

Understanding and Using GitHub Security

Revision 1.0 – 07/08/24
Tech Skills Transformations LLC / Brent Laster

Lab 1 – Getting Started

Purpose: In this lab, we'll get a quick start learning about some general security settings and authentication with Personal Access Tokens

1. Log in to GitHub with your primary GitHub account.
2. Go to <https://github.com/skillrepos/sec-demo> and fork that project into your own GitHub space. Do this by clicking on the **Fork** button. On the next screen, make sure to uncheck the **Copy the main branch only** checkbox and then click the **Create Fork** button.

3. Now you'll be on your fork of the repo. Let's take a look at a couple of developer settings for security. Click on the *icon for your userid* in the upper right of the screen and then click on **Settings** near the bottom of the list that pops up. (Alternatively, you can go to [github.com/settings/profile directly](https://github.com/settings/profile).)

4. In the **Settings** screen for your userid, let's look at a few items related to security at the user level. First, select **the SSH and GPG keys** entry on the left. This is where you can add keys for accessing the repositories via SSH and GPG keys for signing commits/tags.

5. Next, select the **Code security and analysis** entry on the left. You don't have to change anything on this screen, just scroll around and note the different categories for **User** and **Repositories**.

The screenshot shows the GitHub settings interface for 'Code security and analysis'. The left sidebar includes sections for Public profile, Account, Appearance, Accessibility, Notifications, Access (Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys, Organizations, Enterprises, Moderation), Code, planning, and automation (Repositories, Codespaces, Packages, Copilot, Pages, Saved replies), Security (Code security and analysis, Integrations), and Archives (Security log, Sponsorship log). The main content area is titled 'Code security and analysis' and contains sections for 'User' and 'Repositories'. The 'User' section features a message about push protection and a 'Push protection for yourself' feature (Beta) that blocks commits containing supported secrets. It also includes a 'Give feedback' and 'Please elaborate' section. The 'Repositories' section includes a 'Private vulnerability reporting' (Beta) feature and an 'Automatically enable for new public repositories' checkbox.

6. Now, in the left menu, near the bottom, click on **Security log**. You can scroll back through your history, expand entries or search.

The screenshot shows the GitHub settings interface for 'Security log'. The left sidebar includes sections for Account, Appearance, Accessibility, Notifications, Access (Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys, Organizations, Enterprises, Moderation), Code, planning, and automation (Repositories, Codespaces, Packages, Copilot, Pages, Saved replies), Security (Code security and analysis, Integrations, Applications, Scheduled reminders), and Archives (Security log, Sponsorship log). The main content area displays a 'Recent events' log with several entries. One entry is expanded, showing detailed information about a codespace destruction event by user 'gwstudent' on 2024-01-06 at 11:02:03. Other collapsed entries include private vulnerability reporting status changes and dependency alert configuration changes for the same user.

7. Let's now create a Personal Access Token (PAT). Click on **Developer settings**, then click on **Personal access tokens**, then **Fine-grained tokens**. Click on the **Generate new token** button.

The screenshot shows the GitHub developer settings interface. At the top, there are tabs for "Sponsorship log" and "Developer settings". Below the header, the URL is "github.com/settings/tokens?type=beta". The main navigation bar includes "Settings / Developer Settings", a search bar, and various icons. On the left, there are links for "GitHub Apps", "OAuth Apps", and "Personal access tokens". Under "Personal access tokens", the "Fine-grained tokens" tab is selected, indicated by a green "Beta" badge. A sub-menu shows "Tokens (classic)". To the right, the title "Fine-grained personal access tokens" is displayed with a "Beta" badge, and a "Generate new token" button. A note below says "Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#)".

8. Enter text for the **Token Name** field and a **Description**. You can leave the **Expiration** timeframe and **Resource owner** as-is. Under **Repository access**, click on **Only select repositories**, search for your repository “sec-demo” and select it.

The screenshot shows the "New fine-grained personal access token" form. The "Token name" field contains "Security Fundamentals" with a green checkmark. The "Expiration" dropdown is set to "30 days". The "Description" field contains "Token for security fundamentals workshop". The "Resource owner" dropdown is set to "gwstudent". Under "Repository access", the "Only select repositories" option is selected. A dropdown menu titled "Select repositories" shows the repository "sec-demo" selected. At the bottom, a preview section shows the token details: "gwstudent/sec-demo" and "Code repo for GitHub Security Fundamentals workshop".

9. Under the **Repository permissions** section, find the **Contents** row and change the selection to **Access: Read and write**.

The screenshot shows the 'Repository permissions' section with 2 items selected. It lists various repository actions with their current access levels. The 'Contents' row is highlighted, showing 'Access: No access'. A dropdown menu titled 'Select an access level' is open, listing 'No access', 'Read-only', and 'Read and write'. The 'Read and write' option is selected.

Action	Access Level
Actions (i) Workflows, workflow runs and artifacts.	Access: No access ▾
Administration (i) Repository creation, deletion, settings, teams, and collaborators.	Access: No access ▾
Code scanning alerts (i) View and manage code scanning alerts.	Access: No access ▾
Codespaces (i) Create, edit, delete and list Codespaces.	Access: No access ▾
Codespaces lifecycle admin (i) Manage the lifecycle of Codespaces, including starting and stopping.	Access: No access ▾
Codespaces metadata (i) Access Codespaces metadata including the devcontainers and machine type.	Access: No access ▾
Codespaces secrets (i) Restrict Codespaces user secrets modifications to specific repositories.	Access: No access ▾
Commit statuses (i) Commit statuses.	Access: No access ▾
Contents (i) Repository contents, commits, branches, downloads, releases, and merges.	Access: Read and write ▾
Dependabot alerts (i) Retrieve Dependabot alerts.	No access
Dependabot secrets (i) Manage Dependabot repository secrets.	Read-only

10. Click on the green **Generate token** at the bottom. A new token will be generated. [Copy the token](#) and store it somewhere for use in the next steps.

The screenshot shows the GitHub Settings interface under 'Developer Settings'. The left sidebar has 'Personal access tokens' selected, with 'Fine-grained tokens' chosen. A 'Beta' badge is visible next to 'Fine-grained tokens'. The main area is titled 'Fine-grained personal access tokens (Beta)'. It includes a note about the tokens being suitable for personal API use and Git over HTTPS. A specific token is displayed with a copy button and a delete button labeled 'Delete'. The token itself is: `github_pat_11AHIKLH0WbtM3dsCC5mM_ijGxYCR1ITC7odA8i`. It also notes that it 'Expires on Mon, Feb 5 2024.'

11. Go to a terminal, clone your repo down using the https protocol. (Substitute your actual GitHub userid for <github-userid>.) Then cd into the directory, make a simple change or create a new file. Add and commit the change, and then push it.

```
$ git clone https://github.com/<github-userid>/sec-demo
```

```
$ cd sec-demo
```

```
$ git checkout dev
```

```
$ echo "Repo for security demos :lock:" > README.md
```

```
$ git add README.md
```

```
$ git commit -m "Add README file"
```

```
$ git push -u origin dev
```

12. When you do the push, you'll be prompted for username (your GitHub username) and then a sign-in/Private Access Token or password. Wherever it asks for a token or a password, you can just copy and paste in **the token you generated in GitHub in the previous steps**. An example dialog that may come up is shown below.



If instead, you are on the command line and prompted for a password, just paste the token in at the prompt. Note that it will not show up on the line, but you can just hit enter afterwards.

```
developer@Bs-MacBook-Pro sec-demo % git push -u origin dev
Username for 'https://github.com': gwstudent
Password for 'https://gwstudent@github.com':
```



END OF LAB

Lab 2 – Branch protection

Purpose: In this lab, we'll see how to set up some branch protection rules and use them

- Our current repo does not have any protection rules setup. Let's set some up. We have two options, standalone branch protection rules or rulesets. First, we'll look at the standalone option. Switch back to the repository page if not there. Go to the **Settings** tab for the repository, then select **Branches** on the left. In the new screen, click on the **Add classic branch protection** rule button.

- At the end of the last lab, we pushed to branch "dev" in our repository. Let's lock down all dev* branches. In the **Branch name pattern** field, enter the pattern **dev***. Under the **Protect matching branches** section, check the box in the row for **Lock branch**. Also, check the box in the row for **Do not allow bypassing the above settings** - since you're the owner of the repo.

Protect your most important branches

Branch protection rules define whether collaborators can delete or force push to the branch and set requirements for any pushes to the branch, such as passing status checks or a linear commit history.

Your GitHub Free plan can only enforce rules on its public repositories, like this one.

Branch name pattern *

dev*

Protect matching branches

Require a pull request before merging
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

Require status checks to pass before merging
Choose which status checks must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

Require conversation resolution before merging
When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. [Learn more about requiring conversation completion before merging.](#)

Require signed commits
Commits pushed to matching branches must have verified signatures.

