CodeAgent Vulnerability Scanner — Complete API & Operations Guide (FastAPI)

Base URL (dev): `http://localhost:8080`

This document is the authoritative reference for the CodeAgent Vulnerability Scanner service. It covers **how the API works end-to-end**, all endpoints and models, job lifecycle, rate limits, auth, error codes, webhooks/SSE, analyzer configuration, and implementation notes. It is designed so you can hand it to frontend devs, partner integrators, or your future self.

---

1) Overview & Architecture

What it does: Accepts a GitHub repo URL or a ZIP upload → clones/extracts → runs analyzers (Semgrep, Bandit, dependency audit) → produces a **normalized JSON report** grouped by file and severity → exposes it via `/reports/{job_id}`. Optional **async mode**, **SSE progress**, and **webhooks**. Emits an internal `report.created` event for the agentic layer (autofix/test/PR creation).

Components - **FastAPI** app (`/analyze`, `/reports`, `/jobs`, `/events`, `/webhooks`, `/config`). - **Ingestion**: safe clone/extract into `storage/workspace/{job_id}` with ignores. - **Analyzers**: plugin runners (Semgrep, Bandit, pip-audit), concurrent execution with timeouts. - **Orchestrator**: merges findings → normalized report → writes to `storage/reports/{job_id}.json`. - **Events**: optional webhooks (HTTP POST) and in-process bus for agentic subscribers.

---

2) Authentication, Headers, Formats

2.1 Auth

- **Recommended (prod):** `Authorization: Bearer <API_KEY>` (HMAC/DB check server-side)
- **Dev:** unauthenticated

2.2 Global Headers

- `Idempotency-Key: <uuid>` — prevents duplicate scans on retries
- `X-Client-Id: <string>` — (optional) tenant/tracking
- `X-Request-Id` — returned on each response; you can also supply one

2.3 Content Types

- Requests: `multipart/form-data` (uploads), `application/json` (control/config)
- Responses: `application/json` (except SSE stream)

2.4 Versions

- `Accept: application/vnd.codeagent.v1+json` (optional vendor versioning)
- Minor, backward-compatible changes bump `version` field in `/health` and OpenAPI

---

3) Limits, Quotas, Timeouts

- **Upload cap:** 50 MB (413 on exceed)
- **Per job limits:** max files (e.g., 10,000), per-tool timeout (default 600s), overall wall-clock (e.g., 900s)
- **Rate limit:** 60 req/min per API key; max 2 concurrent `running` jobs per key
- **Ignore lists:** `.git`, `node_modules`, `venv`, `.venv`, `dist`, `build`, `__pycache__`, large binaries > 20MB
- **URL allow-list:** `https://github.com/*` initially; can be extended via config

---

4) Data Model

4.1 Severity Enum

`critical | high | medium | low`

4.2 Issue (normalized finding)

```
{
  "tool": "bandit",
  "type": "B608",
  "message": "Possible SQL injection via string building",
  "severity": "high",
  "file": "src/app.py",
  "line": 42,
  "rule_id": "B608",
  "suggestion": "Use parameterized queries"
}
```

4.3 FileIssues

```
{ "path": "src/app.py", "issues": [Issue, ...] }
```

4.4 Report

```
{
  "job_id": "4b3a...",
  "meta": {
    "tools": ["semgrep","bandit","pip-audit"],
    "repo": {"source": "github|zip", "url": "https://github.com/org/repo",
"ref": "main", "commit": null},
    "generated_at": "2025-10-17T14:30:00Z",
    "duration_ms": 23456,
    "labels": ["customer:acme","case:123"]
  },
  "summary": {"critical": 1, "high": 3, "medium": 7, "low": 4},
  "files": [FileIssues, ...]
}
```

4.5 Error

```
{
  "error": {
    "code": "INVALID_INPUT|NOT_FOUND|TIMEOUT|INTERNAL|RATE_LIMIT|
PAYLOAD_TOO_LARGE|UNAUTHORIZED|FORBIDDEN",
    "message": "Human-readable",
    "details": {"field": "github_url"}
  }
}
```

---

5) Job Lifecycle

`queued → running (clone → analyze:N tools → merge → write) → completed | failed | canceled | expired`

- **Creation**: `/analyze` or `/analyze-async` returns `job_id`
- **Progress**: `/jobs/{job_id}` or SSE `/events/{job_id}`
- **Result**: `/reports/{job_id}` once `completed`
- **Cancel**: `DELETE /jobs/{job_id}` when `queued|running`

---

6) Endpoints

6.1 Health & Metadata

GET `/health` → `200 { "status": "ok", "version": "0.1.0" }`

GET `/tools` → `200 { "available": ["bandit","semgrep","dep"], "default": ["bandit","semgrep","dep"], "versions": {"bandit":"1.7.9","semgrep":"1.81.0"} }`

6.2 Submit Scan (Sync / Auto-Async)

POST `/analyze` — may reply `200` (small jobs) or `202` (accepted) based on size/timeout hints.

Headers: `Content-Type: multipart/form-data`, optional `Idempotency-Key`

Form fields - `github_url` *(string, optional)* — `https://github.com/org/repo` - `ref` *(string, optional)* — branch/tag - `commit` *(string, optional)* — SHA to checkout - `file` *(file, optional)* — ZIP archive - `include` *(string, optional)* — comma CSV of globs (`src/**/*.py`) - `exclude` *(string, optional)* — comma CSV of globs (`tests/**,docs/**`) - `analyzers` *(string, optional)* — CSV: `bandit,semgrep,dep` - `timeout_sec` *(int, optional)* — overrides default - `labels` *(string, optional)* — CSV labels (`customer:acme,case:123`)

Responses - `200 OK` — `{ "job_id": "...", "summary": { ... } }` - `202 Accepted` — `{ "job_id": "...", "status": "running" }` - `400/401/403/413/429/500` — *Error*

Validation rules - Require **either** `github_url` **or** `file` (not both) - Validate `github_url` host and scheme; enforce allow-list - Enforce upload size cap and filename checks

---

6.3 Submit Scan (Async)

POST `/analyze-async`

Request — same fields as `/analyze`.

202 Accepted

```
{ "job_id": "uuid", "status": "queued" }
```

Background task performs clone/extract → analyze → write report → emit `report.created` and deliver webhooks.

6.4 Jobs

GET `/jobs/{job_id}` → status & timestamps

```
{
  "job_id": "...",
  "status": "queued|running|completed|failed|canceled|expired",
  "progress": {"phase": "clone|analyze:semgrep|analyze:bandit|merge|write",
"percent": 65},
  "submitted_at": "...",
  "started_at": "...",
  "finished_at": "...",
  "error": null
}
```

DELETE `/jobs/{job_id}` — cancel - `202 { "job_id": "...", "status": "canceling" }` - `409` if not cancellable

POST `/jobs/{job_id}/rerun` — requeue with same inputs - `202 { "job_id": "...", "status": "queued" }`

6.5 Reports

GET `/reports/{job_id}` — full report - `200` Report - `404` Not found (not finished/doesn't exist)

GET `/reports` — paginate/filter **Query:** `page`, `limit` (≤100), `severity=high,critical`, `tool=semgrep`, `repo=https://github.com/org/repo`, `since`, `until`, `label`

```
{
  "items": [Report, ...],
  "page": 1,
  "limit": 20,
  "total": 57
}
```

GET `/reports/{job_id}/summary` — light summary

```
{ "job_id": "...", "summary": {"critical":0,"high":2,"medium":7,"low":5} }
```

6.6 Live Progress (SSE)

GET `/events/{job_id}` - Header: `Accept: text/event-stream` - Server emits events:

```
:event: progress
:data: {"phase":"semgrep","percent":32}

:event: finished
:data: {"job_id":"...","status":"completed"}
```

6.7 Webhooks

POST `/webhooks/register` — register a URL to receive `report.created`

```
{ "url": "https://example.com/hooks", "events": ["report.created"], "secret":
"optional" }
```

- `201 { "id": "wh_123" }`

Delivery - Method: `POST` - Headers: `X-Event: report.created`, `X-Signature: sha256=...` (HMAC over body) - Body:

```
{ "job_id": "...", "repo": {"url":"..."}, "summary": {...}, "report_url": "/
reports/..." }
```

- Retries with exponential backoff on 5xx/timeouts; disable after N attempts

DELETE `/webhooks/{id}` — unregister

6.8 Configuration

GET `/config/analyzers`

```
{ "defaults": ["bandit","semgrep","dep"], "rulesets": {"semgrep":["p/owasp-top-
ten","p/secrets"], "bandit": []}, "allow_list": ["https://github.com/"] }
```

PATCH `/config/analyzers`

```
{ "defaults": ["semgrep","dep"], "rulesets": {"semgrep":["p/owasp-top-ten","p/
secrets"]}, "allow_list": ["https://github.com/", "https://gitlab.com/"] }
```

- `200 { "ok": true }`

7) How Scanning Works (under the hood)

1. **Ingestion**
2. Validate inputs → create `job_id` → make `storage/workspace/{job_id}`
3. `git clone --depth=1 {url}` (or extract ZIP)

4. Remove ignored folders; optionally apply `include`/`exclude` globs
5. **Analyzer selection**
6. Parse `analyzers` CSV or use defaults
7. Enable Bandit only if `.py` files exist; Dep audit only if `requirements.txt` exists (JS/npm audit/OSV later)
8. **Execution**
9. Run in parallel (ThreadPoolExecutor), each with a per-tool timeout
10. Capture JSON output (Semgrep/Bandit/pip-audit); map to normalized Issue
11. **Merge**
12. Compute severity summary; group by file; attach metadata (tools, repo, duration)
13. **Persist & notify**
14. Write `storage/reports/{job_id}.json`
15. Update job state to `completed` or `failed`
16. Emit `report.created` (events + webhooks)

    Security note: The service never executes repo code. It only runs static analyzers over files. All paths are confined under the job workspace.

---

8) Error Handling & Status Codes

- `200 OK` — success (sync)
- `202 Accepted` — queued/running (async or auto-async)
- `400 INVALID_INPUT` — missing/invalid fields; both `github_url` and `file` sent; bad URL host
- `401 UNAUTHORIZED` — missing/invalid API key (prod)
- `403 FORBIDDEN` — plan/tenant restrictions
- `404 NOT_FOUND` — unknown `job_id`/report
- `409 CONFLICT` — cannot cancel/rerun
- `413 PAYLOAD_TOO_LARGE` — upload exceeds limit
- `429 RATE_LIMIT` — throttled; retry after `Retry-After`
- `500 INTERNAL` — unexpected server error
- `504 TIMEOUT` — per-tool or overall timeout exceeded

Each error uses the **Error** shape from §4.5 and returns an `X-Request-Id` header.

---

9) OpenAPI (runnable excerpt)

```
openapi: 3.0.3
info:
  title: CodeAgent Scanner API
  version: 0.1.0
servers:
  - url: http://localhost:8080
paths:
  /health:
```

```yaml
    get:
      summary: Health check
      responses:
        '200': { description: OK }
  /analyze:
    post:
      summary: Submit a scan (may return 200 or 202)
      requestBody:
        required: true
        content:
          multipart/form-data:
            schema:
              type: object
              properties:
                github_url: { type: string, format: uri }
                ref: { type: string }
                commit: { type: string }
                file: { type: string, format: binary }
                include: { type: string }
                exclude: { type: string }
                analyzers: { type: string }
                timeout_sec: { type: integer }
                labels: { type: string }
      responses:
        '200': { description: OK }
        '202': { description: Accepted }
        '400': { description: Bad Request }
        '413': { description: Payload Too Large }
        '500': { description: Internal Error }
  /reports/{job_id}:
    get:
      summary: Get full report
      parameters:
        - in: path
          name: job_id
          required: true
          schema: { type: string }
      responses:
        '200': { description: OK }
        '404': { description: Not Found }
```

Full OpenAPI can be generated directly by FastAPI from the routers; this excerpt anchors required fields.

10) Usage Examples

GitHub URL (auto-async on big repos)

```
curl -F github_url=https://github.com/pallets/flask
     -F analyzers=bandit,semgrep,dep
     http://localhost:8080/analyze
```

ZIP Upload (explicit async)

```
curl -F file=@/path/to/project.zip
     -F include="src/**/*.py"
     http://localhost:8080/analyze-async
```

Get Status & Report

```
curl http://localhost:8080/jobs/<job_id> | jq
curl http://localhost:8080/reports/<job_id> | jq
```

Register Webhook

```
curl -X POST http://localhost:8080/webhooks/register
  -H 'Content-Type: application/json'
  -d '{"url":"https://example.com/hooks","events":
["report.created"],"secret":"xyz"}'
```

---

11) Implementation Notes (for engineers)

- **Routers**: `api/routers/analyze.py` implements `/analyze`, `/analyze-async`, `/jobs/*`, `/reports/*`, `/events/*`.
- **BackgroundTasks**: used to offload large jobs; swap to Celery/RQ later for retries.
- **ThreadPool**: used per job to run analyzers in parallel; cap with env vars.
- **Storage layout**:
- `storage/workspace/{job_id}` — cloned/extracted repo
- `storage/reports/{job_id}.json` — final report
- `storage/logs/{job_id}.json` — status/progress, errors
- **Security**: never execute repo code; sanitize paths; enforce allow-lists; set subprocess timeouts.
- **Observability**: log JSON with `job_id` correlation; expose `X-Request-Id`.

---

12) Agentic Integration Hooks

- After report write: emit `report.created` (internal bus) and deliver webhooks.
- Subscriber (agent orchestrator) clusters issues → proposes patches/tests → opens PR.
- Optionally expose `/agents/actions` in future to attach `patch_unified` back to a job.

---

13) Retention & Privacy (defaults)

- **Workspace** purged after 7 days; **reports** retained 30 days (configurable)
- No PII processed unless present in repo; avoid storing access tokens in logs
- Provide `/jobs/{job_id}` **DELETE** to purge workspace & report early

---

14) FAQ

- **Why 200 vs 202 on `/analyze`?** Small repos may complete within request timeout → `200`. Larger ones return `202` and continue in background.
- **Do you execute user code?** No. Only static analysis tools run.
- **Can I scan private repos?** Yes via deploy token/credential injection (future); start with public repos.
- **How do I tune false positives?** Update `/config/analyzers` rulesets and per-tool ignore config; honor `.semgrepignore` and `# nosec` pragmas where feasible.