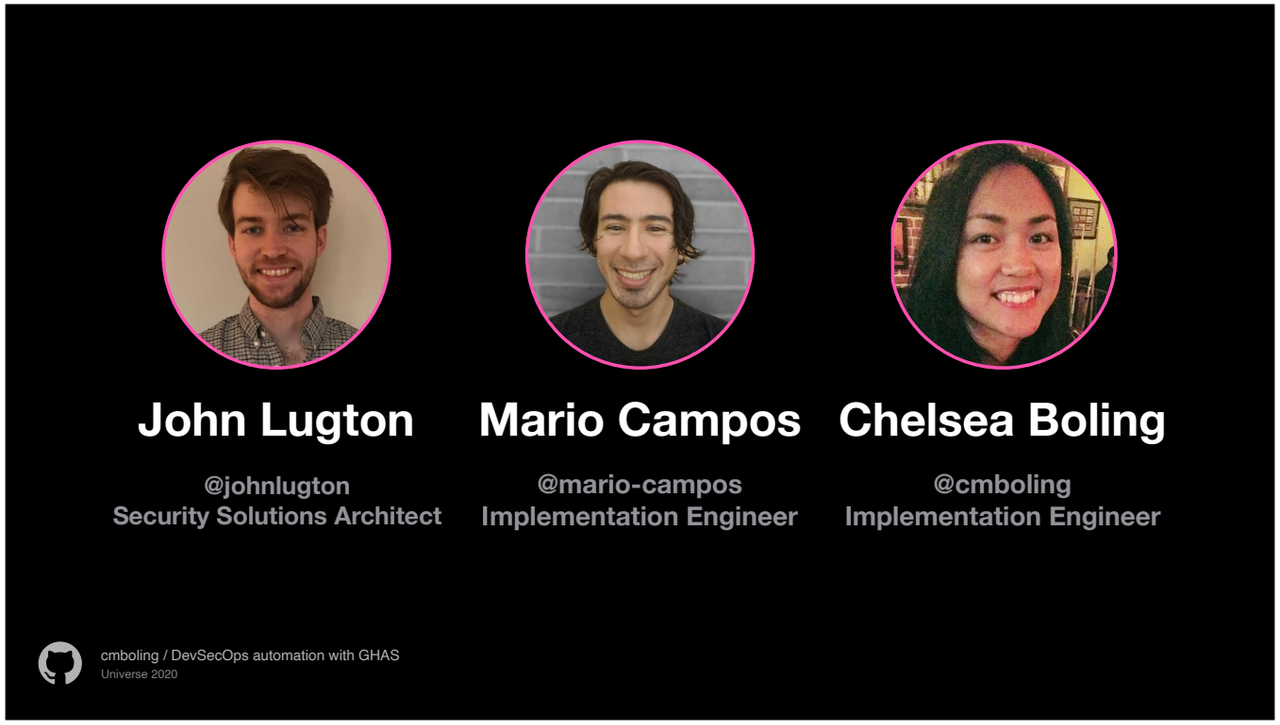
Welcome to the DevSecOps Automation with GitHub Advanced Security aka GHAS!



So let’s kick off with a couple of introductions. Along with myself, I have my colleagues here John Lugton, one of our lead security solutions architects on the team and my other awesome colleague, Mario Campos, who is also an implementation engineer and is moderating this workshop. We’re very glad you’re here to take this workshop. Let’s begin.

