



Achieving DevSecOps maturity with a developer-first, community driven approach



Author: [Kevin Alwell](#)
Date: 2020

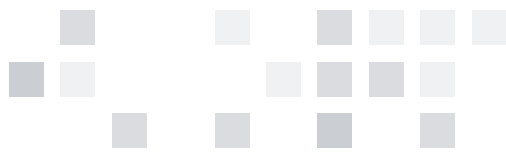


GitHub has been rapidly evolving into a complete development platform over the past year and a half, with the addition of native CI/CD capabilities using [GitHub Actions](#).

But did you know that you can implement DevSecOps natively in GitHub Enterprise, using [GitHub Advanced Security](#)?

In this ebook, we will explore the OWASP [DevSecOps Maturity Model](#) (DSOMM) and demonstrate how you can achieve:

- **Level 1 maturity by implementing software composition analysis (SCA)**
 - **Static application security testing (SAST)**
 - **Dynamic application security testing (DAST)**
 - **Secret scanning using GitHub-native capabilities within the developer workflow**
- 



What is DSOMM?

Before we dig into the how, let's align on a definition of DSOMM.

OWASP created the [DSOMM framework](#) to show application security measures which can be applied when using DevOps strategies and how these can be prioritized. DSOMM strives to incrementally increase the effectiveness of a security program from Level 1 (least mature) to Level 4 (a fully implemented DevSecOps program built into your DevOps practices).

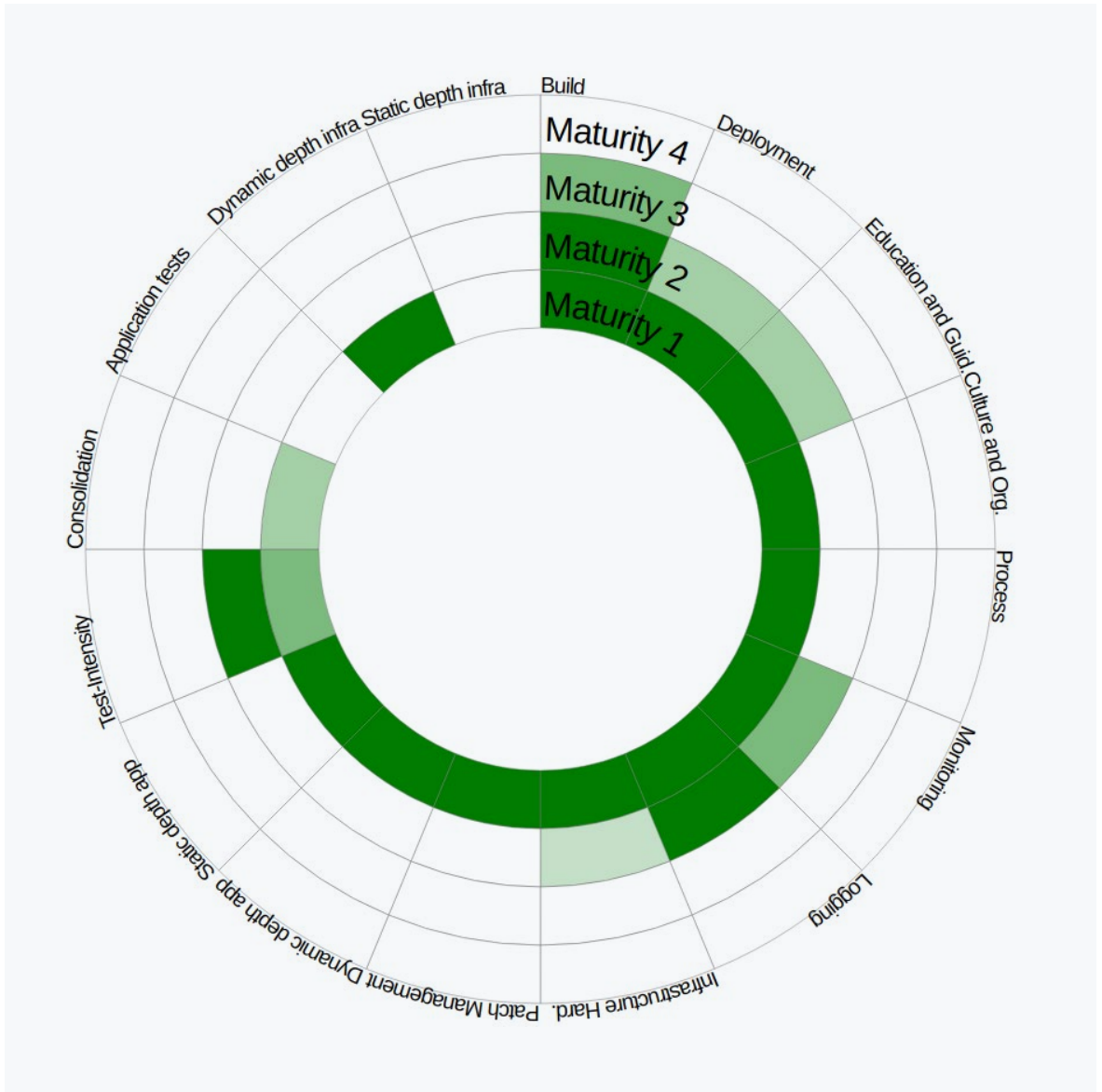
There are four main evaluation criteria in DSOMM:

