Table of Contents

[Manage sensitive data and security policies within GitHub 2](#_Toc132915021)

[Learning objectives 2](#_Toc132915022)

[Prerequisites 2](#_Toc132915023)

[Introduction 3](#_Toc132915024)

[What will you be doing? 4](#_Toc132915025)

[Prerequisites 4](#_Toc132915026)

[Setting security policies 4](#_Toc132915027)

[Documentation 5](#_Toc132915028)

[SECURITY.md 6](#_Toc132915029)

[Other default community health files 7](#_Toc132915030)

[Security settings 7](#_Toc132915031)

[Change settings according to a trust and control position 7](#_Toc132915032)

[At what levels can administrators change security settings? 8](#_Toc132915033)

[What kinds of security settings are available to administrators? 9](#_Toc132915034)

[Security advisories 10](#_Toc132915035)

[Scrub sensitive data from a repository 12](#_Toc132915036)

[Tools to avoid committing sensitive data 12](#_Toc132915037)

[Scrub the repository 13](#_Toc132915038)

[Use the BFG Repo-Cleaner tool 14](#_Toc132915039)

[Use git filter-repo tool 14](#_Toc132915040)

[Request help from GitHub support 17](#_Toc132915041)

[Reporting and logging 19](#_Toc132915042)

[What are log records? 19](#_Toc132915043)

[The GraphQL API 19](#_Toc132915044)

[The Rest API 19](#_Toc132915045)

[Generate reports for auditing 20](#_Toc132915046)

[Summary 21](#_Toc132915047)

[Learn more 21](#_Toc132915048)

**Manage sensitive data and security policies within GitHub**

Familiarize yourself with GitHub's basic security tools, which prepare repositories for secure development and industry-standard response to threats.

**Learning objectives**

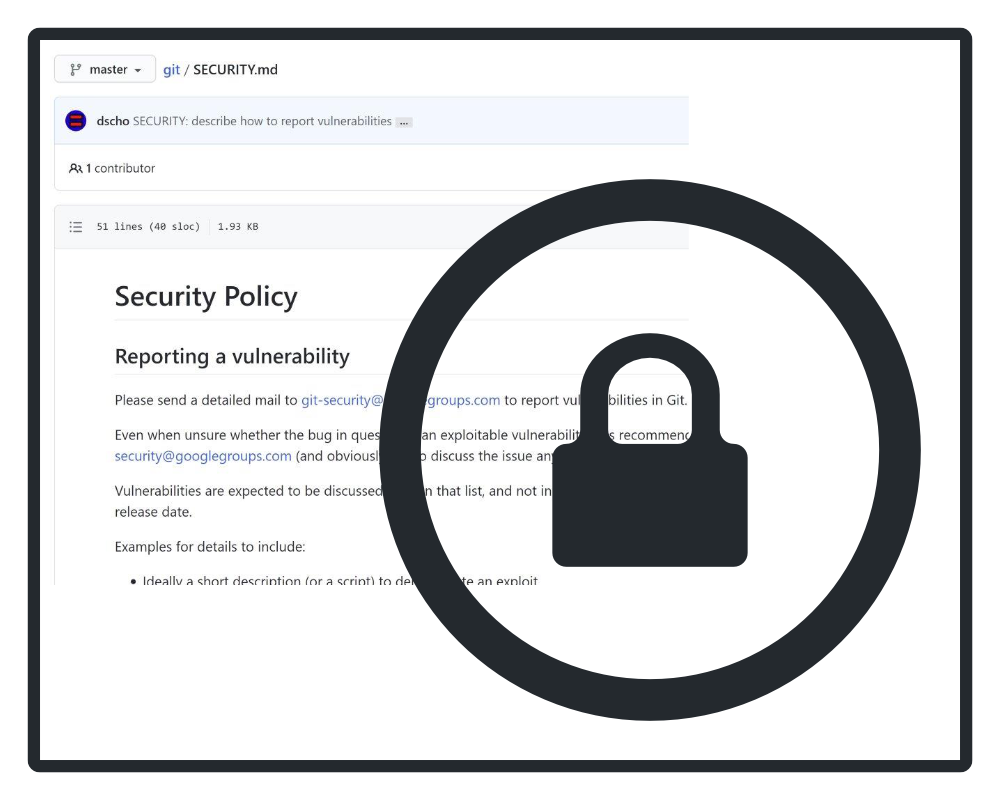
In this module, you'll learn how to:

* Create documentation that details security guidelines and useful information for collaborators.
* Set permissions and other rules.
* Automate processes that prevent security breaches.
* Respond to security breaches.

**Prerequisites**

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

**Introduction**



GitHub security features help to secure your organization's repositories and protect the code and secrets in your repositories.

You need know how to protect against security breaches while collaborating in a GitHub organization, and what to do in the event of a breach. GitHub provides tools to organization members and administrators to accomplish these goals, making it easy for you to guarantee the reputation of your projects and avoid costly mitigation efforts.

