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?

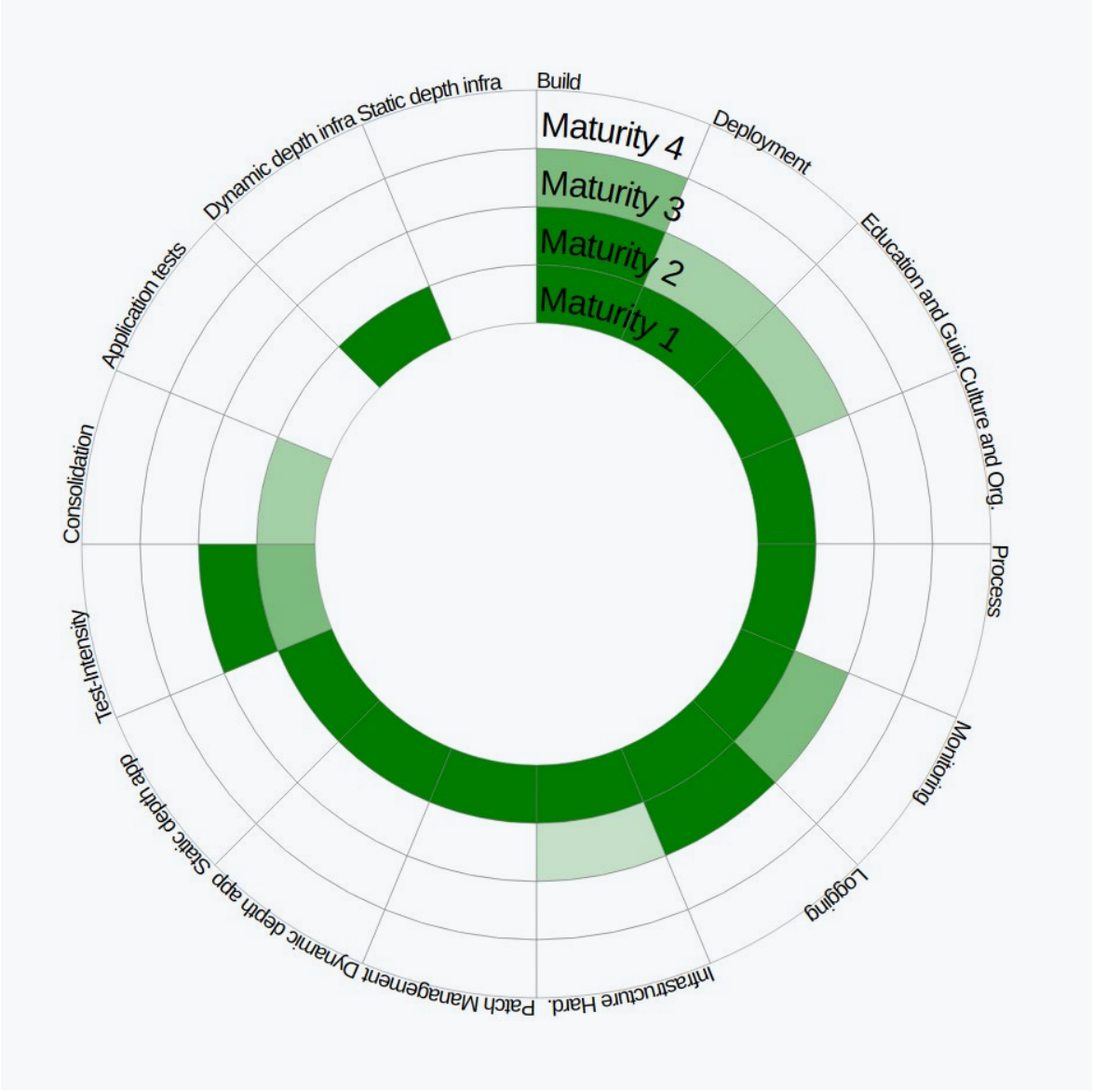
Before we dig into the how, let's align on a definition of DSOMM.

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.

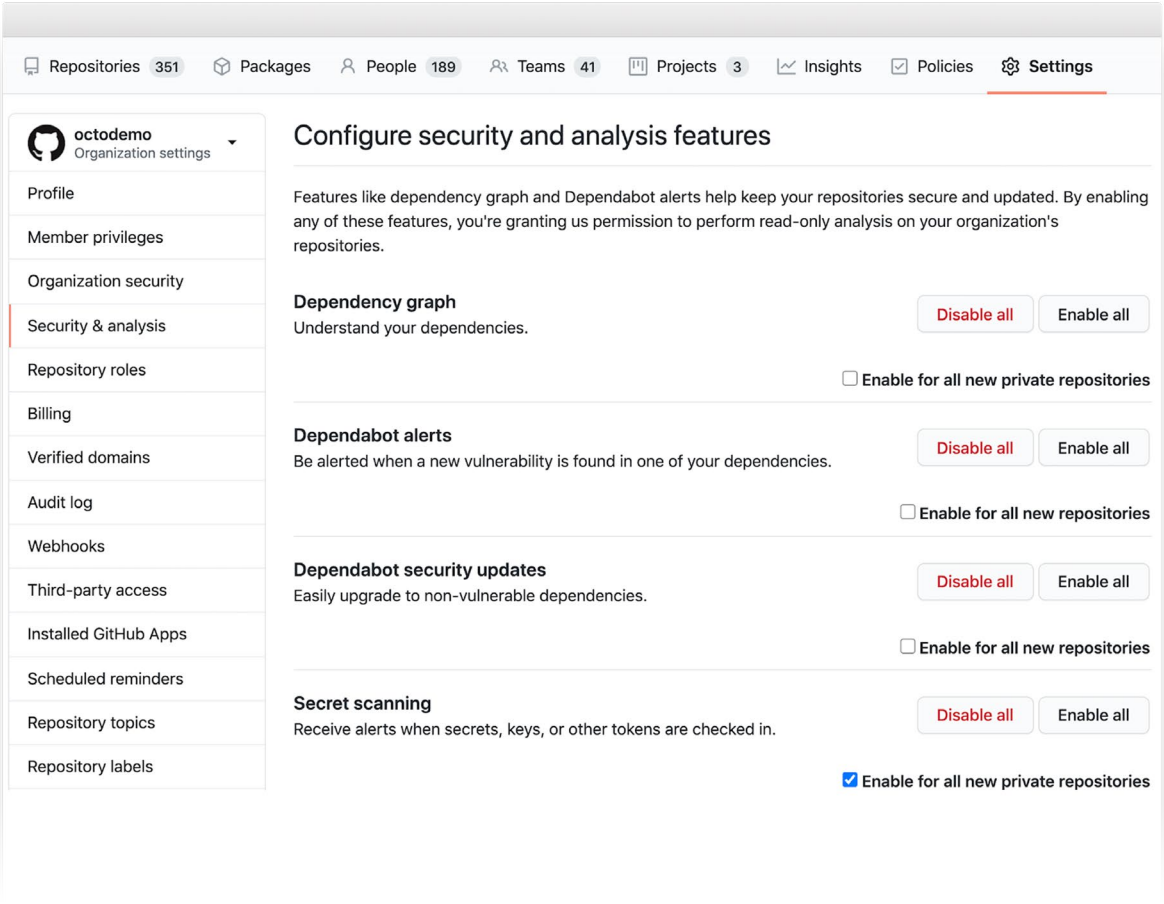
Secret scanning:

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.