Require linear history
Prevent merge commits from being pushed to matching branches.

Require deployments to succeed before merging
Choose which environments must be successfully deployed to before branches can be merged into a branch that matches this rule.

Lock branch
Branch is read-only. Users cannot push to the branch.

Allow fork syncing
Branch can pull changes from its upstream repository

Do not allow bypassing the above settings
The above settings will apply to administrators and custom roles with the "bypass branch protections" permission.

Branch protection rule created.

3. Click the **Create** button at the bottom of the page to save the rule and apply it.

gwstudent / sec-demo

Type ⌘ to search

Code Pull requests Actions Projects Wiki Security Insights Settings

Branch protection rule created.

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Branch protection rules

Add rule

dev*

Currently applies to 1 branch

Edit Delete

4. Let's see how the branch rule works in practice. Go back to your terminal and make another change, commit it, and then push it. (You'll need to have your fine-grained token from the previous lab.) An example change is shown below, but you can make any change you want.

```
$ echo "For educational purposes only" >> README.md
```

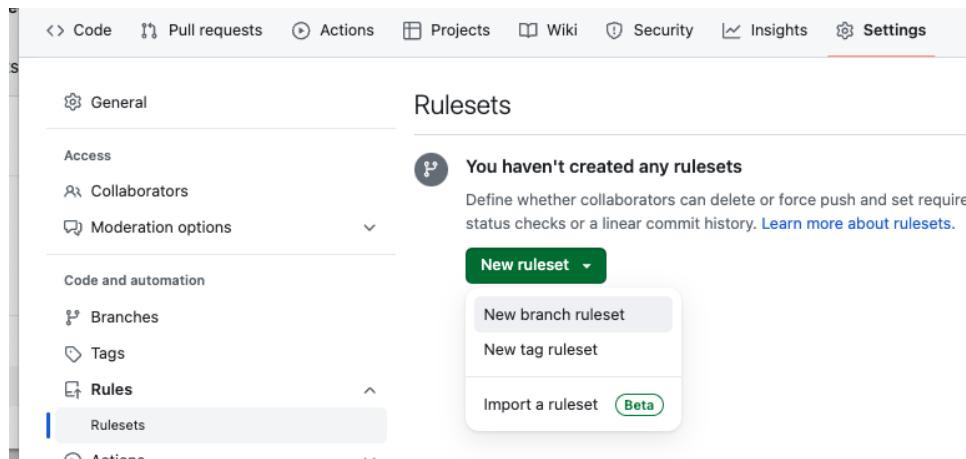
```
$ git commit -am "Update README"
```

```
$ git push origin dev
```

5. At this point, you'll see an error message for your attempted push since the dev branch is locked down.

```
[developer@Bs-MacBook-Pro sec-demo % git push origin dev
Username for 'https://github.com': gwstudent
Password for 'https://gwstudent@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 321 bytes | 321.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote: error: GH006: Protected branch update failed for refs/heads/dev.
remote: error: Cannot change this locked branch
To https://github.com/gwstudent/sec-demo
 ! [remote rejected] dev -> dev (protected branch hook declined)
error: failed to push some refs to 'https://github.com/gwstudent/sec-demo'
```

6. The other option for protecting branches is rulesets. Let's set up a ruleset next. Still in the repository's **Settings** tab, select **Rules** on the left, then **Rulesets**. Then click on the **New ruleset** button and **New branch ruleset**.



7. Enter a **Ruleset Name** (such as "Protect main"), set the **Enforcement** status to **Active**, and for **Target branches**, you can just click **+Add target** and select "*Include default branch*".

8. For the actual **Branch protections**, you can just leave the ones checked that are already selected, and check the box for the one for **Require a pull request before merging**. You don't need to check any of the other boxes for **Additional settings** right now. Then click the **Create** button.

<input checked="" type="checkbox"/> Restrict deletions	Only allow users with bypass permissions to delete matching refs.
<input type="checkbox"/> Require linear history	Prevent merge commits from being pushed to matching refs.
<input type="checkbox"/> Require deployments to succeed	Choose which environments must be successfully deployed to before refs can be pushed into a ref that matches this rule.
<input type="checkbox"/> Require signed commits	Commits pushed to matching refs must have verified signatures.
<input checked="" type="checkbox"/> Require a pull request before merging	Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.
Additional settings	
<input type="checkbox"/> Require status checks to pass	Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.
<input checked="" type="checkbox"/> Block force pushes	Prevent users with push access from force pushing to refs.

Afterwards, if you click on **Rules** or **Rulesets** again, you should see the item you just created.

The screenshot shows the GitHub repository settings page for a repository named 'sec-demo'. On the left, there's a sidebar with sections like General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, and Rules. Under Rules, 'Rulesets' is selected. In the main area, there's a 'Rulesets' section with a dropdown set to 'All'. A card titled 'Protect main' is shown, indicating '3 rules targeting 1 branch'. There's also a 'New ruleset' button.

- Now, let's see this one in practice. Switch back to your terminal and attempt to push the change out to your GitHub repo. Once again, you'll need your fine-grained token.

```
$ git checkout main
$ echo "Copyright 2024" >> LICENSE
$ git commit -am "update license"
$ git push origin main
```

- You should see a message like the following indicating that you can push the changed code due to the repository rule violation.

```
developer@Bs-MacBook-Pro sec-demo % git push origin main
Username for 'https://github.com': gwstudent
Password for 'https://gwstudent@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 323 bytes | 323.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote: error: GH013: Repository rule violations found for refs/heads/main.
remote: Review all repository rules at http://github.com/gwstudent/sec-demo/rules?ref=refs%2Fheads%2Fmain
remote:
remote: - Changes must be made through a pull request.
remote:
To https://github.com/gwstudent/sec-demo
 ! [remote rejected] main -> main (push declined due to repository rule violations)
error: failed to push some refs to 'https://github.com/gwstudent/sec-demo'
```

END OF LAB

Lab 3 – Code Scanning

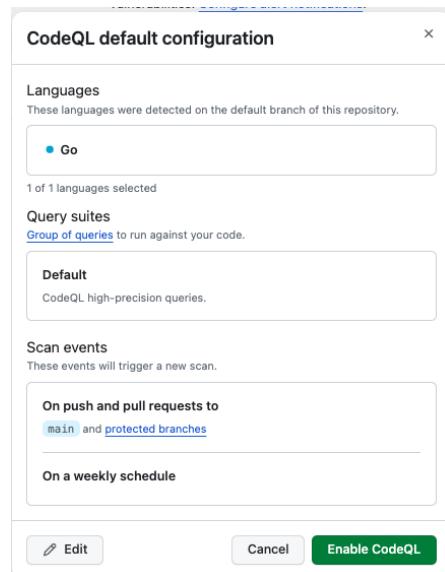
Purpose: In this lab, we'll see how to enable code scanning on our repository

- An important part of keeping your repository secure is code scanning. Let's enable the default setup for code scanning in our repo via CodeQL and GitHub Actions. Go to your repo's **Settings** tab and under "Security" on the left, select **Code security and analysis**. Near the bottom, you'll see the "Code Scanning" section.

The screenshot shows the GitHub repository settings page. The left sidebar has sections like General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages, and Security. Under Security, the 'Code security and analysis' section is expanded, showing 'Code scanning' and 'Tools'. The 'Code scanning' section is highlighted with a blue border. It contains sub-sections for CodeQL analysis, Other tools, Protection rules, and Pull request check failure. Each sub-section has an 'Enable' button. The 'Code scanning' section also has a 'Tools' button at the bottom.

- In the **Tools / CodeQL** analysis section, click on the **Set up** button and select the **Default** configuration. On the dialog that comes up with the default options, click the **Enable CodeQL** button. After that, you'll see a blue bar saying that a scan of the entire repo will happen.

The screenshot shows the 'Code scanning' configuration dialog. It has sections for Tools, CodeQL analysis, Other tools, Protection rules, and Pull request check failure. The 'CodeQL analysis' section has a 'Set up' button. A dropdown menu is open over the 'Default' configuration, showing 'Default' and 'Advanced'. The 'Default' option is selected. The 'Advanced' option has a 'High or higher / Only errors' dropdown.



Repository settings saved. This initial setup might take a while because CodeQL will perform a full scan of the repository. X

- The setup will take several minutes to run. After it starts, you can click on the **Actions** tab in your repository, and you'll see that you have a new Actions workflow named "CodeQL" that is running against your repo. After it completes, if you want, you can click on the **CodeQL Setup** item under **All workflows** on the right and drill in to the jobs that ran.

