i need to develop a module for WAF on API security. so explain what is API and API security .how to attack API and how to protect. give full details

An API is an interface that lets software components communicate by exchanging structured requests and responses over defined contracts; API security is the set of controls, processes, and guardrails that ensure only intended principals can invoke API functionality and data without abuse, leakage, or disruption across the API lifecycle. In practice, modern API risk is dominated by broken authorization, authentication flaws, resource abuse, misconfiguration, and unsafe integrations, as captured in the OWASP API Security Top 10 2023, which should anchor both offensive testing and defensive WAF design for APIs.[1] [2] [3]

What is an API

- An API exposes endpoints that accept inputs (path, query, headers, body) and return outputs per a contract, often over HTTP with JSON, and typically mapped to objects and functions like CRUD operations on resources identified by IDs, which intrinsically increases the attack surface for object- and function-level authorization mistakes.[3]
- APIs underpin web and mobile apps and are often more numerous than traditional pages, making inventory, versioning, and documentation critical to security and change control across environments.[3]

What is API security

- API security aims to preserve confidentiality, integrity, and availability of API functions and data across authentication, authorization, input validation, resource controls, configuration, and supply-chain usage of third-party APIs, considering both implementation bugs and business logic abuse at scale.[3]
- The OWASP API Security Project provides the canonical Top 10 risk categories for APIs, distinct from but overlapping the general OWASP Top 10, and should be treated as a baseline for requirements, testing, and WAF policy coverage.[2] [1] [3]

OWASP API Top 10 (2023)

- API1: Broken Object Level Authorization (BOLA) — missing per-object checks when using user-supplied IDs.[3]
- API2: Broken Authentication — flawed token handling or flows that allow identity compromise or impersonation.[3]

- API3: Broken Object Property Level Authorization — merging excessive data exposure and mass assignment at the field level.[3]
- API4: Unrestricted Resource Consumption — lack of rate/budget limits causing DoS or cost spikes.[3]
- API5: Broken Function Level Authorization — privilege boundaries between regular and admin functions not enforced.[3]
- API6: Unrestricted Access to Sensitive Business Flows — business logic abused at scale even without code bugs.[3]
- API7: Server-Side Request Forgery (SSRF) — backend fetches attacker-controlled URLs, reaching internal networks or metadata.[3]
- API8: Security Misconfiguration — defaults, verbose errors, weak TLS/CORS, unprotected consoles, or debug endpoints.[3]
- API9: Improper Inventory Management — unknown hosts/versions/endpoints lead to forgotten exposures.[3]
- API10: Unsafe Consumption of APIs — trusting third-party responses or contracts without validation or isolation.[3]

How APIs are attacked

- Recon and endpoint enumeration: analyze docs, SDKs, OpenAPI, traffic, and fuzz parameter space to locate hidden or undocumented endpoints and object IDs for later abuse and data exfiltration.[4]
- AuthN/AuthZ bypass: exploit weak tokens, implementation flaws, function/object-level gaps, and role confusion to access other users' resources or admin actions like user listing or configuration changes.[2] [3]
- Parameter manipulation and injection: tamper with IDs, filters, pagination, and body fields; attempt SQL/NoSQL/LDAP/OS/XML/JSON injections via API inputs that are unsafely passed to backends or parsers.[5]
- Enumeration and data exposure: infer valid users through differential responses in login/reset/register flows and scrape over-permissive fields or unfiltered object properties returned by endpoints.[6] [3]
- Resource abuse and DoS: flood endpoints lacking rate limits, exploit expensive operations, or trigger cascades across paid integrations to exhaust quotas and increase costs.[3]
- SSRF and internal pivoting: force server-side fetchers to contact internal services, metadata endpoints, or external attacker hosts to leak credentials or perform lateral movement.[3]
- Supply-chain/API-of-APIs risks: compromise or spoof third-party API integrations, abusing trust and weak validation of upstream responses.[2] [3]