Imagine you've created a new GitHub repository in which your organization will work on a brand new project. In the past, projects collected technical debt or risk when access credentials appeared in the code base or when users worked on files outside the scope of their usual roles. It was also hard to know who to contact to resolve the issues when they were identified. As an administrator, you can employ built-in GitHub tools to ensure that users only make changes appropriate to their role, and that your organization protects its sensitive data from unauthorized access.

In this module, you'll familiarize yourself with GitHub's basic security tools, which prepare repositories for secure development and industry-standard response to threats. You'll also act as a repository or organization administrator to tailor documents and settings to your organization's needs.

**What will you be doing?**

You'll create and manipulate security policies and settings to ensure safe and easy collaboration for a new repository.

* **Create** documentation that details security guidelines and useful information for collaborators.
* **Set** permissions and other rules.
* **Automate** processes that prevent security breaches.
* **Respond** to security breaches.

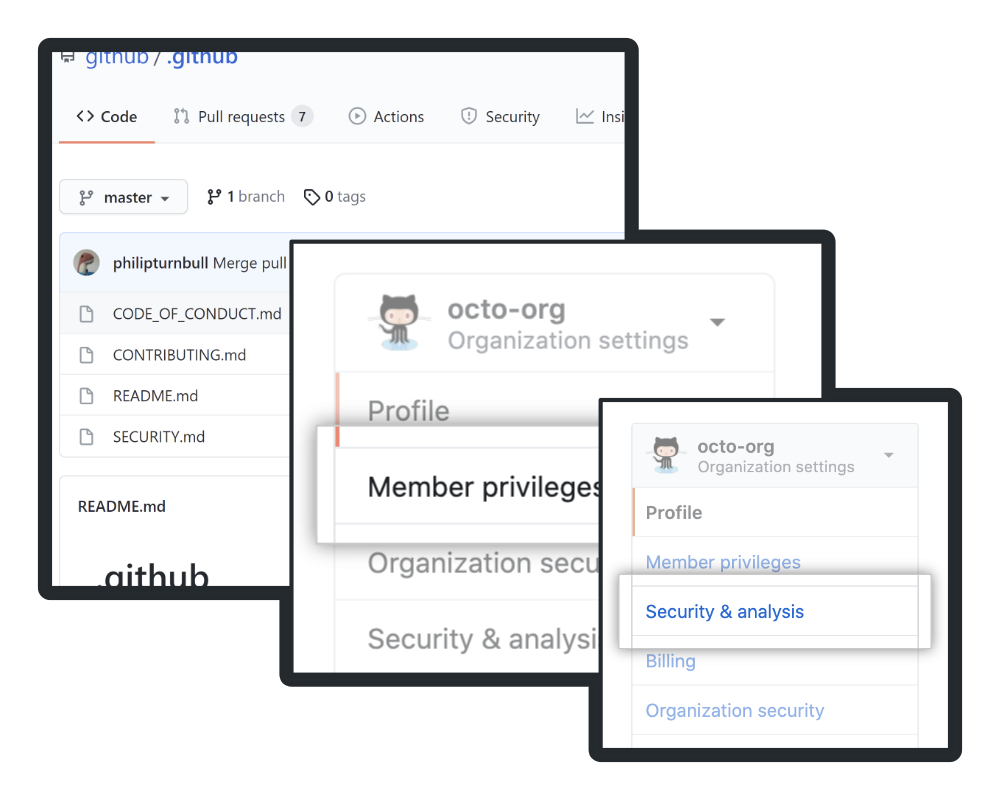
**Prerequisites**

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

# Setting security policies

In this unit, you'll learn more about the preventive measures you can take to maintain the health of your GitHub repositories.

Suppose you're an administrator who is helping to onboard many new collaborators to your organization. You need to make sure they contribute to the proper repositories and have easy access to assistance if they discover a security threat. To do this, you set security policies.



A repository, organization, or enterprise's security policies are the practices and settings that maintain the integrity of users' data and systems. A security policy defines what it means for a system, organization or other entity to be secure, and limits permissions to support that definition. What does that mean for you and your GitHub ecosystem? When you create security policies, you are telling collaborators how to work in ways that maintain code health. Security policies also provide resources for responding to security vulnerabilities, and limit collaborators' access.

As an administrator, you can set some security policies individually at each level of your control (repo, organization, and enterprise). For instance, you can give individual organization owners permission to allow collaborator write-access or other kinds of access, or as the admin, you can require all organizations in an enterprise allow only a certain kind of access.

Documentation and security settings serve three broad purposes.

