



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 'none'",
    "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).json()
        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 ip.is_multicast:
                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[0] not in ALLOWED_API_VERSIONS:
            if segs[0] in DEPRECATED_VERSIONS:
                return await JSONResponse({"detail": "Deprecated API version"}, status_code=404)()
            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=404)()

        # API2: authentication (bearer or mTLS etc.)
        principal = None
        authz = request.headers.get("authorization", "")
        if authz.startswith("Bearer "):
            token = authz.split(" ", 1)[1].strip()
            principal = fake_verify_jwt(token)
            if not principal:
                return await JSONResponse({"detail": "Invalid token"}, status_code=401)()
            if principal.get("iss") not in JWT_ISSUERS or principal.get("aud") != JWT_AUDIENCE:
                return await JSONResponse({"detail": "Token not acceptable"}, status_code=401)()
        else:

```

```

        # allow unauthenticated only for explicitly public endpoints (example: login)
        if rk != ("POST", "/v1/auth/login"):
            return await JSONResponse({"detail": "Authentication required"}, status_code=401)

    # API4: rate limiting / resource control
    identity = hash_identity(request.client.host, principal.get("sub")) if principal else None
    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=413)

    # 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"}, status_code=400)
        except (json.JSONDecodeError, UnicodeDecodeError, ValueError):
            return await JSONResponse({"detail": "Invalid JSON"}, status_code=400)

    # 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.errors()}, status_code=400)

    # 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)"}, status_code=403)

    # API1: object-level authZ
    if principal and rk in OBJECT_OWNERSHIP_FIELDS:
        tpl = rk[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_code=403)

    # API6: protect sensitive business flows from automation
    if (method, rk[1]) in SENSITIVE_BUSINESS_FLOWS:
        # simple anti-automation heuristics (augment with device fingerprint / challenge)

```

```

        ua = request.headers.get("user-agent", "")
        if not ua or len(ua) < 8:
            return await JSONResponse({"detail": "Automation suspected"}, status_code=403)

    # Wrap send to inject security headers and perform property-level filtering on response
    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": None}
    # demo record (replace with DB)
    user = {"id": id, "email": f"user{id}@example.com", "role": "customer", "passwordHash": "demo"}

    # 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": "customer", "passwordHash": "demo"}
    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 sunset 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]

✱

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/OxInfection/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/>