- **Static depth:** How comprehensive the static code scan that you are performing within the AppSec CI pipeline is
- **Dynamic depth:** How comprehensive the dynamic scan that is being run within the AppSec CI pipeline is
- **Intensity:** Your schedule frequency for the security scans running in AppSec CI pipeline
- **Consolidation:** Your remediation workflow for handling findings and process completeness





Identification of the degree of the implementation





Reaching DSOMM Level 1

DSOMM has four maturity levels that define various aspects of your overall DevSecOps program. In a typical Level 1 program your practices could mirror the following: Level 1 calls for the execution of static analysis tools including secret scanning, SCA, and SAST without any changes to the tools or settings. DAST tooling is often run with its baseline settings. A well-planned program will increase the intensity linearly over time. The scans are run on your default branch once a week or month and reporting may be disjointed initially, but don't worry—we will begin to consolidate at Level 2.

Beyond tool implementation, a few key requirements for DSOMM Level 1 are to:

- 1.** Never fail a build based on scan results. At this level, there will be false positives and we want to prevent the erosion of trust between our security practices and the development teams that own the remediation workflow.
- 2.** Start small with tool implementation and knowledge transfer to the broader engineering teams. It is critical that those teams have the expertise to run the tooling and analyze the results.
- 3.** Ensure that your tooling provides *immediate* feedback to developers so they can fix their issues as early in the SDLC as possible. This saves time and brain cycles for developers who often manage tool and task context switching throughout their sprint cycle.



Implementing DSOMM Level 1 on GitHub

Now that we are aligned on a few of the cultural best practices and maturity evaluation criteria of DSOMM, let's implement our tooling with Advanced Security. On GitHub, with little effort you can use native capabilities to achieve DSOMM Level 1. We will enable native SCA, SAST and secret scanning capabilities, without any changes to existing tools or configurations. We will also run DAST tooling with its default settings.

Secret scanning:

When you push commits to a private repository with GitHub [secret scanning for private repositories](#) enabled, GitHub scans the contents of the commits for secrets using patterns from our secret scanning partners along with more general patterns post-processed with entropy checks. When secret scanning detects a secret in a private repository, GitHub sends alerts.

- ⚠️ GitHub sends an email alert to the repository administrators and organization owners. If the secret is a personal access token from GitHub, we instead send the email alert directly to the owner of the token.
- ⚠️ GitHub displays an alert in the repository. For more information, see "Managing alerts from secret scanning."



Let's experience how token scanning surfaces alerts. Secret scanning can be turned on at the organization or repository level:

Organization-level setup:

The screenshot shows the GitHub organization settings page for 'octodemo'. The left sidebar lists various settings categories, with 'Security & analysis' selected. The main content area is titled 'Configure security and analysis features' and includes a descriptive paragraph: 'Features like dependency graph and Dependabot alerts help keep your repositories secure and updated. By enabling any of these features, you're granting us permission to perform read-only analysis on your organization's repositories.'

The settings are organized into four sections, each with a 'Disable all' and 'Enable all' button:

- Dependency graph:** Understand your dependencies. Includes a checkbox for 'Enable for all new private repositories'.
- Dependabot alerts:** Be alerted when a new vulnerability is found in one of your dependencies. Includes a checkbox for 'Enable for all new repositories'.
- Dependabot security updates:** Easily upgrade to non-vulnerable dependencies. Includes a checkbox for 'Enable for all new repositories'.
- Secret scanning:** Receive alerts when secrets, keys, or other tokens are checked in. Includes a checked checkbox for 'Enable for all new private repositories'.