Defensive priorities for a WAF module (API-focused)

- Positive security model per endpoint: enforce schema and method contracts (content-type, required/optional fields, types, ranges, patterns), reject unknown fields to mitigate mass assignment and unsafe payloads.[3]
- Object- and function-level authorization enforcement: require explicit authorization metadata, block cross-tenant access patterns, and prevent invocation of privileged functions by non-privileged principals; do not rely on client routing to differentiate admin vs user APIs.[5] [3]
- Response data governance: enforce field-level allowlists and data shaping to prevent excessive data exposure; normalize and redact verbose error traces at the edge.[5] [3]
- Abuse and resource controls: implement adaptive rate limiting, quotas, concurrency caps, and cost-aware budgets by token, IP, client fingerprint, and endpoint sensitivity; prioritize state-changing and expensive endpoints.[2] [3]
- SSRF hardening: validate and normalize URLs for outbound fetchers; restrict to vetted allowlists, block private/address-literal nets and metadata endpoints, and disallow redirects to internal ranges.[3]
- Authentication hardening: enforce strong token validation (aud/iss/exp/nbf), rotation, mTLS where feasible, and detect replay or anomalous token usage patterns at the edge.[2] [3]
- Security configuration baselines: consistent TLS, CORS policies, header hardening, no debug endpoints, and environment-specific secrets management; detect drift and misconfig states via automated checks.[5] [3]
- API inventory and versioning: maintain live discovery of hosts, routes, methods, and schemas; flag shadow/rogue/deprecated endpoints and enforce deprecation blocks or compensating controls.[2] [3]

Concrete WAF inspection and policies

- Request normalization: canonicalize paths, headers, and query parameters; drop ambiguous encodings and duplicated keys that could bypass route or policy matching prior to schema validation and auth checks.[3]
- JSON/XML schema validation: per-endpoint strict schemas with type/length/pattern constraints; reject unknown properties to stop mass assignment and reduce injection surface; enforce content-type consistency.[5] [3]
- ID and tenancy guardrails: for ID-bearing endpoints, bind subject identifiers to the principal in policy (e.g., allow only IDs within tenant scope) and reject cross-tenant patterns to prevent BOLA.[3]
- Method and verb control: enforce allowed methods per route and block unexpected verbs or method override headers to contain function-level exposure.[3]
- Response filtering: remove sensitive fields at the edge when backend is over-permissive, plus standardized error bodies to avoid info leaks; log redactions for developer feedback loops.[5] [3]

- Rate/quotas: sliding-window and token-bucket policies with per-endpoint baselines; apply stricter thresholds for login, password reset, search, and data export endpoints; add user enumeration suppression with uniform responses.[6] [3]
- SSRF egress policy: if the API brokers outbound fetches, enforce DNS and IP allowlists, denylink-local and RFC1918 by default, and block http metadata paths; surface violations with high-severity alerts.[3]
- Third-party consumption: validate upstream API responses against schemas, timeouts, and signature/mTLS expectations; isolate domains and fail closed on contract drift to address unsafe consumption.[2] [3]

Testing and attack detection playbooks

- BOLA tests: iterate object IDs belonging to other users/tenants and assert denials; alert on access to foreign IDs even if backend returns data, then block on repeat attempts.[3]
- Function-level tests: enumerate admin-only endpoints from docs or versioned paths; ensure WAF policy denies non-admin tokens regardless of client routing or UI reachability.[3]
- Mass assignment: send extra and nested fields beyond the published schema; WAF should reject unknown properties and log attempted field names for backend remediation.[5] [3]
- Enumeration dampening: unify responses for login/reset/register flows and apply per-identity rate limits; trigger tarpits after threshold hits to slow credential stuffing and discovery.[6] [3]
- SSRF probes: attempt internal hosts and metadata IPs/hostnames; confirm hard denials and absence of DNS leak paths; alert on URL schemes or redirects to private ranges.[3]
- Resource abuse: fuzz pagination, sort, and export endpoints; ensure caps on limits, timeouts, and cost-based controls to avoid unbounded queries and fan-outs.[3]

