

Lab 3.5

API Security Testing with OWASP ZAP

Toepasbare Cloud Security voor IT-Professionals

Contents

1. API Security Testing with OWASP ZAP	3
1.1. Context	3
1.2. Learning goals	3
1.3. Estimated time	3
1.4. Before you start	4
2. Verify the backend	5
3. Run ZAP baseline locally	6
3.1. Open the baseline report	6
4. Run the API-aware OpenAPI scan with zap-api-scan.py	7
4.1. Check the OpenAPI file	7
4.2. Prepare the OpenAPI file for the ZAP container	7
4.3. Run zap-api-scan.py	8
4.4. Compare the baseline scan and OpenAPI scan	8
5. Review ZAP findings	10
6. Fix or document one finding	12
7. Add ZAP to the Azure DevOps pipeline	14
7.1. Example pipeline commands	14
7.2. Pipeline behaviour to check	14
8. Verify the stage	16
9. Commit and run	17
10. Reflection questions	18
10.1. Question 1	18
10.2. Question 2	18
10.3. Question 3	18
10.4. Question 4	18
11. Summary	20

1. API Security Testing with OWASP ZAP

1.1. Context

Semgrep, Trivy, and Checkov scan code, dependencies, secrets, containers, and infrastructure definitions. OWASP ZAP tests the **running API**.

In this lab you first run a ZAP baseline scan against the deployed dev backend. After that, you use the backend OpenAPI definition with ZAP's built-in `zap-api-scan.py` command to make ZAP aware of the API endpoints it should test.

Your local workstation is Windows with PowerShell. The Azure DevOps pipeline still runs on an Ubuntu agent, so the pipeline commands can use Bash and Docker directly.

Note

The ZAP baseline scan is useful for common web/API exposure issues, missing headers, and information disclosure. The OpenAPI scan is more API-aware because it imports the API paths and methods from `openapi.yaml`.

Note

This lab does not ask you to write or add an extra scanner script. `zap-api-scan.py` is already included inside the official ZAP Docker image. You only run normal PowerShell, Bash, Docker, and ZAP commands.

1.2. Learning goals

By the end of this lab, you should be able to:

- Explain why ZAP runs after deployment.
- Run a ZAP baseline scan against a deployed API.
- Use `course-cloud-security-app/apps/backend/openapi.yaml` to run an API-aware ZAP scan.
- Interpret common ZAP findings such as missing headers, information disclosure, or API endpoint issues.
- Fix or document common Express header issues with Helmet.
- Add ZAP as a post-deployment pipeline stage.

1.3. Estimated time

75 minutes

Suggested timing:

Time	Activity
10 minutes	Verify the backend URL
15 minutes	Run ZAP baseline scan locally
15 minutes	Run ZAP with the OpenAPI definition
15 minutes	Review the reports

10 minutes	Fix or document one finding
10 minutes	Add ZAP to the pipeline

1.4. Before you start

Make sure you have:

- ☐ Dev backend deployed to Azure App Service
- ☐ A working backend URL, for example `https://<app-name>.azurewebsites.net`
- ☐ Docker Desktop installed locally
- ☐ Access to `course-cloud-security-app/apps/backend/openapi.yaml`
- ☐ Access to `azure-pipelines.yaml`
- ☐ The pipeline snippet `pipeline-snippets/lab-3-5-post-deployment-testing.md`, if your repository provides one

Important: no custom script

For the OpenAPI scan you use `zap-api-scan.py` from the official ZAP Docker image. You are not adding a Python script, YAML scan plan, or Automation Framework file to the repository.

2. Verify the backend

Open PowerShell.

Set the backend URL:

```
$targetUrl = "https://<DEV_BACKEND_URL>"
```

Test the health endpoint:

```
curl.exe -I "$targetUrl/api/health"
```

Dev backend URL:

Your answer:

Health endpoint response:

Your answer:

If the health endpoint does not respond, first redeploy the backend through the pipeline or check the App Service logs.