Workflow	Last Run
CodeQL	CodeQL Setup · 10 minutes ago

Task	Time
Set up job	4s
Checkout repository	3s
Initialize CodeQL	16s
Configure	4s
Setup Go	0s
Autobuild	2m 23s
Perform CodeQL Analysis	33s
Record Successful outcome	0s
Post Perform CodeQL Analysis	0s
Post Initialize CodeQL	1s

4. Now, let's see if any vulnerabilities were found. Click on the **Security** tab of the repo, find the row under **Security Overview** for **Code scanning alerts** and select **View alerts**. The **Security** tab at the top should show a 3 next to it indicating that vulnerabilities were found.

The screenshot shows the GitHub Security overview page. The top navigation bar has tabs: Code, Pull requests, Actions, Projects, Wiki, Security (with a red circle), Insights, and Settings. The left sidebar has sections: Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, Code scanning (with a red circle), and Secret scanning. The main content area is titled "Security overview". It contains several sections: "Security policy • Disabled" (with a "Set up a security policy" button), "Security advisories • Enabled" (with a "View security advisories" button), "Private vulnerability reporting • Disabled" (with a "Enable vulnerability reporting" button), "Dependabot alerts • Disabled" (with a "Enable Dependabot alerts" button), "Code scanning alerts • Enabled" (with a red circle around the "View alerts" button), and "Secret scanning alerts • Disabled" (with a "Enable in settings" button).

5. On the next screen, you should see a set of alerts found in the current code.

The screenshot shows the GitHub Code scanning page. The top navigation bar has tabs: Code, Pull requests, Actions, Projects, Wiki, Security (with a red circle), Insights, and Settings. The left sidebar has sections: Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, Code scanning (with a red circle), and Secret scanning. The main content area is titled "Code scanning". It shows a summary: "All tools are working as expected" (with a green checkmark) and "Tool status 1 + Add tool". There is a search bar with "is:open branch:main". Below, there is a table of alerts: 3 Open, 0 Closed. The first three rows are: "Database query built from user-controlled sources" (High, #3 opened 5 minutes ago, main), "Database query built from user-controlled sources" (High, #2 opened 5 minutes ago, main), and "Database query built from user-controlled sources" (High, #1 opened 5 minutes ago, main). At the bottom, there is a pro tip: "ProTip! You can upload code scanning analyses from other third-party tools using GitHub Actions. [Learn more](#)".

6. Click on the top one and you'll see more detail about the issue. This includes a link to more info about the vulnerability on the right under "Weaknesses".

The screenshot shows a GitHub security alert for a code scanning issue. The alert is titled "Database query built from user-controlled sources" and is categorized as "High" severity. It was opened 6 minutes ago. The alert details a specific line of code (models/models.go:76) where a database query is constructed using `fmt.Sprintf` with a user-provided string. A note states: "This query depends on a user-provided value." Below the code snippet, there's a "CodeQL Show paths" section showing another part of the code where an error is checked. To the right of the alert, there are sections for "Affected branches" (main), "Tags" (security), and "Weaknesses" (CWE-89). Below the alert, a timeline shows the first detection in a commit 6 minutes ago, with an initial add and a reference to models/models.go:76 on branch main.

7. The *dev* branch already has fixes in it for these issues. Go back to the **Code** tab in the repo. Near the top, there may be a yellow bar noting that there were recent pushes to *dev* and showing a button to **Compare & pull request**. If you see that, just click on that button. If you don't see this, you can initiate a new pull request via the **Pull requests** menu at the top.

The screenshot shows the GitHub repository page for "sec-demo". The repository is public and forked from "skillrepos/sec-demo". A yellow banner at the top indicates that the "dev" branch had recent pushes 38 minutes ago, with a "Compare & pull request" button. Below the banner, the repository stats show 2 branches and 0 tags. A message says "This branch is up to date with skillrepos/sec-demo:main.". On the right side, there are navigation links labeled A, R, F, S, ^, and V.

8. Create the pull request and, on the next screen, in the dropdowns on the gray bar, make sure to select the main branch of your current repo as the base on the left and the dev branch of your current repo as the compare selection. Enter a title and description if you want and then click on the green **Create pull request** button.

The screenshot shows the GitHub interface for creating a pull request. At the top, there's a navigation bar with links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below this, a banner says "Open a pull request" and provides instructions for comparing changes across two branches. It shows "base: main" and "compare: dev" dropdowns, both set to their respective branch names. A green checkmark indicates "Able to merge. These branches can be automatically merged." The main form area has sections for "Add a title" (with "Dev" entered) and "Add a description" (with a placeholder "Add your description here..."). There are "Write" and "Preview" tabs above the description area, along with a rich text editor toolbar. At the bottom, it says "Markdown is supported" and "Paste, drop, or click to add files". A large green "Create pull request" button is at the bottom right.

9. On the next screen, notice that the CodeQL analysis is automatically added as a check for being able to merge the changes. While you're waiting for it to complete, you can look at the changes in the **Commits** tab or **Files changed** tab.

Dev #1

Open gwstudent wants to merge 2 commits into `main` from `dev`

Conversation 0 Commits 2 Checks 1 Files changed 2

gwstudent commented now

No description provided.

Brent Laster added 2 commits 1 hour ago

- fixes for models 60509a9
- Add README file e1d195b

Add more commits by pushing to the `dev` branch on [gwstudent/sec-demo](#).

Some checks haven't completed yet
1 in progress check

- CodeQL / Analyze (go) (dynamic) In progress — This check has started... [Details](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

10. Once the checks have completed, the screen will show *All checks have passed*. You can then go ahead, and **Merge pull request** and **Confirm merge**.

Add more commits by pushing to the `dev` branch on [gwstudent/sec-demo](#).

All checks have passed
2 successful checks

- CodeQL / Analyze (go) (dynamic) Successful in 1m [Details](#)
- Code scanning results / CodeQL Successful in 3s — No new alerts in code changed by t... [Details](#)

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

11. When the push is made to main as a result of the merge, this will kick off another run of the *CodeQL* workflow. You'll be able to see that in the Actions menu.

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with categories like Actions, CodeQL, Management, Caches, Runners, and others. The main area is titled "All workflows" and shows "3 workflow runs". The runs listed are:

- Push on main**: CodeQL #3: by gwstudent
- PR #1**: CodeQL #2: by gwstudent
- CodeQL Setup**: CodeQL #1: by gwstudent

12. After this completes, if you go back to the **Security** tab, you'll see that the alerts have been closed as fixed. (Click on the **3 Closed** link and individual items to see details.)

The screenshot shows the GitHub Security tab. On the left, there's a sidebar with Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, and Code scanning (which is selected). The main area is titled "Code scanning" and shows the following information:

- All tools are working as expected**
- is:closed branch:main**
- Clear current search query, filters, and sorts**
- 0 Open 3 Closed**
- Database query built from user-controlled sources [High]** (3 closed as fixed 2 minutes ago - Detected by CodeQL in models/models.go:76 28 minutes ago)
- Database query built from user-controlled sources [High]** (2 closed as fixed 2 minutes ago - Detected by CodeQL in models/models.go:57 28 minutes ago)
- Database query built from user-controlled sources [High]** (#1 closed as fixed 2 minutes ago - Detected by CodeQL in models/models.go:38 28 minutes ago)

Below this, there's a "ProTip!" message: "♀ ProTip! You can run CodeQL locally using Visual Studio Code. [Learn more](#)".

At the bottom, there's a detailed view of one of the closed alerts:

Database query built from user-controlled sources

Fixed in `main` 3 minutes ago

`models/models.go:76`

```
73 // the query, you should be using a parameterized query.
74 func ReadBook(r string) ([]Book, error) {
75     rows, err := DB.Query("SELECT * FROM books WHERE read = ?")
76     rows, err := DB.Query("SELECT * FROM books WHERE read = 's'", r)
}
This query depends on a user-provided value.
```

Tool: CodeQL
Rule ID: go/sql-injection
Query: View source

If a database query (such as an SQL or NoSQL query) is built from user-provided data without sufficient sanitization, a malicious user may be able to run commands that exfiltrate, tamper with, or destroy data stored in the database.

Severity: High
Affected branches: main
Tags: security
Weaknesses: CWE-89

Show more

First detected in commit 29 minutes ago
initial add
models/models.go:76 on branch main ✓ 5b7b949

Fixed in branch main 3 minutes ago
Merge pull request #1 from gwstudent/dev → Verified ✓ 895af97

END OF LAB

© 2024 Tech Skills Transformations LLC & Brent Laster

Lab 4 – Secret Scanning

Purpose: In this lab, we'll see how to enable secret scanning on our repository

- Just as we used code scanning to find vulnerabilities in the code in our repositories, GitHub can also scan for secrets that may be exposed in our repository files. This functionality should be enabled by default. To see that, go to the repository's **Settings**, then back to **Code security and analysis** on the left and then scroll down to the bottom of the page. In the **Secret scanning** section, you can see the main setting and also the setting for the **Push protection** option to *Block commits that contain supported secrets*.