Build vs runtime controls

- Shift-left: require OpenAPI-first design, contract tests, and SAST/DAST aligned to OWASP API Top 10 with field-level access control verification in CI.[3]
- Runtime: combine API discovery, schema enforcement, authN/Z checks, behavior analytics, and WAF signatures; many vendors highlight discovery, central policy, and layered defenses as core capabilities for API protection at scale.[2] [3]

Notable changes since 2019

- 2023 added sensitive business flow abuse, SSRF, and unsafe consumption of APIs; several categories merged or refocused toward property-level authorization and resource consumption updates, reflecting modern abuse patterns and cloud integrations.[7]
- Inventory and misconfiguration remain prominent, emphasizing continuous visibility and hardened defaults as APIs proliferate across teams and environments.[7] [3]

Useful checklists and cheat sheets

- Security misconfiguration, injection, and function-level authorization cheat-sheets stress repeatable hardening, input validation, and explicit role-based access at the API tier; these are actionable for WAF rulebooks and developer guardrails.[5]
- High-level explainers and best-practice guides from industry references can help map OWASP categories to platform features and operational runbooks for enforcement and monitoring.[8] [2]

⁂

now suggest some research papers on API security.

Here are solid research papers, surveys, and practitioner studies to ground an API security WAF module. Items emphasize BOLA/IDOR detection, schema-aware defenses, SSRF/consumption risks, and empirical “state of API” findings.

Canonical frameworks

- OWASP API Security Project and Top 10 (latest edition and docs) — definitive taxonomy, guidance, and references for API-specific risks to anchor threat models and WAF policies.[21] [22]

BOLA/IDOR detection and authorization flaws

- Automatic detection of access control vulnerabilities via API (IDOR/BOLA) — proposes a systematization of BOLA techniques and automated detection approaches; useful for modeling test cases and WAF telemetry heuristics.[23]
- Rethinking Broken Object Level Authorization Attacks: BOLAZ framework (2025 preprint) — static/dataflow analysis of resource IDs across code layers to locate BOLA injection points; informs shift-left checks and runtime correlation ideas for WAF.[24] [25]
- Harnessing LLMs for Automating BOLA Detection (BOLABuster, Unit 42) — methodology to generate BOLA test cases from OpenAPI using LLM reasoning; relevant for integrating spec-driven fuzzing alongside WAF schema enforcement.[26]

Security API misuse and design issues

- Detecting Misuse of Security APIs: A Systematic Review (2023, arXiv) — systematizes 30 misuses across security API categories and detection techniques; maps well to “unsafe consumption of APIs” and input misuse patterns affecting downstream components.[27]

OWASP API Top 10 deep dives and mappings

- OWASP Top 10 API Security Risks – 2023 (official detail pages) — category-level rationales, examples, and mitigations directly translatable to WAF rule families (BOLA, property-level auth, resource consumption, SSRF, misconfig, inventory).[28]
- OWASP API Security Top 10 summaries and best-practice mappings by industry explainers (Cloudflare, F5) — concise operationalization references for policy design and control coverage.[29] [30]
- What's changed in OWASP API Top 10 2023 (comparative overview) — highlights the new categories (e.g., SSRF, consumption, unsafe API consumption) to prioritize in modern API WAFs.[31]

Empirical "state of API security" reports

- Imperial/industry State of API Security 2024 (Imperva) — incident data, attack patterns, and organizational gaps; helpful for prioritizing detection and response metrics in a WAF roadmap.[32]
- Salt Security 2024 State of API Security — empirical trends on API sprawl, auth abuses, and sensitive data exposure; aligns with inventory and schema enforcement needs.[33]
- CMU/SEI: API Vulnerabilities and Risks (2024 report) — structured risk review from a research institute for programmatic controls and measurement.[34]
- API Security Impact Study 2025 (APAC) — survey-based insights on adoption and gaps; supports executive-level justification for WAF module scope and KPIs.[35]

