i need to develop a WAAP where i will develop a module for API security.so give me rule set for the owasp top 10 API vulnerability. i need the exact working rule set in python for the API security .

Here is a practical, production-grade Python rule set you can drop into a WAAP (reverse proxy/middleware) to mitigate the OWASP API Security Top 10 (2023). It implements request/response validation, authentication, authorization (object, property, and function level), rate limiting and burst control, anti-automation on sensitive business flows, SSRF egress control, security headers, schema-based allow-lists, version/inventory enforcement, and upstream API hardening. The categories and controls map to OWASP's 2023 API Top 10 list and recommended mitigations. Adapt the stubbed policy callbacks to your identity store, RBAC/ABAC engine, schemas, and data layer.[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

Complete Python middleware (FastAPI example)

```
# requirements:
#   fastapi==0.115.*
#   uvicorn==0.30.*
#   pydantic==2.*
#   httpx==0.27.*
#   redis==5.* (or use in-memory dicts for demo)
# This module wraps your API app with WAAP controls for OWASP API Top 10:2023.

import re
import time
import ipaddress
import json
import hashlib
from typing import Dict, Any, Optional, List, Tuple
from urllib.parse import urlparse

import httpx
from fastapi import FastAPI, Request, Response
from fastapi.responses import JSONResponse
from pydantic import BaseModel, ValidationError

###############################################################################
# Policy sources (replace with real sources: DB/Redis/config service)
###############################################################################

ALLOWED_API_VERSIONS = {"v1", "v2"}  # API9: inventory/version gating
DEPRECATED_VERSIONS = {"v0"}         # API9
```

```
DOCUMENTED_ENDPOINTS = {             # API9
    ("GET", "/v1/users/{id}"),
    ("PATCH", "/v1/users/{id}"),
    ("POST", "/v1/auth/login"),
    ("POST", "/v1/tickets"),
    ("POST", "/v1/payments"),
}

SENSITIVE_BUSINESS_FLOWS = {         # API6: sensitive flows subject to anti-automation
    ("POST", "/v1/tickets"),
    ("POST", "/v1/payments"),
}

FUNCTION_ROLE_MAP = {                # API5: function-level authorization
    ("PATCH", "/v1/users/{id}"): {"admin", "support"},
    ("POST", "/v1/payments"): {"customer"},
}

PROPERTY_ACL = {                     # API3: property-level authorization
    # model_name -> field -> allowed roles
    "User": {
        "id": {"admin", "support", "customer"},
        "email": {"admin", "support", "customer"},
        "role": {"admin"},            # hidden from non-admins
        "passwordHash": set(),        # never exposed
    }
}

OBJECT_OWNERSHIP_FIELDS = {          # API1: object-level authorization
    # route key -> lambda(principal, path_params, body)->bool
    ("GET", "/v1/users/{id}"): lambda p, path, body: (p["role"] in {"admin", "support"})
    ("PATCH", "/v1/users/{id}"): lambda p, path, body: (p["role"] in {"admin", "support"}
}

RATE_LIMITS = {                      # API4: per-identity limits
    # key -> (limit, window_sec, burst)
    "default": (100, 60, 30),
    "/v1/auth/login": (20, 300, 10),
    "/v1/payments": (30, 60, 15),
}

EGRESS_SSRF_ALLOWLIST = {            # API7: egress network allow-list
    # hostname -> allowed_schemes
    "api.stripe.com": {"https"},
    "identity.example.com": {"https"},
}

BLOCK_PRIVATE_EGRESS = True          # API7

SECURITY_HEADERS = {                 # API8: security headers
    "X-Content-Type-Options": "nosniff",
    "X-Frame-Options": "DENY",
    "Referrer-Policy": "no-referrer",
    "Content-Security-Policy": "default-src 'none'; frame-ancestors 'none'; base-uri 'non
    "Cache-Control": "no-store",
}
```

```
REJECT_HTTP_METHODS = {"TRACE", "TRACK", "CONNECT"}  # API8
MAX_BODY_BYTES = 1_000_000           # API4
MAX_ARRAY_LENGTH = 10_000            # API4
MAX_DEPTH = 25                       # API4

REJECT_QUERY_KEYS = {                # API8
    "__proto__", "constructor", "$where"
}

JWT_ISSUERS = {"https://identity.example.com/"}  # API2
JWT_AUDIENCE = "api://your-service"              # API2
# Replace with proper JWT verification using a JWKS
def fake_verify_jwt(token: str) -> Optional[Dict[str, Any]]:
    try:
        # WARNING: replace with jose/jwt + JWKS verification. This is a stub.
        payload_json = httpx.get("https://identity.example.com/introspect", headers={"Aut
        if not payload_json.get("active"):
            return None
        return {
            "sub": str(payload_json["sub"]),
            "role": payload_json.get("role", "customer"),
            "iss": payload_json.get("iss"),
            "aud": payload_json.get("aud"),
            "exp": payload_json.get("exp"),
        }
    except Exception:
        return None

################################################################################
# Schemas: define allow-lists for payload shape to stop mass assignment (API3)
################################################################################

class UserPatch(BaseModel):
    email: Optional[str] = None

ALLOWED_REQUEST_SCHEMAS: Dict[Tuple[str, str], Optional[BaseModel]] = {
    ("PATCH", "/v1/users/{id}"): UserPatch,   # Only email allowed here
    ("POST", "/v1/tickets"): None,           # none -> only type check & size limits
    ("POST", "/v1/payments"): None,
}

################################################################################
# Helpers
################################################################################

def route_key(method: str, path: str) -> Tuple[str, str]:
    # naive templating: normalize variable segments by {id}
    parts = path.strip("/").split("/")
    norm = []
    for p in parts:
        norm.append("{id}" if re.fullmatch(r"[0-9a-fA-F-]{6,}", p) else p)
    return (method.upper(), "/" + "/".join(norm))

def extract_path_params(route_tpl: str, actual: str) -> Dict[str, str]:
    tpl = route_tpl.strip("/").split("/")
    act = actual.strip("/").split("/")
    out = {}
    for t, a in zip(tpl, act):
        if t == "{id}":
            out["id"] = a
    return out

def deep_limits(value, depth=0):
    if depth > MAX_DEPTH:
        raise ValueError("Max depth exceeded")
    if isinstance(value, list):
        if len(value) > MAX_ARRAY_LENGTH:
            raise ValueError("Array too large")
        for v in value:
            deep_limits(v, depth + 1)
    elif isinstance(value, dict):
        if len(value) > MAX_ARRAY_LENGTH:
            raise ValueError("Object too large")
        for v in value.values():
            deep_limits(v, depth + 1)

def is_private_ip(host: str) -> bool:
    try:
        ips = httpx.get(f"https://dns.google/resolve?name={host}&type=A", timeout=2.0).js
        for ans in ips:
            ip = ipaddress.ip_address(ans["data"])
            if ip.is_private or ip.is_loopback or ip.is_link_local or ip.is_reserved or i
                return True
        return False
    except Exception:
        # fail closed
        return True

def hash_identity(ip: str, sub: Optional[str]) -> str:
    b = f"{ip}|{sub or ''}".encode()
    return hashlib.sha256(b).hexdigest()

class TokenBucket:
    def __init__(self, capacity: int, refill_rate_per_sec: float):
        self.capacity = capacity
        self.tokens = capacity
        self.rate = refill_rate_per_sec
        self.ts = time.time()

    def allow(self, cost=1) -> bool:
        now = time.time()
        delta = now - self.ts
        self.ts = now
        self.tokens = min(self.capacity, self.tokens + delta * self.rate)
        if self.tokens >= cost:
            self.tokens -= cost
            return True
        return False

# in-memory limiter (swap with Redis for distributed)
rate_buckets: Dict[str, TokenBucket] = {}
```

```
def rate_key(identity: str, path: str) -> Tuple[int, int, int]:
    spec = RATE_LIMITS.get(path, RATE_LIMITS["default"])
    return spec

def check_rate_limit(identity: str, path: str) -> bool:
    limit, window, burst = rate_key(identity, path)
    key = f"{identity}:{path}:{window}"
    if key not in rate_buckets:
        # refill rate = limit per window
        rate_buckets[key] = TokenBucket(burst, limit / window)
    return rate_buckets[key].allow(1)

###############################################################################
# WAAP Middleware
###############################################################################

class WAAP:
    def __init__(self, app: FastAPI):
        self.app = app

    async def __call__(self, scope, receive, send):
        if scope["type"] != "http":
            return await self.app(scope, receive, send)

        request = Request(scope, receive=receive)
        method = request.method.upper()