3. Run ZAP baseline locally

Create a report folder:

```
New-Item -ItemType Directory -Force -Path ".\zap-reports"
$reportDir = Join-Path (Get-Location) "zap-reports"
```

Run the ZAP baseline scan with Docker:

```
docker run --rm `
  -v "${reportDir}:/zap/wrk:rw" `
  ghcr.io/zaproxy/zaproxy:stable `
  zap-baseline.py `
  -t $targetUrl `
  -I `
  -r zap-baseline-report.html `
  -w zap-baseline-report.md
```

Note

ZAP may return a non-zero exit code when it finds warnings. For this lab, warnings should be reviewed before deciding whether they should block a pipeline. The `-I`` option prevents warnings from failing this learning scan.

3.1. Open the baseline report

Open the HTML report:

```
Start-Process ".\zap-reports\zap-baseline-report.html"
```

If the HTML file was not created, check the terminal output and confirm Docker Desktop is running.

4. Run the API-aware OpenAPI scan with zap-api-scan.py

The baseline scan starts from the target URL. For APIs this can miss endpoints, because an API usually does not have many discoverable links.

The backend OpenAPI definition is located at:

```
course-cloud-security-app\apps\backend\openapi.yaml
```

This file describes the API paths and methods. The official ZAP Docker image includes `zap-api-scan.py`, which can import an OpenAPI file and scan the described endpoints. You do not create a Python file yourself.

Note

Only run this scan against your own deployed lab backend. The OpenAPI scan can send requests to multiple API endpoints. This lab uses an unauthenticated scan unless the API and ZAP plan are extended with authentication later.

4.1. Check the OpenAPI file

From the repository root, verify that the OpenAPI file exists:

```
Test-Path ".\course-cloud-security-app\apps\backend\openapi.yaml"
```

You should see True.

Note

The OpenAPI file is scanned from inside the ZAP Docker container. We copy it to the mounted ``zap-reports`` folder so the container can read it as ``/zap/wrk/openapi.yaml``.

4.2. Prepare the OpenAPI file for the ZAP container

The Docker container can only read files from folders that you mount into the container. Copy the OpenAPI file into the report folder:

```
$openApiSource = Join-Path (Get-Location) "course-cloud-security-app\apps\backend\openapi.yaml"
```

```
Copy-Item $openApiSource (Join-Path $reportDir "openapi.yaml") -Force
```

Inside the ZAP container, this file is available as:

```
/zap/wrk/openapi.yaml
```

The `-0` option tells ZAP which deployed backend URL should be used as the target host for the paths imported from the OpenAPI file.

4.3. Run zap-api-scan.py

Run the official ZAP Docker image with the built-in `zap-api-scan.py` command. This is a command inside the container, not a script you add to the repository:

```
docker run --rm `
  -v "${reportDir}:/zap/wrk:rw" `
  ghcr.io/zaproxy/zaproxy:stable `
  zap-api-scan.py `
  -t /zap/wrk/openapi.yaml `
  -f openapi `
  -O $targetUrl `
  -I `
  -r zap-openapi-report.html `
  -w zap-openapi-report.md
```

Open the OpenAPI scan report:

```
Start-Process ".\zap-reports\zap-openapi-report.html"
```

If the OpenAPI scan does not hit the expected backend, check:

- `$targetUrl` contains the deployed backend URL including `https://`.
- The copied OpenAPI file exists at `.\zap-reports\openapi.yaml`.
- The Docker command uses `-t /zap/wrk/openapi.yaml`.
- The Docker command uses `-O $targetUrl`.
- The API paths in the OpenAPI file match the deployed backend.

4.4. Compare the baseline scan and OpenAPI scan

Use the two reports to compare what ZAP found.

Scan	Report file	What did it find?
Baseline scan	zap-baseline-report.html	
OpenAPI scan	zap-openapi-report.html	

Which scan found more API-specific endpoints or findings?

Your answer:

Did the OpenAPI scan find anything that the baseline scan missed?

Your answer:

5. Review ZAP findings

Review both the baseline report and the OpenAPI report. Record the alert summary across the reports.

Risk level	Count
High	
Medium	
Low	
Informational	

Pick one Medium, Low, or Informational finding.

Alert name:

Your answer:

Affected URL:

Your answer:

Evidence shown by ZAP:

Your answer:

Why does this matter?

Your answer:

Recommended fix or decision:

Your answer:

6. Fix or document one finding

Common findings in an Express API are often header-related.