Practitioner guides for testing and mapping to controls

- APIsecurity.io Top 10 2023 overview and cheat sheets — actionable testing and mitigation checklists useful for turning OWASP categories into concrete rules and playbooks.[36] [31]
- Cloudflare OWASP API Top 10 explainer — maps risks to operational mitigations including schema enforcement, rate limits, and discovery; helpful for designing layered controls.[29]

How to use these in a WAF module

- Use OWASP API Top 10 as the control taxonomy and acceptance criteria; derive per-category WAF rules (schema/contract enforcement for API3, ID/tenant scoping for API1, rate/quotas for API4, SSRF egress policies for API7, inventory drift monitors for API9, upstream response validation for API10).[21] [28] [29]
- Incorporate BOLA/IDOR research to build detections: correlate subject identifiers to resource IDs, flag cross-tenant patterns, and leverage OpenAPI-driven test generation in CI to align runtime policies with contracts.[25] [26] [23]
- Translate empirical reports into guardrails and dashboards: prioritize login/reset/export endpoints for enumeration and abuse controls, track sensitive field exposure, and enforce strict rate caps and token anomaly detections.[32] [33] [34] [35]

If you want, a curated reading pack PDF (with direct links and brief annotations) can be prepared around BOLA, schema enforcement, SSRF prevention, and inventory discovery for API WAF engineering.[34] [26] [25] [23] [28] [29]

⁂

how to test vulnearability of API using tools in kali linux or burpsuite?is there any such tools to test it and report?

Yes—Kali Linux and Burp Suite have multiple tools and workflows to test API vulnerabilities and generate reports. The most effective approach is to load an API definition (OpenAPI/Swagger, WSDL, Postman) into a scanner, exercise authenticated flows, and combine active scanning with manual tests for BOLA/authorization and business logic gaps.[45] [46] [47]

Burp Suite workflows

- API-only scans: In Burp Suite Professional, start a New scan → select API-only scan → upload OpenAPI/WSDL/Postman to enumerate endpoints and launch an automated scan with report output; Burp validates and parses the definition, then probes parameters, headers, and bodies with API-focused checks.[46]
- OpenAPI import via extensions: If you prefer extensions, use OpenAPI parsers like Swurg (BApp Store) or openapi-parser to ingest definitions directly into Burp for spidering and scanning, which helps when running Community or integrating into manual testing flows.[48] [49]
- Manual testing baseline: Use the Proxy to capture traffic from your client, map endpoints and parameters, then send requests to Repeater/Intruder for BOLA, auth bypass, and mass assignment testing; this remains essential for object- and function-level authorization issues that scanners alone cannot prove.[50]

OWASP ZAP for API scanning

- ZAP API Scan: Use zap-api-scan.py with an OpenAPI/GraphQL/WSDL spec to run a headless API scan and produce HTML/MD/XML/JSON reports; supports custom scan rules, timeouts, and authentication headers via config files in CI or standalone.[51] [47]
- Automation example: zap-api-scan.py -t https://your/api/openapi.json -f openapi -r report.html -j report.json -c rules.conf to generate a pass/fail report and artifacts for pipelines or manual review.[47]

Kali Linux toolset you can leverage

- Burp Suite (Community/Pro) comes prepackaged in Kali and is the primary interactive DAST for APIs; launch via burpsuite and configure the browser/system proxy to 127.0.0.1:8080 for interception and testing.[52] [45]

- OWASP ZAP is available in Kali and pairs well for automated API scans and CI; its Docker image includes zap-api-scan.py for OpenAPI/GraphQL/WSDL targets with turn-key reports. [45] [47]
- Curated installers like APISecTools can bootstrap multiple API-specific testing utilities on Kali to complement Burp/ZAP with format parsers and fuzzers, easing setup for engagements. [53]

