Table of Contents

[Maintain a secure repository by using GitHub best practices 1](#_Toc132914220)

[Learning objectives 1](#_Toc132914221)

**Maintain a secure repository by using GitHub best practices**

Beginner

DevOps Engineer

Administrator

Developer

Solution Architect

GitHub

Azure

Learn best practices for building, hosting, and maintaining a secure repository on GitHub.

**Learning objectives**

In this module, you will:

* Identify the tools and GitHub features to establish a secure development strategy
* Enable vulnerable dependency detection for private repositories
* Detect and fix outdated dependencies with security vulnerabilities
* Automate the detection of vulnerable dependencies with Dependabot
* Add a security policy with a SECURITY.md file
* Remove a commit exposing sensitive data in a pull request
* Keep sensitive files out of your repository by applying the use of a .gitignore file
* Remove historical commits exposing sensitive data deep in your repository

**Introduction**

Software security is always an important topic, and it spans the entire software-development lifecycle. While a lot of focus is dedicated towards writing secure code and locking down infrastructure, it's also important to protect the processes that occur during every stage of the software-development lifecycle.

Suppose you're managing an important GitHub repository. You want to enforce the highest level of security, but also want to offer a welcoming experience for contributors. Unfortunately, being secure often introduces friction that hampers everyone's productivity. To mitigate this overhead, GitHub offers various automated features that allow you to efficiently administer a secure repository without slowing everyone down throughout the entire development process.

**Learning objectives**

In this module, you'll:

* Recognize the importance of securing your repository and shifting left in the development lifecycle
* Identify the tools, GitHub features, and best practices to establish a secure development strategy
* Keep sensitive files out of your repository by applying the use of a .gitignore file
* Identify how to detect and fix outdated dependencies with security vulnerabilities
* Recognize advanced security features such as code scanning and secret scanning

**Prerequisites**

* A GitHub account
* The ability to navigate and edit files in GitHub

# How to maintain a secure GitHub repository

Here, we'll discuss some of the essential security tools and techniques available to GitHub repository administrators.

Note

The following content does not cover the fundamentals of writing secure code, but rather important security considerations, tools, and features to use within a GitHub repository.

## The importance of a secure development strategy

It's now commonplace that security for applications is extremely important. News services frequently carry stories about some company's systems that have been breached, and private company and customer data that was stolen.

So, what are the issues to think about when planning a secure development strategy? Clearly, we need to protect information from being disclosed to people that shouldn't have access to it. But more importantly than that, we need to ensure that the information isn't inappropriately altered or destroyed, and that it's actually destroyed when it's supposed to be.

We need to make sure we properly authenticate who's accessing the data, and that they have the correct permissions to do so. Through historical or archival data or logs, we need to be able to find evidence when something has gone wrong.

There are many aspects to building and deploying secure applications. Here are three things to consider:

* **There is a general knowledge problem.** Many developers and other staff members assume they understand security, but they don't. Cybersecurity is a constantly evolving discipline. A program of ongoing education and training is essential.
* **Code must be created correctly and securely.** We need to be sure that the code is created correctly and securely implements the required features. We also need to make sure that the features were designed with security in mind in the first place.
* **Applications must comply with rules and regulations.** We need to make sure that the code complies with the rules and regulations that it's required to meet. We have to test for compliance while building the code and retest periodically, even after deployment.

### Security at every step



Security isn't something you can just add to an application or a system later. Secure development must be part of every stage of the software-development life cycle. This is even more important for critical applications and those that process sensitive or highly confidential information.

In practice, to hold teams accountable for what they develop, processes need to **shift left** earlier in the development lifecycle. By moving steps from a final gate at deployment time to an earlier step, fewer mistakes are made, and developers can move more quickly.

Application security concepts haven't been a focus for developers in the past. Apart from the education and training issues, it's because their organizations have emphasized fast development of features.

With the introduction of DevOps practices however, security testing is much easier to integrate into the pipeline. Rather than being a task performed by security specialists, security testing should just be part of the day-to-day delivery processes.

