.
Action 4	Brief the CISO and relevant stakeholders — this posture carries direct regulatory exposure.

41 – 55 pts -- VULNERABLE *(41–55%)*

Action 1	Prioritise remediation of BOLA, BFLA, and mass assignment findings — these are the highest-probability exploitation paths.
Action 2	Review and enforce authorization logic at the framework level, not per-endpoint.
Action 3	Implement rate limiting across all authenticated endpoints, not just login flows.
Action 4	Schedule a full manual VAPT within the next 30 days.

56 – 70 pts -- DEVELOPING *(56–70%)*

Action 1	Map your CI/CD DAST scanner's actual coverage to OWASP API Top 10 — identify the blind spots.
Action 2	Extend manual VAPT scope to cover API types (GraphQL, WebSocket) not reached by automated tools.
Action 3	Conduct business logic testing — this is the category automated tools cannot cover.
Action 4	Introduce API spec drift monitoring to prevent shadow endpoints accumulating.

71 – 85 pts -- MANAGED *(71–85%)*

Action 1	Focus next engagement on business logic and race condition testing — the categories where your automated coverage cannot reach.
Action 2	Validate that retest verification is truly independent and not self-certified by developers.
Action 3	Review logging completeness — ensure attack patterns are generating SIEM alerts, not just being logged.
Action 4	Formalise the lessons-learned process to prevent recurring finding categories.

86 – 100 pts -- RESILIENT *(86–100%)*	
Action 1	Maintain cadence: full manual VAPT quarterly or after significant architectural changes.
Action 2	Expand red team exercises to test multi-API attack chains across microservice boundaries.
Action 3	Benchmark against new OWASP API Top 10 releases and update your test coverage accordingly.
Action 4	Consider threat modelling for new API designs before implementation — shift testing further left.

COMPANION RESOURCES
Blog: *API VAPT Testing Framework: A Step-by-Step Guide for Enterprise Security Teams*
Pillar: *API VAPT Explained: How to Secure Endpoints & Stop Data Leaks*

Need an expert team to validate and secure your APIs beyond automated scans? Let Infosprint Technologies perform a comprehensive API VAPT assessment for your enterprise environment.

Secure Your APIs Now