Suggested step-by-step pipelines

- With Burp Suite Pro:
  - Import API definition and configure API-only scan; add authentication (JWT/cookie/mTLS), scope, and rate throttling; run the scan and export HTML/PDF/Issue reports from the Dashboard for tracking. [46]
  - Augment with Swurg/openapi-parser to ensure full endpoint coverage, then manual tests in Repeater/Intruder for BOLA (ID/tenant swapping), mass assignment (extra fields), and function-level auth by calling admin routes with non-admin tokens. [49] [48]
- With OWASP ZAP:
  - Use zap-api-scan.py against your OpenAPI URL/file; pass headers (e.g., -z "auth.auto…") or config files for tokens; generate HTML/JSON outputs; integrate the same into CI for recurring scans on schema changes. [51] [47]
  - For black-box, proxy your client through ZAP to collect traffic, then run active scans on discovered endpoints and export consolidated reports for dev teams. [47]

What each tool covers well vs gaps

- Burp/ZAP scanners excel at injection, insecure headers, weak TLS, basic auth/session issues, and many input validation problems; they also exercise common API patterns when a spec is provided. [46] [47]
- Authorization logic (BOLA/IDOR, function-level auth), excessive data exposure, business logic abuse, and resource consumption issues need manual/semiautomated tests: parameter tampering, field allowlist enforcement checks, pagination abuse, and rate-limit verification. [50]

Practical test cases to run

- BOLA/IDOR: Change resource IDs across tenants/users and assert access is denied; try numeric sequences and UUIDs in path/query/body; verify responses and error uniformity. [50]
- Mass assignment/property-level: Add undocumented fields to POST/PUT/PATCH, nested objects, or boolean admin flags; check if state changes occur or sensitive properties leak. [50]
- Auth robustness: Tamper with alg=none or claim values in JWTs if applicable; test expired tokens; replay tokens from different IPs or clients if permitted by policies. [50]

- Rate limiting/resource abuse: Hammer login, password reset, and export/search endpoints; try large page sizes, deep sorts, file exports; confirm server enforces sensible limits and returns consistent responses.[50]

Reporting

- Burp: Export scan issue lists and HTML reports from the Dashboard; tag findings per endpoint for developer routing and attach Repeater/Intruder evidence.[46]
- ZAP: Use -r/-j/-x/-w flags to output HTML/JSON/XML/Markdown; include config and evidence links to reproduce and track regression in CI.[51] [47]

Helpful references to get started

- Burp API-only scans documentation with definition requirements and setup steps for OpenAPI, WSDL, and Postman Collections.[46]
- ZAP API Scan user guide and command-line usage, including options for auth, custom rules, and report generation.[47]
- Swurg and openapi-parser repositories for importing OpenAPI documents into Burp workflows effectively.[48] [49]
- Kali tools index for installing and updating Burp/ZAP and related utilities on Kali Linux images.[45]

⁂

1. https://owasp.org/www-project-api-security/
2. https://www.cloudflare.com/learning/security/api/owasp-api-security-top-10/
3. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
4. https://www.traceable.ai/blog-post/the-anatomy-of-an-api-abuse-attack-a-hackers-process-unveiled
5. https://apisecurity.io/encyclopedia/content/owasp-api-security-top-10-cheat-sheet-a4.pdf
6. https://www.onelogin.com/blog/user-enumeration-attacks-what-you-need-to-know
7. https://apisecurity.io/owasp-api-security-top-10/
8. https://www.f5.com/glossary/owasp-api-security-top-10
9. https://www.indusface.com/blog/whats-new-in-owasp-api-top-10-2023/
10. https://portswigger.net/web-security/api-testing/top-10-api-vulnerabilities
11. https://www.reddit.com/r/netsec/comments/1hbo1tz/a_complete_owasp_api_top_10_manual_testing_guide/
12. https://www.upguard.com/blog/what-is-an-enumeration-attack
13. https://www.apisecuniversity.com/courses/owasp-api-security-top-10-and-beyond
14. https://www.stackhawk.com/blog/understanding-the-2023-owasp-top-10-api-security-risks/
15. https://www.wiz.io/academy/api-attacks
16. https://learn.microsoft.com/en-us/azure/api-management/mitigate-owasp-api-threats
17. https://www.cequence.ai/blog/owasp/owasp-top-10-lists-end-state-or-starting-point/
18. https://appsentinels.ai/blog/api-hacking-cheat-sheet/