Overall, when the time for rework is taken into account, adding security to your DevOps practices earlier in the development lifecycle allows development teams to catch issues earlier and can actually reduce the overall time it takes to develop quality software.

Shifting left is a process change, but it isn’t a single control or specific tool; it’s about making all of your security more developer-centric and giving developers security feedback where they are.

Below are some ways to distribute security and operational responsibilities across all phases of the software-development lifecycle.

## Communicate a security policy with SECURITY.md

The community benefits of GitHub are substantial, but they also carry potential risks. The fact that anyone can propose bug fixes publicly comes with certain responsibilities. The most important is the responsible disclosure of information that could lead to security exploits before their underlying bugs can be fixed. Developers looking to report or address security issues look for a SECURITY.md file in the root of a repository in order to responsibly disclose their concerns. Providing guidance in this file will ultimately speed up the resolution of these critical issues.

To learn more about SECURITY.md, see [Adding a security policy to your repository](https://docs.github.com/code-security/getting-started/adding-a-security-policy-to-your-repository).

## GitHub Security Advisories

GitHub Security Advisories allow repository maintainers to privately discuss and fix a security vulnerability in a project. After collaborating on a fix, repository maintainers can publish the security advisory to publicly disclose the security vulnerability to the project's community. By publishing security advisories, repository maintainers make it easier for their community to update package dependencies and research the impact of the security vulnerabilities. GitHub stores the published advisories in the Common Vulnerabilities and Exposures (CVE) list, which is used for automatically notifying affected repositories that leverage software that has a listed vulnerability.

## Keep sensitive files out of your repository with .gitignore

It's easy for developers to overlook files included in a commit. Sometimes these overlooked files are benign, such as intermediate build files. However, there is always the risk that someone may inadvertently commit sensitive data, such as an API key or private configuration data, that a malicious actor could use. One technique to help avoid the majority of this risk is to build and maintain .gitignore files. These files instruct client tools, such as the git command line utility, to ignore paths and patterns when aggregating files for a commit.

The sample below illustrates some of the common use cases for ignoring files.

.gitignore

# User-specific files - Ignore all files ending in ".suo"

\*.suo

# Mono auto generated files - Ignore all files starting with "mono\_crash."

mono\_crash.\*

# Build results - Ignore all files in these folders found at any folder depth

[Dd]ebug/

[Rr]elease/

x64/

x86/

# Root config folder - Ignore this directory at the root due to leading slash

# Removing the slash would ignore "config" directories at all depths

/config

# Ignore intermediate JS build files produced during TypeScript build at any

# folder depth under /Web/TypeScript. This won't ignore JS files elsewhere.

/Web/TypeScript/\*\*/\*.js

Your repository may include multiple .gitignore files. Settings are inherited from parent directories, with overriding fields in new .gitignore files taking precedence over parent settings for their folders and subfolders. It's common for the majority of effort to go into maintaining the root .gitignore file where feasible, although adding a .gitignore into a project directory can be helpful when that project has specific requirements that are easier to maintain separately from the parent, such as files that should not be ignored.

To learn more about .gitignore, see [Ignoring files](https://docs.github.com/get-started/getting-started-with-git/ignoring-files). Also check out the collection of starter .gitignore files offered for various platforms in the [gitignore repository](https://github.com/github/gitignore?azure-portal=true).

## Remove sensitive data from a repository

While .gitignore can be useful in helping contributors avoid committing sensitive data, it's just a strong suggestion. Developers can still work around it to add files if they are motivated enough, and sometimes files may slip through because they don't meet the .gitignore configuration. Project participants should always be on the lookout for commits containing data that should not be included in the repository or its history.

Important

You should assume that any data committed to GitHub at any point has been compromised. Simply overwriting a commit isn't enough to ensure the data will not be accessible in the future. For the complete guide to removing sensitive data from GitHub, see [Removing sensitive data from a repository](https://docs.github.com/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository).

## Branch protection rules

You can create a [branch protection rule](https://docs.github.com/github/administering-a-repository/defining-the-mergeability-of-pull-requests/managing-a-branch-protection-rule) to enforce certain workflows for one or more branches, such as requiring an approving review or passing status checks for all pull requests merged into the protected branch.