The screenshot shows the 'Code security and analysis' settings page. It includes sections for Dependabot security updates, Grouped security updates, Dependabot version updates, Dependabot on Actions runners, Code scanning (which is currently disabled), Tools (CodeQL analysis and Other tools), Protection rules (Check runs failure threshold), Secret scanning (disabled), and Push protection (disabled). A red oval highlights the 'Secret scanning' and 'Push protection' sections.

- Let's add a file with secrets to see how the blocking works. In the **Code** tab, click on **Add File** and then **+ Create new file**.

The screenshot shows the GitHub Code tab with a pull request. The status bar indicates 'Merge pull request #1 from gwstudent/dev'. The 'Code' tab has a 'Create new file' button with a context menu open, showing options like '+ Create new file' and 'Upload files'.

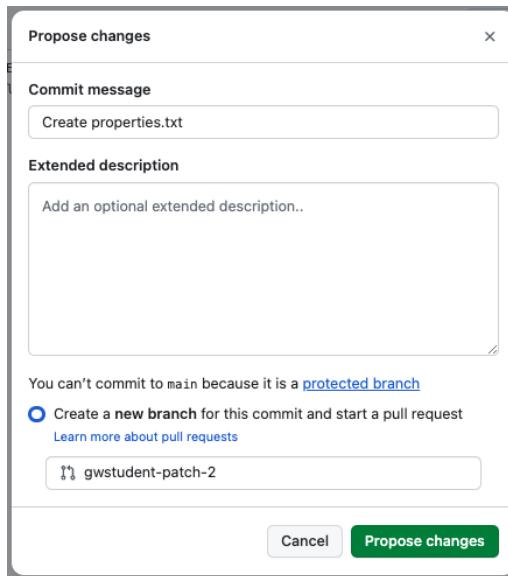
- In the new file screen, name the file properties/properties.txt (add that path in the text entry box at the top after “sec-demo”) and copy and paste the following content into the contents area under the “Edit” button.

```
AWS_ACCESS_KEY_ID="AKIAZBQE445JKPTEAHQD"
AWS_SECRET_ACCESS_KEY="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"
```

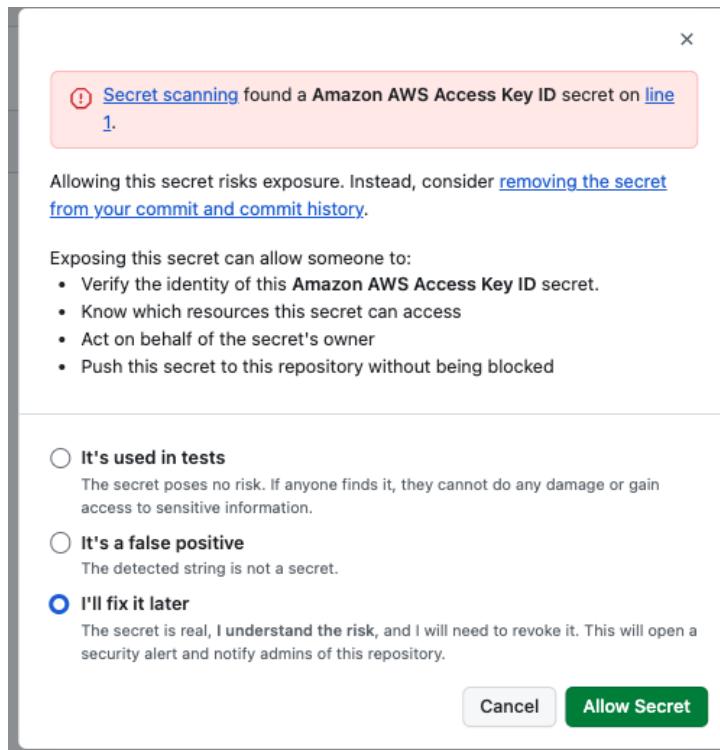
The screenshot shows a GitHub repository interface. At the top, there are navigation links: Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below this, the repository path is shown as sec-demo / properties / properties.txt in the main branch. On the right, there are buttons for Cancel changes and Commit changes... (which is highlighted in green). Below these buttons, there are options for Edit, Preview, Spaces (set to 2), and No wrap. The main content area shows the code content:

```
1 AWS_ACCESS_KEY_ID="AKIAZBQE445JKPTEAHQD"
2 AWS_SECRET_ACCESS_KEY="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"
```

- Click on the **Commit changes...** button. Since we have the current branches protected, go ahead and accept the option to create a new branch and click the **Propose changes** button.



- At this point, you'll see a warning dialog because we have secret scanning turned on in the repository and we are trying to push a file with a secret in it. For purposes of this exercise, just select the "I'll fix it later" option and then click on the **Allow Secret** button.



- At this point, you'll see a screen acknowledging that the secret is allowed and you can commit. Click on the **Commit changes...** button.

✓ Secret allowed. You can now commit these changes.

sec-demo / properties / properties.txt in main

Edit Preview Code 55% faster with GitHub Copilot

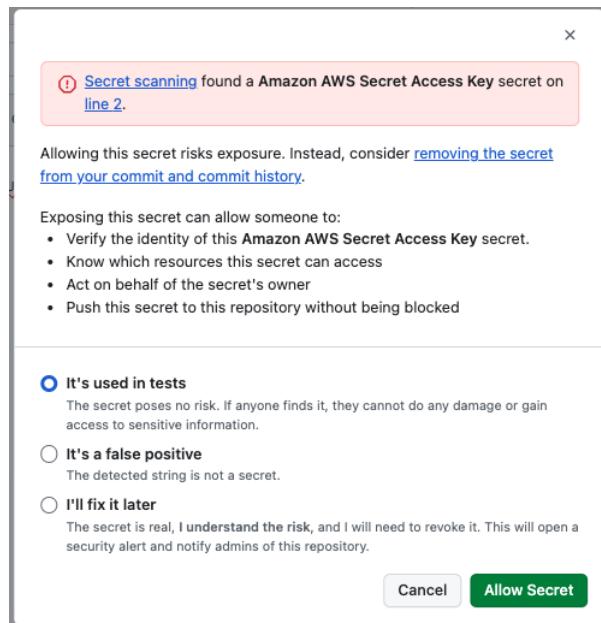
Cancel changes **Commit changes...**

```

1 AWS_ACCESS_KEY_ID="AKIAZBQE445JKPTEAHQD"
2 AWS_SECRET_ACCESS_KEY="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"
3

```

- Click on the **Propose changes** button on the next dialog. Then you'll be at a similar dialog for the second secret as you had for the first one. This time, select "*It's used in tests*" and then click on the **Allow Secret** button.



8. As before, you'll see the screen saying the secret is allowed. Click on the ***Commit changes...*** button and then the ***Propose changes*** button on the next dialog.
9. Now, if you click on the **Security** tab, the **Secret Scanning alerts** section on the page, and the ***View detected secrets*** options on the side, you'll see the one secret we said we would fix later.

Secret scanning alerts • Enabled

Get notified when a secret is pushed to this repository

View detected secrets

Overview Secret scanning alerts

Reporting Policy Advisories Vulnerability alerts Dependabot Code scanning Secret scanning 1

is:open

1 Open 1 Closed Bypassed Validity Secret type Provider Sort

Amazon AWS Access Key ID AKIAZBQE445JKPTEAHQD
#1 opened 1 minute ago • Detected secret in properties/properties.txt

10. Assume that we have actually revoked this secret and now want to close out the alert. Check the box next to the secret to select it. A new button titled ***Close as*** will appear. Click on that and select

Revoked as the reason to close the alert. Enter a comment if desired and then click on the **Close alert** button.

The screenshot shows the GitHub Secrets scanning interface. On the left, there's a sidebar with links like Overview, Reporting, Policy, and Secret scanning. The Secret scanning link is selected and has a count of 1. In the main area, there's a search bar with 'is:open' and a button to 'Close as'. A dropdown menu is open, showing 'Amazon' with '#1 opened 1' and a checked checkbox. Below this, there's a section titled 'Select a close reason' with four options: 'Revoked' (selected), 'Used in tests', 'False positive', and 'Won't fix'. Under 'Comment', there's a text input field with 'Add a comment'. At the bottom right are 'Cancel' and 'Close alert' buttons.

11. After closing the alert, you can click on the 2 Closed link and see the alert that was automatically closed when you said it was used in tests and the one you recently closed as revoked.

This screenshot shows the same GitHub interface after closing the alert. The sidebar and search bar remain the same. In the main area, the 'Closed as' dropdown now shows '2 Closed'. Below it, there are two entries in the list: 'Amazon AWS Secret Access Key' and 'Amazon AWS Access Key ID', both with their respective details and 'Closed as used in tests' or 'Closed as revoked' status.

