Table of Contents

[Introduction to GitHub Advanced Security 2](#_Toc132917294)

[Learning objectives 2](#_Toc132917295)

[Prerequisites 2](#_Toc132917296)

[Introduction 3](#_Toc132917297)

[Keep your development environment secure with GitHub Advanced Security 3](#_Toc132917298)

[Learning objectives 3](#_Toc132917299)

[Prerequisites 4](#_Toc132917300)

[What is GitHub Advanced Security? 5](#_Toc132917301)

[Secure your software supply chain 5](#_Toc132917302)

[The security overview 6](#_Toc132917303)

[Dependency graphs 6](#_Toc132917304)

[The GitHub Advisory Database 7](#_Toc132917305)

[Dependabot for automated dependency management 8](#_Toc132917306)

[Find and fix security issues as you code 8](#_Toc132917307)

[Code scanning 8](#_Toc132917308)

[Secret scanning and push protection 9](#_Toc132917309)

[Features behind the scenes 10](#_Toc132917310)

[Secure your automated workflow environment 10](#_Toc132917311)

[Licensing considerations 11](#_Toc132917312)

[Create an organizational culture around security 12](#_Toc132917313)

[Adopt a security mindset 12](#_Toc132917314)

[Shift left 12](#_Toc132917315)

[Faulty security models 12](#_Toc132917316)

[The ideal security model 13](#_Toc132917317)

[Security policies 13](#_Toc132917318)

[Responding to security alerts 15](#_Toc132917319)

[Evaluate security alerts 15](#_Toc132917320)

[Triage security alerts 15](#_Toc132917321)

[Manage security alerts 16](#_Toc132917322)

[Use security advisories 16](#_Toc132917323)

[Summary 17](#_Toc132917324)

[Learn more 17](#_Toc132917325)

**Introduction to GitHub Advanced Security**

This module will help you become familiar with GitHub's Advanced Security features and best practices. As you learn about these features, you'll identify critical areas for eliminating security gaps.

## Learning objectives

By the end of this module, you'll be able to:

* Define GitHub Advanced Security
* Identify the purpose of specific GitHub Advanced Security features
* Understand the value of a security-focused team culture
* Highlight the roles involved in securing your workflow
* Recognize best practices for identifying and responding to security vulnerabilities

## Prerequisites

* A GitHub Enterprise Cloud or Enterprise Server account with GitHub Advanced Security
* Working knowledge of GitHub Actions and workflows

# Introduction

Imagine that you're a security administrator for an enterprise organization. Your responsibility is to protect your organization from cyber security threats. You know that the constant menace of data breaches makes securing your development environment more important than ever. Bad actors can cause devastating events. However, the negligence of users and administrators can also create serious security incidents. As hackers become more sophisticated, the impact and damage to businesses and people have proven to be far-reaching and occasionally tragic.

A few examples from recent years come to mind:

* **2017:** A vulnerability found on Equifax servers allows hackers to execute malicious code remotely. The event exposes more than 143 million records, including 209,000 credit card numbers. An attack that human error makes possible breaks customer trust. The Apache Foundation had released a patch to address the vulnerability just two months before the breach. Unfortunately, Equifax delayed applying the necessary system updates.
* **2018:** Hackers successfully breach the Marriott Hotels systems. The breach compromises 500 million users and exposes data records for 327 million guests. This data includes names, travel dates, passport numbers, and home addresses.
* **2021:** The discovery of a vulnerability in the open-source logging software Log4j marks the latter part of 2021. Cybersecurity experts label the fallout caused by the bug as the most serious to date due to the popularity of the software. Because of this popularity, hackers seeking to break into digital spaces exploit a massive field of operation, which puts millions of servers at risk. The vulnerability gives these hackers complete control over some of these applications.

These incidents show the importance of having every resource at your disposal to successfully deal with threats.

### Keep your development environment secure with GitHub Advanced Security

Luckily, your organization has purchased a GitHub Advanced Security license. GitHub provides advanced security features to help secure your software-development lifecycle from attacks like the ones mentioned above. GitHub Advanced Security is available for enterprise accounts on GitHub Enterprise Cloud and GitHub Enterprise Server 3.0 or higher. GitHub Advanced Security is also part of all public repositories on GitHub.com.

## Learning objectives

This module will help you become familiar with GitHub's Advanced Security features and best practices. As you learn about these features, you'll identify critical areas for eliminating security gaps.

At the end of the module, you'll be able to:

* Define GitHub Advanced Security.
* Identify the purpose of specific GitHub Advanced Security features.
* Understand the value of a security-focused team culture.
* Highlight the roles involved in securing your workflow.
* Recognize best practices for identifying and responding to security vulnerabilities.

## Prerequisites

* Familiarity with GitHub, repositories, and the basics of managing individual accounts is required
* Familiarity with personal and organizational authentication technologies and processes would be helpful

# What is GitHub Advanced Security?

The previous unit discussed the importance of securing your environment from malicious actors. The real-life cases highlighted some of the ways in which hackers can exploit systems. In this unit, you'll be introduced to GitHub Advanced Security. Let's focus on the various product features and the challenge or problem that these features address.

GitHub Advanced Security is a powerful suite of tools and features that gives you the ability to identify security vulnerabilities in your codebase and environment. These features allow you to secure your code at every step of the software-development lifecycle, not just in your production environment. You stay ahead of security threats and breaches, leverage the expertise of the security community, and continue to use open-source software securely. GitHub Advanced Security also includes implementing security best practices that help create a security-minded culture within your organization.

GitHub Advanced Security focuses on protecting your organization in three primary areas:

* Supply chain
* Code
* Environments

Let's look at these three categories to see how you can use advanced security to improve security within your GitHub organization.

## Secure your software supply chain

When you think about supply chains, you may think of manufacturing goods within an industrial warehouse. However, the concept of a supply chain also applies to what happens during your software-development lifecycle. Your workflow produces chunks of code that are then reviewed for quality assurance before reaching the next stage. Protections and guidelines are enforced along the way to optimize production and make an acceptable end product.

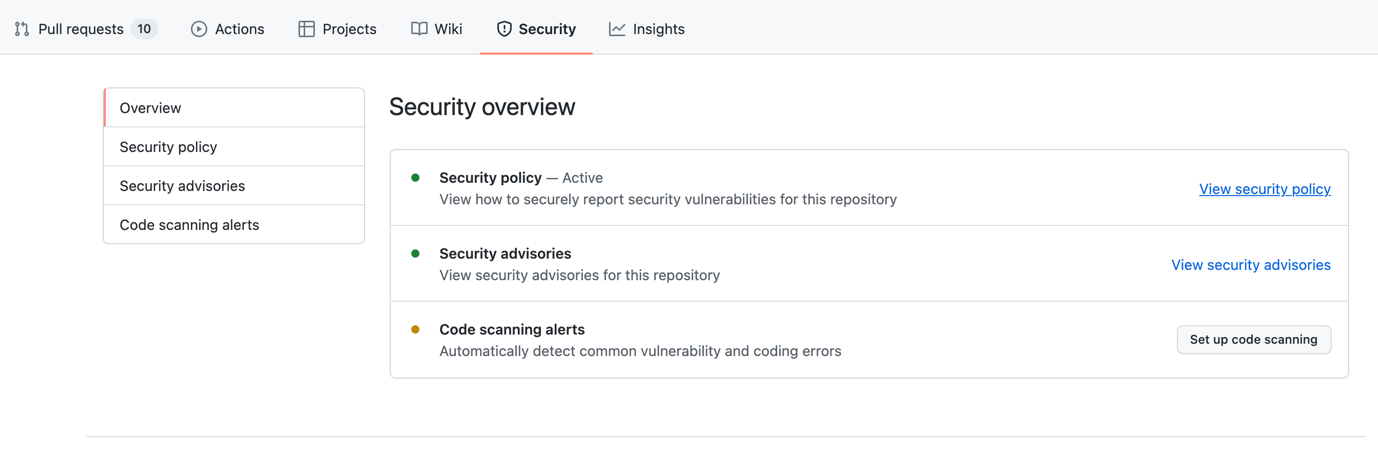
In the context of software development, supply chain also refers to the integration of any third-party or open-source software. Your software is most likely more than the code you've written in-house. Up to 94% of active repositories rely on open-source, so you may rely on many components you didn't produce. These components are called dependencies. You need to actively review them for vulnerabilities.

In 2021, Log4j was a dependency included in many applications worldwide. The events surrounding the Log4j vulnerability show how crucial dependency management is to securing your codebase. If you use any third-party software with security vulnerabilities, your project inherits the same weaknesses.

Fortunately, GitHub has automated features that flag these dependency vulnerabilities. These warnings alert specific team members to potential dangers and enable them to act accordingly. These powerful features allow you to automate the process that monitors and secures your project dependencies and searches for vulnerabilities.

Let's look at the features you can use to monitor and secure your software supply chain.

### The security overview

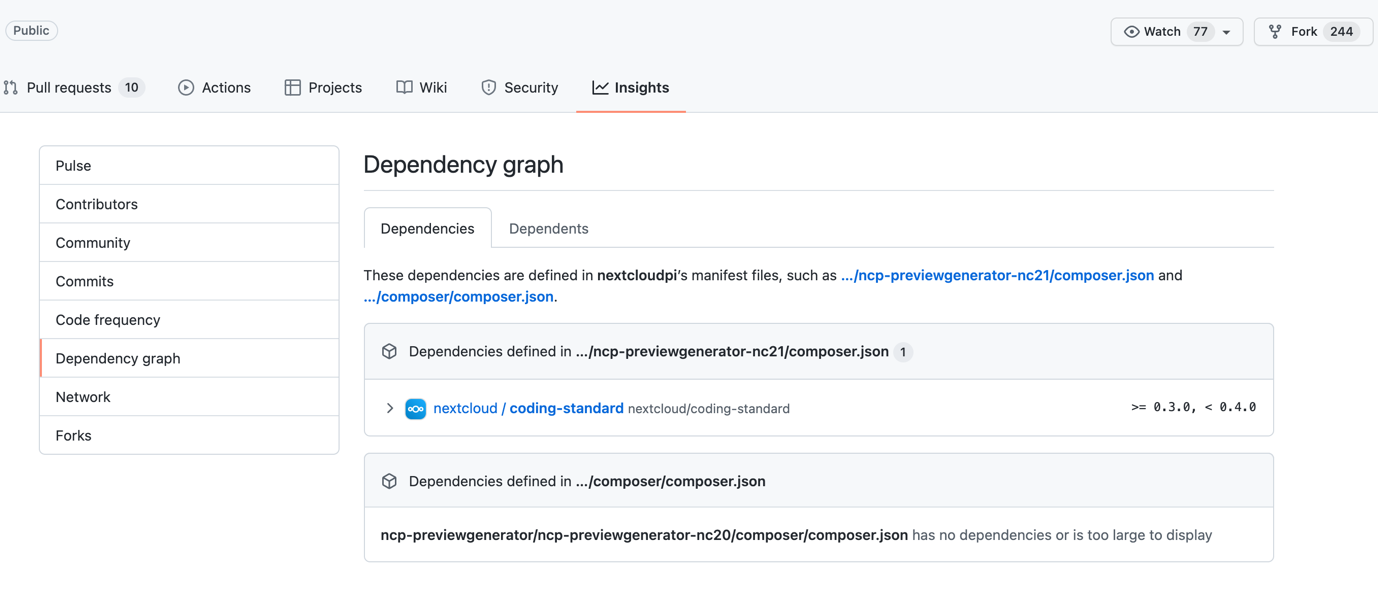


Once you have an established supply chain, GitHub's security overview provides you with a central location to properly oversee it. This makes it easier to monitor the state of your project's dependencies. The security overview allows you to identify problematic repositories and understand how your dependencies impact your code's security. You can also enable security features from the security overview. View, filter, and sort automated security alerts generated for your organization or a specific team. The overview also lets you filter alerts by their risk level so you can prioritize your response.

The security overview includes icons for different types of alerts. These icons also display the quantity of each alert type. In the next section, you'll learn about the GitHub features that generate these alerts.

### Dependency graphs

Visualizations can simplify and summarize large amounts of data. Dependency graphs provide important insights into your project's direct and indirect package dependencies and their current state. The graphs display a summary of manifest and lock files in the project's ecosystem, repository, and packages. You can also find out if dependencies are up-to-date and secure.



The dependency graph lets you monitor any known issues within dependencies that may affect the security of your application. Once you activate the dependency graph, you have access to dependency reviews for both private and public repositories belonging to your organization. These reviews help keep dependency vulnerabilities from being introduced into your environment.

### The GitHub Advisory Database

The GitHub Advisory Database provides information on the state of your dependencies. You can browse or search the GitHub Advisory Database for vulnerabilities in third-party solutions. This is useful because the packages on which your project depends also have other packages on which they depend. In this chain of package relation, you most likely won't be familiar with each piece of software that's been pulled in to your project. Having an overarching database to track what's happening in the ecosystem is an important resource.

The GitHub Advisory Database is curated by a dedicated team. It lists security vulnerabilities mapped to packages tracked by dependency graphs. Security threats appear in the GitHub Advisory Database ranked from low to critical. GitHub automatically updates this list regularly, using authoritative sources such as The National Vulnerability Database and the npm security advisories database.

Now, let's look at another component that relies on the GitHub Advisory Database.

### Dependabot for automated dependency management

Imagine a project that includes 20 to 30 third-party integrations. Manually monitoring so many dependencies and keeping them up to date is time consuming and tedious. Dependabot is GitHub's answer to this challenge. This built-in mechanism detects vulnerable dependencies by first examining the project's manifest file (such as the package.json file used in NodeJS). Next, Dependabot consults the GitHub Advisory Database to see if the detected dependencies have a flag that indicates they're out of date or vulnerable. Dependabot alerts authorized team members if there's an issue with any dependency or when a dependency graph related to a repository's changes.

This automated process helps avoid maintenance oversights and increases efficiency.

## Find and fix security issues as you code

After securing the supply chain, the next area of focus for GitHub Advanced Security is code analysis. Human error can introduce information into the codebase that puts your project at risk. You'll now learn about the features that address this risk, code scanning, secret scanning, and push protection.

### Code scanning

Even the best developers produce code with errors that their team must find, triage, and fix. GitHub's code-scanning feature allows you to complete all of those tasks. Code scanning is a static analysis of every git push. The scan happens at the same time as development and checks for common misconfigurations, errors, and vulnerabilities. This ensures that you catch problems before they make it into production. GitHub alerts authorized developers in the affected repository if the system detects an issue. You'll receive alerts about the state of your code and guidance with suggestions on how to fix any errors. Code scanning also allows members of your organization to learn how to write more secure code.

To use code scanning, you'll first have to enable it for your repository by completing these steps:

1. Navigate to the **Security** tab.
2. Select **Code scanning alerts**.
3. Click **Set up this workflow for CodeQL Analysis**.

GitHub code scanning supports the following languages:

* C
* C++
* C#
* Java
* JavaScript
* TypeScript
* Python
* Go

Once you enable code scanning, the default workflow triggers scans:

* When you push code to a specified branch
* When a pull request is created for a branch
* According to a schedule

You can monitor your code scanning workflow from the GitHub user interface. This workflow returns the following results:

* The name of the file that contains the vulnerability
* Highlighted code lines depicting the actual line on which to find the code in question
* An explanation of the problem
* A recommendation on how to fix the problem
* Examples that model the corrected code
* References to consult

### Secret scanning and push protection

When applications communicate with external systems, they often need to authenticate to gain access. This authentication is performed using keys known as secrets. Because developers work with secrets as they build code, it's not uncommon for them to accidentally commit these sensitive pieces of data to repositories. However, these keys create significant security risks if exposed. GitHub Advanced Security offers the following capabilities to prevent the exposure of secrets:

* **Secret scanning.** When you enable secret scanning, GitHub searches existing code on all branches present in your GitHub repository for keys. You can configure alerts to notify certain users if a secret has been found. Users can then take the appropriate remediation actions. For example, they can remove the key from the code and revoke and reissue the key that was compromised. Or in some cases, the users may choose to ignore the alert; for example, if the scan identified a false positive.
* **Push protection.** For an additional layer of security, you can enable the push-protection option of secret scanning. Push protection lets you find and remove secrets proactively before code is even added to your GitHub repositories. So instead of having to clean up leaked secrets, you can prevent secrets from being exposed in your repository in the first place.

Push protection integrates seamlessly with the command line or web UI tools your developers use. When a developer issues a git push command, GitHub scans the code to be added to your repository for secrets. If the code contains a secret, GitHub blocks the code from being committed and prompts the developer to take action. The developer can remove the secret, identify the detected secret as a false positive or a test key, or bypass the alert. As an additional security measure, administrators can configure GitHub push protection to send alert notifications if a developer bypasses a block.

### Features behind the scenes

Some GitHub Advanced Security features are not accessible from the user interface. They work behind the scenes to improve the performance of your platform.

The following table highlights some of these features:

|  |  |
| --- | --- |
| Feature | Use |
| **Protection rules** | Administrators can set protection rules that specify reviewers that must approve workflow runs on an environment. The reviewers have read-only access to the repository sp that they can approve or reject workflow runs without having permission to edit the code. |
| **Environment secrets** | Systems such as GitHub Actions or Azure can execute a workflow on an environment. Workflow jobs can only access the environment that they reference using the secrets of that environment. A reviewer can approve access for the job if needed. |
| **Wait timers** | In situations where you want to delay code deployment, you can set a timer that can wait up to 30 days before deployment takes place. |

## Secure your automated workflow environment

Developers often use GitHub Actions or Microsoft Azure to automate or customize software-development workflows. When defining a workflow, developers specify an environment in which their code should execute. These environments differ from repositories in that the environments are targets for workflow jobs. The target may be a runner that is similar to a virtual machine. For the runner to execute code received by the workflow, it may need to use an environment secret.

GitHub uses protection rules, environment secrets, and wait timers to keep environment access secure and give you control. Review these tools in the following table:

|  |  |
| --- | --- |
| **Feature** | **Use** |
| **Protection rules** | Administrators can set protection rules that specify reviewers that must approve workflow runs on an environment. The reviewers have read-only access to the repository sp that they can approve or reject workflow runs without having permission to edit the code. |
| **Environment secrets** | Systems such as GitHub Actions or Azure can execute a workflow on an environment. Workflow jobs can only access the environment that they reference using the secrets of that environment. A reviewer can approve access for the job if needed. |
| **Wait timers** | In situations where you want to delay code deployment, you can set a timer that can wait up to 30 days before deployment takes place. |

You should treat environment secrets with the same level of security as repository and organization secrets. Here are a few tips:

* **Do not use structured data as a secret.** Doing so can cause secret redaction within logs to fail.
* **Register the secrets you use in workflows.** This applies when a secret is used to generate another sensitive value. The value that the secret is to generate needs to be registered.
* **Use required reviewers to protect environment secrets.** This means that a workflow job cannot access environment secrets until approval is granted by a reviewer.

Remember these tips to safely use runners:

* Use self-hosted runners only on private repositories.
* Be cautious when using self-hosted runners on private or internal repositories. Anyone with read access can compromise the runner environment and gain access to secrets.

## Licensing considerations

It's GitHub's mission to be the home for all developers. That's why GitHub makes advanced security features such as dependency review, code scanning, and secret scanning automatically available free of charge for public repositories on GitHub.com. Open-source projects can benefit from these features. Smaller teams and individual developers have the opportunity to secure their development.

To use GitHub Advanced Security features on private repositories, you'll need a GitHub Advanced Security license (GHAS). GHAS is available with GitHub Enterprise Cloud or GitHub Enterprise Server. Each license for GHAS specifies a maximum number of accounts or seats.

As you survey your security requirements, prioritize in which repositories you'd like to run GitHub Advanced Security. Consider which codebases have the highest frequency of commits and are most critical to your company's success.

# Create an organizational culture around security

This module looks at how your organization can use GitHub Advanced Security features to improve security. You'll learn what it means to "shift left." You'll also develop an understanding of how shifting left helps secure your Software Development Life Cycle (SDLC). Finally, you'll review different security workflow models to identify best practices.

At the end of this unit, you'll understand how the right tools, combined with organizational culture, increase the security of your environments and codebases. This begins with adopting a security-focused mindset.

## Adopt a security mindset

The more security-centric your organizational culture is, the more everyone remembers to view their actions, work, and decisions through the lens of security. This perspective provides better outcomes for your software development and planning. However, a sound security strategy involves more than simply putting protective tools in place. You also have to account for human error. A user could accidentally enable an attack, or a developer could write insecure code. Think about how many attacks begin by tricking or forcing a member of an organization.

## Shift left

You may have heard that a company or development team is beginning to "shift left." To understand this phrase, you must first realize that security has commonly been an isolated topic in many software-development paradigms. The development lifecycle has figuratively begun "on the left" with design and development. Then the lifecycle ends to the right, bringing in quality control and deployment tasks. Workflows often include security only after the development stage. At that point, only a specific group would perform a security review.

In the SDLC, shifting left means adopting security practices early. When development shifts left, everyone considers security when they contribute to the code. They integrate security into the design phase, making it easier to write secure code and policies.

### Faulty security models

Developers that view security as an isolated consideration can struggle when quality assurance or security discovers errors in the project's code. The development team has to fix the code. At this point, they may have already begun a new project. Passing code back to be repaired results in two open work streams. This is an inefficient use of time and resources. Although some elements of a defective security model may be common, you should still work to avoid them.

This table outlines traits regularly found in faulty organizational security models:

|  |  |
| --- | --- |
| Facet | Comment |
| Scope of team involvement | Only a limited number of team members are responsible for security. The people building the application aren't the same ones involved in ensuring security. Security experts are outside of the development workflow. |
| Prioritization | Security reviews and quality control are separate steps at the end of the development stage. |
| Documentation | Security documentation is nonexistent. Written standards and guidelines haven't been established to granularly regulate what type of information enters the codebase and by whom. Subsequently, team members lack a common understanding of expectations and directives. |
| Platform configuration | The development environment isn't configured to align with and enforce established security guidelines. |

### The ideal security model

Ideally, everyone is responsible for securing the project within their scope of work. The team that writes the code is the same team performing tests and dealing with the results of those tests in real time. Policies exist for each phase of the operation to ensure code quality. Automation plays a significant role in creating an efficient workflow.

This table shows key elements of the ideal security model:

|  |  |
| --- | --- |
| **Facet** | **Comment** |
| Scope of team involvement | Teams shift left. Each role from design to deployment is concerned with security. |
| Prioritization | Throughout all stages of development, measures are in place to protect assets |
| Documentation | Standards and guidelines are in writing. Teams know the process to report bugs. |
| Tooling | The development platform enforces standards and keeps code and environments secure. |

### Security policies

A strong security culture needs documentation to define the people or roles that can perform specific actions within a repository or branch. In GitHub, these documents are policies. Policies may require a code review before merging. They ensure that the code passes certain status checks. For example, you may want to specify who can push code to or delete a branch. In GitHub, you'll store policy details in your repository in the SECURITY.md file. This file will also include instructions on reporting any bugs or vulnerabilities found in the project.

#### Enforce policies

GitHub Advanced Security includes compliance and policy-management features. You can use policies to stipulate standards and procedures that keep your code healthy. Policies also identify who to contact when an error is found.

Policies alone don't enforce your organization's requirements. Policies are organizational measures. Administrators ensure user adherence by setting up protected branches. The configuration of a branch allows you to force specific requirements. For example, tests such as code scanning must pass before merging changes. Another example is requiring a reviewer to approve a pull request before reviewers release it for merging.

# Responding to security alerts

At the beginning of this module, you read case studies that showed how serious security breaches can be. In the example of the 2017 Equifax breach, a lack of proper dependency management by administrators was a root cause of the breach. Humans failed to respond when they received information that a dependency needed to be fixed quickly.

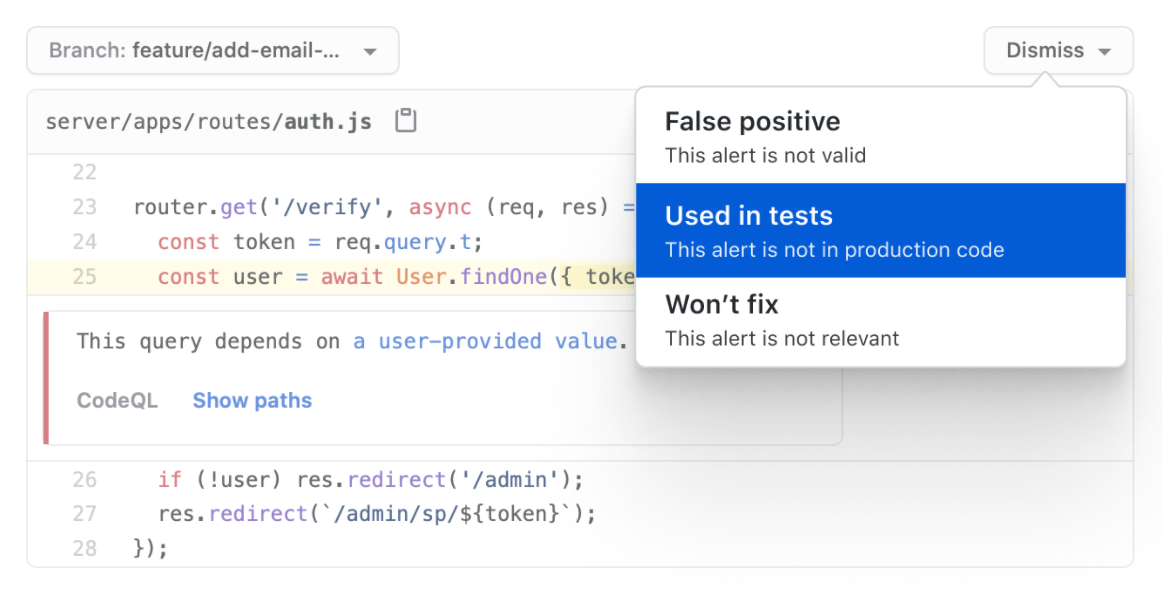
In this unit, you'll learn the importance of responding to security alerts and what that response should look like. You'll also see GitHub functionality that enables you to quickly react to these alerts.

## Evaluate security alerts

Although teams cannot always avoid errors, quick action reduces risks. Failing to act promptly leaves your system or project open to exploitation by bad actors. You can reduce costs, retain customers, and protect your brand reputation when you avoid sensitive data leaks by quickly responding to security alerts.

### Triage security alerts

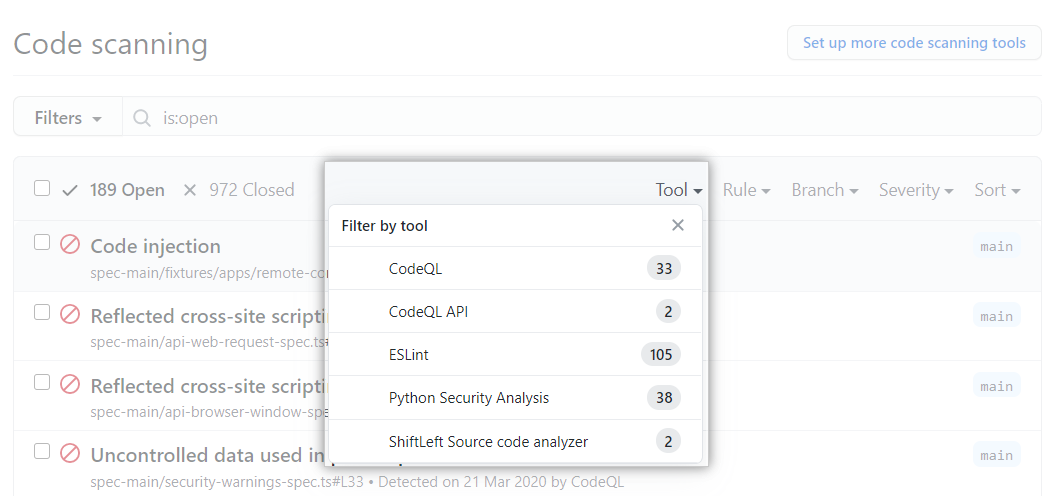
In the "What is GitHub Advanced Security" unit, you learned that GitHub's code scanning helps to identify vulnerabilities in your code. When the system identifies a problem, it notifies your team with alerts highlighting the issue. This helps you to quickly identify the issue. Once GitHub flags a vulnerability in your codebase, analyzing the flaw is the next step.



As shown in the preceding image, you control the code scanning process. If the scan has misunderstood your intent, you can dismiss the alert by identifying a reason for the dismissal. Indicate if the scan has highlighted a false positive, code used in tests, or an irrelevant alert.

You can also interact with scanning alerts through pull requests. You can get details related to the alert if you have write permission to the repository. A check must pass before you can merge the pull request.

### Manage security alerts



GitHub provides you with an overview of your repository alerts along with the triage functionality. This summary facilitates the prioritization of security alerts. Navigate to the security tab of your repository and select **Code scanning alerts**. Use the free-text search box or the drop-down menus to filter alerts.

GitHub can also suggest fixes for errors in your code. This functionality is more than a simple convenience; the direction saves you valuable time. The time you save goes toward your goal of quickly reacting to vulnerabilities.

## Use security advisories

Imagine a scenario where you find a vulnerability. You don't want to announce the vulnerability to the world. In situations like these, security advisories can help.

Think of security advisories as a private forum that code maintainers use to discuss bugs or vulnerabilities found within their projects. With security in mind, maintainers keep these exchanges in this forum instead of using the standard method of GitHub issues for their conversations. When you use security advisories, you ensure that knowledge of any error or bug is not exposed to the public before you can fix it.

After you resolve the problem, the security advisory is published back into the GitHub Advisory Database so that affected external parties affected can take any necessary steps.

# Summary

You now have a solid understanding of GitHub Advanced Security and how its features defend against gaps in security. This module also explained the importance of responding quickly to threats. You reviewed examples of best practices to keep your supply chain, codebase, and development environment safe. Now, you can start improving how teams work within your organization.

Security is about much more than passing a complete project to a security team. Security needs to be front of mind at each step of the software development lifecycle.

In this unit, you learned how to:

* Define GitHub Advanced Security.
* Identify the purpose of specific GitHub Advanced Security features.
* Understand the value of a security-focused team culture.
* Highlight the roles involved in securing your workflow.
* Recognize best practices for identifying and responding to security vulnerabilities.

## Learn more

Use the following resources to learn more about GitHub Advanced Security:

* [GitHub Security Overview](https://docs.github.com/en/code-security/security-overview/about-the-security-overview)
* [Dependency graphs](https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph)
* [GitHub Advisory Database](https://github.com/advisories)
* [Dependabot](https://docs.github.com/en/rest/dependabot)
* [Code scanning](https://docs.github.com/en/code-security/code-scanning)
* [Secret scanning](https://docs.github.com/en/code-security/secret-scanning)
* [CodeQL](https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning-with-codeql#about-code-scanning-with-codeql)
* [About billing for GitHub Advanced Security](https://docs.github.com/en/billing/managing-billing-for-github-advanced-security/about-billing-for-github-advanced-security)