        path = request.url.path

        # API8: block dangerous methods
        if method in REJECT_HTTP_METHODS:
            return await JSONResponse({"detail": "Method not allowed"}, status_code=405)(

        # API9: version and undocumented endpoint enforcement
        segs = [s for s in path.split("/") if s]
        if len(segs) > 0 and segs[^1_0] not in ALLOWED_API_VERSIONS:
            if segs[^1_0] in DEPRECATED_VERSIONS:
                return await JSONResponse({"detail": "Deprecated API version"}, status_c
            return await JSONResponse({"detail": "Unknown API version"}, status_code=404)

        rk = route_key(method, path)
        if rk not in DOCUMENTED_ENDPOINTS:
            # allow only explicitly documented surfaces
            return await JSONResponse({"detail": "Undocumented endpoint"}, status_code=40

        # API2: authentication (bearer or mTLS etc.)
        principal = None
        authz = request.headers.get("authorization", "")
        if authz.startswith("Bearer "):
            token = authz.split(" ", 1)[^1_1].strip()
            principal = fake_verify_jwt(token)
            if not principal:
                return await JSONResponse({"detail": "Invalid token"}, status_code=401)(s
            if principal.get("iss") not in JWT_ISSUERS or principal.get("aud") != JWT_AUD
                return await JSONResponse({"detail": "Token not acceptable"}, status_code
        else:
            # allow unauthenticated only for explicitly public endpoints (example: login)
            if rk != ("POST", "/v1/auth/login"):
                return await JSONResponse({"detail": "Authentication required"}, status_c

        # API4: rate limiting / resource control
        identity = hash_identity(request.client.host, principal.get("sub") if principal e
        if not check_rate_limit(identity, path):
            return await JSONResponse({"detail": "Rate limit exceeded"}, status_code=429)

        # API4: body size guard
        body = b""
        if method in {"POST", "PUT", "PATCH"}:
            body = await request.body()
            if len(body) > MAX_BODY_BYTES:
                return await JSONResponse({"detail": "Payload too large"}, status_code=41

        # Parse JSON if needed
        json_body = None
        ctype = request.headers.get("content-type", "")
        if body and "application/json" in ctype:
            try:
                json_body = json.loads(body.decode("utf-8"))
                deep_limits(json_body)  # API4: depth/size limits
                # API8: basic key allow-list on query/payload
                for k in (request.query_params.keys()):
                    if k in REJECT_QUERY_KEYS:
                        return await JSONResponse({"detail": "Disallowed query key"}, sta
            except (json.JSONDecodeError, UnicodeDecodeError, ValueError):
                return await JSONResponse({"detail": "Invalid JSON"}, status_code=400)(sc

        # API3: schema allow-list + mass-assignment prevention
        schema = ALLOWED_REQUEST_SCHEMAS.get(rk)
        if schema is not None and json_body is not None:
            try:
                # Only fields in schema are permitted
                allowed = schema.model_validate(json_body).model_dump(exclude_none=True)
                json_body = allowed
            except ValidationError as ve:
                return await JSONResponse({"detail": "Payload not allowed", "errors": ve.

        # API5: function-level authZ
        if principal and rk in FUNCTION_ROLE_MAP:
            if principal["role"] not in FUNCTION_ROLE_MAP[rk]:
                return await JSONResponse({"detail": "Forbidden (function-level)"}, statu

        # API1: object-level authZ
        if principal and rk in OBJECT_OWNERSHIP_FIELDS:
            tpl = rk[^1_1]
            path_params = extract_path_params(tpl, path)
            allowed = OBJECT_OWNERSHIP_FIELDS[rk](principal, path_params, json_body)
            if not allowed:
                return await JSONResponse({"detail": "Forbidden (object-level)"}, status_

        # API6: protect sensitive business flows from automation
        if (method, rk[^1_1]) in SENSITIVE_BUSINESS_FLOWS:
            # simple anti-automation heuristics (augment with device fingerprint / challe
            ua = request.headers.get("user-agent", "")
            if not ua or len(ua) < 8:
                return await JSONResponse({"detail": "Automation suspected"}, status_code

        # Wrap send to inject security headers and perform property-level filtering on re
        async def send_wrapper(message):
            if message["type"] == "http.response.start":
                headers = list(message.get("headers", []))
                for k, v in SECURITY_HEADERS.items():
                    headers.append((k.encode(), v.encode()))
                message["headers"] = headers
            return await send(message)

        # Bind principal and sanitized json_body to downstream via state
        scope["state"]["principal"] = principal
        scope["state"]["sanitized_json_body"] = json_body

        return await self.app(scope, receive, send_wrapper)

app = FastAPI()
app.middleware("http")(WAAP(app))

################################################################################
# Example upstream handlers demonstrating response property filtering (API3)
################################################################################

def filter_properties(model_name: str, data: Dict[str, Any], role: str) -> Dict[str, Any]
    acl = PROPERTY_ACL.get(model_name, {})
    out = {}
    for field, value in data.items():
        allowed_roles = acl.get(field, {"admin", "support", "customer"})
        if role in allowed_roles:
            out[field] = value
    return out

@app.get("/v1/users/{id}")
async def get_user(request: Request, id: str):
    principal = request.scope["state"].get("principal") or {"role": "customer", "sub": No
    # demo record (replace with DB)
    user = {"id": id, "email": f"user{id}@example.com", "role": "customer", "passwordHash

    # API3: response property filtering
    filtered = filter_properties("User", user, principal["role"])
    return JSONResponse(filtered)

@app.patch("/v1/users/{id}")
async def patch_user(request: Request, id: str):
    principal = request.scope["state"]["principal"]
    body = request.scope["state"]["sanitized_json_body"] or {}
    # apply update (demo)
    user = {"id": id, "email": body.get("email", f"user{id}@example.com"), "role": "custo
    filtered = filter_properties("User", user, principal["role"])
    return JSONResponse(filtered)

@app.post("/v1/auth/login")
async def login():
    # intentionally minimal; real flow handled by IdP
    return JSONResponse({"detail": "Use IdP"}, status_code=501)

###############################################################################
# Safe egress client for upstream API consumption (API7 & API10)
###############################################################################

class SafeEgressClient:
    def __init__(self, timeout=5.0):
        self.client = httpx.Client(timeout=timeout)

    def _check_url(self, url: str):
        p = urlparse(url)
        if p.scheme not in {"http", "https"}:
            raise ValueError("Scheme not allowed")
        host = p.hostname
        if not host:
            raise ValueError("Invalid host")
        if BLOCK_PRIVATE_EGRESS and is_private_ip(host):
            raise ValueError("Private or internal address blocked")
        allowed = EGRESS_SSRF_ALLOWLIST.get(host)
        if allowed and p.scheme not in allowed:
            raise ValueError("Scheme not allowed for host")
        if not allowed:
            raise ValueError("Host not allow-listed")

    def get(self, url: str, headers=None):
        self._check_url(url)
        return self.client.get(url, headers=headers)

    def post(self, url: str, json=None, headers=None):
        self._check_url(url)
        return self.client.post(url, json=json, headers=headers)

safe_client = SafeEgressClient()

@app.post("/v1/payments")
async def create_payment(request: Request):
    principal = request.scope["state"]["principal"]
    # Example: consume upstream payment API safely
    try:
        r = safe_client.post("https://api.stripe.com/v1/payment_intents", json={"amount":
        data = r.json()
    except Exception as e:
        return JSONResponse({"detail": "Upstream blocked or failed"}, status_code=502)
    # Do not trust upstream blindly; validate and minimize
    if not isinstance(data, dict) or "id" not in data or "status" not in data:
        return JSONResponse({"detail": "Unexpected upstream shape"}, status_code=502)
    return JSONResponse({"paymentId": data["id"], "status": data["status"]})
```