You can leverage the workflows that protect the branch to:

* Run a build to verify the code changes can be build
* Run a linter to check for typos and conformation to the internal coding conventions
* Run automated tests to check for any behavior changes of the code
* And so on

## Add a CODEOWNERS file

By adding a [CODEOWNERS](https://docs.github.com/github/creating-cloning-and-archiving-repositories/creating-a-repository-on-github/about-code-owners#codeowners-syntax) file to your repository, you can assign individual team members or entire teams as code owners to paths in your repository. These code owners are then required for pull-request reviews on any changes to files in a path that they are configured for.

# Changes to files with the js extensions need to be reviewed by the js-owner user/group:

\*.js @js-owner

# Changes to files in the builds folder need to be reviewed by the octocat user/group:

/build/ @octocat

You can create the CODEOWNERS file in either the root of the repository, or in the docs or .github folder.

# Automated security

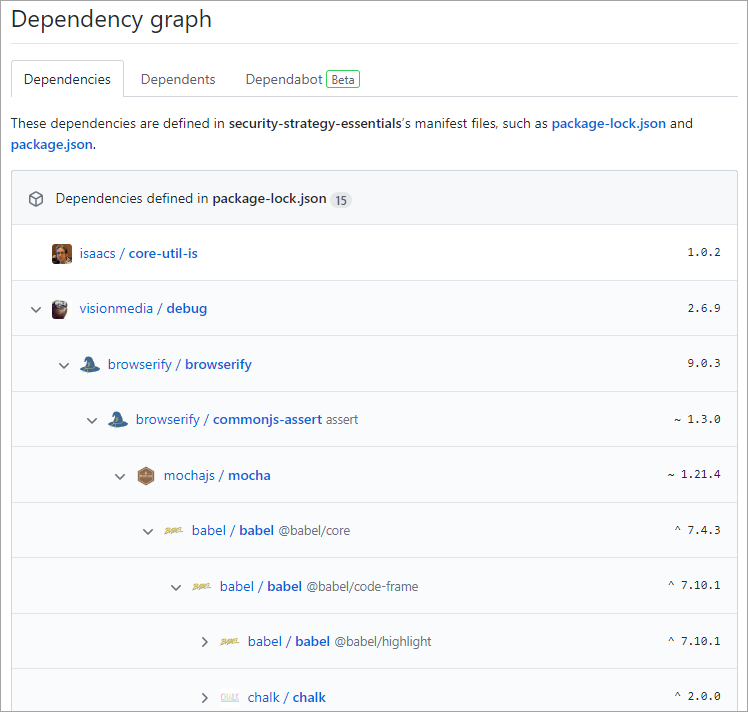
Here, we'll discuss some ways you can automate security checks in a repository that are available to GitHub repository administrators.

## Detect and fix outdated dependencies with security vulnerabilities

Virtually every project these days takes dependencies on external packages. While these components can offer substantial benefits in productivity, they can introduce other security risks. Staying on top of these packages and their vulnerability status can be time consuming, especially given how each dependency may have its own dependencies that can become difficult to track and maintain. Fortunately, GitHub provides features that reduce this workload.