END OF LAB

Lab 5 – Working with Dependabot

Purpose: In this lab, we'll see how to use Dependabot to keep dependencies up to date.

- Let's enable Dependabot functionality in our repository. Go to the **Settings** tab for the repository, then select **Code security and analysis** on the left side again and in the **Dependabot** section of that page, click the button for **Dependabot alerts** to enable them. In the dialog that comes up about Enable Dependabot alerts, you can just click on the **Enable** button.

The screenshot shows the GitHub Settings page for a repository. On the left sidebar, under Security, the **Code security and analysis** tab is selected. In the main content area, the **Dependabot** section is visible. A modal dialog titled "Enable Dependabot alerts" is open. It contains a message: "Dependabot alerts needs the dependency graph to be enabled, so we'll turn that on too." At the bottom right of the modal, there is a large red circle around the "Enable" button. Below the modal, another "Enable" button is also circled in red.

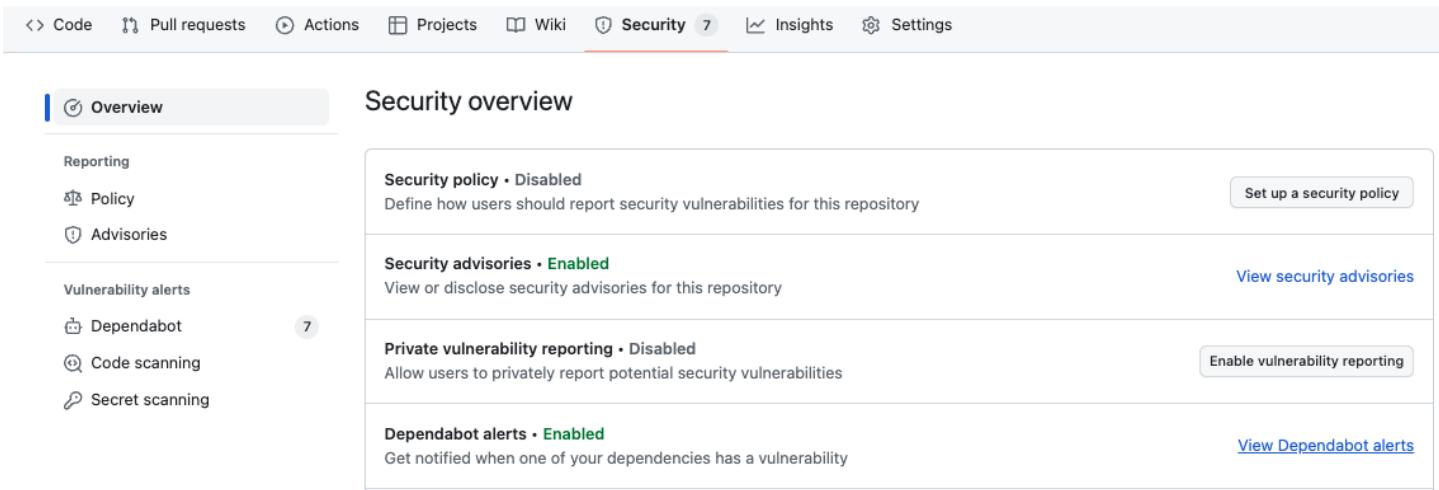
- After this, you'll see that Dependabot has 1 rule enabled. You can click on the gear if you want to see more details about the rule.

The screenshot shows the GitHub Settings page for a repository. The **Code security and analysis** tab is selected. In the main content area, the **Dependabot alerts** section is shown. It says "Receive alerts for vulnerabilities that affect your dependencies and manually generate Dependabot pull requests to resolve these vulnerabilities." A "Disable" button is visible. Below it, the **Dependabot rules** section shows "1 rule enabled" and a gear icon. A red circle highlights the "1 rule enabled" text.

- If you want to see the dependency graph info, you can click on the **Insights** tab at the top and then the **Dependency graph** item on the left.

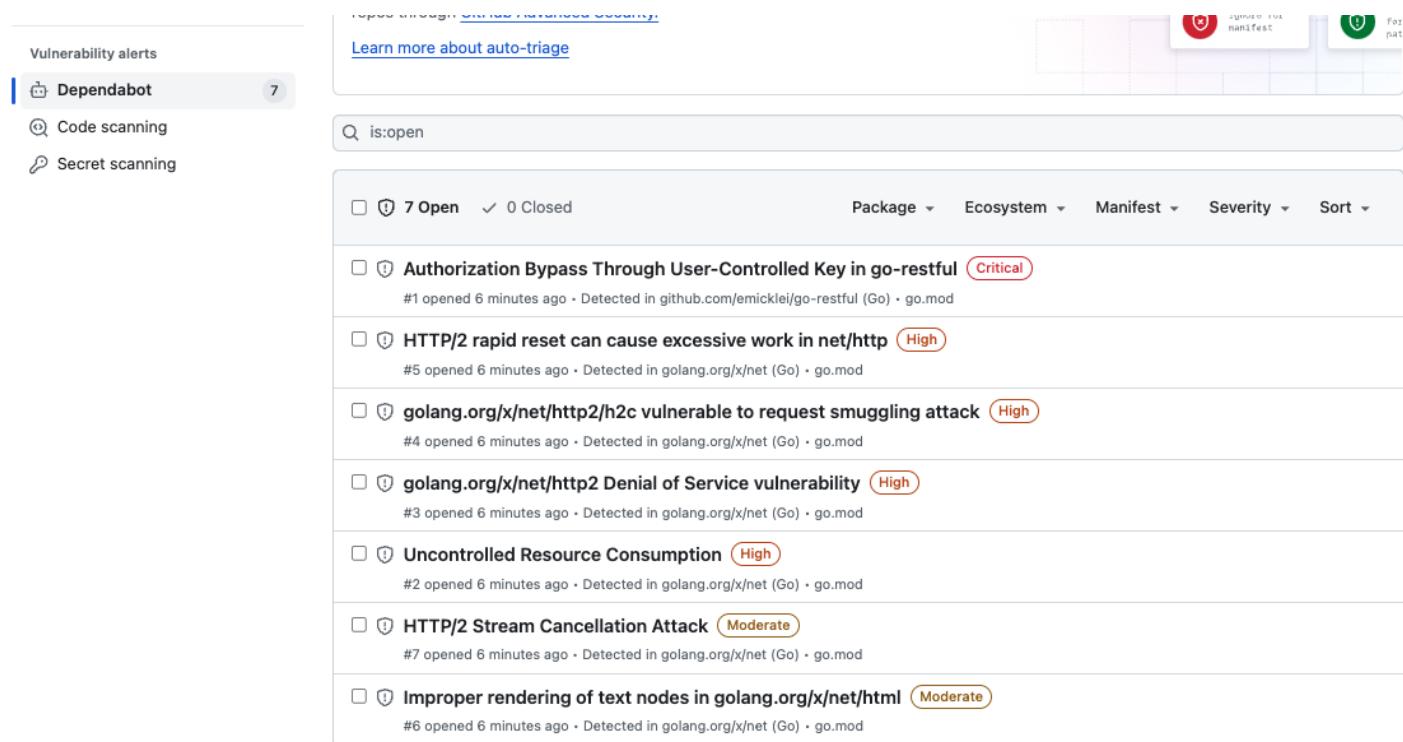
The screenshot shows the GitHub Insights page for a repository. The **Insights** tab is selected in the top navigation bar. On the left, a sidebar menu includes items like Pulse, Contributors, Community, Traffic, Commits, Code frequency, **Dependency graph** (which is selected and highlighted in red), Network, and Forks. The main content area is titled "Dependency graph". It features tabs for Dependencies, Dependents, and Dependabot, with "Dependencies" selected. There is a search bar labeled "Search all dependencies". Below the search bar, three dependency entries are listed with their status: "github.com/emicklei/go-restful 2.9.5+incompatible" (1 critical), "golang.org/x/net 0.0.0-20220127200216-cd36cc0744dd" (4 high), and "github.com/mattn/go-sqlite3 1.14.8" (no status shown). Each entry includes a link to the dependency's page and a note about when it was detected.

4. To see the issues Dependabot found, next click on the **Security** tab and then in the **Dependabot alerts** row, click on [View Dependabot alerts](#).



The screenshot shows the GitHub Security overview page. On the left, there's a sidebar with links like Overview, Reporting, Policy, and Dependabot (which has 7 alerts). The main area is titled "Security overview" and contains several sections: "Security policy" (disabled), "Security advisories" (enabled), "Private vulnerability reporting" (disabled), and "Dependabot alerts" (enabled). The "Dependabot alerts" section lists 7 open alerts, each with a title, description, and severity level (e.g., Critical, High, Moderate).