What this enforces (mapping to OWASP API Top 10 2023):

- API1 Broken Object Level Authorization: Ownership checks per route, using path params to enforce BOLA on read/write.[4] [5] [1]

- API2 Broken Authentication: Bearer token required by default, issuer/audience checks, and a stub for real JWKS verification; tie into IdP/OIDC in production.[9] [10] [1] [4]
- API3 Broken Object Property Level Authorization: Request schema allow-listing to prevent mass assignment; response field filtering via PROPERTY_ACL to avoid excessive data exposure.[3] [5] [1] [4]
- API4 Unrestricted Resource Consumption: Token bucket rate limiting per identity+path; body size limits, JSON depth/collection limits, and method gating.[10] [1] [3] [9]
- API5 Broken Function Level Authorization: Role gates per method+route for admin vs user operations.[1] [4] [9]
- API6 Unrestricted Access to Sensitive Business Flows: Anti-automation heuristics on ticket/purchase flows; extend with challenges/fingerprints.[6] [3] [1]
- API7 Server-Side Request Forgery: Safe egress client with host allow-list, scheme restrictions, and private-IP blocking before making outbound requests.[8] [3] [10] [1]
- API8 Security Misconfiguration: Security headers, disallowed HTTP verbs, basic key blacklist, strict JSON parsing, and default-deny endpoints.[7] [9] [1]
- API9 Improper Inventory Management: Version gating, deprecated version blocking, and documented-endpoint allow-list.[3] [10] [1]
- API10 Unsafe Consumption of APIs: Validate upstream responses, minimize fields, and never trust third-party payloads without schema checks.[8] [9] [1] [3]

Integration notes

- Authentication: Replace fake_verify_jwt with standards-based verification (OIDC discovery + JWKS, token lifetime checks, nonce/at_hash as needed).[4] [9] [10]
- Authorization: Swap FUNCTION_ROLE_MAP and PROPERTY_ACL for centralized ABAC/RBAC policies; log all denials for forensics.[1] [4]
- Rate limiting: Back with Redis/cluster to enforce globally; consider per-scope and per-endpoint weights; add concurrency caps and timeouts.[9] [3]
- SSRF: If you host egress in a separate microservice, still enforce DNS pinning and disallow redirects to private IP; add URL normalization and block IPv6 literals if not needed.[10] [8] [1]
- Schemas: Use OpenAPI-derived Pydantic models to generate ALLOWED_REQUEST_SCHEMAS automatically for tight allow-listing.[5] [3] [4]
- Security headers and CORS: Add strict CORS rules per audience; include HSTS at the edge/ingress.[7] [9] [10]
- Inventory: Fail closed on undocumented routes; use spec linting in CI to ensure DOCUMENTED_ENDPOINTS and version policy stay current.[3] [10] [1]

References and mappings

- OWASP API Security Top 10 (2023) official list, definitions, and risks.[2] [1]
- Practical mitigations for each item and cheat-sheet style guidance.[4] [3]
- WAF/WAAP concepts and community best practices.[5] [7]
- Platform examples for implementing mitigations at gateways.[6] [9] [10]
- Overview and explanations suitable for stakeholder alignment.[8]

⁂

explain all the codes above in detail

The module implements a WAAP-style middleware that enforces controls mapped directly to the OWASP API Security Top 10 (2023) by inspecting every request/response, authenticating callers, authorizing functions, objects, and properties, rate-limiting, constraining payload resources, protecting sensitive flows from automation, restricting egress to prevent SSRF, enforcing security headers, and gating versions/endpoints to ensure proper API inventory. Each control aligns with OWASP's 2023 list and its recommended mitigations for API1–API10.[39]

High-level structure

- Policy/config blocks define which API versions/endpoints are allowed, who can call which functions, what properties are exposed, rate limits, security headers, and SSRF allow-lists; these are the "rules" the middleware enforces on every request and select responses.[39]
- Schemas (Pydantic) act as allow-lists to prevent mass assignment and overposting and ensure only explicit fields are accepted per route.[39]
- The WAAP middleware sits in front of your FastAPI app, failing closed by default for undocumented or deprecated versions and injecting security headers on all responses.[39]
- Dedicated egress client validates destination URLs before any server-side call, blocking private networks and non-allow-listed hosts to mitigate SSRF.[39]

Constants and policy sources

- ALLOWED_API_VERSIONS, DEPRECATED_VERSIONS, and DOCUMENTED_ENDPOINTS implement inventory/version controls so only documented, current surfaces respond; deprecated versions return 410 and unknown versions 404, limiting shadow and zombie APIs. This addresses Improper Inventory Management (API9).[39]
- SENSITIVE_BUSINESS_FLOWS enumerates critical flows (e.g., ticket purchase, payments) to apply anti-automation heuristics, covering Unrestricted Access to Sensitive Business Flows (API6).[39]
- FUNCTION_ROLE_MAP ties method+route to allowed roles to enforce distinct function-level permissions (e.g., admin-only PATCH), addressing Broken Function Level Authorization (API5).[39]

- PROPERTY_ACL defines which response fields are visible per role; coupled with request schema allow-lists, this prevents excessive data exposure and mass assignment, handling Broken Object Property Level Authorization (API3).[39]
- OBJECT_OWNERSHIP_FIELDS provides per-route lambdas that verify the principal owns the resource or has privileged role, closing Broken Object Level Authorization/BOLA (API1).[39]
- RATE_LIMITS defines per-path limits with a token bucket (limit/window/burst), plus MAX_BODY_BYTES, MAX_ARRAY_LENGTH, and MAX_DEPTH to cap resource use, addressing Unrestricted Resource Consumption (API4).[39]
- EGRESS_SSRF_ALLOWLIST and BLOCK_PRIVATE_EGRESS restrict outbound destinations by host and scheme, and block private/loopback, mitigating Server-Side Request Forgery (API7).[39]
- SECURITY_HEADERS and REJECT_HTTP_METHODS enforce response hardening (e.g., X-Content-Type-Options, CSP) and deny risky verbs (TRACE/TRACK/CONNECT), countering Security Misconfiguration (API8).[39]
- REJECT_QUERY_KEYS blacklists dangerous keys often abused in prototype pollution or server-side injection, contributing to API hardening and misconfiguration defenses (API8).[39]
- JWT_ISSUERS and JWT_AUDIENCE define acceptable token issuers/audience; fake_verify_jwt is a stub to be replaced by standard OIDC/JWKS validation for Broken Authentication (API2).[39]

Schemas and mass assignment defense

- UserPatch permits only the email field on PATCH /v1/users/{id}; ALLOWED_REQUEST_SCHEMAS maps routes to models, ensuring only whitelisted properties pass validation and others are rejected, stopping overposting/mass assignment and preventing exposure via property-level authorization (API3).[39]

Helper functions

- route_key normalizes paths by replacing ID-like segments with {id} so rules match templated routes; extract_path_params recovers dynamic path variables for object ownership checks (API1/5).[39]
- deep_limits traverses JSON to enforce depth and collection-size caps, preventing resource exhaustion and parsing bombs (API4).[39]
- is_private_ip resolves host A records and blocks private/loopback/link-local/multicast/reserved ranges to fail closed on private targets for SSRF controls (API7).[39]
- hash_identity combines client IP and subject claim into a stable token for rate limiting; TokenBucket implements burst and steady refill; check_rate_limit ties it together per identity+path (API4).[39]

WAAP middleware pipeline

- Method filtering: blocks TRACE/TRACK/CONNECT up front for safer defaults (API8).[39]
- Version and surface gating: enforces version policy and rejects undocumented endpoints with 404, preventing accidental exposures and shadow routes (API9).[39]
- Authentication: requires Bearer tokens by default, except explicit public routes like login; validates issuer and audience, and returns 401 for invalid/missing tokens, addressing Broken Authentication (API2).[39]
- Rate limiting and resource control: computes an identity key and enforces token bucket limits; checks payload size before parsing to reduce DoS risk (API4).[39]
- Strict JSON handling: decodes JSON with error handling, applies deep_limits, and rejects disallowed query keys to reduce misconfig/injection surface (API4/API8).[39]
- Schema allow-listing: validates and rewrites the request body to only allowed fields; Pydantic errors cause a 400 with detail, preventing mass assignment and property-level misuse (API3).[39]
- Function-level authorization: checks role membership against FUNCTION_ROLE_MAP and returns 403 if forbidden (API5).[39]
- Object-level authorization: for routes with IDs, enforces ownership/privilege checks from OBJECT_OWNERSHIP_FIELDS and returns 403 if access is not allowed (API1).[39]
- Anti-automation on sensitive flows: applies simple user-agent heuristic to throttle likely bots on flows like payments; extendable to challenges/fingerprints (API6).[39]
- Response security headers: wraps send to add standard security headers on every response, reducing attack surface from clickjacking/MIME sniffing/referrer leakage (API8).[39]
- State propagation: attaches principal and sanitized_json_body to scope.state so downstream handlers consume sanitized input and caller identity consistently (cross-cutting).[39]

Example upstream handlers

- get_user filters response fields based on PROPERTY_ACL and caller role; hidden fields like passwordHash never leave the service for non-admins, handling property-level authorization (API3).[39]
- patch_user uses sanitized_json_body from the middleware so only allowed fields are applied; then filters output by role (API3).[39]
- login stays unauthenticated by design as a public route and returns 501 placeholder; in real systems, this is delegated to an IdP/OIDC flow to avoid building custom auth (API2).[39]

SafeEgressClient (SSRF and API10)

- Validates outbound URL scheme and host; blocks private addresses and denies non-allow-listed hosts or scheme mismatches, preventing SSRF pivots to internal metadata services and network scans (API7).[39]
- Payment example calls an allow-listed upstream (e.g., Stripe) and then validates the returned shape, returning minimal fields to the client; this avoids trusting third-party responses blindly and reduces blast radius from compromised partners, addressing Unsafe Consumption of APIs (API10).[39]

OWASP API Top 10 mapping summary

- API1 Broken Object Level Authorization: Per-route ownership checks using path params deny cross-tenant/object access with 403.[39]
- API2 Broken Authentication: Default-deny unauthenticated requests; issuer/audience validation; replace stub with OIDC/JWKS to harden token verification.[39]
- API3 Broken Object Property Level Authorization: Request schema allow-lists plus response field ACLs prevent overposting and sensitive field leakage.[39]
- API4 Unrestricted Resource Consumption: Token-bucket rate limits per identity+path, body size caps, depth/array bounds, and restricted methods.[39]
- API5 Broken Function Level Authorization: Route-level role gates for admin/support vs. customer-only endpoints.[39]
- API6 Unrestricted Access to Sensitive Business Flows: Extra throttling/heuristics on business-critical flows to deter automated abuse even without a code bug.[39]
- API7 Server-Side Request Forgery: DNS-based private IP detection, host allow-list, scheme restrictions, and closed-by-default egress.[39]
- API8 Security Misconfiguration: Secure headers, rejected dangerous verbs, strict parsing and key blacklists, and default-deny on undocumented endpoints.[39]
- API9 Improper Inventory Management: Version gating, deprecation blocking, and a curated set of documented endpoints that fail closed.[39]
- API10 Unsafe Consumption of APIs: Upstream response validation and minimization, never trusting third-party payloads implicitly.[39]

How to adapt for production

- Replace fake_verify_jwt with standards-based verification: OIDC discovery, JWKS caching/rotation, clock skew checks, token type validation, and optional mTLS for service-to-service. This directly strengthens API2.[39]
- Centralize authorization: Use ABAC/RBAC policy engines (e.g., OPA, Cedar) and bind FUNCTION_ROLE_MAP/OBJECT_OWNERSHIP_FIELDS/PROPERTY_ACL to policy decisions and attribute stores, strengthening API1/API3/API5.[39]
- Distributed rate limiting: Back token buckets with Redis and add concurrency caps, timeouts, circuit breakers, and per-scope weights; apply separate budgets to expensive

downstream integrations to limit cost amplification (API4).[39]

- Stronger anti-automation: Add device fingerprinting, risk scoring, progressive challenges, and anomaly detection for sensitive flows to reduce business flow abuse (API6).[39]
- Harden SSRF further: Add DNS rebinding protections, forbid redirects to private IPs, normalize/resolve IPv6 and dotless hosts, and pin to allow-listed netblocks if feasible (API7).[39]
- Automate inventory: Generate DOCUMENTED_ENDPOINTS and request/response schemas from OpenAPI in CI/CD; fail deployments if drift is detected, and set sunsetting windows for deprecated versions (API9).[39]
- Observability: Log denials and decisions with structured fields (principal, route, reason) and feed SIEM/UEBA to detect abuse patterns across API1–API10 categories.[39]

This code and configuration scaffold provide a fail-closed, policy-driven approach aligned with the OWASP API Security Top 10 (2023), emphasizing least privilege, explicit allow-listing, and strict validation across the full request/response lifecycle.[39]

⁂

soo the owasp top 10 vulnerabilities are common, but when i develop a WAF with pre defined rules, if any attack signatures are detected then it can be blocked, and that same WAF can be put in as a proxy between user and server in almost all web applications, but in case of API security, the API endpoints and all are custom and different for each web application,so how can i develop a WAAP?