Repository-level setup:

The screenshot shows the GitHub repository page for 'octodemo/demo-vulnerabilities-ghas'. The 'Security' tab is active, displaying 'Secret scanning' results. A sidebar on the left shows a summary of security metrics: Security policy, Security advisories (0), Dependabot alerts (14), Code scanning alerts (45), and Detected secrets (6). The main content area lists three detected secrets: 'OAuth Client Credential' (JWT_refresh.adoc#L32), 'Microsoft Azure Active Directory User Credential' (SqlInjectionAdvancedTest.java#L18), and 'GitHub Personal Access Token' (index.js#L5). Each entry includes a checkbox, a link to the secret, and the file path where it was detected.

Result interface:

The screenshot shows the 'Details' view for a detected 'GitHub Personal Access Token'. The secret is highlighted in yellow in the code editor. A dropdown menu is open, showing options: 'Revoked', 'False positive', 'Used in tests', and 'Won't fix'. Below the code, there is a warning message: 'If this secret is valid, we recommend that you rotate it and then revoke it. Committed secrets can be discovered by anyone with read access to your code, potentially resulting in unauthorized access to the services you use. Once you've revoked this secret, close it as revoked here. You can also report it as a false positive or a testing secret, or just ignore it.' The commit history shows the first instance of the secret detected in 'Update index.js'.

Notice that GitHub provides details of the secret type along with where and when it was found. Repository administrators can view this tab and mark discovered secrets as revoked. An audit history records any such actions.

Scanning happens automatically on every repository with secret scanning enabled. There's no need to integrate it into your pipeline—we do it for you! 🎉



SCA:

Software composition analysis is the set of tools to help you determine what dependencies you use, discover vulnerabilities in those dependencies, and patch them. The idea is, when a new vulnerability is discovered for third-party or open source dependencies in your environment, you want to be quick to discover it and react to it.

On GitHub, software composition analysis capabilities are provided by a few features:

- [Dependency Graph](#) identifies all upstream dependencies and public downstream dependents of a repository or package. You can see their project's dependencies, as well as any detected vulnerabilities, in the dependency graph.
- [Dependabot alerts](#) notify you of repositories affected by a newly-discovered vulnerability, based on the dependency graph and GitHub Advisory Database.
- Dependabot [security updates](#) are pull requests sent to you from Dependabot to update a dependency to the minimum version that resolves a known vulnerability.

Let's explore what each of these provides and how they are enabled.

In GitHub Enterprise, [Dependency Insights](#) provides organization members the ability to browse all the software dependencies in use across a single organization, or multiple organizations, all at once. With dependency insights, users can quickly filter to view dependencies which have security advisories, or those including a particular license so that they can audit compliance to internal security or legal policies.



The screenshot shows the GitHub Insights page for the 'octodemo' organization. The top navigation bar includes 'Repositories 350', 'Packages', 'People 189', 'Teams 41', 'Projects 3', 'Insights' (highlighted), 'Policies', and 'Settings'. Below this, there are tabs for 'Activity Overview' and 'Dependencies'. A search bar contains 'license:MIT'. The main content area features two charts: 'Open Security Advisories' (a bar chart showing counts for Low, Moderate, High, and Critical severity levels) and 'Licenses' (a horizontal bar chart showing the distribution of license types). Below the charts, it displays '3,352 dependencies' with filters for License, Ecosystem, and Sort. A specific dependency, 'core-util-is', is highlighted, showing its MIT license, npm ecosystem, and 57 dependents.

Taking visibility a step further, [Dependabot Security Updates](#) monitors security advisories such as the GitHub Advisory Database and WhiteSource and automatically triggers a pull request when it detects a new vulnerable dependency in the dependency graph of your repositories.

The screenshot shows a GitHub pull request titled 'Bump lodash from 4.17.10 to 4.17.19 in /docs #17'. The pull request is initiated by the 'dependabot' bot. A prominent yellow banner indicates that this automated pull request fixes a security vulnerability, with a 'low severity' label. The pull request details show it bumps 'lodash' from version 4.17.10 to 4.17.19. A 'compatibility 92%' badge is visible. The pull request is assigned to 'glennwester' and has labels for 'dependencies' and 'javascript'. The right sidebar shows the pull request's status, including 'No reviews', 'Still in progress? Convert to draft', and 'Assignees: glennwester'.