5. At this point, you'll see 7 dependabot alerts that show up, ranging from Critical to Moderate in severity.



The screenshot shows the GitHub Dependabot alerts page. The sidebar indicates 7 open alerts. The main table lists the following alerts:

Alert ID	Description	Severity
#1	Authorization Bypass Through User-Controlled Key in go-restful	Critical
#5	HTTP/2 rapid reset can cause excessive work in net/http	High
#4	golang.org/x/net/http2/h2c vulnerable to request smuggling attack	High
#3	golang.org/x/net/http2 Denial of Service vulnerability	High
#2	Uncontrolled Resource Consumption	High
#7	HTTP/2 Stream Cancellation Attack	Moderate
#6	Improper rendering of text nodes in golang.org/x/net/html	Moderate

6. Click on the first alert. You can review the information shown and then click on the **Create Dependabot security update** button.

The screenshot shows a GitHub repository page for 'emicklei/go-restful'. The 'Security' tab is selected. A specific alert titled 'Authorization Bypass Through User-Controlled Key in go-restful #1' is highlighted. Below the alert, there's a green button labeled 'Create Dependabot security update'. This button is circled in red. To the right of the alert, there's a 'Severity' section indicating 'Critical' (9.1 / 10) and a 'CVSS base metrics' table.

CVSS base metrics	
Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	None

- After the creation step runs, GitHub will generate a new security update. Click on the ***Review security update*** button to see details of the update.

The screenshot shows the same GitHub repository page for 'emicklei/go-restful'. The 'Security' tab is still selected. The same alert is shown, but now it includes a 'Review security update' button. This button is highlighted with a green box. The alert message indicates a bump from '2.9.5+incompatible' to '2.16.0+incompatible'.

- This will open up the security update which is actually a pull request. Click on the ***Commits*** tab and then the ***Files changed*** tab to see the changes. In this case, the two dependencies that were causing issues weren't needed, so Dependabot removed them. More typically, it would create a pull request to update the version.

The screenshot shows a GitHub pull request page. At the top, it says "Bump github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible #2". Below this, there's a green button labeled "Open" and a note that "dependabot wants to merge 1 commit into main from dependabot/go_modules/github.com/emicklei/go-restful-2.16.0+incompatible". A message box states: "Merging this pull request will resolve a critical severity Dependabot alert on github.com/emicklei/go-restful." The "Files changed" tab is selected, showing two files: go.mod and go.sum. The go.mod file shows a change from version 1.17 to 1.18, requiring github.com/mattn/go-sqlite3 v1.14.8. The go.sum file shows a change from v1.14.1 to v1.14.8. Both files have a green status bar at the bottom.

- When ready, change back to the **Conversation** tab of the pull request and click the **Merge pull request** button, followed by the **Confirm merge** button. After the merge is complete, you'll see a green bar telling you the pull request resolved the Dependabot alert.

The screenshot shows the same GitHub pull request page after merging. It now displays a green bar at the top stating "Bump github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible #2" and "dependabot wants to merge 1 commit into main from dependabot/go_modules/github.com/emicklei/go-restful-2.16.0+incompatible". Below this, the "Merge pull request" button is shown with a dropdown menu. A modal window is open, showing "All checks have passed" (1 neutral check), "Code scanning results / CodeQL" (1 configuration not found), and "This branch has no conflicts with the base branch" (Merging can be performed automatically). At the bottom of the modal is a "Merge pull request" button.

The screenshot shows the GitHub pull request page again, this time with a blue "Merged" button at the top. It says "Bump github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible #2" and "dependabot merged 1 commit into main from dependabot/go_modules/github.com/emicklei/go-restful-2.16.0+incompatible 1 minute ago". A green message box says "This pull request resolved a Dependabot alert on github.com/emicklei/go-restful.". Below this, the "Files changed" tab is selected, showing a comment from "dependabot (bot)" that says "commented on behalf of github 7 minutes ago" and "Bumps github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible." The "Reviewers" section indicates "No reviews".

10. In preparation for the next lab, enable the Issues functionality on your repo by going to the **Settings** tab, scrolling down to **Features**, and then checking the box in the **Issues** section.

The screenshot shows the 'Features' section of the GitHub repository settings. It includes options for Wikis, Restrict editing to collaborators only, and Issues. The 'Issues' checkbox is checked and highlighted with a blue border. A blue callout bubble with the text 'Check this box' points to the 'Issues' checkbox.

Social preview
Upload an image to customize your repository's social media preview.
Images should be at least 640x320px (1280x640px for best display).
[Download template](#)

Features

- Wikis**
Wikis host documentation for your repository.
- Restrict editing to collaborators only**
Public wikis will still be readable by everyone.
- Issues**
Issues integrate lightweight task tracking into your repository. Keep projects on track reference them in commit messages.

END OF LAB

Lab 6 – Securing GitHub Actions

Purpose: In this lab, we'll see how to use do some preventative security changes for GitHub Actions.

- When the dev branch was merged with the pull request earlier, it added a GitHub Actions workflow that can be used to automatically create a GitHub issue. In the *Code* tab, find the file `.github/workflows/create-issue.yml` and click on it and take a look at the contents.

The screenshot shows the GitHub code editor with the file `.github/workflows/create-issue.yml` open. The file content is as follows:

```

name: create-issue
# Controls when the workflow will run
on:
# Allows you to call this manually from the Actions tab

```

2. We can run this workflow manually to create an issue. Click on the **Actions** tab and then under the *All workflows* section on the left, select the **Create issue** item. Then click on the **Run workflow** button and in the dialog that comes up, select **Branch: main**, and then you can put in any text for **Issue title** and **Issue body**. Finally, click on the green button to **Run workflow**.

The screenshot shows the GitHub Actions interface. On the left sidebar, under the 'Actions' section, 'Create Issue' is selected. A modal window titled 'Create Issue' is open, showing the workflow file 'create-issue.yml'. The modal contains fields for 'Issue title' (set to 'This is a title') and 'Issue body' (set to 'This is the body text'). A large green 'Run workflow' button is at the bottom right of the modal. Above the modal, a message says 'This workflow has a workflow_dispatch event trigger.' and there is a small icon of a person with a checkmark.

3. After the run of the workflow completes, you should have a new *GitHub Issue* visible under the **Issues** tab.

The screenshot shows the GitHub Issues interface. The 'Issues' tab is selected, showing 6 open issues. One issue is highlighted with a green circle and the text 'This is a title'. Below the issue list are filters for Author, Label, Projects, Milestones, Assignee, and Sort. At the bottom of the list, it says '#10 opened 27 minutes ago by github-actions bot'.

4. The code in this workflow is vulnerable to script injection because of the way the variables are evaluated in the code. To see this, in the **Actions** tab, run the workflow again, put what you want in the **Issue title** field, but in the **Issue body** field, enter the following as the arguments (NOTE: That is two backquotes around ls -la) ``ls -la``

The screenshot shows the GitHub Actions interface. The top navigation bar includes links for Code, Issues (6), Pull requests, Discussions, Actions (highlighted in red), Projects, and Wiki. A success message "Workflow run was successfully requested." is displayed. The left sidebar has sections for Actions, All workflows, CodeQL, Create Issue (selected), Management, Caches, and Runners. The main area is titled "Create Issue" with the file name "create-issue.yml". It shows 0 workflow runs. A table header includes Event, Status, Branch, and Actor dropdowns. A note states "This workflow has a workflow_dispatch event trigger." with a "Run workflow" button. A card for the most recent run is shown, with a green checkmark icon and the title "Create Issue". Below it, the log entry "Create Issue #11: Manually run by gwstudent" is listed, along with the branch "main". To the right, a modal window titled "Use workflow from" shows "Branch: main" selected. The modal contains fields for "Issue title *" (value: "This is a title") and "Issue body *" (value: "`ls -la`"). A "Run workflow" button is at the bottom of the modal.

5. After the workflow runs, you'll have another issue created, but the more interesting part is in the execution of the action. Click on the row for the most recent run.

The screenshot shows the GitHub Actions interface with the same layout as the previous one. The left sidebar shows "Create Issue" is selected. The main area shows 2 workflow runs. The first run, which is the most recent, is circled in red. It has a green checkmark icon and the title "Create Issue". Below it, the log entry "Create Issue #13: Manually run by gwstudent" is listed, along with the branch "main". The second run is partially visible below it.

Then click on the job name in the next screen.

The screenshot shows the GitHub Actions job details for the "create_issue" job of the "Create Issue" workflow. The top navigation bar is identical. The left sidebar shows "create_issue" is selected. The main area has a "Summary" tab selected, showing a table with columns for Job, Status, Total duration, and Artifacts. One job entry is shown: "gwstudent -> 9652e60 main" with Status "Success" and Total duration "12s". Below the summary is a "create-issue.yml" section with the trigger "on: workflow_dispatch". A table shows the steps: "create_issue" (status "Success", duration "4s"), followed by "Run details", "Usage", and "Workflow file". The "create_issue" step is circled in red.