A reusable WAF can block generic web attacks with static signatures, but a WAAP for APIs must be policy-driven and API-aware: it needs your API inventory, schemas, identities, and business logic to enforce per-endpoint authentication, authorization, and usage rules that signatures alone can't capture. The practical way to build a broadly reusable WAAP is to separate the generic controls from API-specific policy and load that policy from specs (OpenAPI), identity providers, and per-service configuration at deploy time.[67] [68] [69]

Core idea: policy + engine

- Build a generic enforcement engine (authn, authz checks, schema validation, rate limiting, SSRF egress control, security headers), and keep all API specifics in versioned policies derived from OpenAPI and org standards; at runtime the same engine works for different apps by loading different policies.[68] [69]

- Treat the WAAP as a sidecar/gateway pattern: place it between clients and services (or in front of an API gateway) so it can apply identity and schema-aware controls per route, not just regex signatures; this aligns with secure API gateway blueprints.[69] [67]

What is generic and reusable

- Common OWASP API Top 10 mitigations are reusable: token verification, issuer/audience checks, function/object/property-level authorization hooks, rate limits, payload size/depth caps, SSRF allow-lists, secure headers, deprecations/version gating, and upstream response validation; these are engine capabilities, not per-API logic.[68] [69]
- Deploy pattern: Internet → Edge WAF/WAAP → API Gateway/APIM → Services; edge enforces generic Layer-7 controls, and APIM/WAAP enforces API-aware policies (schemas, roles, quotas) with centralized policies and telemetry.[69]

What must be per-API policy

- Endpoints and methods: generated from your OpenAPI spec to enumerate documented surfaces, versions, and parameter shapes; everything else fails closed to curb shadow/zombie APIs.[68]
- Schemas: request and response contracts (types, required fields, enums, max sizes) compiled to validators for each operation; this blocks mass assignment and excessive data exposure.[68]
- Authorization matrices: per-operation role/scope rules and object-ownership checks derived from your domain model; this closes BOLA and function-level auth gaps.[68]
- Sensitive flows: identify business-critical operations (payments, ticketing, password reset) and attach anti-automation, stronger quotas, and step-up auth to just those routes.[68]

How to make it portable across apps

- Spec-driven: ingest OpenAPI at deploy time and auto-generate allow-lists (paths, params), validators, and default rate-limit classes; keep overrides for sensitive flows and admin endpoints; fail deploy if spec and code drift.[69] [68]
- Identity-driven: integrate with your IdP (OIDC/JWKS) and map tokens to roles/scopes/attributes; the engine stays the same while mappings differ per tenant/app. [69] [68]
- Policy bundles: ship policies as versioned bundles (YAML/JSON) that declare per-route auth, quotas, and SSRF egress rules; the same WAAP binary can load different bundles at startup or via control plane.[67] [69]
- Reference blueprint: follow a secure API gateway blueprint with threat modeling, policy enforcement points, and CI/CD checks so each app onboards by providing spec + policy, not code changes.[67]

Minimal architecture

- Control plane: stores OpenAPI specs, policy bundles, JWKS metadata, and distributes compiled policies to gateways; also runs CI validation.[67] [69]
- Data plane: stateless WAAP instances that enforce policies: TLS termination, token checks, schema validation, rate limiting, SSRF guard, security headers, and logging; can run as ingress, sidecar, or gateway plugin.[69]
- Observability: structured logs and metrics per route and decision (allow/deny/reason) for forensics and tuning; feed SIEM and anomaly detection for business logic abuse.[69]

Concrete build plan

- Start with an engine like the Python middleware shared earlier, but replace hardcoded maps with hot-loaded policies compiled from OpenAPI and YAML; add a loader that watches a policy store and rebuilds validators and route tables on the fly.[68]
- Implement policy objects: Routes (method, path template), Schemas (compiled from OpenAPI), AuthZ (RBAC/ABAC rules), Quotas (per role/route), SensitiveFlows, EgressRules (allow-lists), Versions (active/deprecated), and DefaultDeny.[68]
- Provide adapters: FastAPI/Starlette, Flask, ASGI, and Envoy/NGINX plugin variants so the same logic runs at different layers; this mirrors vendor blueprints pairing WAF + APIM.[69]
- Add discovery guardrails: only publish and allow routes present in the spec; tag deprecated operations with sunset dates; block undocumented parameters; inventory is a first-class control against API9.[68]

Why signatures alone aren’t enough for APIs

- Many API risks are authorization and workflow issues (BOLA/BFLA, sensitive business flows) that don’t present unique payload signatures; they require identity, object ownership, and rate/sequence analysis per operation, which is inherently policy-driven.[68]
- Unsafe consumption of third-party APIs and SSRF require outbound validation and schema checks, not inbound signatures; a WAAP must validate destinations and upstream responses.[68]

Deployment patterns that work

- Edge WAF + APIM: Edge blocks common web exploits; APIM/WAAP enforces API-aware policies (rate by key, private networking, bot protection), a pattern documented in platform architectures.[69]
- Gateway + WAAP synergy: Combine API gateway traffic management with WAAP threat detection and business logic abuse defenses to cover gaps across layers.[70] [69]

First steps to implement

- Standardize: require every team to maintain OpenAPI with security schemes and object identifiers; enforce CI linting and publish to the control plane.[68]
- Automate compilation: turn OpenAPI into validators and route maps; generate default deny policies; require explicit approvals for sensitive flows and egress hosts.[68]
- Integrate IdP: enable JWKS rotation, token introspection, and scope mapping; default-deny anonymous access unless marked public.[69] [68]
- Instrument and iterate: monitor denials and anomalies; tune quotas and anti-automation where abuse appears; sunset deprecated versions per policy.[69] [68]

Bottom line: design the WAAP as a reusable engine plus per-app policy generated from OpenAPI and identity mappings; deploy it as a gateway/sidecar that enforces object-, property-, and function-level rules, resource controls, SSRF egress allow-lists, and versioned inventories so it adapts to each application without rewriting code. This aligns with OWASP API Top 10 guidance and secure API gateway blueprints used in production architectures.[67] [69] [68]

⁂

i need to develop a global and common API ,not a per APP one, so give me the vulnerabilities that it can address and give me python code of that WAAP module.

A single, global WAAP can only enforce controls that are API-agnostic: identity/authentication checks, default-deny surfaces, generic authorization patterns, schema conformance from specs, abuse throttling, SSRF-safe egress, security headers, and upstream response validation. These map to OWASP API Top 10 risks that do not require per-app business logic: API2 (authn), API4 (resource consumption), API6 (flow abuse throttling), API7 (SSRF), API8 (misconfiguration), API9 (inventory/versioning), and API10 (unsafe consumption); API1, API3, and API5 can be covered generically by pattern- and spec-driven policies (e.g., per-route ownership and role gates) when the WAAP loads OpenAPI and IdP metadata at deploy time.[98] [99] [100] [101]

Vulnerabilities addressed globally

- Broken Authentication (API2): Enforce OIDC/JWKS token validation, issuer/audience checks, token age and revocation; default-deny unauthenticated calls except explicitly public operations.[99] [98]
- Unrestricted Resource Consumption (API4): Per-identity rate limiting, quotas, payload size/depth/array caps, timeouts, and concurrency limits to prevent DoS and cost spikes.[102] [98]
- Unrestricted Access to Sensitive Business Flows (API6): Route tagging for flows like payments or password reset with higher throttles, anomaly/bot heuristics, and optional step-up auth.[100] [98]

- Server-Side Request Forgery (API7): Outbound allow-list by host/scheme, private IP/DNS-rebinding protection, strict redirect policy, and per-integration egress keys.[98]
- Security Misconfiguration (API8): Secure defaults: TLS, strict methods, header hardening, strict JSON parsing, parameter validation from spec, and default-deny for undocumented routes.[103] [98]
- Improper Inventory Management (API9): Version gating, deprecation blocks, only documented endpoints from OpenAPI are exposed; shadow/zombie endpoints 404/410.[99] [98]
- Unsafe Consumption of APIs (API10): Validate third-party responses against schemas, enforce timeouts/retries, minimize data, and isolate partner traffic.[103] [98]
- Partially generic coverage for BOLA/BFLA/Property-level (API1/5/3): Apply spec-driven role/scope checks and object-ownership hooks based on a standard resource model (owner_id in path/body), then let apps optionally extend; many APIs benefit without per-app code.[100] [98]

Global Python WAAP module (ASGI/FastAPI adapter)

This module is policy-driven and loads OpenAPI-derived routes/schemas plus IdP settings. Drop it in front of diverse APIs; per-app specifics come from the supplied spec/policy file, not code.

```
# pip install fastapi uvicorn pydantic httpx pyjwt cryptography
import json, time, ipaddress, hashlib, re
from typing import Dict, Any, Optional, Tuple, Set
from urllib.parse import urlparse

import httpx, jwt
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from pydantic import BaseModel, ValidationError

# ---- Control plane-fed policy (replace loader to read OpenAPI + YAML) ----
class Policy(BaseModel):
    allowed_versions: Set[str]
    deprecated_versions: Set[str]
    documented_endpoints: Set[Tuple[str, str]]   # (METHOD, /vX/route/{id})
    public_endpoints: Set[Tuple[str, str]]       # endpoints that don’t need auth
    function_roles: Dict[str, Set[str]]          # f"{M} {tpl}" -> roles
    resource_owner_hint: Dict[str, str]          # f"{M} {tpl}" -> field name like "id"
    sensitive_flows: Set[Tuple[str, str]]
    egress_allow: Dict[str, Set[str]]            # host -> allowed schemes
    jwt_issuers: Set[str]
    jwt_audience: str

# Example: default global policy (can be overridden per tenant via config)
POLICY = Policy(
    allowed_versions={"v1","v2"},
    deprecated_versions={"v0"},
    documented_endpoints=set(),  # load from OpenAPI; empty -> deny all by default
    public_endpoints={("GET","/v1/health")},
    function_roles={},            # fill from policy
    resource_owner_hint={},       # fill if BOLA checks possible
    sensitive_flows=set(),
    egress_allow={"api.stripe.com":{"https"}, "identity.example.com":{"https"}},
    jwt_issuers={"https://identity.example.com/"},
    jwt_audience="api://global"
)

SECURITY_HEADERS = {
    "X-Content-Type-Options": "nosniff",
    "X-Frame-Options": "DENY",
    "Referrer-Policy": "no-referrer",
    "Cache-Control": "no-store",
}

REJECT_METHODS = {"TRACE","TRACK","CONNECT"}
MAX_BODY = 1_000_000
MAX_DEPTH = 20
MAX_ITEMS = 10000
REJECT_QUERY_KEYS = {"__proto__","constructor","$where"}

def route_key(method: str, path: str) -> Tuple[str,str]:
    parts = path.strip("/").split("/")
    norm = []
    for p in parts:
        norm.append("{id}" if re.fullmatch(r"[0-9a-fA-F-]{6,}", p) else p)
    return method.upper(), "/" + "/".join(norm)

def deep_limits(value, depth=0):
    if depth > MAX_DEPTH: raise ValueError("depth")
    if isinstance(value, list):
        if len(value) > MAX_ITEMS: raise ValueError("array")
        for v in value: deep_limits(v, depth+1)
    elif isinstance(value, dict):
        if len(value) > MAX_ITEMS: raise ValueError("object")
        for v in value.values(): deep_limits(v, depth+1)

def is_private_ip(host: str) -> bool:
    try:
        resp = httpx.get(f"https://dns.google/resolve?name={host}&type=A", timeout=2.0).
        for ans in resp.get("Answer", []):
            ip = ipaddress.ip_address(ans["data"])
            if ip.is_private or ip.is_loopback or ip.is_link_local or ip.is_multicast or
                return True
        return False
    except Exception:
        return True

class TokenBucket:
    def __init__(self, capacity, refill_per_sec):
        self.capacity = capacity; self.tokens = capacity; self.rate = refill_per_sec; sel
    def allow(self, n=1):
        now = time.time(); dt = now - self.t; self.t = now
        self.tokens = min(self.capacity, self.tokens + dt*self.rate)
        if self.tokens >= n: self.tokens -= n; return True
        return False
```

```
rate_buckets: Dict[str, TokenBucket] = {}
DEFAULT_LIMIT = (120, 60, 40)   # limit, window, burst
PER_PATH_LIMITS: Dict[str, Tuple[int,int,int]] = {}

def rate_key(identity: str, path: str) -> Tuple[int,int,int]:
    return PER_PATH_LIMITS.get(path, DEFAULT_LIMIT)

def check_rate(identity: str, path: str) -> bool:
    limit, window, burst = rate_key(identity, path)
    k = f"{identity}:{path}:{window}"
    if k not in rate_buckets:
        rate_buckets[k] = TokenBucket(burst, limit/window)
    return rate_buckets[k].allow()

def hash_identity(ip: str, sub: Optional[str]) -> str:
    return hashlib.sha256(f"{ip}|{sub or ''}".encode()).hexdigest()

def verify_jwt(authz: str) -> Optional[Dict[str,Any]]:
    if not authz.startswith("Bearer "): return None
    token = authz.split(" ",1)[^4_1].strip()
    try:
        # In production: discover JWKS for issuer, pick key by kid, verify sig+claims
        unverified = jwt.get_unverified_header(token)
        claims = jwt.decode(token, options={"verify_signature": False}, algorithms=["RS25
        if claims.get("iss") not in POLICY.jwt_issuers: return None
        if claims.get("aud") != POLICY.jwt_audience: return None
        if claims.get("exp") and time.time() > claims["exp"]: return None
        return {"sub": str(claims.get("sub")), "role": claims.get("role","user")}
    except Exception:
        return None

class SafeEgress:
    def __init__(self, allow: Dict[str,Set[str]]): self.allow = allow; self.client=httpx.
    def _check(self, url: str):
        u = urlparse(url)
        if u.scheme not in {"http","https"}: raise ValueError("scheme")
        if not u.hostname: raise ValueError("host")
        if is_private_ip(u.hostname): raise ValueError("private")
        allowed = self.allow.get(u.hostname)
        if not allowed or u.scheme not in allowed: raise ValueError("not allowed")
    def get(self, url, **kw): self._check(url); return self.client.get(url, **kw)
    def post(self, url, **kw): self._check(url); return self.client.post(url, **kw)

egress = SafeEgress(POLICY.egress_allow)

class GlobalWAAP:
    def __init__(self, app: FastAPI): self.app = app
    async def __call__(self, scope, receive, send):
        if scope["type"] != "http": return await self.app(scope, receive, send)
        req = Request(scope, receive=receive)
        m = req.method.upper(); p = req.url.path; rk = route_key(m, p)

        if m in REJECT_METHODS:
            return await JSONResponse({"detail":"method not allowed"}, status_code=405)(s

        segs = [s for s in p.split("/") if s]
        if segs and segs[^4_0] not in POLICY.allowed_versions:
            if segs[^4_0] in POLICY.deprecated_versions:
                return await JSONResponse({"detail":"deprecated"}, status_code=410)(scope
            return await JSONResponse({"detail":"unknown version"}, status_code=404)(scop

        if rk not in POLICY.documented_endpoints:
            return await JSONResponse({"detail":"undocumented"}, status_code=404)(scope,

        principal = None
        if rk not in POLICY.public_endpoints:
            principal = verify_jwt(req.headers.get("authorization",""))
            if not principal: return await JSONResponse({"detail":"unauthorized"}, status

        identity = hash_identity(req.client.host, principal.get("sub") if principal else
        if not check_rate(identity, p):
            return await JSONResponse({"detail":"too many requests"}, status_code=429)(sc

        body = b""
        if m in {"POST","PUT","PATCH"}:
            body = await req.body()
            if len(body) > MAX_BODY:
                return await JSONResponse({"detail":"payload too large"}, status_code=413

        json_body = None
        ctype = req.headers.get("content-type","")
        if body and "application/json" in ctype:
            try:
                json_body = json.loads(body.decode("utf-8"))
                deep_limits(json_body)
                for k in req.query_params.keys():
                    if k in REJECT_QUERY_KEYS:
                        return await JSONResponse({"detail":"bad query key"}, status_code
            except Exception:
                return await JSONResponse({"detail":"invalid json"}, status_code=400)(sco

        # Function-level generic RBAC if declared
        key = f"{rk[^4_0]} {rk[^4_1]}"
        roles = POLICY.function_roles.get(key)
        if roles and principal and principal.get("role") not in roles:
            return await JSONResponse({"detail":"forbidden"}, status_code=403)(scope, rec

        # Generic BOLA: if a resource owner hint exists and path contains {id}, compare t
        hint = POLICY.resource_owner_hint.get(key)
        if principal and hint and "{id}" in rk[^4_1]:
            # naive extract: last segment
            victim_id = p.rstrip("/").split("/")[-1]
            if str(principal.get("sub")) != str(victim_id) and principal.get("role") not
                return await JSONResponse({"detail":"forbidden"}, status_code=403)(scope,

        # Sensitive flow throttle: simple UA heuristic
        if (rk[^4_0], rk[^4_1]) in POLICY.sensitive_flows:
            ua = req.headers.get("user-agent","")
            if not ua or len(ua) < 8:
                return await JSONResponse({"detail":"automation suspected"}, status_code=

        async def send_wrap(message):
            if message["type"] == "http.response.start":
                headers = list(message.get("headers", []))
                for k,v in SECURITY_HEADERS.items(): headers.append((k.encode(), v.encode
                message["headers"] = headers
            return await send(message)

        scope["state"]["principal"] = principal
        scope["state"]["json_body"] = json_body
        return await self.app(scope, receive, send_wrap)

# Usage:
app = FastAPI()
app.middleware("http")(GlobalWAAP(app))

@app.get("/v1/health")
async def health(): return JSONResponse({"ok":True})

# Example egress use with SSRF guard:
@app.get("/v1/payments/providers")
async def providers():
    try:
        r = egress.get("https://api.stripe.com/v1/capabilities")
        if r.status_code != 200: return JSONResponse({"detail":"upstream error"}, status_
        data = r.json()
        if not isinstance(data, dict): return JSONResponse({"detail":"bad upstream"}, sta
        return JSONResponse({"provider":"stripe", "capabilities": list(data.keys())[:10]}
    except Exception:
        return JSONResponse({"detail":"blocked"}, status_code=502)
```