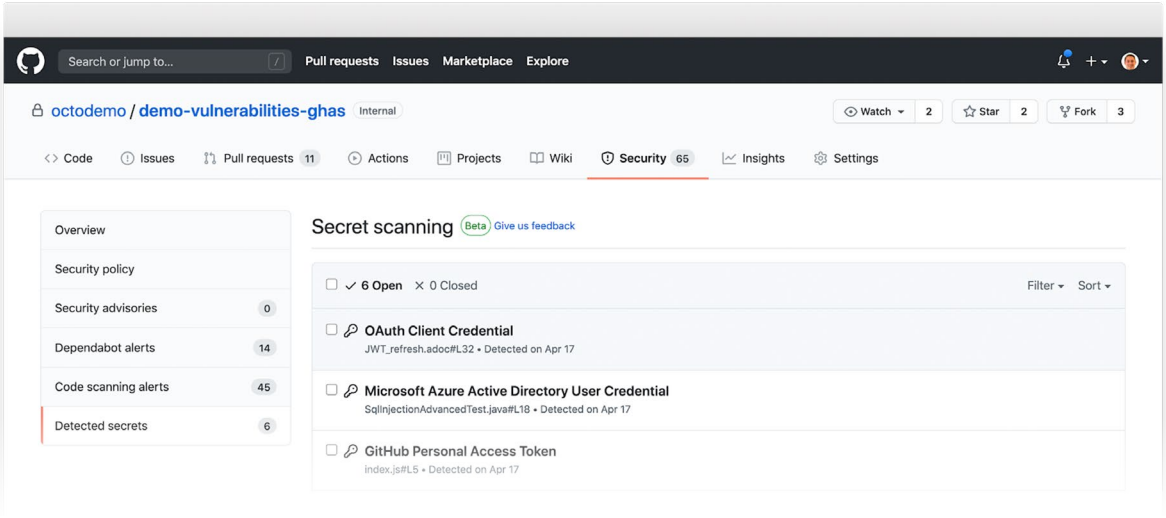
- ⚠ GitHub sends an email alert to the repository administrators and organization owners. If the secret is a personal access token from GitHub, we instead send the email alert directly to the owner of the token.
- ⚠ GitHub displays an alert in the repository. For more information, see "Managing alerts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

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

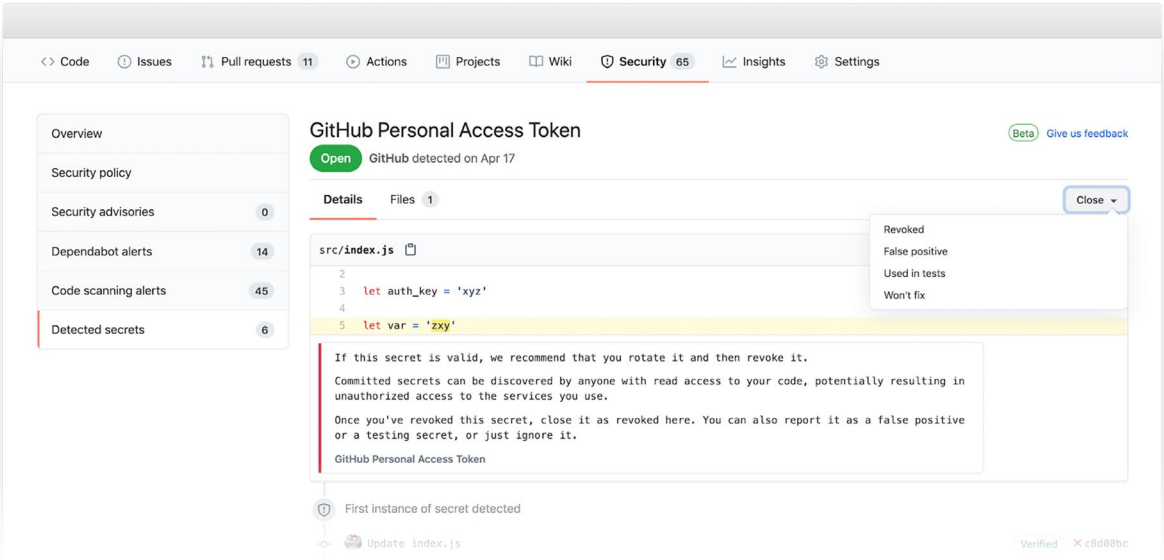
Organization-level setup:



Repository-level setup:



Result interface:



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:

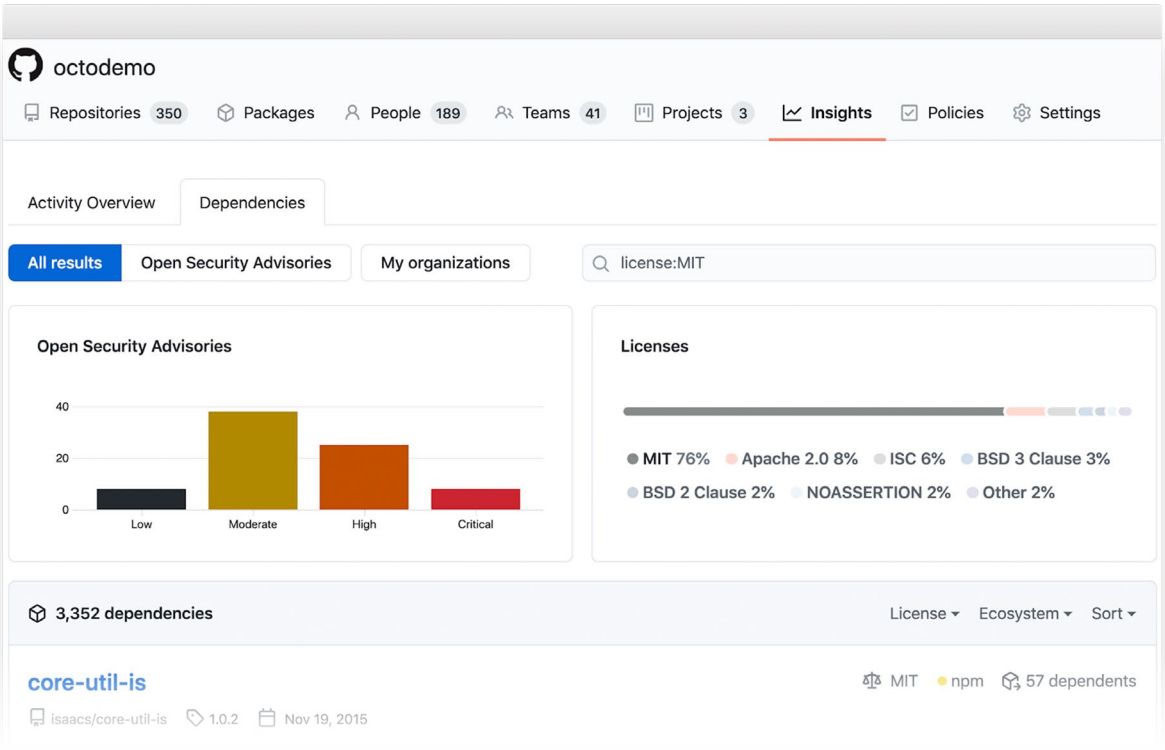
Software composition analysis is the set of tools to help you determine what dependencies you use, discover vulnerabilities in those dependencies, and patch them. The idea is, when a new vulnerability is discovered for third-party or open source dependencies in your environment, you want to be quick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