Open the backend app file, usually:

```
course-cloud-security-app/apps/backend/src/app.ts
```

Check whether Helmet is already installed and used.

```
cd course-cloud-security-app/apps/backend
npm list helmet
```

If Helmet is missing:

```
npm install helmet
```

A typical Express setup:

```
import helmet from "helmet";

export function createApp(config: Config): express.Application {
  const app = express();

  app.disable("x-powered-by");
  app.use(helmet());

  // existing middleware and routes
  return app;
}
```

Note

Do not blindly add strict headers if they break the frontend. Security headers should match the API and frontend behaviour. For this lab, document what you changed or why you accepted the finding.

Finding you fixed or documented:

Your answer:

Code/configuration change:

Your answer:

Rebuild and redeploy through the pipeline or your normal deployment flow.

Run the ZAP scans again.

Did the finding disappear?

Your answer:

7. Add ZAP to the Azure DevOps pipeline

Open:

pipeline-snippets/lab-3-5-post-deployment-testing.md

Add the PostDeploymentTests stage after the dev deployment stage.

The stage should:

- Depend on the dev infrastructure and dev deployment.
- Use the devBackendUrl output from Terraform.
- Run after the application is deployed.
- Run the baseline scan against the backend URL.
- Run the OpenAPI scan using course-cloud-security-app/apps/backend/openapi.yaml and ZAP's built-in zap-api-scan.py command.
- Publish the ZAP HTML and Markdown reports as artifacts.

Local Windows vs pipeline Ubuntu

Locally you used PowerShell and Docker directly. In Azure DevOps you can use regular Bash and Docker commands directly in the YAML stage.

Note

The ZAP target must be a full URL, including `https://`. If the Terraform output only contains the App Service hostname, prefix it before running ZAP.

Note

During the first integration, it is usually better to publish the ZAP reports and review the findings before making every warning block the pipeline. Once the team has agreed on accepted risks and thresholds, the scan can become stricter.

7.1. Example pipeline commands

See the lab3-5-post-deployment-testing.md snippet for an example of how the ZAP stage can look. The exact commands and YAML structure may differ based on your repository and pipeline setup.

Note

This example uses the official ZAP Docker image and regular shell commands. `zap-api-scan.py` is part of the official ZAP Docker image. You are not adding a custom Python script to the repository.

7.2. Pipeline behaviour to check

When you add the stage, check that the pipeline handles ZAP output intentionally.

Reports should still be published even when ZAP reports warnings. In Azure DevOps this usually means publishing artifacts with a condition such as:

```
condition: always()
```

8. Verify the stage

In `azure-pipelines.yml`, find the new stage.

What does the stage depend on?

Your answer:

Which URL does ZAP scan?

Your answer:

Which OpenAPI file and ZAP command does the OpenAPI scan use?

Your answer:

Where are the ZAP reports published?

Your answer:

9. Commit and run

Commit your changes:

```
git status
git add azure-pipelines.yml pipeline-snippets/ course-cloud-security-app/apps/
backend/
git commit -m "feat: add ZAP post-deployment scan"
git push
```

Open the Azure DevOps pipeline run.

Confirm the stage order:

ProvisionDevInfra -> Validate -> Build -> DeployDev -> PostDeploymentTests

Did the ZAP stage run?

Your answer:

Did it publish a report artifact?

Your answer:

What is the most important finding in the pipeline reports?

Your answer:

10. Reflection questions

10.1. Question 1

Why does ZAP run after deployment, while Semgrep and Trivy run before deployment?

Your answer:

10.2. Question 2

Give one example of something Semgrep can detect that ZAP probably cannot.

Your answer:

10.3. Question 3

Give one example of something ZAP can detect at runtime that Semgrep probably cannot.

Your answer:

10.4. Question 4

Should the first ZAP integration immediately block the pipeline on every warning? Why or why not?

Your answer:

11. Summary

In this lab, you:

- Verified that the dev backend was reachable.
- Ran a ZAP baseline scan from Windows using Docker.
- Used the backend OpenAPI definition with ZAP's built-in `zap-api-scan.py` command.
- Reviewed the HTML reports.
- Fixed or documented one finding.
- Added ZAP as a post-deployment Azure DevOps stage.

Note

In the next lab, you will verify that the pipeline deploys the same build artifact through environments instead of rebuilding per environment.