How this global module maps to OWASP API Top 10

- API2 Broken Authentication: OIDC/JWT verification with issuer/audience/expiry and default-deny on protected routes. Replace decode stub with JWKS verification per issuer. [98] [99]
- API4 Unrestricted Resource Consumption: Token-bucket rate limiting per identity+path; caps on payload size, JSON depth and item counts. [102] [98]
- API6 Unrestricted Access to Sensitive Business Flows: Tag sensitive routes in policy and apply extra throttling/heuristics. [100] [98]
- API7 SSRF: SafeEgress class enforces outbound allow-list and blocks private IP/DNS tricks. [98]
- API8 Security Misconfiguration: Deny risky methods, strict header hardening, strict JSON parsing, and reject suspicious query keys. [103] [98]
- API9 Improper Inventory Management: Only serve requests for versions and endpoints declared in policy (typically compiled from OpenAPI); unknown or deprecated versions return 404/410. [99] [98]
- API10 Unsafe Consumption: Validate upstream responses, enforce timeouts, and minimize returned data. [103] [98]

- API1/3/5 Partial generic coverage: Optional policy keys enable route RBAC and a simple ownership check when resource IDs are in paths; for richer models, apps can extend via attributes/scopes while reusing the same engine.[100] [98]

This design stays global by keeping the enforcement engine generic while loading API-specific facts (routes, versions, public operations, optional RBAC and ownership hints) from a policy file derived from each API's OpenAPI, enabling common protections without embedding per-app business logic.[101] [99] [98] [100]

⁂

1. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
2. https://owasp.org/API-Security/editions/2023/en/0x00-header/
3. https://apisecurity.io/owasp-api-security-top-10/
4. https://www.pynt.io/learning-hub/owasp-top-10-guide/owasp-api-top-10
5. https://portswigger.net/web-security/api-testing/top-10-api-vulnerabilities
6. https://konghq.com/blog/engineering/owasp-top-10-api-security-2023
7. https://owasp.org/www-community/Web_Application_Firewall
8. https://www.cloudflare.com/learning/security/api/owasp-api-security-top-10/
9. https://www.f5.com/glossary/owasp-api-security-top-10
10. https://learn.microsoft.com/en-us/azure/api-management/mitigate-owasp-api-threats
11. https://ieeexplore.ieee.org/document/10276919/
12. https://ieeexplore.ieee.org/document/10374730/
13. https://ieeexplore.ieee.org/document/10111955/
14. https://journal.uad.ac.id/index.php/JITEKI/article/view/25987
15. https://ieeexplore.ieee.org/document/10182656/
16. https://ojs.unud.ac.id/index.php/jitter/article/view/104690
17. https://www.mdpi.com/1424-8220/23/14/6307
18. https://ieeexplore.ieee.org/document/10401373/
19. https://www.mdpi.com/2076-3417/13/21/11784
20. https://onepetro.org/JPT/article/75/05/40/522705/Location-Location-Tanzania-Mozambique-Soon-To-Join
21. https://www.techrxiv.org/articles/preprint/An_OWASP_Top_Ten_Driven_Survey_on_Web_Application_Protection_Methods/13265180/files/25545929.pdf
22. https://www.mdpi.com/2076-3417/12/8/4077/pdf?version=1650281818
23. https://dl.acm.org/doi/pdf/10.1145/3658644.3690294
24. https://arxiv.org/html/2306.08869v2
25. https://www.techscience.com/cmc/v64n3/39444
26. https://arxiv.org/pdf/2405.01111.pdf
27. https://ccsenet.org/journal/index.php/cis/article/download/4279/3726
28. https://arxiv.org/pdf/2311.10450.pdf

29. https://owasp.org/www-project-api-security/
30. https://owasp.org/www-project-top-ten/
31. https://bugbase.ai/blog/top-10-ways-to-bypass-waf
32. https://www.cequence.ai/blog/owasp/owasp-top-10-lists-end-state-or-starting-point/
33. https://www.wiz.io/academy/owasp-api-security
34. https://aws.amazon.com/about-aws/whats-new/2017/07/use-aws-waf-to-mitigate-owasps-top-10-web-application-vulnerabilities/
35. https://www.apisecuniversity.com/courses/owasp-api-security-top-10-and-beyond
36. https://www.practical-devsecops.com/top-api-security-vulnerabilities-guide/
37. https://github.com/0xInfection/Awesome-WAF
38. https://owasp.org/www-project-enterprise-security-api/
39. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
40. https://www.sec.gov/Archives/edgar/data/1796209/000162828025047400/apg-20250930.htm
41. https://www.mdpi.com/1424-8220/23/14/6307
42. https://ieeexplore.ieee.org/document/10100357/
43. http://www.cmaj.ca/lookup/doi/10.1503/cmaj.230240
44. https://www.mdpi.com/2076-3417/13/21/11784
45. https://www.semanticscholar.org/paper/b16847a51eb9d1b0edda61344633a9d0a121e80c
46. https://ieeexplore.ieee.org/document/11004742/
47. https://dl.acm.org/doi/10.1145/3655693.3655701
48. https://ieeexplore.ieee.org/document/11157553/
49. https://www.techrxiv.org/articles/preprint/An_OWASP_Top_Ten_Driven_Survey_on_Web_Application_Protection_Methods/13265180/files/25545929.pdf
50. https://www.mdpi.com/2076-3417/12/8/4077/pdf?version=1650281818
51. https://www.sec.gov/Archives/edgar/data/2047190/000149315225014170/formf-1a.htm
52. https://arxiv.org/pdf/2405.01111.pdf
53. https://dl.acm.org/doi/pdf/10.1145/3658644.3690227
54. https://ccsenet.org/journal/index.php/cis/article/download/4279/3726
55. https://publications.eai.eu/index.php/cs/article/download/3011/2609
56. https://arxiv.org/pdf/2311.10450.pdf
57. https://arxiv.org/pdf/2310.14137.pdf
58. https://owasp.org/www-project-top-ten/
59. https://www.sec.gov/Archives/edgar/data/1796209/000162828025036919/apg-20250630.htm
60. https://www.sec.gov/Archives/edgar/data/2047190/000149315225018140/form424b4.htm
61. https://www.sec.gov/Archives/edgar/data/2047190/000164117225026169/formf-1a.htm
62. https://www.sec.gov/Archives/edgar/data/2047190/000164117225024899/formf-1a.htm
63. https://www.sec.gov/Archives/edgar/data/2047190/000164117225021315/formf1.htm
64. https://www.sec.gov/Archives/edgar/data/1796209/000119312525125091/d56815d8k.htm
65. https://ieeexplore.ieee.org/document/10276919/

66. https://ieeexplore.ieee.org/document/10182656/
67. https://owasp.org/www-project-secure-api-gateway-blueprint/
68. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
69. https://learn.microsoft.com/en-us/azure/architecture/web-apps/api-management/architectures/protect-apis
70. https://www.radware.com/blog/application-protection/the-synergy-of-api-gateway-and-waap/
71. https://zenodo.org/record/5727094/files/main.pdf
72. https://downloads.hindawi.com/journals/wcmc/2022/1657627.pdf
73. https://arxiv.org/pdf/2212.06606.pdf
74. http://arxiv.org/pdf/2402.13696.pdf
75. https://arxiv.org/pdf/2105.02031.pdf
76. https://arxiv.org/pdf/2108.00033.pdf
77. https://owasp.org/www-project-api-security/
78. https://www.devopsdigest.com/waap-web-application-and-api-security-the-essential-guide
79. https://appsentinels.ai/blog/waf-vs-api-gateway/
80. https://www.checkpoint.com/cyber-hub/cloud-security/what-is-application-security-appsec/best-practices-for-secure-api-design/
81. https://www.indusface.com/blog/whats-new-in-owasp-api-top-10-2023/
82. https://qodex.ai/blog/15-api-security-best-practices-to-secure-your-apis-in-2026
83. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-228.pdf
84. https://learn.microsoft.com/en-us/azure/api-management/mitigate-owasp-api-threats
85. https://www.imperva.com/learn/application-security/api-security/
86. https://docs.fortinet.com/document/fortiweb/8.0.0/waf-concept-guide/637600/waf-features-against-owasp-top-10-api-security-risks
87. https://www.cycognito.com/learn/api-security/owasp-api-security.php
88. https://content.salt.security/rs/352-UXR-417/images/Salt Security Solution Brief - WAAP 2021.02.26.pdf
89. https://www.wiz.io/academy/owasp-api-security
90. https://konghq.com/blog/engineering/owasp-top-10-api-security-2023
91. https://www.f5.com/company/blog/api-security-checklist
92. https://owasp.org/API-Security/
93. https://www.multidisciplinaryfrontiers.com/search?q=FMR-2025-1-099&search=search
94. https://journalwjaets.com/node/980
95. https://ijsrcseit.com/index.php/home/article/view/CSEIT23112568
96. https://arxiv.org/pdf/1606.07479.pdf
97. https://zenodo.org/record/4550441/files/MAP-EuroPlop2020aPaper.pdf
98. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
99. https://learn.microsoft.com/en-us/azure/api-management/mitigate-owasp-api-threats
100. https://apisecurity.io/owasp-api-security-top-10/
101. https://owasp.org/API-Security/editions/2023/en/0x00-header/

102. https://www.cycognito.com/learn/api-security/owasp-api-security.php
103. https://www.pynt.io/learning-hub/owasp-top-10-guide/owasp-api-top-10
104. https://ijsrem.com/download/api-attack-vectors-understanding-and-mitigating-emerging-threats/
105. https://arxiv.org/abs/2507.02309
106. https://ieeexplore.ieee.org/document/9833760/
107. http://hdl.handle.net/10125/59641
108. https://www.semanticscholar.org/paper/196d11cc439417830130fc20dc2c87313b0881e6
109. https://www.semanticscholar.org/paper/ab14e25b24943d4d2521fcd87d0e2cdc4a4f2eee
110. https://www.techrxiv.org/articles/preprint/An_OWASP_Top_Ten_Driven_Survey_on_Web_Application_Protection_Methods/13265180/files/25545929.pdf
111. https://arxiv.org/pdf/2212.06606.pdf
112. https://arxiv.org/html/2306.08869v2
113. https://www.mdpi.com/1424-8220/24/15/4859
114. https://publications.eai.eu/index.php/cs/article/download/3011/2609
115. https://dl.acm.org/doi/pdf/10.1145/3658644.3690227
116. https://arxiv.org/pdf/1307.6649.pdf
117. https://arxiv.org/pdf/2405.01111.pdf
118. https://owasp.org/www-project-api-security/
119. https://42crunch.com/owasp-api-security-top-10/
120. https://qodex.ai/blog/owasp-top-10-for-api-security-a-complete-guide
121. https://owasp.org/www-project-top-ten/
122. https://www.openappsec.io/post/the-developer-s-guide-to-owasp-api-security
123. https://www.indusface.com/blog/whats-new-in-owasp-api-top-10-2023/
124. https://www.wiz.io/academy/owasp-api-security
125. https://dev.to/freelancingsolutions/how-to-create-a-high-performant-api-gateway-using-python-4059
126. https://portswigger.net/web-security/api-testing/top-10-api-vulnerabilities
127. https://escape.tech/blog/how-to-secure-fastapi-api/
128. https://www.apisecuniversity.com/courses/owasp-api-security-top-10-and-beyond
129. https://securedebug.com/api-gateway-security-implementation-best-practices/
130. https://www.cloudflare.com/learning/security/api/owasp-api-security-top-10/
131. https://www.freecodecamp.org/news/owasp-api-security-top-10-secure-your-apis/
132. https://ieeexplore.ieee.org/document/10276919/
133. https://www.mdpi.com/1424-8220/23/14/6307
134. https://www.mdpi.com/2076-3417/13/21/11784