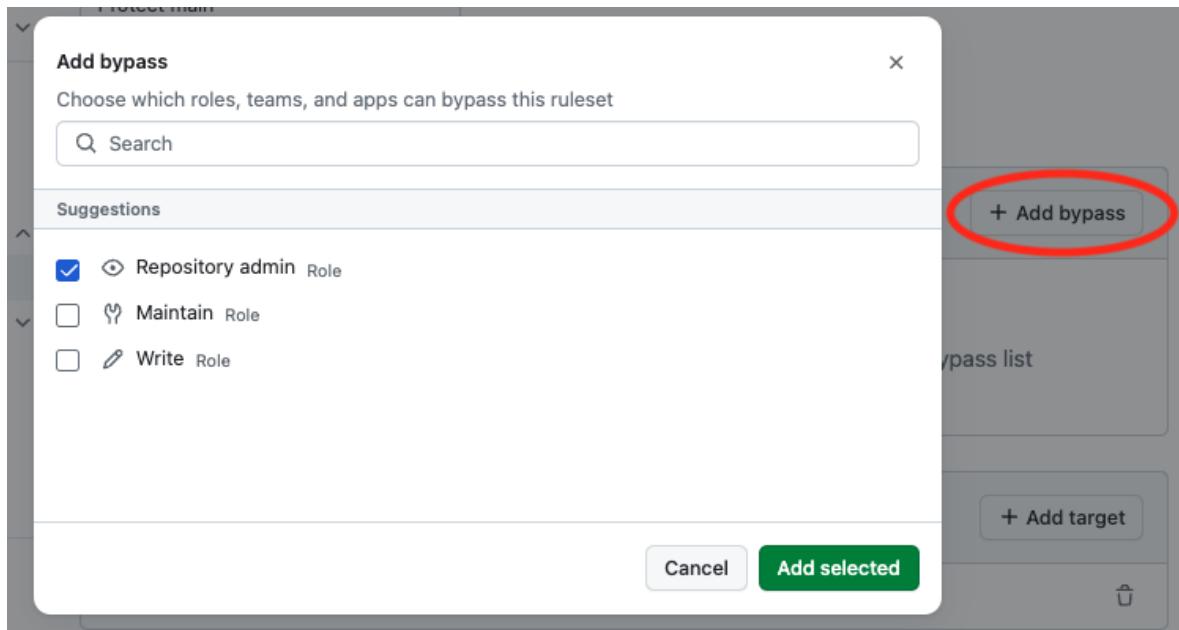
Then click to expand the *echo inputs* step in the log. Notice that the text we put in as the body of the issue was actually executed and run as a command.

The screenshot shows the GitHub Actions interface for a workflow named 'create_issue'. On the left, there's a sidebar with options like 'Summary', 'Jobs', 'Run details', 'Usage', and 'Workflow file'. The 'Jobs' section is expanded, showing a single job named 'create_issue'. This job has two steps: 'Set up job' and 'echo inputs'. The 'echo inputs' step is expanded, revealing its log output. The log shows the command 'Run echo This is a title `ls -la`' followed by the output of the command 'ls -la', which includes files named 'runner' and 'docker' with specific permissions and timestamps. A red oval highlights the log output of the 'echo inputs' step.

- Now, let's change the code to not directly interpret the values passed in, but to use environment variables instead. To make things simpler, let's add a bypass to the ruleset to allow us to commit directly to main for this activity. Go to the repository's **Settings**, then select **Rules / Rulesets** on the left, and click on the ruleset to the right that you previously setup.

The screenshot shows the 'General' settings page for a GitHub repository. On the left, there's a sidebar with sections like 'Access', 'Collaborators', 'Moderation options', 'Code and automation', 'Branches', 'Tags', and 'Rules'. The 'Rules' section is expanded, showing the 'Rulesets' tab selected. On the right, there's a 'Rulesets' section with a heading 'Protect main' and a sub-section '3 rules • targeting 1 branch'. A dropdown menu next to 'All' shows 'All' is selected. A red oval highlights the 'Protect main' section.

Next, in the page for the ruleset, in the **Bypass list** section, click on **+ Add bypass**. In the **Add bypass** dialog that comes up, select the **Repository admin** role, and check that box. Then click on the **Add selected** button.



Be sure to click on ***Save changes*** at the bottom of the screen to persist the bypass. After that, you should see a quick pop-up acknowledging the change.

- Now let's go back to the **Code** tab and fix the workflow. Click on the `.github/workflows/create-issue.yml` file to open it and click on the pencil icon to edit it.

The screenshot shows the GitHub repository interface. On the left, the navigation bar includes "Code", "Issues 11", "Pull requests", "Discussions", "Actions", "Projects", "Wiki", "Security", "Insights", and "Settings". The main area shows the file structure under "Files": "main" (selected), ".github/workflows" (expanded), "create-issue.yml" (selected), and "models". The "create-issue.yml" file is displayed with its content: "name: Create Issue". There are "Raw" and "Edit this file" buttons at the bottom of the code editor.

- Replace the code at line 28 "`run: echo ${{ inputs.title }} ${{ inputs.body }}`" with this code:

```
env:
  ARGS: ${{ inputs.title }} ${{ inputs.body }}
run: echo "$ARGS"
```

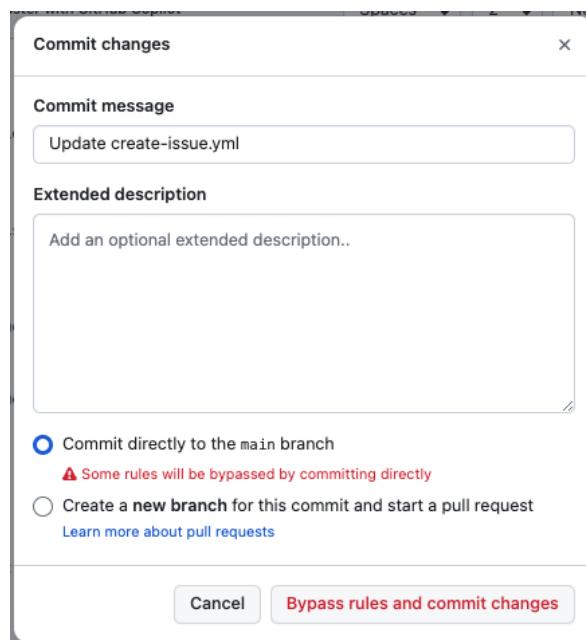
Be sure that it lines up as shown in the next screenshot.

```

10     inputs:
11       title:
12         description: 'Issue title'
13         required: true
14       body:
15         description: 'Issue body'
16         required: true
17
18
19   jobs:
20
21     create_issue:
22       runs-on: ubuntu-latest
23
24     permissions:
25       issues: write
26     steps:
27       - name: echo inputs
28         env:
29           ARGS: ${{ inputs.title }} ${{ inputs.body }}
30         run: echo "$ARGS"
31

```

9. After making the change, click on the **Commit changes...** button and commit the change directly to the main branch. Notice the warning message because of the bypass we put in place. Click on the **Bypass rules and commit changes** button when ready.



10. At this point, another workflow run will have kicked off, but without any parameters. We can prove out if the changes fixed the potential script injection. Click back on the **Actions** tab, select the **Create Issue** workflow from the left, and then fill in the *Run workflow* dialog as before. This time the command should not be executed but simply echoed out.

The screenshot shows the GitHub Actions interface. On the left, under the 'Actions' tab, the 'Create Issue' workflow is selected. The main area displays two workflow runs. The first run shows the command 'ls -la' being echoed. A modal window is open for the second run, showing the inputs 'Issue title' (This is a title) and 'Issue body' ('ls -la').

11. After the workflow run is completed, you can drill into the logs and verify the arguments were echoed correctly and not executed.