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

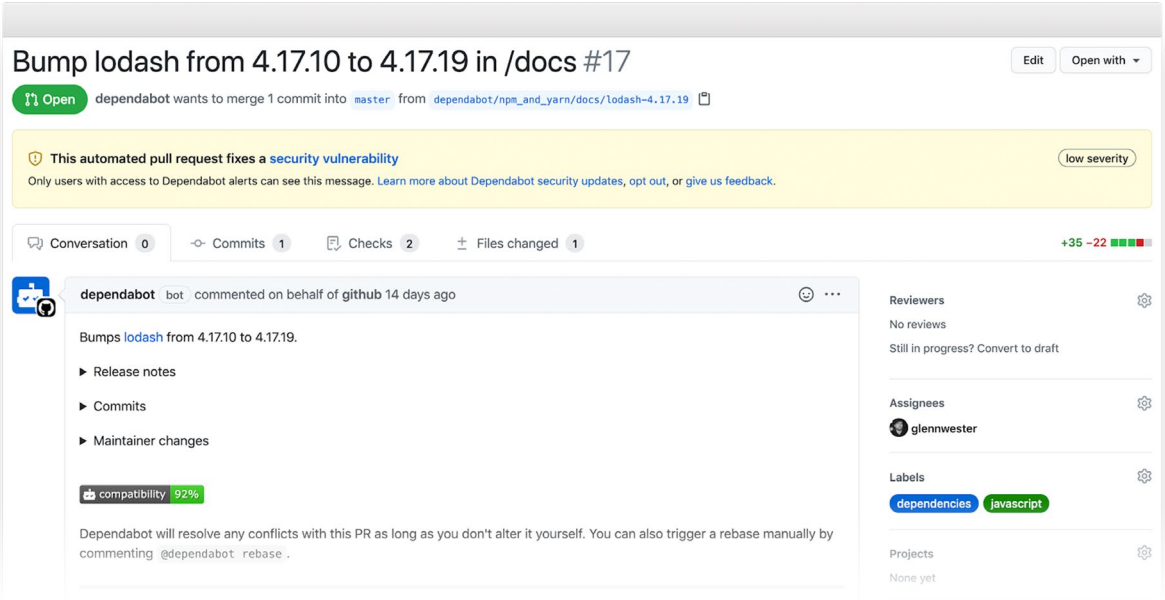
- Dependency Graph identifies all upstream dependencies and public downstream dependents of a repository or package. You can see their project's dependencies, as well as any detected vulnerabilities, in the dependency graph.
- Dependabot alerts notify you of repositories affected by a newly-discovered 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.



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.

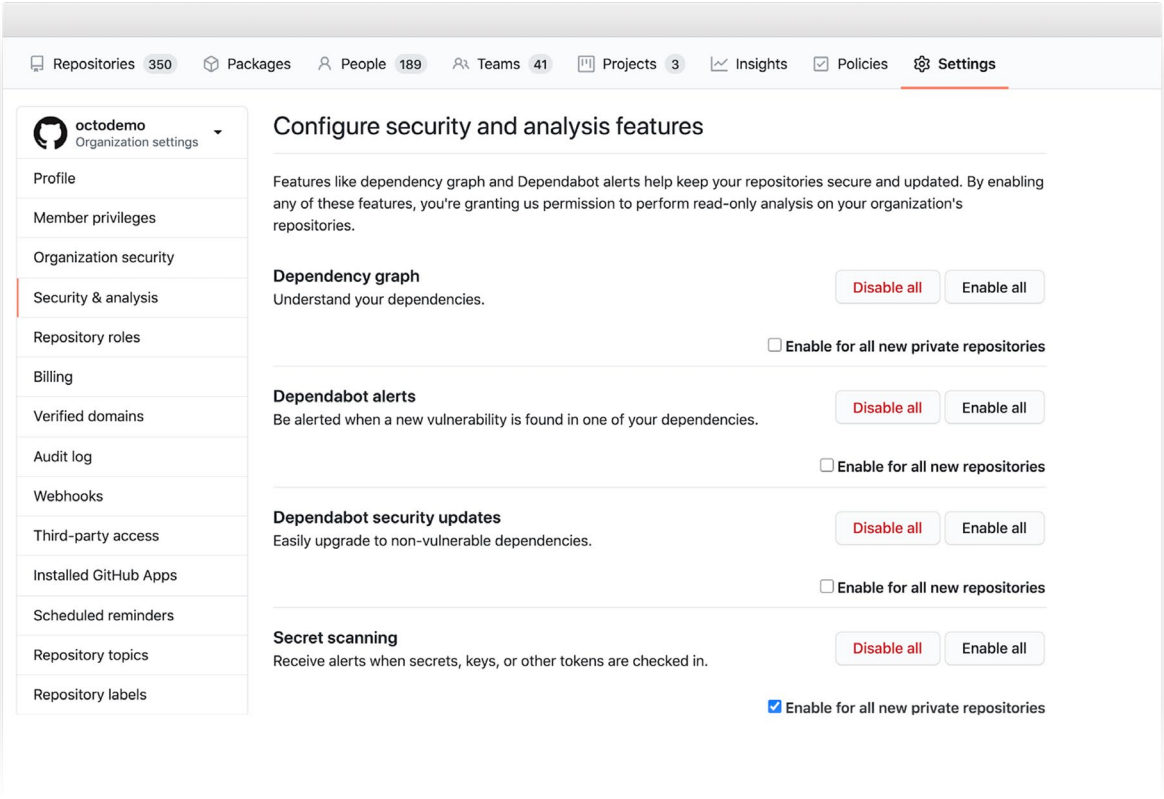


Let's look at how to enable Dependabot for our organization or repositories.

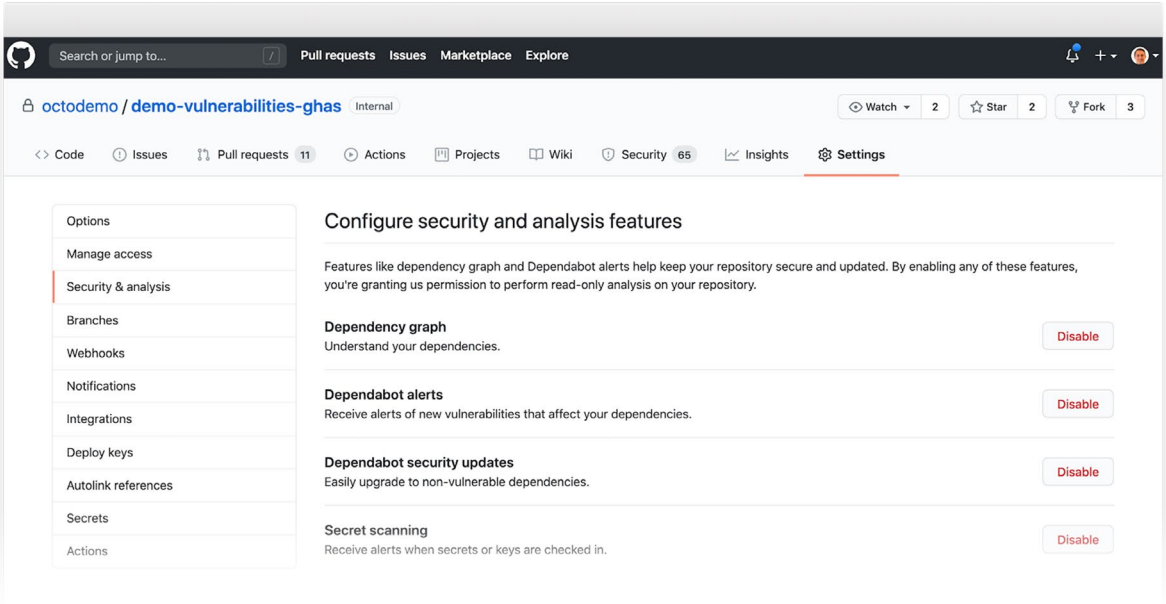
You will need to enable:

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.

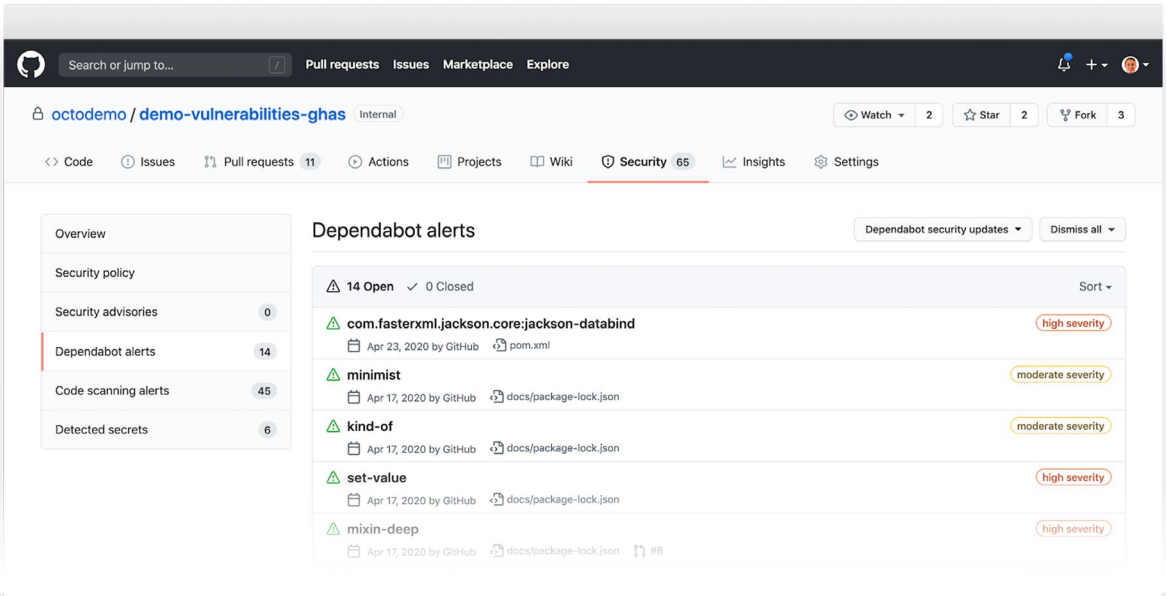
Organization security:



Repository security:



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.



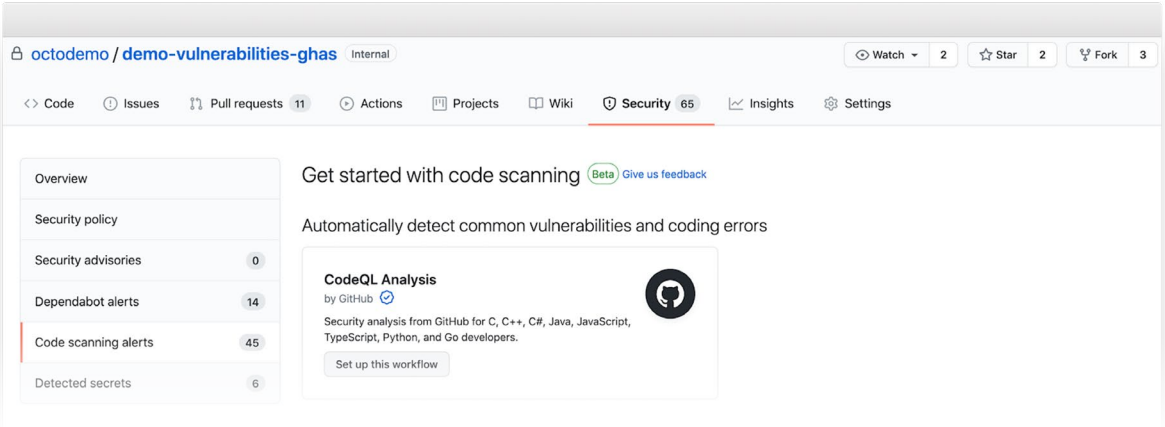
SAST:

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."

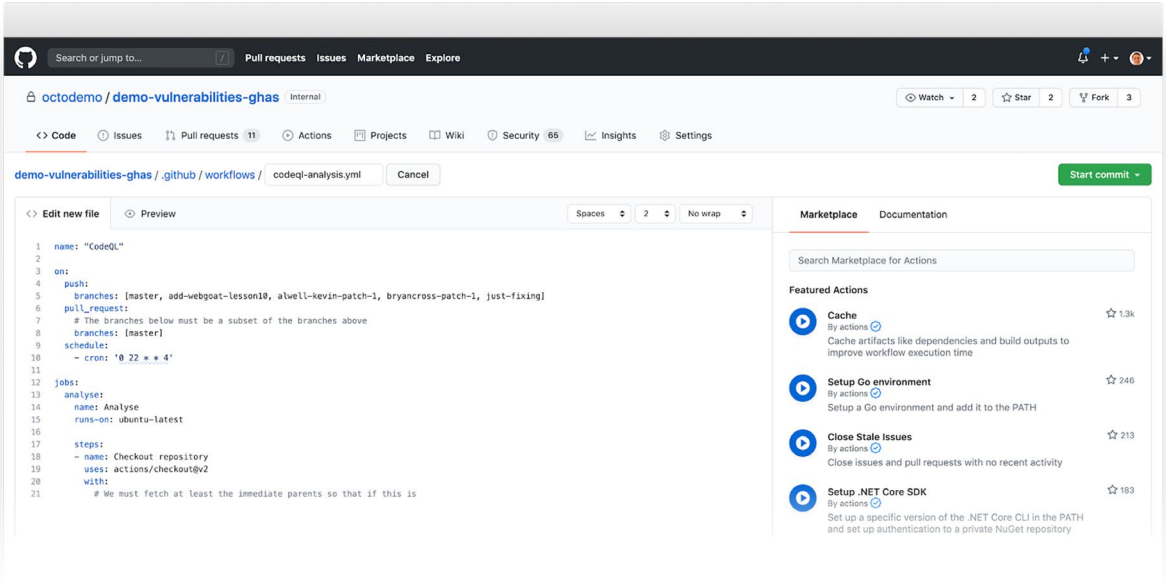
Lets enable code scanning on a repository together. First, go to your repository security tab and click “Set up this workflow”.



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

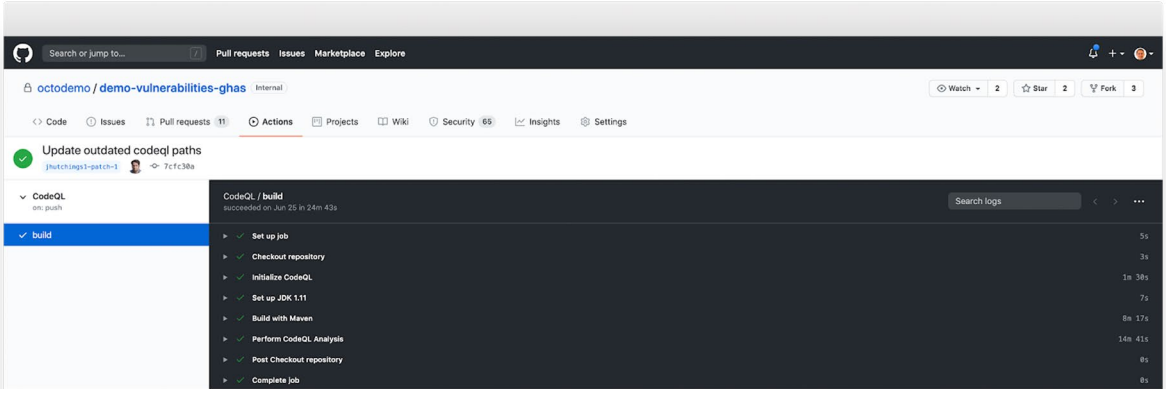
increasingly important as your automation practices mature and your teams adopt GitHub more heavily.

Lets quickly dissect the code scanning workflow.



- **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.

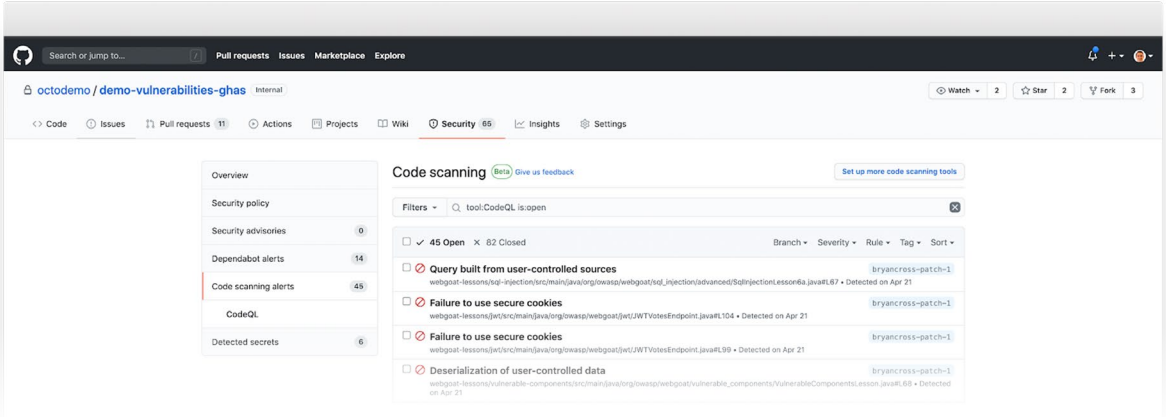
After committing your application and running code scanning successfully for the first time, you should see something like this in your CI logs (we are using GitHub Actions).



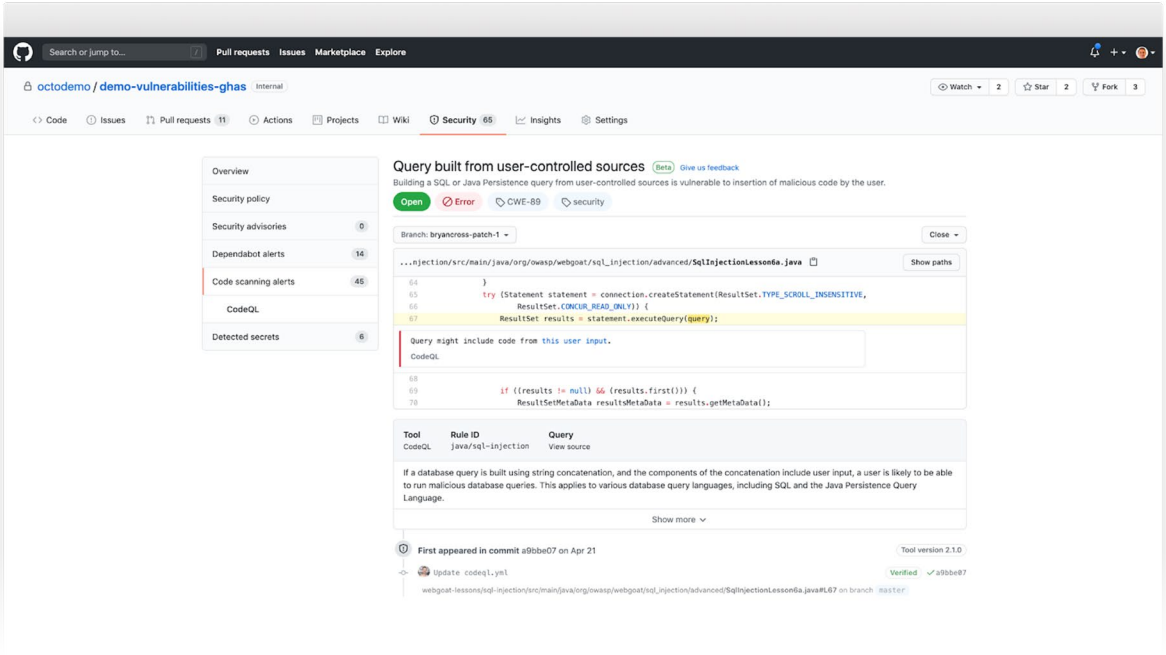
Expand the “Perform CodeQL Analysis” step to observe each query that is being run against your application. You should find something like:

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

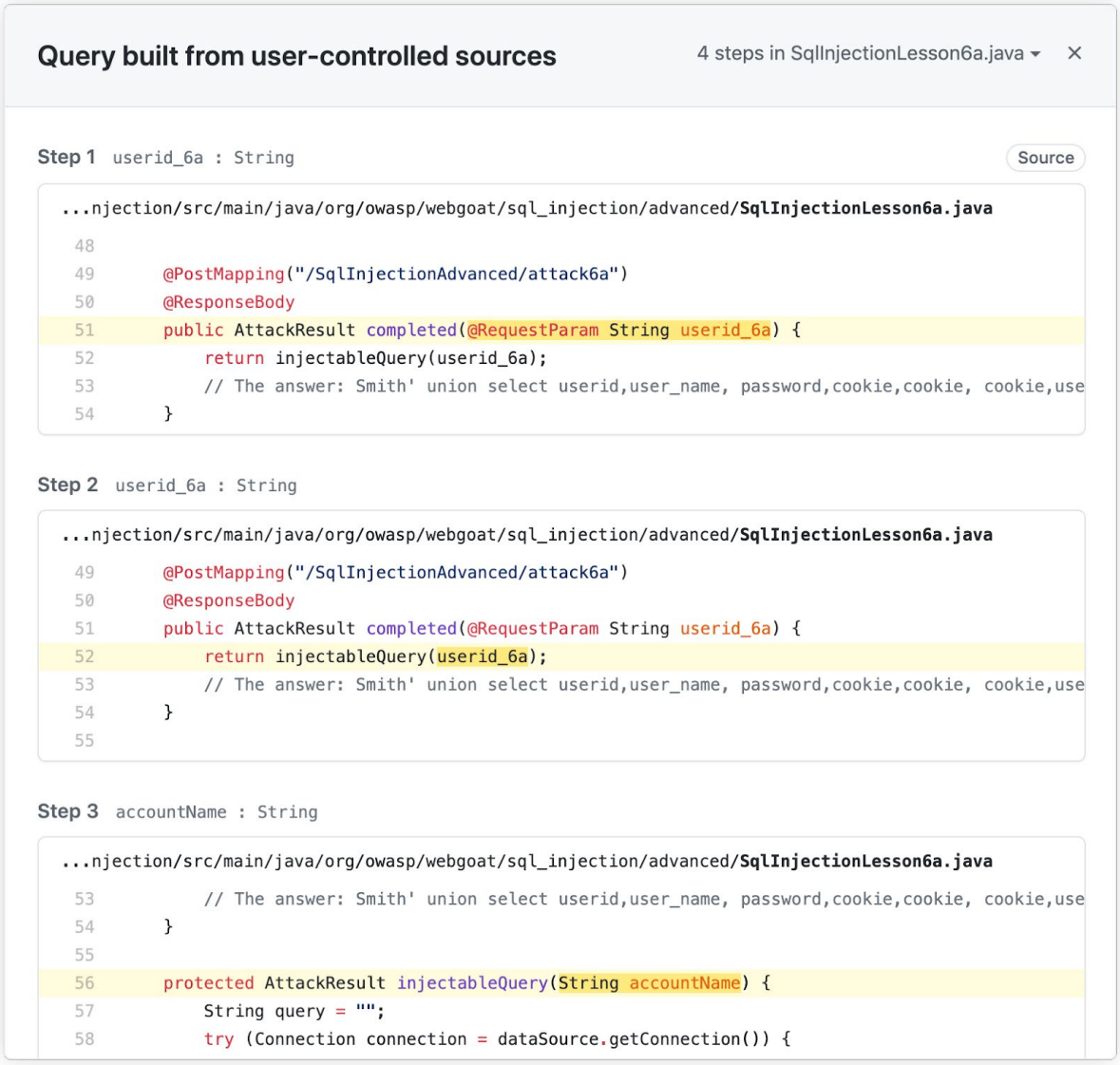
If a vulnerability is found, an alert (or many) will surface under the “Security” tab of your repository. This may take a few moments for the first scan.



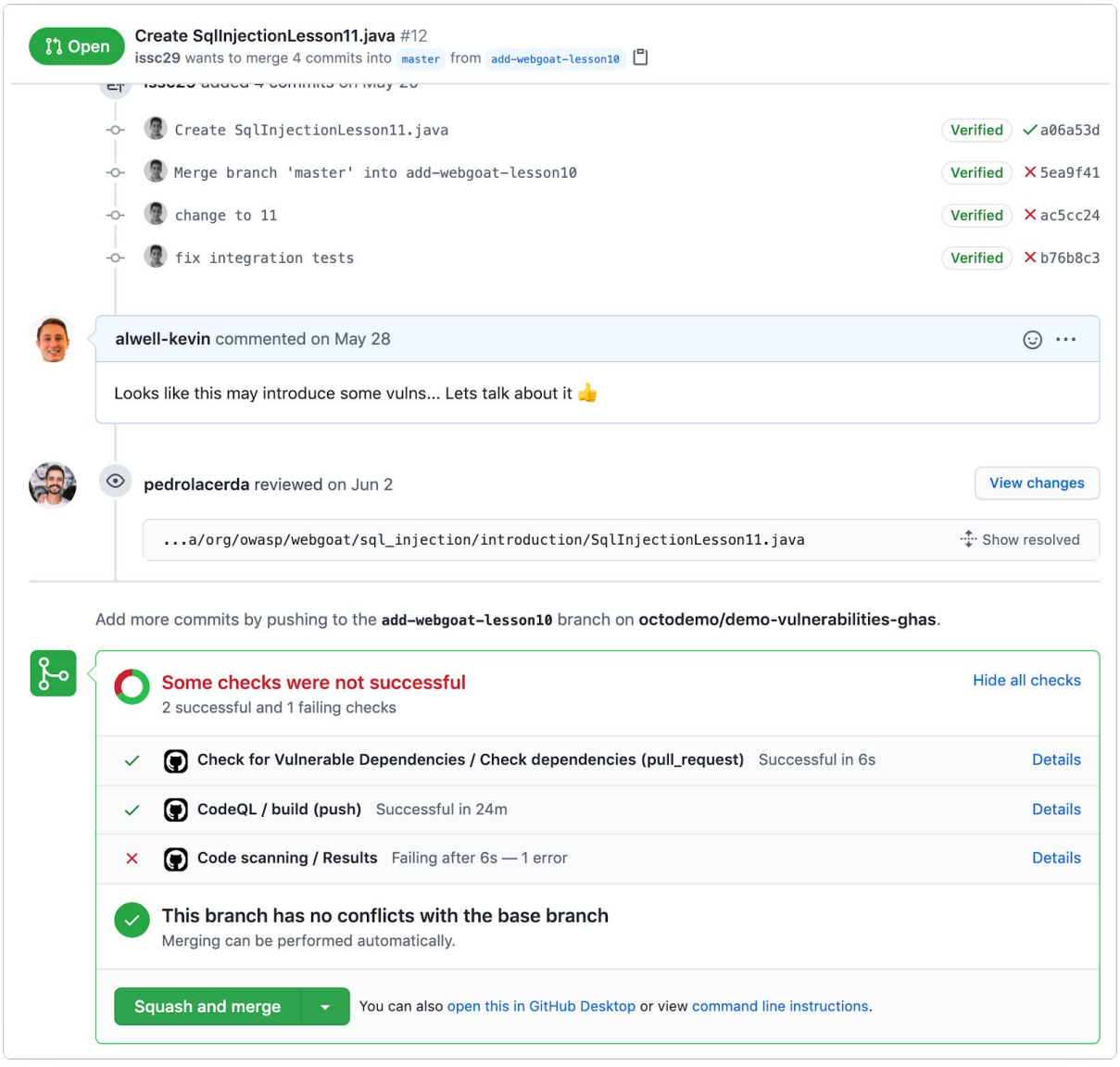
In my example, you can see many open vulnerabilities. Notice we include the type of vulnerability, filters can be applied based on the Branch, Severity, Rule, Tag and Date/Time. The alert includes data like the file(s) impacted, a source -> sink view, remediation recommendations and an audit history.



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



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).

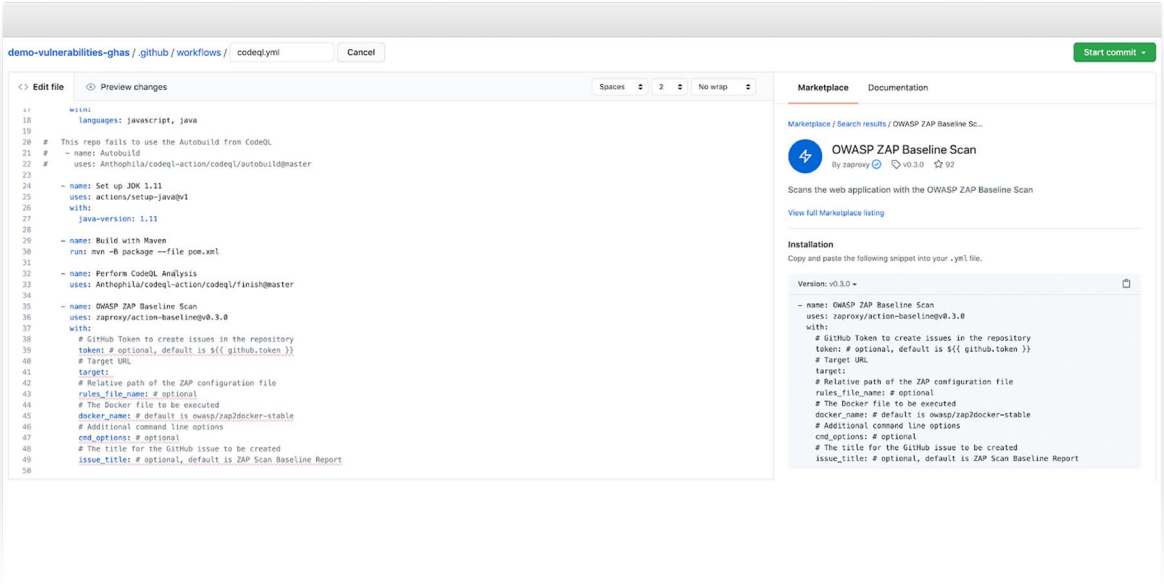


Congratulations, you have now integrated GitHub code scanning as your SAST tool! 🎉

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.

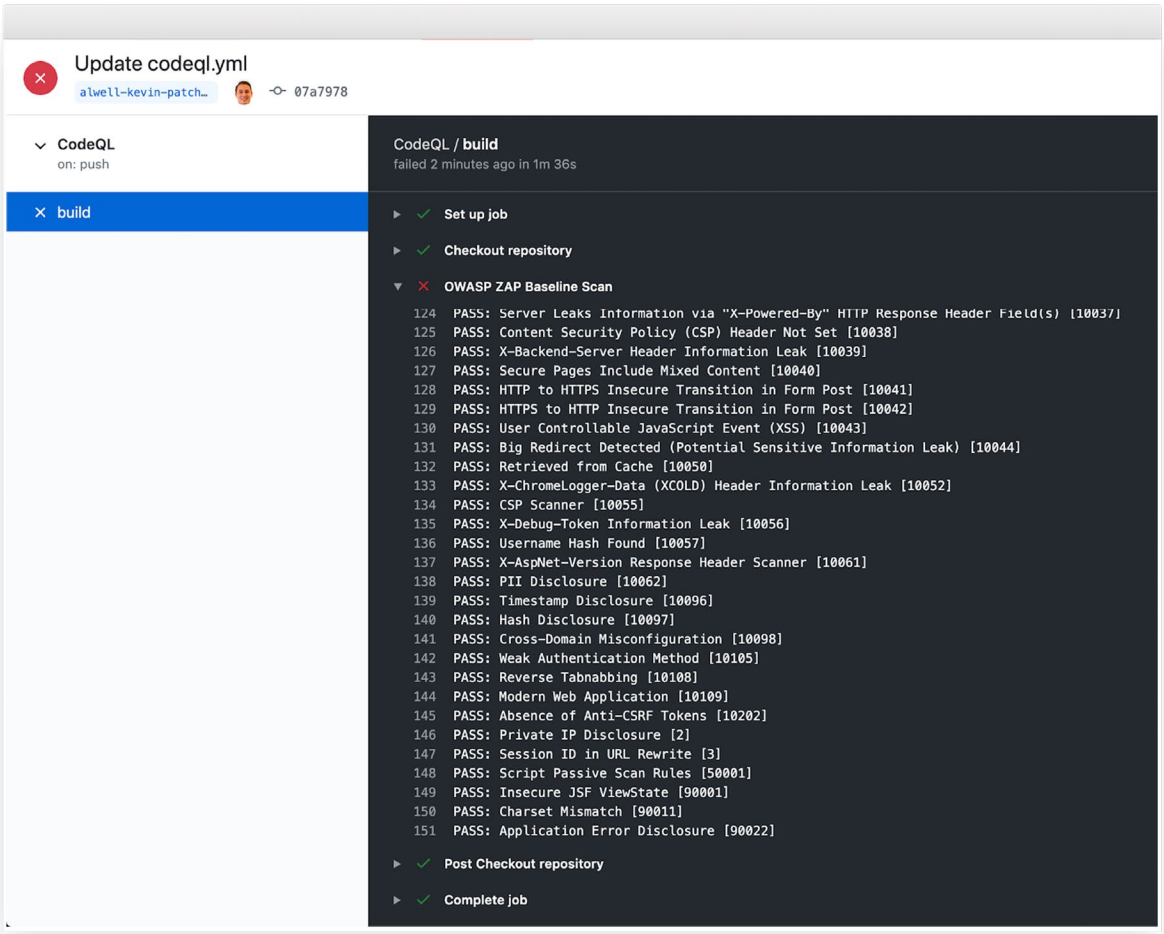
Open the code scanning workflow file we just created. You should see the ability to search for and single click integrate the ZAP GitHub Action. Lets select "OWASP ZAP Baseline Scan" for our initial Level 1 implementation. You should now see something like this:



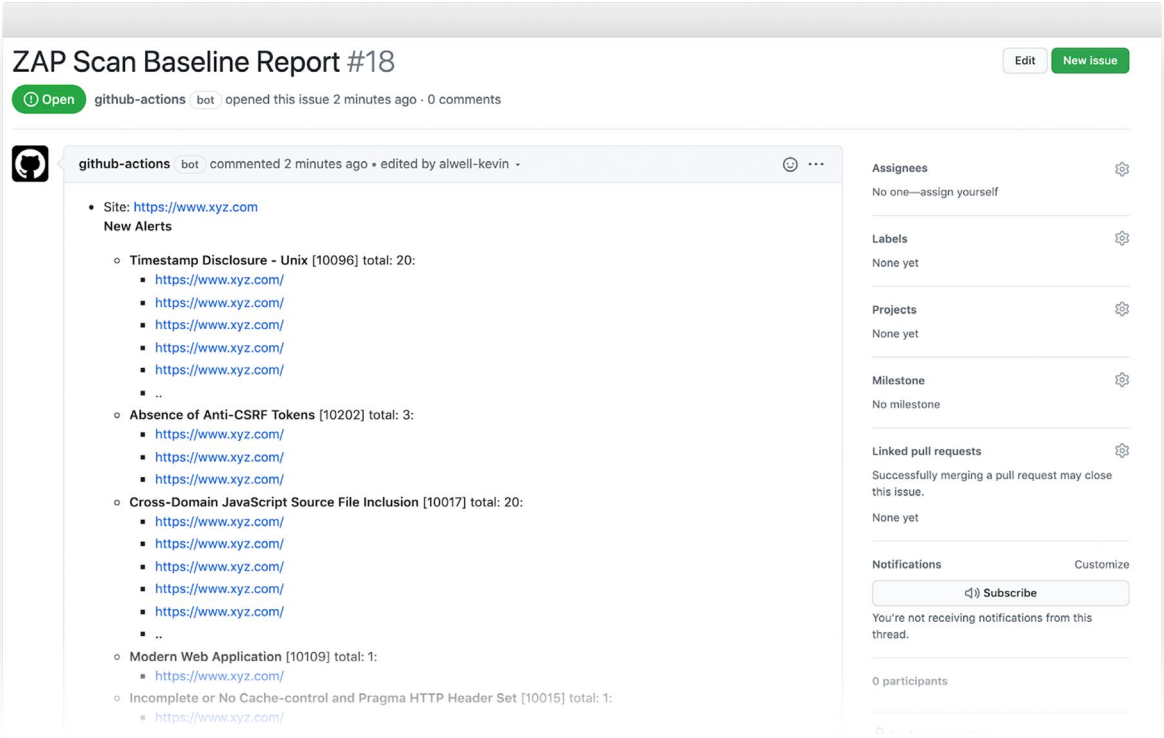
For the purposes of our implementation, I want to keep the implementation simple. This is what my YAML looks like now.

```
- 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.



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.



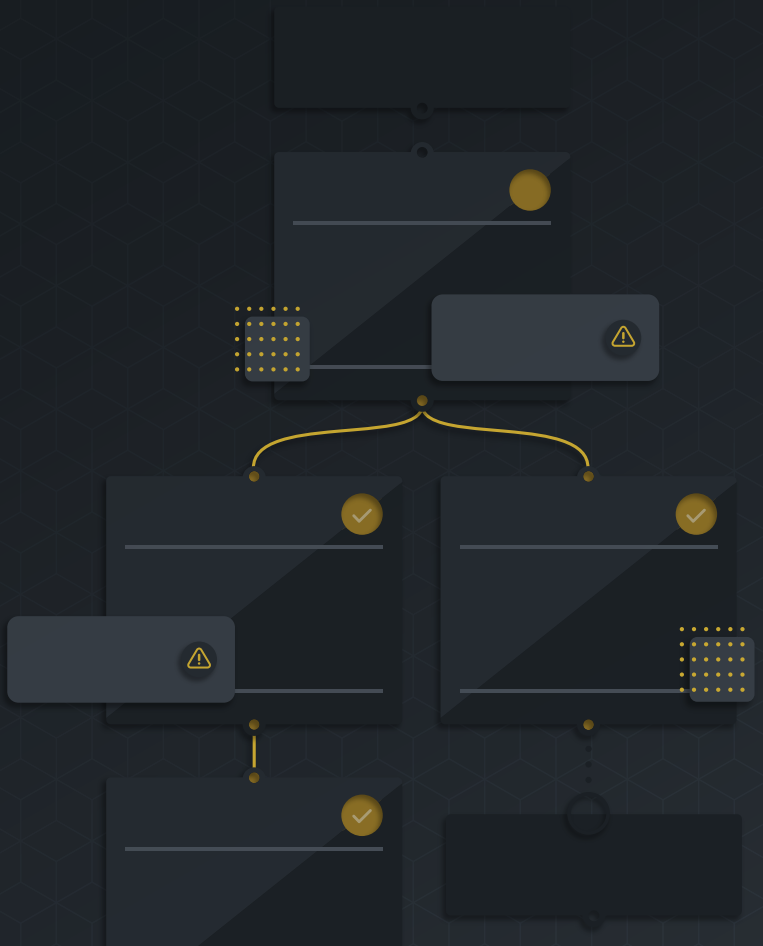
Congratulations, you have now implemented baseline DAST on your application and embedded it into your CI pipeline! 🎉

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.

GitHub

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