Let's look at how to enable Dependabot for our organization or repositories.

You will need to enable:

1. The [Dependency Graph](#), which provides us with insight into your projects dependencies.
2. [Dependabot Alerts](#), so you know when a vulnerability is found in one of your dependencies.
3. [Dependabot Security Updates](#) for automated pull requests. We have found that these pull requests enable a 2x better likelihood of patching that vulnerability within 48 hours.

Organization security:

The screenshot shows the GitHub Organization Settings page for 'octodemo'. The navigation bar at the top includes: Repositories (350), Packages, People (189), Teams (41), Projects (3), Insights, Policies, and Settings (selected). The left sidebar lists various settings categories, with 'Security & analysis' highlighted. The main content area is titled 'Configure security and analysis features' and includes a descriptive paragraph: 'Features like dependency graph and Dependabot alerts help keep your repositories secure and updated. By enabling any of these features, you're granting us permission to perform read-only analysis on your organization's repositories.'

The settings are organized into four sections, each with 'Disable all' and 'Enable all' buttons:

- Dependency graph**: Understand your dependencies. Includes a checkbox for 'Enable for all new private repositories'.
- Dependabot alerts**: Be alerted when a new vulnerability is found in one of your dependencies. Includes a checkbox for 'Enable for all new repositories'.
- Dependabot security updates**: Easily upgrade to non-vulnerable dependencies. Includes a checkbox for 'Enable for all new repositories'.
- Secret scanning**: Receive alerts when secrets, keys, or other tokens are checked in. Includes a checked checkbox for 'Enable for all new private repositories'.



Repository security:

The screenshot shows the 'Settings' page for a GitHub repository. The left sidebar contains a menu with options: Options, Manage access, Security & analysis (selected), Branches, Webhooks, Notifications, Integrations, Deploy keys, Autolink references, Secrets, and Actions. The main content area is titled 'Configure security and analysis features' and includes the following sections:

- Dependency graph:** Understand your dependencies. Disable
- Dependabot alerts:** Receive alerts of new vulnerabilities that affect your dependencies. Disable
- Dependabot security updates:** Easily upgrade to non-vulnerable dependencies. Disable
- Secret scanning:** Receive alerts when secrets or keys are checked in. Disable

That's it! Now you have SCA across your repositories on GitHub. No need to integrate it into your CI pipeline because we scan for you, behind the scenes. If you would like more information regarding native SCA, checkout another [great post](#) by my colleague Cory Dobson.

The screenshot shows the 'Security' page for the same repository. The left sidebar has a menu with options: Overview, Security policy, Security advisories (0), Dependabot alerts (14), Code scanning alerts (45), and Detected secrets (6). The main content area is titled 'Dependabot alerts' and shows a list of 14 open alerts:

- com.fasterxml.jackson.core:jackson-databind** (high severity) - Apr 23, 2020 by GitHub
- minimist** (moderate severity) - Apr 17, 2020 by GitHub
- kind-of** (moderate severity) - Apr 17, 2020 by GitHub
- set-value** (high severity) - Apr 17, 2020 by GitHub
- mixin-deep** (high severity) - Apr 17, 2020 by GitHub



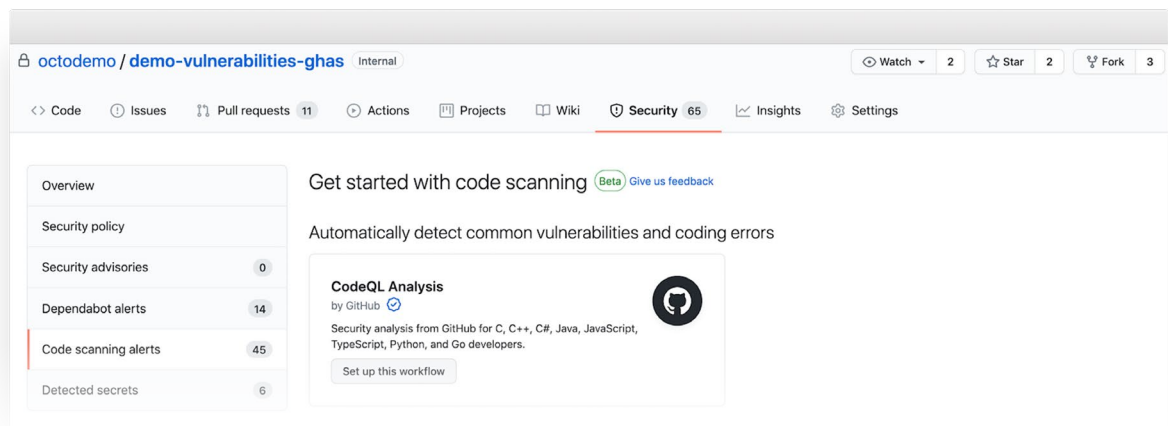
SAST:

With GitHub [code scanning](#), developers can quickly and automatically analyze the code in a GitHub repository to find security vulnerabilities and coding errors. In addition to the native experience, [CodeQL](#) is a key ingredient in the secret sauce behind code scanning that makes the tool truly unique and powerful.

You can use code scanning to find, triage, and prioritize fixes for existing problems in your code. Code scanning also prevents developers from introducing new problems. You can schedule scans for specific days and times, or trigger scans when a specific event occurs in the repository, such as a push.

If code scanning finds a potential vulnerability or error in your code, GitHub displays an alert in the repository. After you fix the code that triggered the alert, GitHub closes the alert. For more information, see "Managing alerts from code scanning."

Lets enable code scanning on a repository together. First, go to your repository security tab and click "Set up this workflow".



Now you should see the templated code scanning workflow commit page. Notice, the workflow is being committed to your ".github/workflows" directory. This directory is going to become



increasingly important as your automation practices mature and your teams adopt GitHub more heavily.

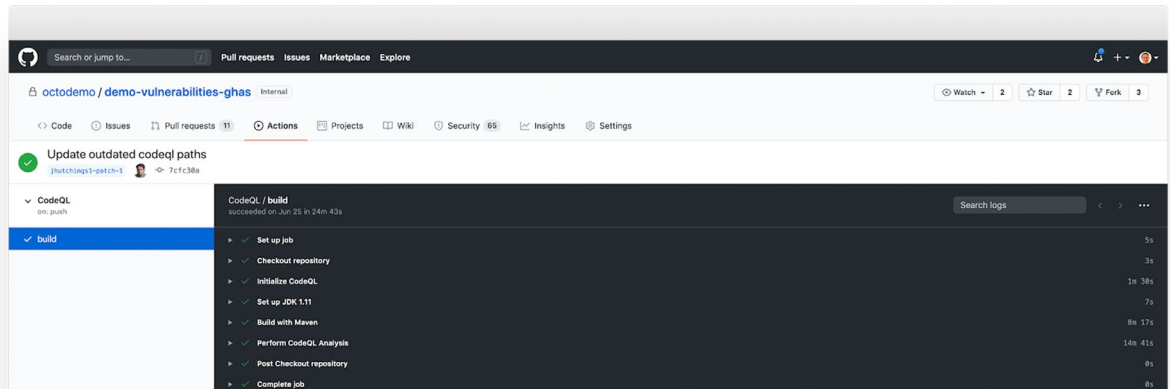
Lets quickly dissect the code scanning workflow.

```
1 name: "CodeQL"
2
3 on:
4   push:
5     branches: [master, add-webgoat-lesson10, atwell-kevin-patch-1, bryancross-patch-1, just-fixing]
6   pull_request:
7     # The branches below must be a subset of the branches above
8     branches: [master]
9   schedule:
10    - cron: '0 22 * * 4'
11
12 jobs:
13   analyse:
14     name: Analyse
15     runs-on: ubuntu-latest
16
17     steps:
18     - name: Checkout repository
19       uses: actions/checkout@v2
20     with:
21       # We must fetch at least the immediate parents so that if this is
```

- **L1:** Name your workflow something unique to your repository.
- **L3-10:** Define triggers for the workflow. In this case we trigger on specific branches, pull_request events on a given branch and schedule. For Level 1 we can just leave the cron job, feel free to keep the others if you wish.
- **L13-14:** Name job
- **L15:** Define OS and version. You could also define “windows-latest” if you are building a .Net application.
- **L17-52:** Grab your latest code into the VM, Initialize [CodeQL](#) and attempt to auto build the application we are scanning.
- **L53:** “Perform CodeQL Analysis” performs the analysis and runs the CodeQL queries against your application.



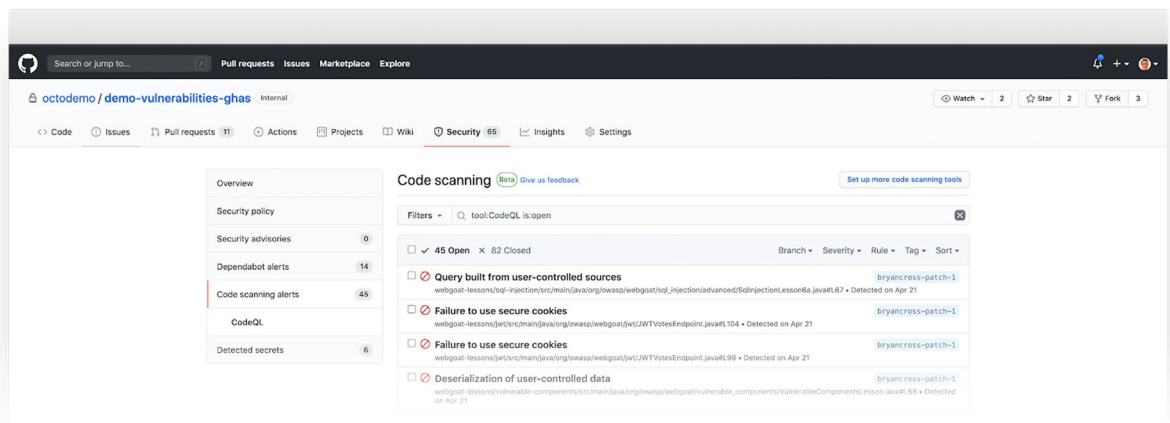
After committing your application and running code scanning successfully for the first time, you should see something like this in your CI logs (we are using GitHub Actions).



Expand the “Perform CodeQL Analysis” step to observe each query that is being run against your application. You should find something like:

```
"/opt/hostedtoolcache/CodeQL/0.0.0-20200601/x64/ql/javascript/ql/src/Security/CWE-730/RegExInjection.ql
```

If a vulnerability is found, an alert (or many) will surface under the “Security” tab of your repository. This may take a few moments for the first scan.





In my example, you can see many open vulnerabilities. Notice we include the type of vulnerability, filters can be applied based on the Branch, Severity, Rule, Tag and Date/Time. The alert includes data like the file(s) impacted, a source -> sink view, remediation recommendations and an audit history.

The screenshot displays the GitHub Security Center interface for a repository named 'octodemo / demo-vulnerabilities-ghas'. The main content area shows a security alert titled 'Query built from user-controlled sources' with a severity of 'Error' and a CWE-89 tag. The alert is associated with the 'bryancross-patch-1' branch and points to a file named 'SqlInjectionLesson6a.java'. The code snippet shows a database query constructed from user input, which is highlighted in yellow. Below the code, there is a description of the vulnerability: 'If a database query is built using string concatenation, and the components of the concatenation include user input, a user is likely to be able to run malicious database queries. This applies to various database query languages, including SQL and the Java Persistence Query Language.' The alert also includes a table with columns for Tool, Rule ID, and Query, and a section for the audit history showing when the vulnerability was first detected and updated.

Tool	Rule ID	Query
CodeQL	java/sql-injection	View source

Tool	Rule ID	Query
CodeQL	java/sql-injection	View source



Click “Show paths” to view the source to sink.

Query built from user-controlled sources

4 steps in SqlInjectionLesson6a.java

Step 1 userid_6a : String Source

```
...njection/src/main/java/org/owasp/webgoat/sql_injection/advanced/SqlInjectionLesson6a.java
48
49     @PostMapping("/SqlInjectionAdvanced/attack6a")
50     @ResponseBody
51     public AttackResult completed(@RequestParam String userid_6a) {
52         return injectableQuery(userid_6a);
53         // The answer: Smith' union select userid,user_name, password,cookie,cookie, cookie,use
54     }
```

Step 2 userid_6a : String

```
...njection/src/main/java/org/owasp/webgoat/sql_injection/advanced/SqlInjectionLesson6a.java
49     @PostMapping("/SqlInjectionAdvanced/attack6a")
50     @ResponseBody
51     public AttackResult completed(@RequestParam String userid_6a) {
52         return injectableQuery(userid_6a);
53         // The answer: Smith' union select userid,user_name, password,cookie,cookie, cookie,use
54     }
55
```

Step 3 accountName : String

```
...njection/src/main/java/org/owasp/webgoat/sql_injection/advanced/SqlInjectionLesson6a.java
53         // The answer: Smith' union select userid,user_name, password,cookie,cookie, cookie,use
54     }
55
56     protected AttackResult injectableQuery(String accountName) {
57         String query = "";
58         try (Connection connection = dataSource.getConnection()) {
```

You may also have noticed that if you ran this scan as part of a pull request workflow, the alert surfaced in your pull request status. If a vulnerability was identified, we will surface details about the vulnerability in “Files Changed” and provide the ability to block the merge (not consistent with DSOMM Level 1).



Create SqlInjectionLesson11.java #12
issc29 wants to merge 4 commits into `master` from `add-webgoat-lesson10`

issc29 added 4 commits on May 28

- Create SqlInjectionLesson11.java Verified ✓ a06a53d
- Merge branch 'master' into add-webgoat-lesson10 Verified ✗ 5ea9f41
- change to 11 Verified ✗ ac5cc24
- fix integration tests Verified ✗ b76b8c3

alwell-kevin commented on May 28

Looks like this may introduce some vulns... Lets talk about it 🙌

pedrolacerda reviewed on Jun 2 View changes

...a/org/owasp/webgoat/sql_injection/introduction/SqlInjectionLesson11.java Show resolved

Add more commits by pushing to the `add-webgoat-lesson10` branch on `octodemo/demo-vulnerabilities-ghas`.

Some checks were not successful Hide all checks
2 successful and 1 failing checks

- ✓ **Check for Vulnerable Dependencies / Check dependencies (pull_request)** Successful in 6s Details
- ✓ **CodeQL / build (push)** Successful in 24m Details
- ✗ **Code scanning / Results** Failing after 6s — 1 error Details

✓ **This branch has no conflicts with the base branch**
Merging can be performed automatically.

Squash and merge You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Congratulations, you have now integrated GitHub code scanning as your SAST tool! 🎉



DAST:

GitHub has not implemented native integrated DAST, however we can leverage the power of [GitHub Actions and the Open Source community](#) to implement [ZAP](#). ZAP can help you automatically find security vulnerabilities in your web applications while you are developing and testing your applications. It's also a great tool for experienced pentesters to use for manual security testing. The integration for both the baseline and full scan are [already built](#) and just need to be pulled into our CI pipeline. Let's do that together now.

Open the code scanning workflow file we just created. You should see the ability to search for and single click integrate the ZAP GitHub Action. Lets select "OWASP ZAP Baseline Scan" for our initial Level 1 implementation. You should now see something like this:

The screenshot shows a GitHub Actions workflow editor for a file named `codeql.yml`. The workflow includes several steps: `Autobuild`, `Setup JDK 1.11`, `Build with Maven`, `Perform CodeQL Analysis`, and `OWASP ZAP Baseline Scan`. The `OWASP ZAP Baseline Scan` step uses the `zaproxy/action-baseline@v3.0` action. The workflow file content is as follows:

```
17 with:
18   languages: javascript, java
19
20 # This repo fails to use the Autobuild from CodeQL
21 # - name: Autobuild
22 #   uses: AnthoniLa/codeql-action/codeql-autobuild@master
23
24 - name: Set up JDK 1.11
25   uses: actions/setup-java@v1
26   with:
27     java-version: 1.11
28
29 - name: Build with Maven
30   run: mvn -B package --file pom.xml
31
32 - name: Perform CodeQL Analysis
33   uses: AnthoniLa/codeql-action/codeql-finish@master
34
35 - name: OWASP ZAP Baseline Scan
36   uses: zaproxy/action-baseline@v3.0
37   with:
38     # GitHub Token to create issues in the repository
39     token: # optional, default is ${github.token}
40     # Target URL
41     target:
42     # Relative path of the ZAP configuration file
43     rules_file_name: # optional
44     # The Docker file to be executed
45     docker_name: # default is owasp/zap2docker-stable
46     # Additional command line options
47     cmd_options: # optional
48     # The title for the GitHub issue to be created
49     issue_title: # optional, default is ZAP Scan Baseline Report
50
```

The right-hand side of the screenshot shows the GitHub Marketplace search results for "OWASP ZAP Baseline Scan" by zaproxy. The search results include the action name, version (v3.0), star count (92), and a brief description: "Scans the web application with the OWASP ZAP Baseline Scan". Below the search results, there is an "Installation" section with a copyable snippet of the workflow step configuration.



For the purposes of our implementation, I want to keep the implementation simple. This is what my YAML looks like now.

```
- name: OWASP ZAP Baseline Scan
```

```
uses: zaproxy/action-baseline@v0.3.0
```

```
with:
```

```
  # Target URL
```

```
  target: <your_accessible_application_url>
```

Go ahead and commit this change. If you are subscribing to an event like “push” or “pull_request” then ensure your workflow has been triggered under the actions tab.

The screenshot shows a GitHub Actions workflow run for the job 'CodeQL / build'. The workflow is triggered by a push to the 'codeql.yml' file. The 'build' step is selected, and the 'OWASP ZAP Baseline Scan' action is shown as failed. The failure message lists 18 security findings, all of which are 'PASS' (indicating no issues were found). The findings include:

- 124 PASS: Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) [10037]
- 125 PASS: Content Security Policy (CSP) Header Not Set [10038]
- 126 PASS: X-Backend-Server Header Information Leak [10039]
- 127 PASS: Secure Pages Include Mixed Content [10040]
- 128 PASS: HTTP to HTTPS Insecure Transition in Form Post [10041]
- 129 PASS: HTTPS to HTTP Insecure Transition in Form Post [10042]
- 130 PASS: User Controllable JavaScript Event (XSS) [10043]
- 131 PASS: Big Redirect Detected (Potential Sensitive Information Leak) [10044]
- 132 PASS: Retrieved from Cache [10050]
- 133 PASS: X-ChromeLogger-Data (XCOLD) Header Information Leak [10052]
- 134 PASS: CSP Scanner [10055]
- 135 PASS: X-Debug-Token Information Leak [10056]
- 136 PASS: Username Hash Found [10057]
- 137 PASS: X-AspNet-Version Response Header Scanner [10061]
- 138 PASS: PII Disclosure [10062]
- 139 PASS: Timestamp Disclosure [10096]
- 140 PASS: Hash Disclosure [10097]
- 141 PASS: Cross-Domain Misconfiguration [10098]
- 142 PASS: Weak Authentication Method [10105]
- 143 PASS: Reverse Tabnabbing [10108]
- 144 PASS: Modern Web Application [10109]
- 145 PASS: Absence of Anti-CSRF Tokens [10202]
- 146 PASS: Private IP Disclosure [2]
- 147 PASS: Session ID in URL Rewrite [3]
- 148 PASS: Script Passive Scan Rules [50001]
- 149 PASS: Insecure JSF ViewState [90001]
- 150 PASS: Charset Mismatch [90011]
- 151 PASS: Application Error Disclosure [90022]



It looks like our scan ran successfully, though it found new warnings on the site. A report should have been opened as an issue in your repository. Your developers can now triage and remediate any relevant vulnerabilities.

ZAP Scan Baseline Report #18 Edit New Issue

Open github-actions bot opened this issue 2 minutes ago · 0 comments

github-actions bot commented 2 minutes ago · edited by alwell-kevin

- Site: <https://www.xyz.com>
New Alerts
 - Timestamp Disclosure - Unix [10096] total: 20:
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - ...
 - Absence of Anti-CSRF Tokens [10202] total: 3:
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - Cross-Domain JavaScript Source File Inclusion [10017] total: 20:
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - <https://www.xyz.com/>
 - ...
 - Modern Web Application [10109] total: 1:
 - <https://www.xyz.com/>
 - Incomplete or No Cache-control and Pragma HTTP Header Set [10015] total: 1:
 - <https://www.xyz.com/>

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Linked pull requests: Successfully merging a pull request may close this issue. None yet

Notifications: Customize Subscribe

You're not receiving notifications from this thread.

0 participants

Congratulations, you have now implemented baseline DAST on your application and embedded it into your CI pipeline! 🎉

Once your Development team is practicing DSOMM and has achieved Level 1, you can try to tackle maturing to Level 2 in six to 12 months. Let's do this! Stay tuned for future GitHub blog posts on DevSecOps.

Questions?

Contact your Account Management team or the [GitHub Sales team](#) to see how GitHub can help your Engineering team build better, more secure software together.

Additional Details / Notes

[GitHub Enterprise Twitch](#)
[Practical DevSecOps Course](#)
[OWASP DSOMM](#)
[GitHub Advanced Security](#)
[SCA in Depth](#)