### Repository dependency graphs

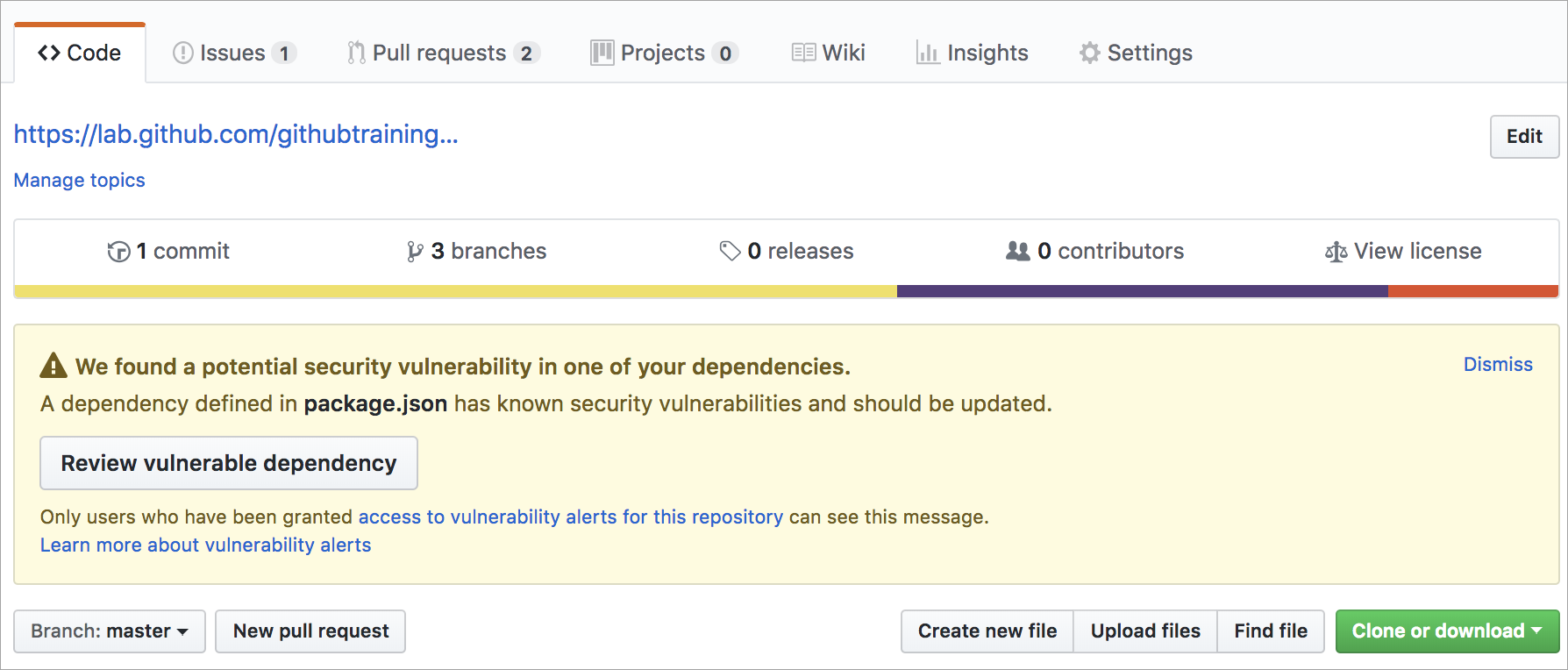
One default feature every repository enjoys is dependency graphs. GitHub scans common package manifests, such as package.json, requirements.txt, and others. These graphs allow project owners to recursively track all of the dependencies their project relies on.



For the list of supported dependency manifests, see [About the dependency graph](https://docs.github.com/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph).

### Automated dependency alerts

Even with a visual dependency graph, it can still be overwhelming to stay on top of the latest security considerations for every dependency a project has. To reduce this overhead, GitHub provides [automated dependency alerts](https://docs.github.com/code-security/supply-chain-security/managing-vulnerabilities-in-your-projects-dependencies/about-alerts-for-vulnerable-dependencies#dependabot-alerts-for-vulnerable-dependencies) that watch your dependency graphs for you. It then cross-references target versions with versions on known vulnerability lists. When a risk is discovered, the project is alerted. Input for the analysis comes from [GitHub Security Advisories](https://docs.github.com/code-security/security-advisories/about-github-security-advisories#dependabot-alerts-for-published-security-advisories).



### Automated dependency updates with Dependabot

Most of the time, a dependency alert leads to a project contributor bumping the offending package reference to the recommended version and creating a pull request for validation. Wouldn't it be great if there was a way to automate this effort? Well, good news! That's exactly what **Dependabot** does. It scans for dependency alerts and creates pull requests so that a contributor can validate the update and merge the request.