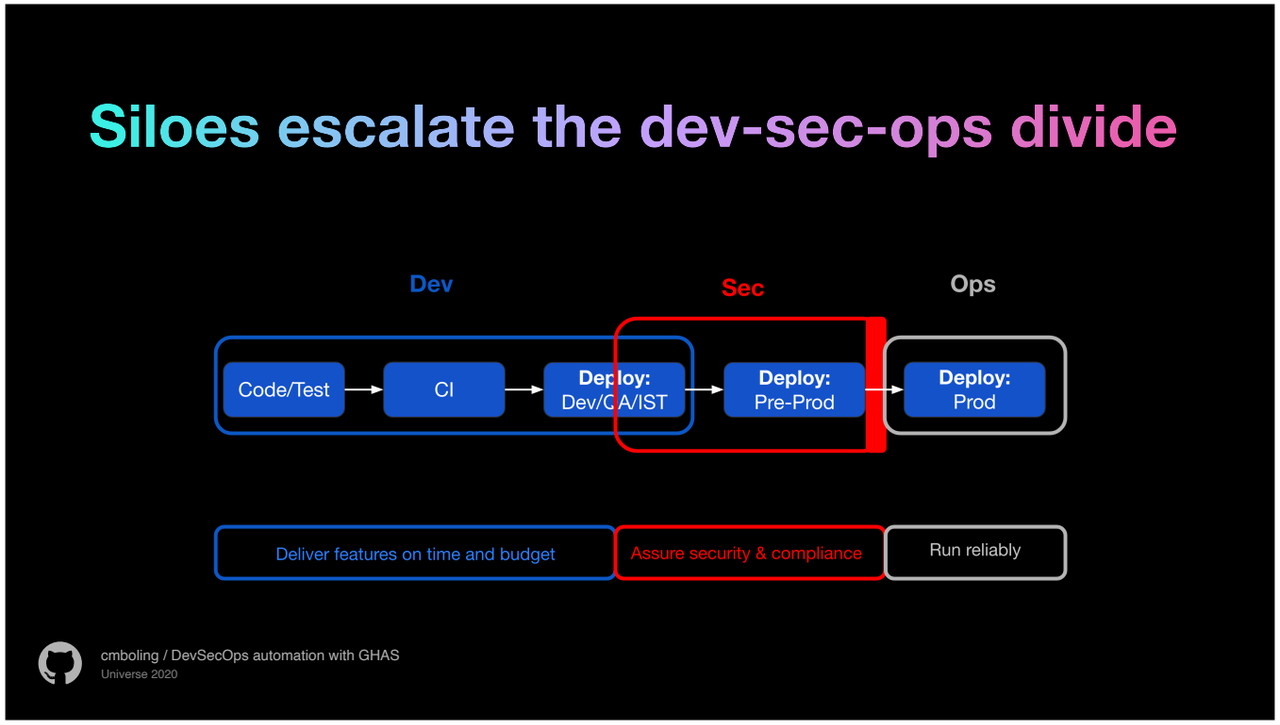
What is Advanced Security?
Before we dive in GitHub Advanced Security…
Let’s discuss what DevSecOps is and the importance of it in the software development lifecycle.

DevSecOps brings security into development and operations teams to ensure that security is a priority at every step of the software development lifecycle.

In other words, to decrease the number of vulnerabilities in production code, we should make security a part of everyone's daily work instead of testing for security concerns at the end of the process.

Putting developers at the center of application security is an effective way to shift security left and succeed against the technical debt. This means integrating security testing and controls into the daily work of development, QA, and operations.

Ideally, we want to automate this process, and that’s what we’re going to do in this workshop: Automating DevSecOps by using GitHub Advanced Security.



Here’s a siloes diagram:

So as you can see from this, why not be more developer centric?

Why continue with this model of:
- Siloed security
- Testing just before deployment
- Static testing and dynamic testing happened at the end of the delivery cycle?

Here’s how we can close this gap.

GHAS is a suite of security tools used to enable developers to identify security vulnerabilities in their codebases:

Code scanning, Secret scanning, Dependabot, Security Advisories, Security policies, Dependency insights

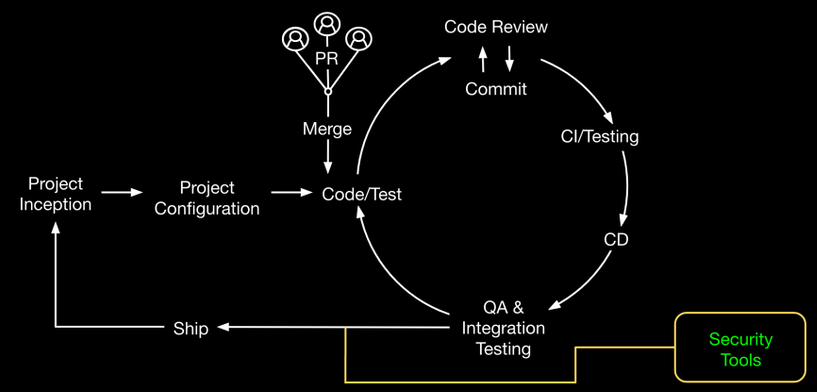
And let’s pretend we have Dependency Review squeezed into this list as well! I’m sure you’ve heard about this during this week’s plethora of Universe talks. Dependency Review is another fantastic tool that offers tighter integration of security by helping contributors understand dependency changes and their security impact at every pull request.

We want to bring a native and automated experience when it comes to securing your code on GitHub. When we think about application security, part of the challenge with application security today is that they're very much focused on the security team's experience. We want this experience to be dev-centri - the developers, who are the ones who are responsible for remediating and fixing their issues.

Developers can come to a single point to utilize security tools alongside all of the other software development initiatives that they're running inside their pipeline.

In this workshop, we’re going to explore secret scanning, Dependabot (security alerts and security updates), and code scanning.

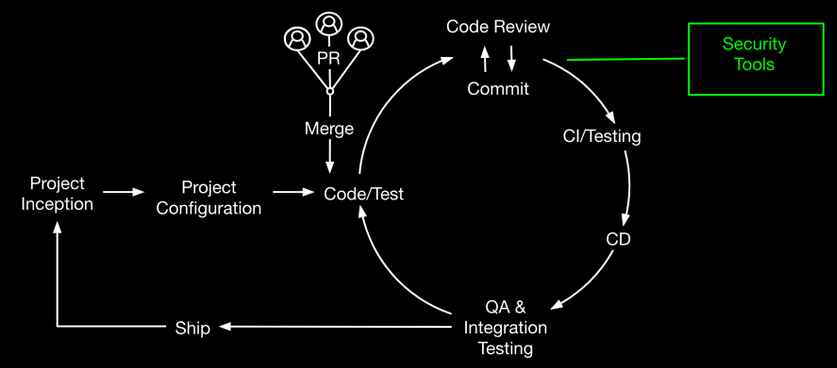
Basic application security scenario



Here again… in this basic scenario: in some developer workflows that I've seen myself and heard about from my colleagues and their experiences, the usage of security analysis happens right before we're about to ship.

Why not use these security tools earlier in the software development workflow, which could be alongside secure coding practices and code quality checks?

Improved application security scenario



Here’s how it can be better…

Instead of software dev teams, IT operations and security teams working independently… Why not commit to sharing the responsibility of security and follow security guidelines in all lines of work.

This is where we want to be: to shift security left and have security integrated at the PR level.

Here’s a tiny recap on what’s going to happen during the workshop:
How are we going to shift left with these Advanced Security features at our hands? Let’s take a moment to pause and think about this.

What we’re going to do in the next hour is enable these features as mentioned previously.

Getting started

- Are you logged into GitHub?
- Have you accepted the invitation to be a collaborator of the Universe Workshop organization?
- Do you see your private repository within the Universe Workshop organization?

Before we get started….
Let’s do a small health check with you all and make sure you’re all set.

Our demo application

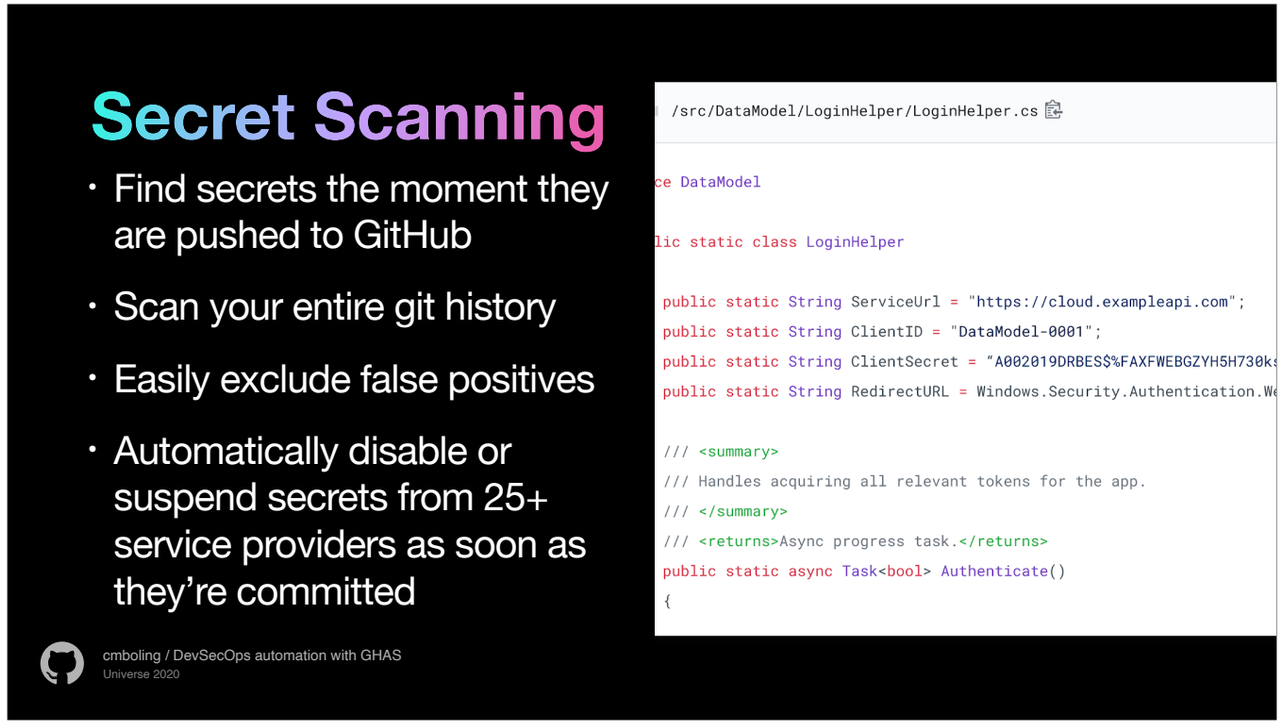
- It's a simple gallery application with a handful of services
  - Frontend is written in Javascript.
  - Backend services are written in Java, Go and Python.
- There's a variety of security problems we've written into the application; can you find them all?

Authentication service written in python
Frontend written in javascript
Gallery service written in go.
Storage service written in java
A scripts folder which contains a python script that generates secrets, which will be handy in one of our exercises.

And just a note… it's not required that you try to spin up this application. Most of what we're doing will be in the browser, enabling secret scanning, Dependabot and code scanning in the UI and modifying a couple of files here and there.

What's an easy thing that we can do to improve team security?

Feel free to add your comments in the chat!



Self-reflection questions for introspection:

Have you pushed something to a repo and you know you shouldn't have?
Who has seen a someone commit/expose secrets to a repository? Who has received a security audit because of a key you exposed?
Who's ever been in a situation where you're working too fast, maybe you have 3 features to do and someone like your teach lead asks you to rotate secret keys, you do so but accidentally committed these credentials to production?

We make mistakes; things happen…
Please share your stories in the Slack channel.

This is where secret scanning comes in.

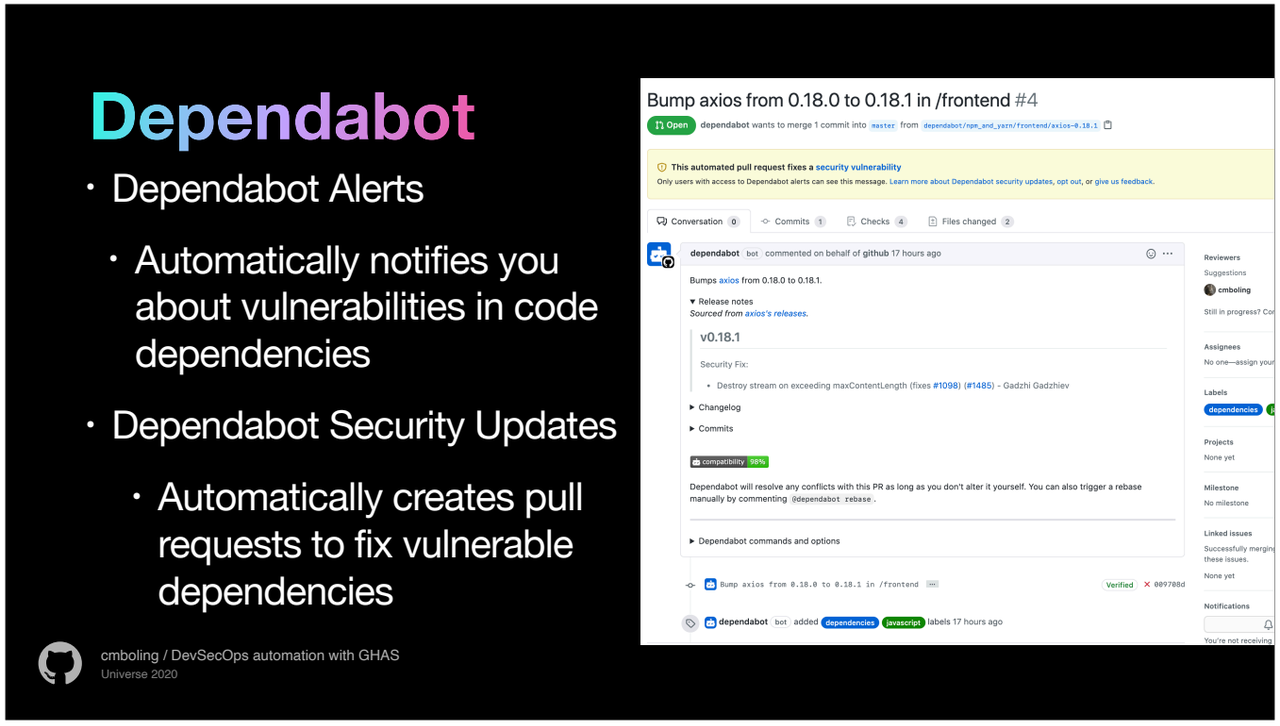
We use secret scanning to scan for known types of secrets to prevent fraudulent, deceitful use of accidentally committed secrets.
When you push to a public repository, GitHub scans the content of the commits for secrets. If you switch a private repository to public, GitHub scans the entire repository for secrets.

If you check a secret into a repository, anyone who has read access to the repository can use the secret to access the external service with your privileges.

For safety measures we recommend storing secrets in a dedicated, secure location outside of the repository for your project. P.S. work with your sys admins/engineers. :)

When secret scanning detects a set of credentials, GitHub notify the service provider who issued the secret. The service provider validates the credential and then decides whether they should revoke the secret, issue a new secret, or reach out to you directly…depending on what the risk is.

We support a long list of providers which can be found in our GitHub docs. For instance, AWS, Strip, Atlassian, Azure… to name a few.



Now that we've talk about secret scanning, what about the dependencies that our projects rely on? Can you think of a time when you worked on a project that had a handful of vulnerable dependencies?

That's where Dependabot comes in. Dependabot security updates make it easier for you to fix vulnerable dependencies in your repository .When you enable this feature during this workshop, when a Dependabot alert is raised for a vulnerable dependency in the dependency graph of your repository, Dependabot automatically triggers a PR.

This is pretty cool to do since this is done natively and there's no need to integrate this functionality into your CI pipeline.

Let’s take a look at secret scanning and Dependabot in action!

Feedback and questions are welcome in the chat.

Practical Exercise 1

Let's enable secret scanning and Dependabot together.

Practical exercise 1 recap

This is something you can do in 5 minutes to have a big impact on security!

- Great job on the first exercise! How did it go? For each feature:
  - Were you able to enable it?
  - Were you able to view and manage results?
  - Were you able to modify its respective configuration file?
- Stretch goals:
  - Using GraphQL API to retrieve vulnerability information
    - This is in the additional references section.

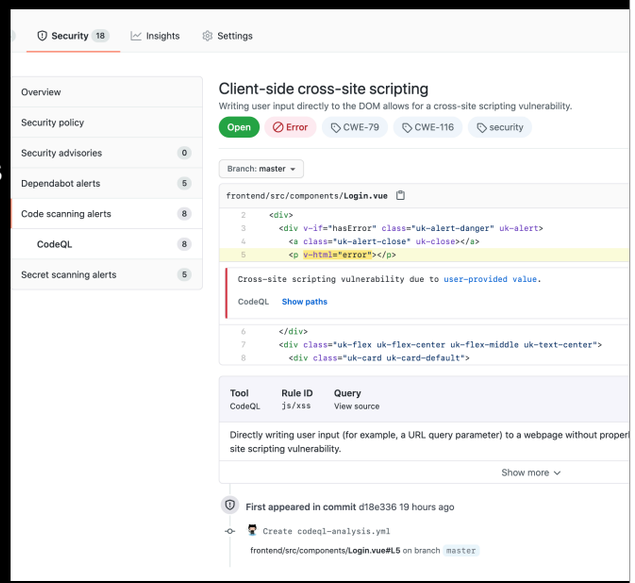
Pausing for alerts to show up and answer any questions

How do we find vulnerabilities in the code that we write?

Again, feel free to add your comments in the chat!

Code scanning

- Run CodeQL analysis automatically on pushes and PRs by using Actions
- Deliver results to developers within the GitHub platform
- Use the standard security analysis from GitHub and the community, along with custom queries
- Extensible with support for other SAST tools that product SARIF



We’ve gone over how to scan secrets; we’ve found some vulnerable dependencies along the way too. Now what about our code?

Suppose you’re in a sprint and have three features to develop or how ever many features/tasks you’re used to working on. You develop these features, and submit PRs. With code scanning enabled, you can actually scan your code in your pull requests and see any security introduced… rather than seeing security issues later.

It’s awesome to have code scanning integrated into Github because
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 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.

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.

Let’s enable code scanning.

Feedback and questions are welcome in the chat.

Practical Exercise 2

Let’s enable code scanning together.

cmboling / DevSecOps automation with GHAS

Universe 2020

Practical exercise 2 recap

This is something that you can do quickly to have an impact on security.

- How did it go? For each feature:
  - Were you able to enable code scanning?
  - Were you able to go to the Actions page and see the jobs running?
  - Were you able to fix the build?
  - Were you able to see and review the alerts?
- Stretch goals:
  - Were you able to tune tune the workflow file with contexts and expressions?
  - Were you able to identify false positives?

Pausing for workflows to catch up and answer any questions

Should we block merges?

Common question: Should we stop PRs from being merged with unfixed security vulnerabilities?

- **Not developer-centric:**
  - What if there are false positives or bad results?
  - It can take a long time/lots of discussions to get exception
- **Alternative ways:**
  - Expectation that developers fix the resolve the alerts.
  - Security team can track fixes/FPs and reach out to dev teams that aren't.
  - Checks can be tuned/adjusted or can decide to provide extra training.

This is a common question that we get from security engineers and developers. The idea with using Advanced Security is to enable the contributor to take the best approach when it comes to securing their code. This isn't a tool that's meant to slow down the development workflow; rather it's a tool that should help you resolve vulnerabilities the earliest it comes up, which usually can be prevented at the PR level. So knowing that… doesn't is sound better knowing you have these security tools at your disposable and it's *you* that can make a good choice, a great impact on securing code?

Conversations like these open up a really great discussion, especially when it's both security and development teams making those choices about application security together.

How do we refine CodeQL analysis to address false positives or catch more vulnerabilities?

Feel free to address results in the chat!

Let’s modify our CodeQL analysis configuration.

Feedback and questions are welcome in the chat.

Practical Exercise 3

Let's improve the CodeQL analysis configuration.

cmboling / DevSecOps automation with GHAS

Universe 2020

Practical exercise 3 recap

Increase your confidence easily by tuning your setting with GHAS. No one size fits all with security.

- How did it go? For each feature:
  - Were you able to add your own code scanning suites?
  - Were you able to add additional query suites?
  - Were you able to ignore paths?
- Stretch goals:
  - Were you able to add a custom query?
  - Were you able to identify false positives?
    - Were you able to resolve them?

Pausing for workflows to catch up and answer any questions

Which checks should we run?

- **More rules means more results and that could be great, but it could also mean more false positives.**
- **Focus on rules with the greatest impact. Alert fatigue is a real thing!**
- **Aim to be dev-centric:**
  - Dev and security teams work to agree on security coding standards.
  - Enable new rules to include feedback from devs and tracking FP rates.
  - Catch vulnerabilities earlier. Trying to catch everything is ***not*** realistic.

So we've introduced other query suites such as security extended and security and quality which could search for those low severity alerts that can be in your code. When it comes to this question, the bigger picture is to focus on the rules that matter at large - What are the checks that have the greatest impact? With that in mind, again, this isn't a tool to slow you down; it isn't a tool that should create fatigue or rather alert fatigue (burn out), which is a real thing that many developers have experienced, so let's do what matters the most and aim to be more dev-centric.

What have we learned so far?

You've seen how to use GHAS to shift security left.

- Remediating secrets and vulnerabilities
- Reviewing code scanning alerts
- Enabling additional queries
- Filtering false positives

What do you think about secret scanning, Dependabot and code scanning so far?

Call to Action

Try it yourself

Setup GHAS features on your own repository

Collaborate with security

Make the most out of collaborating with security teams. Publish security advisories to notify users of your project about a vulnerability.

Reach out to us

CodeQL is open source and any contributions you make helps secure the software we rely on.

Consider a shifting left

Embrace catching any vulnerabilities earlier in your development workflow.

Any questions?