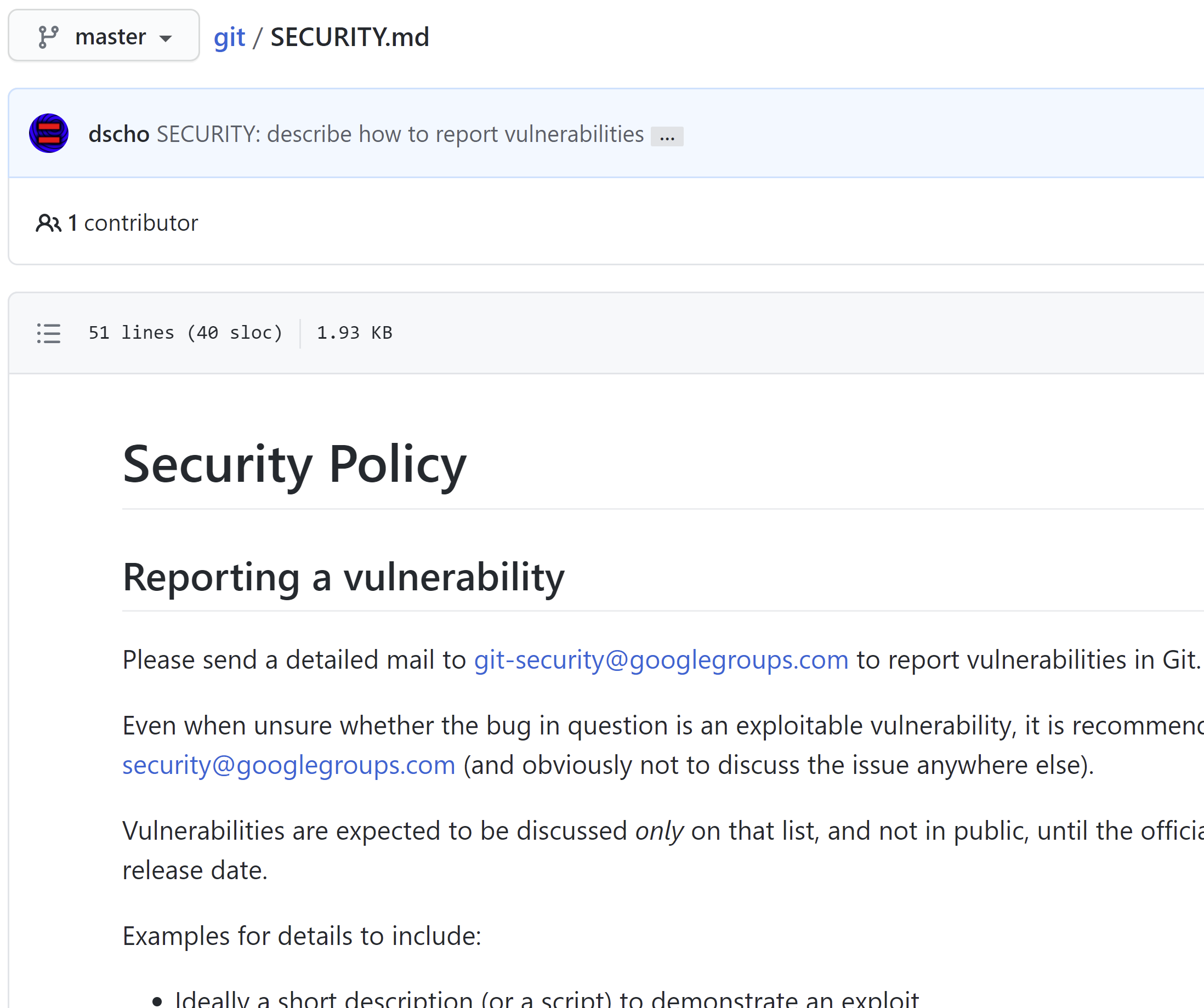
* Standardization: security policies ensure that everyone on your project responds in the same way to common issues, making it easier to know what is happening and why.
* Compliance: the countries/regions where you plan to develop and publish your projects may require that you follow industry-standard practices in coding and recording security events.
* Prevention of critical failure: most importantly, solid policies defend against events that threaten continued development, like publishing trade secrets or intellectual property.

## Documentation

As an administrator working on the team responsible for setting up a new repository, the first thing you should do is create certain default community health files. GitHub assigns special significance to these files that makes it easy for collaborators to find them. In this section, you'll learn about the characteristics of these key files.

If you've ever contributed to or looked at an open source project, you'll quickly be able to tell if that project has a healthy contribution level. This is directly related to how closely the project follows recommended community standards, which include a list of recommended documentation files like a README, CODE\_OF\_CONDUCT.md, or CONTRIBUTING.md file. Just like an open source project with documentation on how to best contribute, your repository, organization, or enterprise benefits from a list of recommended documentation on your specific security policies.

These files record information about security preparation, key contacts, and the project background. They tell collaborators and other users what measures you've taken to protect them, what to do or who to call in the event of a security issue, and provide guidelines for contributing to the project.



### SECURITY.md

SECURITY.md is the primary document for communicating security information. It is a Markdown file in a repository's root, docs, or .github folder. The SECURITY.md file should include:

* A list of supported versions of the project
* A way to report a security vulnerability

It may also include:

* Information about the project's compliance with key privacy and security laws
* Technologies that administrators and stakeholders use to secure information
* Known risks

### Other default community health files

A user may add other community health documentation at the organization level or in their user account. GitHub supports the following community health documentation:

* CODE\_OF\_CONDUCT.md
* CONTRIBUTING.md
* FUNDING.yml
* Issue and pull request templates and config.yml
* SECURITY.md
* SUPPORT.md

GitHub considers files of these types to have specific purpose, and requires you to follow the listed naming scheme when creating or updating them.

GitHub will use and display these default files for any repository owned by the account that does not have its own file of that type in any of the following places:

* The root of the repository
* The .github folder
* The docs folder

## Security settings

The other component of creating robust security policies is taking advantage of GitHub's built-in security settings and features. Imagine you're onboarding collaborators whose scope of work will vary--some will be part of focused teams to implement a feature, while others will be responsible for watching over the code base for issues, and a few others may need to assist you with administrative duties. In this section, you'll learn about settings that define user permissions and allow automation of common security tasks.

### Change settings according to a trust and control position

Every organization has a trust and control position: circumstances that determine how much trust you can safely extend to individual collaborators and teams, and how much control you'll need to maintain over basic permissions.

If your organization is a new business with a small team, it'll have few moving parts and few areas of potential security vulnerability. After all, when team members work in the same office or in nearby time zones, it's easy to identify who can take specific actions and how to contact them. In this case, you can safely trust most or all collaborators with high levels of access and capability.

On the other hand, if your organization is very large, with teams in many separate time zones or continents, determining who to contact to resolve incidents can be difficult. Even getting in touch with the right person once you've identified them may require significant planning. This means that team leaders are in a position to trust fewer people with high levels of access and capability, because troubleshooting problems takes more time as the team extends privileges to more users.

### At what levels can administrators change security settings?

Before we go further, you should know that you can configure security settings at the organization and enterprise level. These choices are also part of "organization policies" and "Enterprise policies"; choosing settings for individual organizations or for the entire enterprise goes hand in hand with creating the proper documentation as discussed previously.

Here's an example of a basic security setting you could change at the organization level or for a whole enterprise.

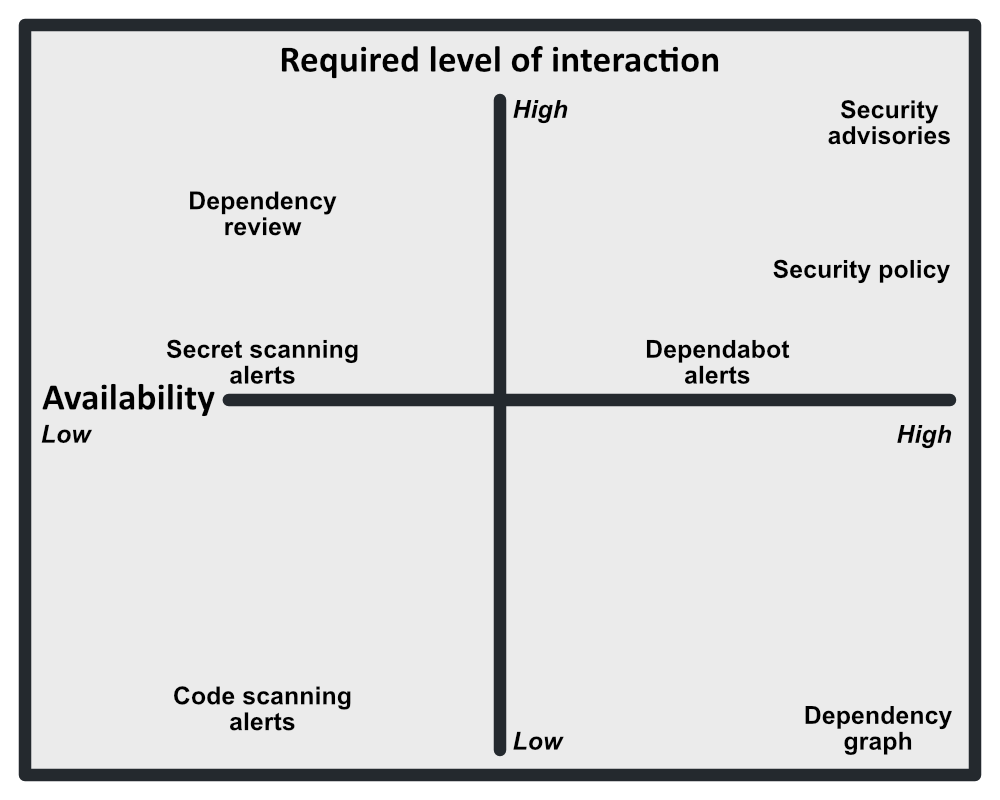
* You may choose to set base permissions for all members of one organization working in any repository owned by that organization. Select **Settings** in **Your organizations** to give collaborators a uniform level of access ("None," "Read," "Write," or "Admin"). With this setting, one organization's members could have read access to its repositories, while another organization's members could have write access.
* Alternately, you could apply that setting to every organization covered by the Enterprise plan. Select an enterprise from **Your enterprises** and click **Policies**. The **Repository policies** tab enables you to review the base permissions for every user in every organization in the enterprise, and change them to "None," "Read," "Write," or "Admin." With this setting, only specific individuals to whom you grant special privileges will be able to exceed the base permission level.

As you can see, settings that Enterprise administrators enforce cascade down to all organizations covered by the GitHub Enterprise plan, while settings not covered by Enterprise administrators are free to be customized by organization administrators. Only GitHub Enterprise administrators and organization owners can configure organization security settings.

Changing security settings at the organization level or for all organizations covered by the Enterprise plan is powerful, because it can standardize user capabilities during unusual circumstances. In the example above, you may need to restrict capabilities to all but a few users for an organization--or even for the whole enterprise--in response to a security threat. In contrast, you could temporarily allow greater capabilities to all users in an organization during a rare development effort where you need help from extra personnel.

Note

Available settings and tools differ based on the type of repository. In addition, these settings and features differ in their level of required user interaction.



### What kinds of security settings are available to administrators?

Access restrictions, security documentation, advisories, Dependabot alerts and security updates, Dependabot version updates, and the GitHub dependency graph are available for all repositories. Documentation and advisories require the most significant manual interaction, but applying Dependabot to your code base automates parts of the security process, up to and including updating dependencies.

Code scanning alerts, secret scanning alerts, and dependency review provide further automation to the security process. Enabling these GitHub features will flag vulnerabilities in code submitted to a repository, highlighting suspicious code. However, these features are only available for private repositories with an Advanced Security license or public repositories.

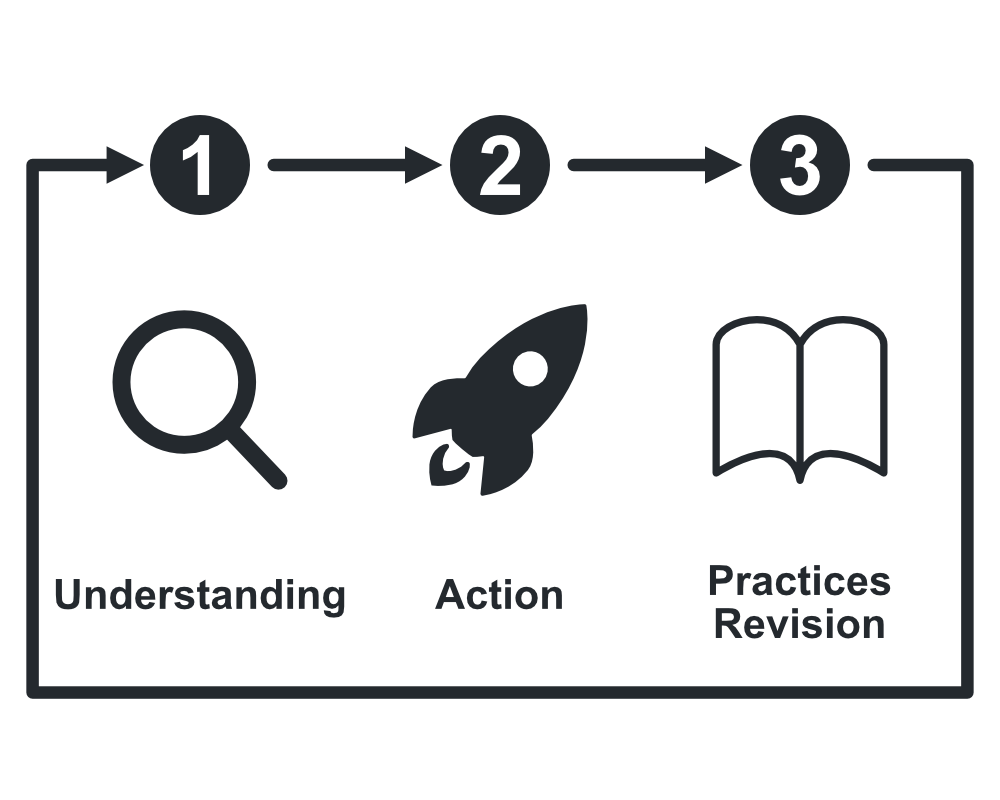
If you have a security vulnerability, you can create a security advisory to privately discuss and fix the vulnerability.

## Security advisories

You've been vigilant in your efforts to maintain healthy code, establishing clear policies and enacting settings to help collaborators work within their scope. But despite your team's efforts, someone identified a vulnerability in published code. This happens to every team sooner or later--no one is perfect.

When you identify a security threat, your team's response will go beyond patching offending sections of code. In this section, you'll learn the basics of the GitHub security advisory tools that allow you to draft and publish comprehensive documentation on the nature of the threat.

GitHub security advisories provide a historical narrative for users that demonstrates that you understand the nature of the threat, have addressed the immediate threat, and have taken steps to prevent similar threats in the future.



In service of these goals, a security advisory should be comprehensive and include following information:

* Product and versions affected
* Severity
* Types of security weaknesses addressed by the project owners' actions
* Impact, status of patches, and workarounds

Administrators are responsible for publishing security advisories. GitHub's system for publishing security advisories guides the administrator through the process of writing comprehensive documentation.

Check if the vulnerability you're patching matches an existing entry on the Common Vulnerabilities and Exposures list, or if it fits into a Common Weakness Enumeration category. GitHub's tools for creating security advisories are geared toward presenting this information, because it allows users to quickly understand the security vulnerabilities they're facing.

# Scrub sensitive data from a repository

As a GitHub organization administrator, you may need to remove, or scrub, sensitive data from a repository. You may also need to take additional security steps, like changing security policies in response to an incident.

You may remember from the scenario in the introduction that some of the most stressful and risky problems can involve sensitive data appearing in the code base. As you administer a new project with unfamiliar collaborators, you know that it is important to prevent these kinds of security risks from happening as much as possible. You also know that these problems will happen sooner or later, and it is important to be prepared, in order to minimize time spent resolving the issue and up-front costs (like paying a ransomware actor).

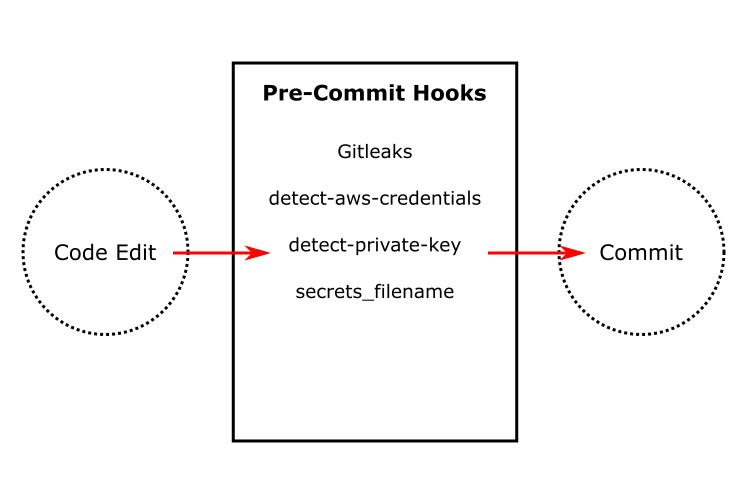
In this unit, you'll learn about the tools to help prevent committing sensitive data. You'll also learn what to do if there is a security breach, and how to evaluate potential changes to security policies.

## Tools to avoid committing sensitive data

You know that your new software build will need to authenticate with external services in order to test functions. This could lead to the most common type of accidental commission of sensitive data: including login credentials in the code base.

This section discusses the best way to deal with interactions between sensitive data and your code base: prevention. Prevention is key, because it can form part of a routine, and your team can automate routines.

It's possible to automate this process with freely available pre-commit hooks (scripts that Git executes prior to the commit event) that scan for sensitive data.



Some of the most popular pre-commit hooks that serve this purpose include:

* Gitleaks
* pre-commit plugins
  + detect-aws-credentials
  + detect-private-key
  + secrets\_filename

## Scrub the repository

This section explains the two primary methods for removing unwanted files from a repository's history: the git filter-repo command and the BFG Repo-Cleaner open source tool. It also covers what GitHub support technicians can do if you determine that high-risk data has made its way into a repository.

Suppose a collaborator was testing the software build's interactions with an external service, but the service required an unforeseen method of authentication. What if the user accidentally included credentials in the code base that pre-commit hooks weren't prepared to catch? You would need to take action.

Note

Whether you choose git filter-repo or the BFG to remove sensitive data from a repository, a user that pushes a commit to GitHub containing sensitive data has already crossed an important line; the organization should consider that data to be compromised, and change it immediately.

Both git filter-repo and the BFG have tradeoffs. git filter-repo is powerful, allowing the removal of passwords or renaming of specific files or file sets. However, its syntax is complex and it has strong risk of creating unforeseen problems in a repository's integrity, especially if you're running the Windows operating system. In contrast, the BFG Repo-Cleaner open source tool is extremely efficient and easy to use, because it provides you with a set of default actions to choose from that match many common use cases related to sensitive data. However, the BFG tool is limited in its capabilities to these use cases, giving you little opportunity to customize for unusual circumstances.

[The following material appears in the GitHub Doc "Removing sensitive data from a repository."](https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository)

The git filter-repo command and the BFG Repo-Cleaner rewrite your repository's history, which changes the SHAs for existing commits that you alter and any dependent commits. Changed commit SHAs may affect open pull requests in your repository. We recommend merging or closing all open pull requests before removing files from your repository.

You can remove the file from the latest commit with git rm.

### Use the BFG Repo-Cleaner tool

To remove your file with sensitive data and leave your latest commit untouched, run:

shell

$ bfg --delete-files YOUR-FILE-WITH-SENSITIVE-DATA

To replace all text listed in passwords.txt wherever it can be found in your repository's history, run:

shell

$ bfg --replace-text passwords.txt

After the sensitive data is removed, you must force push your changes to GitHub.

shell

$ git push --force

See the [BFG Repo-Cleaner](https://rtyley.github.io/bfg-repo-cleaner/)'s documentation for full usage and download instructions.

### Use git filter-repo tool

Warning

If you run git filter-repo after stashing changes, you won't be able to retrieve your changes with other stash commands. Before running git filter-repo, we recommend unstashing any changes you've made. To unstash the last set of changes you've stashed, run git stash show -p | git apply -R

To illustrate how git filter-repo works, we'll show you how to remove your file with sensitive data from the history of your repository and add it to .gitignore to ensure that it is not accidentally re-committed.

1. Install the latest release of the [git filter-repo](https://github.com/newren/git-filter-repo) tool. You can install git-filter-repo manually or by using a package manager. For example, to install the tool with HomeBrew, use the brew install command.

brew install git-filter-repo

For more information, see [INSTALL.md](https://github.com/newren/git-filter-repo/blob/main/INSTALL.md) in the newren/git-filter-repo repository.

1. If you don't already have a local copy of your repository with sensitive data in its history, [clone the repository](https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/cloning-a-repository-from-github/cloning-a-repository) to your local computer.

 $ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY

> Initialized empty Git repository in /Users/YOUR-FILE-PATH/YOUR-REPOSITORY/.git/

> remote: Counting objects: 1301, done.

> remote: Compressing objects: 100% (769/769), done.

> remote: Total 1301 (delta 724), reused 910 (delta 522)

> Receiving objects: 100% (1301/1301), 164.39 KiB, done.

> Resolving deltas: 100% (724/724), done.

 Navigate into the repository's working directory.

$ cd YOUR-REPOSITORY

 Run the following command, replacing PATH-TO-YOUR-FILE-WITH-SENSITIVE-DATA with the path to the file you want to remove, not just its filename. These arguments will:

* Force Git to process, but not check out, the entire history of every branch and tag
* Remove the specified file, as well as any empty commits generated as a result
* Remove some configurations, such as the remote URL, stored in the .git/config file. You may want to back up this file in advance for restoration later.
* Overwrite your existing tags

 $ git filter-repo --invert-paths --path PATH-TO-YOUR-FILE-WITH-SENSITIVE-DATA

Parsed 197 commits

New history written in 0.11 seconds; now repacking/cleaning...

Repacking your repo and cleaning out old unneeded objects

Enumerating objects: 210, done.

Counting objects: 100% (210/210), done.

Delta compression using up to 12 threads

Compressing objects: 100% (127/127), done.

Writing objects: 100% (210/210), done.

Building bitmaps: 100% (48/48), done.

Total 210 (delta 98), reused 144 (delta 75), pack-reused 0

Completely finished after 0.64 seconds.

Note

If the file with sensitive data used to exist at any other paths (because it was moved or renamed), you must run this command on those paths, as well.

 Add your file with sensitive data to .gitignore to ensure that you don't accidentally commit it again.

 $ echo "YOUR-FILE-WITH-SENSITIVE-DATA" >> .gitignore

$ git add .gitignore

$ git commit -m "Add YOUR-FILE-WITH-SENSITIVE-DATA to .gitignore"

> [main 051452f] Add YOUR-FILE-WITH-SENSITIVE-DATA to .gitignore

> 1 files changed, 1 insertions(+), 0 deletions(-)

 Double-check that you've removed everything you wanted to from your repository's history, and that all of your branches are checked out.

 Once you're happy with the state of your repository, force-push your local changes to overwrite your GitHub repository, as well as all the branches you've pushed up:

 $ git push origin --force --all

> Counting objects: 1074, done.

> Delta compression using 2 threads.

> Compressing objects: 100% (677/677), done.

> Writing objects: 100% (1058/1058), 148.85 KiB, done.

> Total 1058 (delta 590), reused 602 (delta 378)

> To https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git

> + 48dc599...051452f main -> main (forced update)

 In order to remove the sensitive file from [your tagged releases](https://docs.github.com/en/github/administering-a-repository/releasing-projects-on-github/about-releases), you'll also need to force-push against your Git tags:

 $ git push origin --force --tags

> Counting objects: 321, done.

> Delta compression using up to 8 threads.

> Compressing objects: 100% (166/166), done.

> Writing objects: 100% (321/321), 331.74 KiB | 0 bytes/s, done.

> Total 321 (delta 124), reused 269 (delta 108)

> To https://github.com/YOUR-USERNAME/YOUR-REPOSITORY.git

> + 48dc599...051452f main -> main (forced update)

### Request help from GitHub support

GitHub support technicians can assist you in removing confidential information that poses a security risk to you or your organization from a repository that you or another organization owns. For instance, you might have removed access credentials from your organization's repository, but you need to make sure they don't appear in forks of that repository (you should also change the access credentials to invalidate them as soon as possible).

#### Private information removal requests are appropriate for:

* Access credentials, such as user names combined with passwords, access tokens, or other sensitive secrets that can grant access to your organization's server, network, or domain.
* AWS tokens and other similar access credentials that grant access to a third party on your behalf. You must be able to show that the token does belong to you.
* Documentation (such as network diagrams or architecture) that poses a specific security risk for an organization.
* [Information](https://docs.github.com/en/github/site-policy/github-community-guidelines#doxxing-and-invasion-of-privacy) related to, and posing a security risk to, you as an individual (such as social security numbers or other government identification numbers).

GitHub needs you to fulfill other conditions before asking for the help of support technicians.

* **Ask Nicely First.** When your request to support will affect a repository your organization doesn't own, get in touch with the repository's owner to ask them to remove the data.
* **No Bots.** You should have a trained professional evaluate the facts of every request you send. No party making requests of this type to GitHub support should use automated bots, whether they're direct representatives of your organization or third parties to which you have outsourced the handling of these requests.
* **Send In The Correct Request.** This process works best when you only send requests related to the types of private information listed above, and when they're the only requests from you that GitHub is handling at the time.
* **Processing Time.** If you make a request to GitHub to remove private information, please be patient and send only one request for each discrete issue.

#### Request requirements

When you're ready to make your request, assemble the following information.

1. A working, clickable link to each file containing private information. (Note that we're not able to work from search results, examples, or screenshots.)
2. Specific line numbers within each file containing the private information.
3. A brief description of how each item you've identified poses a security risk to you or your organization. **It is important that you provide an explanation of how the data poses a security risk beyond merely stating that it does.**
4. If you are a third party acting as an agent for an organization facing a security risk, include a statement that you have a legal right to act on behalf of that organization.
5. OPTIONAL: Let us know if your request is particularly urgent, and why. We respond to all private information removal requests as quickly as possible. However, if this request is especially time-sensitive, such as a very recent credential exposure, please explain why.

**Reporting and logging**

Sometimes you need more details about actions performed in your organization, like who performed the action, when, and how. You may also need to check who deleted a repository, or when a workflow ran? These pieces of information can be helpful in situations where something went wrong in your organization after a specific event.

Here you'll learn what your organization's audit log records, and how to access and export it.

**What are log records?**

Your organization's log records actions taken by organization members. The log is available to organization owners, and records information about actions that affect the organization including:

* The repository in which the action was performed
* The user that performed the action
* The action that was performed
* Which country/region in which the action took place
* The date and time of the action

Access the audit log through GitHub.com, GitHub Enterprise Server, or GitHub AE to review actions from the past 90 days. However, interacting with the audit log using either the GraphQL API or the Rest API can allow easy retrieval of specific information types, with additional limitations.

**The GraphQL API**

The GraphQL API is available for organizations using GitHub Enterprise, and can retrieve information about actions up to 120 days old. It can monitor:

* Access to your organization or repository settings
* Changes in permissions
* Added or removed users in an organization, repository, or team
* Users being promoted to admin
* Changes to permissions of a GitHub App

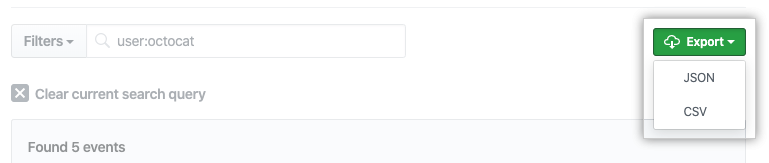
It cannot provide information about Git events.

**The Rest API**

The Rest API is available for organizations using GitHub Enterprise Cloud, and can retrieve information about actions up to 90 days old. It can monitor the same actions as the GraphQL API, as well as Git events. However, information about Git events only remains in the log for 7 days.

**Generate reports for auditing**

For compliance and record-keeping purposes, you'll want to export your audit logs in a common format. This is a simple process that you can complete by accessing the **Audit log** from the **Settings** sidebar in **Your organizations** and choosing the **Export** drop-down menu.



If you want to export an audit log that focuses on specific events, like those enacted by one user or that happened on a given day, you can filter the audit log results by adding a qualifier and accompanying value to the **Filters** field next to the **Export** drop-down menu.

| **Qualifier** | **Example value** |
| --- | --- |
| action | team.create |
| actor | octocat |
| user | codertocat |
| org | octo-org |
| repo | octo-org/documentation |
| created | 2019-06-01 |

# Summary

The goal of this module was to help you understand how to protect your GitHub organization from security breaches as well as what to do in the event of a breach.

You learned how to:

* Create documentation to outline security guidelines and provide useful information for collaborators
* Set permissions and rules
* Automate processes that prevent security breaches.
* Respond to security breaches

You reviewed GitHub's basic security tools including the documents and settings available to secure your GitHub organization when working with collaborators.

This module explained how to create a new GitHub repository in a way that limits technical debt and risk. You also learned how to use built-in GitHub tools to manage user permissions and prevent unauthorized access.

## Learn more

The following are links related to the information covered in this module:

* [Adding a security policy to your repository.](https://docs.github.com/en/code-security/getting-started/adding-a-security-policy-to-your-repository)
* [Creating a default community health file](https://docs.github.com/communities/setting-up-your-project-for-healthy-contributions/creating-a-default-community-health-file#supported-file-types)
* [About GitHub Security Advisories](https://docs.github.com/code-security/security-advisories/about-github-security-advisories)
* [Creating a security advisory](https://docs.github.com/code-security/security-advisories/creating-a-security-advisory)
* [GitHub security features](https://docs.github.com/code-security/getting-started/github-security-features)
* [About GitHub security advisories.](https://docs.github.com/code-security/security-advisories/about-github-security-advisories)
* [Reference for working with an audit log using the GraphQL API.](https://docs.github.com/en/graphql)
* [Reference for working with an audit log using the Rest API.](https://docs.github.com/en/rest)
* [git-filter-repo Manual Page](https://htmlpreview.github.io/?https://github.com/newren/git-filter-repo/blob/docs/html/git-filter-repo.html)