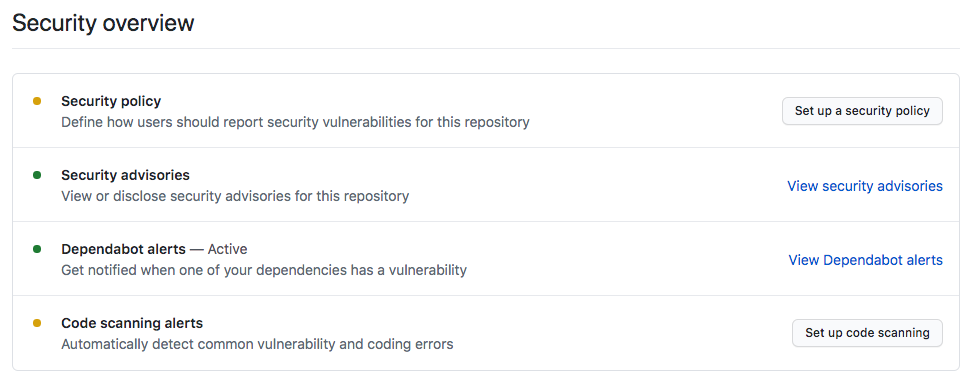
To learn more about Dependabot's flexibility, see [Configuring Dependabot security updates](https://docs.github.com/code-security/dependabot/dependabot-security-updates/configuring-dependabot-security-updates).

### Automated code scanning

Similar to how Dependabot scans your repository for dependency alerts, you can use code scanning to analyze and find security vulnerabilities and errors in the code in a GitHub repository. Code scanning has several benefits; you can use it to find, triage, and prioritize fixes for existing problems or potential security vulnerabilities. It's also useful to help prevent developers from introducing any new security problems into the code.

Another advantage to code scanning is its ability to use CodeQL. CodeQL lets you query code as data, which lets you create custom queries or use queries maintained by the open-source community. Code scanning gives you the freedom to customize and maintain how the code within your repository is being scanned.

You can enable code-scanning alerts and workflows in the security tab of a GitHub repository:



Learn more about [Code scanning and CodeQL](https://docs.github.com/free-pro-team@latest/github/finding-security-vulnerabilities-and-errors-in-your-code/about-code-scanning#about-code-scanning).

### Secret scanning

Another automated scanning feature within a GitHub repository is secret scanning. Similar to the previous security scanning features, secret scanning looks for known secrets or credentials committed within the repository. This scanning is done to prevent the use of fraudulent behavior and to secure the integrity of any sensitive data. By default, secret scanning occurs on public repositories and can be enabled on private repositories by repository administrators or organization owners.

When secret scanning detects a set of credentials, GitHub notifies the service provider who issued the secret. The service provider validates the credential, then decides whether they should revoke the secret, issue a new secret, or reach out to you directly, which will depend on the associated risks to you or the service provider.

Learn more about [Secret scanning for public and private repositories](https://docs.github.com/free-pro-team@latest/github/administering-a-repository/about-secret-scanning).

# Summary

In this module, we talked about the importance of securing and maintaining a GitHub repository.

You learned about:

* The importance of securing your repository and shifting left in the development lifecycle
* Security features and best practices within a GitHub repository
* Detection of outdated dependencies with security vulnerabilities
* How to add a .gitignore file to a repository
* Advanced security features such as code scanning and secret scanning

Now that you're familiar with security best practices, learn to [Automate DevOps processes by using GitHub Apps](https://learn.microsoft.com/en-us/training/modules/automate-devops-github-apps/).

## Learn more

Here are some links to more information on the topics we discussed in this module.

* [Viewing and updating Dependabot alerts](https://docs.github.com/code-security/dependabot/dependabot-alerts/viewing-and-updating-dependabot-alerts)
* [Dependabot official site](https://github.com/dependabot)
* [Security apps on GitHub Marketplace](https://github.com/marketplace/category/security?azure-portal=true)
* [Adding a security policy to your repository](https://docs.github.com/code-security/getting-started/adding-a-security-policy-to-your-repository)
* [Ignoring files](https://docs.github.com/get-started/getting-started-with-git/ignoring-files)
* [Removing sensitive data from a repository](https://docs.github.com/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository)