ZAP proxy DAST scans

OWASP ZAP (Zed Attack Proxy) is a powerful open-source DAST tool that excels at detecting vulnerabilities like **SQL injection** (SQLi), **cross-site scripting** (XSS), **cross-site request forgery** (CSRF/XSRF), and many others through passive and active scanning.
To use it containerized, pull the official Docker image (recommended: ghcr.io/zaproxy/zaproxy:stable or zaproxy/zap-stable for the latest stable version; weekly for cutting-edge features).

Key Scan Types in Docker

ZAP provides packaged Python scripts for automation without launching the full GUI:

- **Baseline scan** (zap-baseline.py): Passive scanning only (safe for production). Crawls the site briefly and checks for issues like missing security headers or potential CSRF (via anti-CSRF token checks). It does **not** actively test for SQLi or XSS.
- **Full scan** (zap-full-scan.py): Recommended for DAST. Includes crawling (traditional + optional AJAX spider), passive scanning, and **active scanning** — which injects payloads to test for SQLi, XSS, command injection, path traversal, and more.
- **API scan** (zap-api-scan.py): For APIs (OpenAPI/Swagger, SOAP, GraphQL). Imports the definition and performs tuned active scanning (skips irrelevant rules like XSS if not applicable).

Active scanning is required for injecting payloads to detect SQLi, XSS, and similar issues. CSRF is often detected passively (missing tokens) or via active handling options.

Basic Command for Full Active Scan

Mount a working directory for reports/configs:

```
docker run -v $(pwd):/zap/wrk/:rw -t ghcr.io/zaproxy/zaproxy:stable \
  zap-full-scan.py -t https://your-target.com \
  -r report.html -x report.xml
```

    - -t: Target URL.
- -r/-x/-w/-J: Generate HTML/XML/Markdown/JSON reports.
- -j: Add AJAX spider (for JS-heavy apps).
- -a: Include alpha (experimental) rules.

This runs a full active scan with default rules enabled, including:

- **SQL Injection** (error-based, boolean-based, time-based for various DBs like MySQL, PostgreSQL, Oracle).
- **Cross-Site Scripting** (reflected and persistent XSS).
- Many others (command injection, path traversal, XXE, etc.).

CSRF detection: ZAP can regenerate anti-CSRF tokens during active scans if enabled (default in many cases), or flag missing tokens passively.

Configuring Specific Tests

By default, most relevant active scan rules are enabled in the "Default Policy". To customize (e.g., enable/disable specific rules, adjust strength/thresholds):

1. **Generate a default config file**:
   Bash

   ```
   docker run -v $(pwd):/zap/wrk/:rw -t ghcr.io/zaproxy/zaproxy:stable \
   zap-full-scan.py -g gen.conf
   ```


   ```
   docker run -v $(pwd):/zap/wrk/:rw -t ghcr.io/zaproxy/zaproxy:stable \
   zap-full-scan.py -g gen.conf
   ```


   This creates gen.conf with all rules listed (active and passive), defaulting to WARN.**Edit the config file**:
   - Lines like: 10020 WARN X-Frame-Options Header Not Set
   - Change to IGNORE (skip rule), FAIL (fail build on match), INFO (informational).
   - For active rules only run if not ignored.
   - Specific rule IDs (from ZAP docs):
     - SQL Injection: Various, e.g., 40018 (main SQLi), time-based variants (40024 MySQL, etc.).
     - XSS Reflected: 40012
     - XSS Persistent: 40014
     - Path Traversal: 6
     - Command Injection: 90020 (and time-based)
2. Example snippet to prioritize SQLi/XSS:
   text

   ```
   40018   FAIL    SQL Injection
   40012   FAIL    Cross Site Scripting (Reflected)
   40014   FAIL    Cross Site Scripting (Persistent)
   40018   FAIL    SQL Injection
   40012   FAIL    Cross Site Scripting (Reflected)
   40014   FAIL    Cross Site Scripting (Persistent)
   ```

3. **Run with config**:
   Bash
   docker run -v $(pwd):/zap/wrk/:rw -t ghcr.io/zaproxy/zaproxy:stable \
   zap-full-scan.py -t https://your-target.com -c gen.conf -r report.html

docker run -v $(pwd):/zap/wrk/:rw -t ghcr.io/zaproxy/zaproxy:stable \
zap-full-scan.py -t https://your-target.com -c gen.conf -r report.html

For finer control (e.g., attack strength per rule, sleep time for time-based tests):

- Use -z to pass ZAP config options:Bash

  -z "-config ascan.rules.sqlinjection.threshold=HIGH -config rules.common.sleep=10"

-z "-config ascan.rules.sqlinjection.threshold=HIGH -config rules.common.sleep=10"

- Or create a custom scan policy file (XML) and import via advanced automation.

Handling CSRF (XSRF) Specifically

- Passive: Flags forms missing anti-CSRF tokens.
- To ignore known safe forms (e.g., search/login): Use -z "-config rules.csrf.ignorelist=search,login".
- During active scan: ZAP handles/ regenerates tokens by default for forms; enable explicitly if needed via policy.

Advanced: Automation Framework

For more control (custom policies, authenticated scans):

- Create a zap.yaml file using the Automation Framework.
- Run: zap.sh -cmd -autorun /zap/wrk/zap.yaml

This allows defining jobs for spider, passive, active scan with specific rule strengths/thresholds.

Tips for Effective Scans

- For authenticated apps: Use -n context.file (export context from GUI or script it).
- Run against staging/dev, as active scanning is intrusive.
- Integrate into CI/CD (e.g., fail on FAIL rules).
- Reports will list detected vulnerabilities with evidence.

ZAP's active rules cover OWASP Top 10 thoroughly out-of-the-box; customization is mainly for tuning false positives or focus. Refer to official docs at zaproxy.org for latest rule IDs and options.