19. https://www.wattlecorp.com/owasp-api-security-top-10/
20. https://owasp.org/API-Security/
21. https://owasp.org/www-project-api-security/
22. https://owasp.org/API-Security/
23. https://arxiv.org/abs/2201.10833
24. https://arxiv.org/html/2507.02309v1
25. https://arxiv.org/abs/2507.02309
26. https://unit42.paloaltonetworks.com/automated-bola-detection-and-ai/
27. https://arxiv.org/html/2306.08869v2
28. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
29. https://www.cloudflare.com/learning/security/api/owasp-api-security-top-10/
30. https://www.f5.com/glossary/owasp-api-security-top-10
31. https://apisecurity.io/owasp-api-security-top-10/
32. https://www.imperva.com/resources/resource-library/reports/the-state-of-api-security-in-2024/
33. https://content.salt.security/rs/352-UXR-417/images/2024 State of API Security_x.pdf
34. https://www.sei.cmu.edu/documents/5908/api-vulnerabilities-and-risks-2024sr004-1.pdf
35. https://mysecuritymarketplace.com/reports/api-security-impact-study-2025/
36. https://apisecurity.io/encyclopedia/content/owasp-api-security-top-10-cheat-sheet-a4.pdf
37. https://www.scribd.com/document/854161474/Owasp-Top-10-API-Security-Risks
38. https://www.traceable.ai/2025-state-of-api-security
39. https://www.apisecuniversity.com/courses/owasp-api-security-top-10-and-beyond
40. https://www.apisecuniversity.com/2024marketreport
41. https://www.traceable.ai/blog-post/a-deep-dive-on-the-most-critical-api-vulnerability----bola-broken-object-level-authorization
42. https://www.secopsolution.com/blog/owasp-top-10-api-security-risks-2023
43. https://apisecurity.io/owasp-api-security-top-10/owasp-api-security-top-10-project/
44. https://www.globaldots.com/resources/blog/api-security/
45. https://www.kali.org/tools/
46. https://portswigger.net/burp/documentation/desktop/running-scans/api-scans
47. https://www.zaproxy.org/docs/docker/api-scan/
48. https://github.com/aress31/openapi-parser
49. https://github.com/robelgado/swurg
50. https://www.pynt.io/learning-hub/burp-suite-guides/api-testing-with-burp-suite-a-practical-guide
51. https://notes.kodekloud.com/docs/DevSecOps-Kubernetes-DevOps-Security/DevSecOps-Pipeline/Demo-OWASP-ZAP
52. https://www.checkops.com/kali-linux/
53. https://github.com/aashishsec/APISecTools
54. https://www.kali.org

55. https://www.infosecinstitute.com/resources/penetration-testing/top-19-kali-linux-tools-for-vulnerability-assessments/
56. https://www.stationx.net/penetration-testing-tools-for-kali-linux/
57. https://stackoverflow.com/questions/51706298/owasp-zap-testing-rest-api
58. https://www.pixelqa.com/blog/post/using-kali-linux-for-software-security-testing
59. https://www.jit.io/resources/owasp-zap/api-scanner-with-owasp-zap
60. https://phoenixnap.com/kb/kali-linux-tools
61. https://www.pluralsight.com/labs/aws/api-testing-with-burp-suite
62. https://www.simplilearn.com/top-kali-linux-tools-article
63. https://help.hostedscan.com/help/scanning-open-api-endpoints
64. https://www.geeksforgeeks.org/linux-unix/kali-linux-web-penetration-testing-tools/