The screenshot shows the GitHub Actions log for 'Create Issue #3'. The 'create_issue' job is selected. The log shows the following steps:

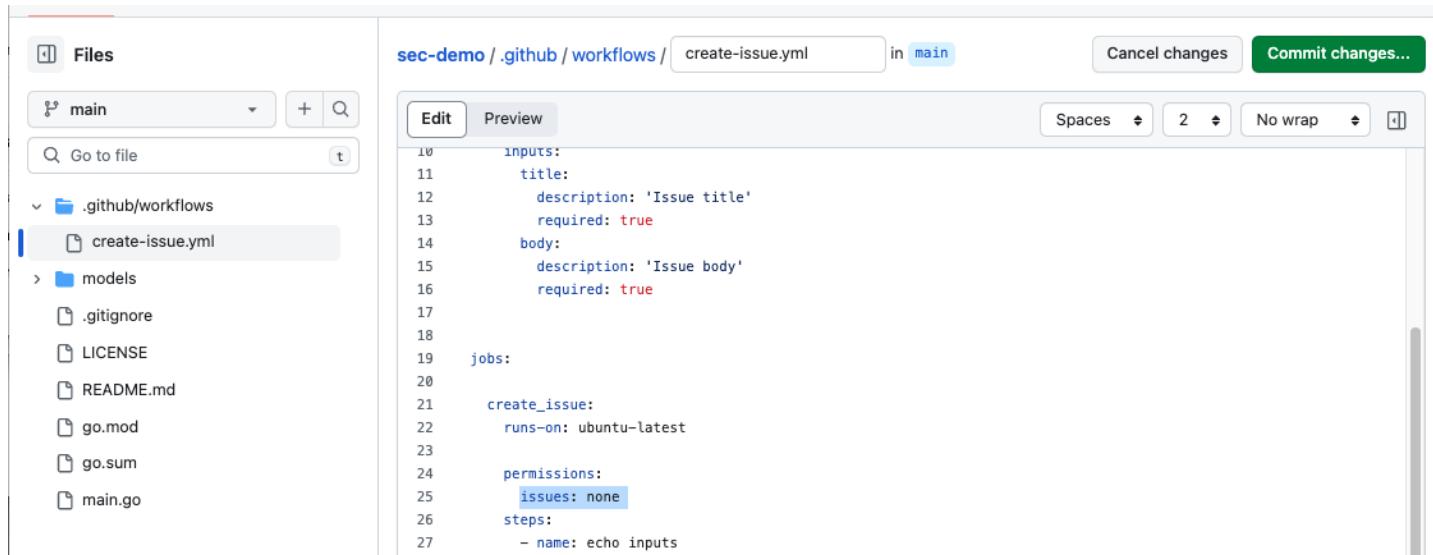
- > Set up job 1s
- > echo inputs 0s
 - 1 ► Run echo "\$ARGS"
 - 6 This is a title `ls -la`
- > Create issue using REST API 0s

END OF LAB

Lab 7 – Using a personal access token with a secret in a workflow

Purpose: In this lab, we'll see how to encapsulate a personal access token in a secret and use that in a workflow.

1. The workflow to create a GitHub issue that we used in the last lab authenticates using the "built-in" GITHUB_TOKEN. It must have permissions to execute the necessary code. Let's see what happens if we remove that permission. Edit the file `.github/workflows/create-issue.yml` and change the line under `permissions`: from `issues: write` to `issues: none`. **Commit your changes** to the **main** branch when done.



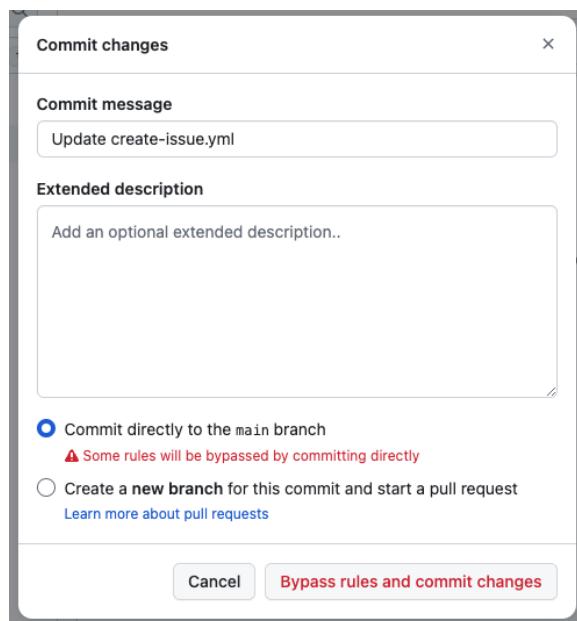
```

Files          sec-demo / .github / workflows / create-issue.yml in main
Cancel changes Commit changes...
main
Go to file
.github/workflows
create-issue.yml
models
.gitignore
LICENSE
README.md
go.mod
go.sum
main.go

Edit Preview
10   inputs:
11     title:
12       description: 'Issue title'
13       required: true
14     body:
15       description: 'Issue body'
16       required: true
17
18
19   jobs:
20
21     create_issue:
22       runs-on: ubuntu-latest
23
24   permissions:
25     issues: none
26   steps:
27     - name: echo inputs

```

You will need to bypass the rules to commit the changes again.



2. Go to the **Actions** tab, select the **Create Issue** workflow and run it as before.

The screenshot shows the GitHub Actions interface for the 'Create Issue' workflow. On the left, a sidebar lists 'Actions', 'New workflow', 'All workflows', 'CodeQL', 'Create Issue' (which is selected), 'Management', 'Caches', and 'Runners'. The main area displays the workflow details for 'create-issue.yml'. It shows '6 workflow runs' with the most recent four being successful ('Create Issue #17', '#16', '#15', and '#14'). Each run was manually triggered by 'gwstudent' and completed successfully ('main'). To the right, a configuration sidebar allows setting the 'Branch' to 'main', defining the 'Issue title' as 'This is a title', and the 'Issue body' as 'This is the body text'. A green 'Run workflow' button is at the bottom. A timestamp '1 hour ago' is visible at the bottom right.

3. This run will fail because the token does not have permissions. You can see this by drilling into the logs for the job. Click on the commit message in the most recent run, select the *create_issue* job and expand the step to “Create issue using REST API”. You’ll see a 403 error.

The screenshot shows the GitHub Actions job logs for the 'create_issue' job in the most recent run (#19). The logs are displayed in a dark-themed terminal window. The job starts with 'Set up job' and 'echo inputs', both of which succeed. The 'Create issue using REST API' step fails with a red error icon. The log output shows the command 'curl --request POST \', followed by a progress table, and finally the error message 'curl: (22) The requested URL returned error: 403' at line 18. The status bar at the bottom indicates 'Error: Process completed with exit code 22.'

4. Let's update our personal access token (PAT) to use instead in the workflow. As you did in the first lab, go to your **Developer Settings** (or github.com/settings/apps). Then, under **Personal access tokens**, select **Fine-grained tokens**. Your token that you created in lab 1 should show up. Click on the name you gave it.

The screenshot shows the GitHub Developer Settings interface. On the left, there's a sidebar with options like GitHub Apps, OAuth Apps, Personal access tokens (with sub-options: Fine-grained tokens and Tokens (classic)), and a Beta button. The main area is titled "Fine-grained personal access tokens (Beta)". It says, "These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS." A token named "Security Fundamentals" is listed, with its name circled in red. The token details show it was last used within the last week and has an expiration date of Monday, March 11, 2024. There are "Delete" and "Generate new token" buttons.

5. On the next screen, click on the **Edit** button.

The screenshot shows the "Edit" screen for the "Security Fundamentals" token. The left sidebar is identical to the previous screenshot. The main area shows the token details: "Token for security fundamentals workshop", "Created today.", and "Expires on Mon, Mar 11 2024.". Below this, there's a section for "Access on gwstudent" with an "Edit" button circled in red. Further down is a "Repository access" section.

6. On the next screen, find the **Permissions** section, then click to expand the **Repository permissions** section.

The screenshot shows the "Permissions" section. It includes a link to the "permissions documentation". A "Repository permissions" section is shown, with a note that 2 items are selected and a description: "Repository permissions permit access to repositories and related resources." An "Edit" button is circled in red.

7. In the **Repository Permissions** section, find the **Issues** row and change the permissions to **Read and write**.

Environments ⓘ Manage repository environments. Access: No access ▾

Issues ⓘ Issues and related comments, assignees, labels, and milestones. Access: Read and write ▾

Merge queues ⓘ Manage a repository's merge queues

Metadata ⓘ mandatory Search repositories, list collaborators, and access repository metadata.

Select an access level X

- No access
- Read-only
- Read and write

7. Scroll to the bottom of the page and click on the **Update** button to save your changes.

Overview

3 permissions for 1 of your repositories >

0 Account permissions >

Update **Cancel**

This token will be ready for use immediately.

8. To use our token with the workflow, we need to store it in a secret for the action. Go back to the repository's Settings, and on the left side, in the **Secrets and variables** section, select **Actions**. Then click on **New repository secret**.

gwstudent / sec-demo

Code Issues 13 Pull requests Discussions Actions Projects Wiki Security Insights Settings 1

General

Access Collaborators Moderation options

Code and automation

Branches Tags Rules Actions Webhooks Environments Codespaces Pages

Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Secrets Variables

Environment secrets

This repository has no environment secrets. Manage environment secrets

Repository secrets

This repository has no secrets. New repository secret 3

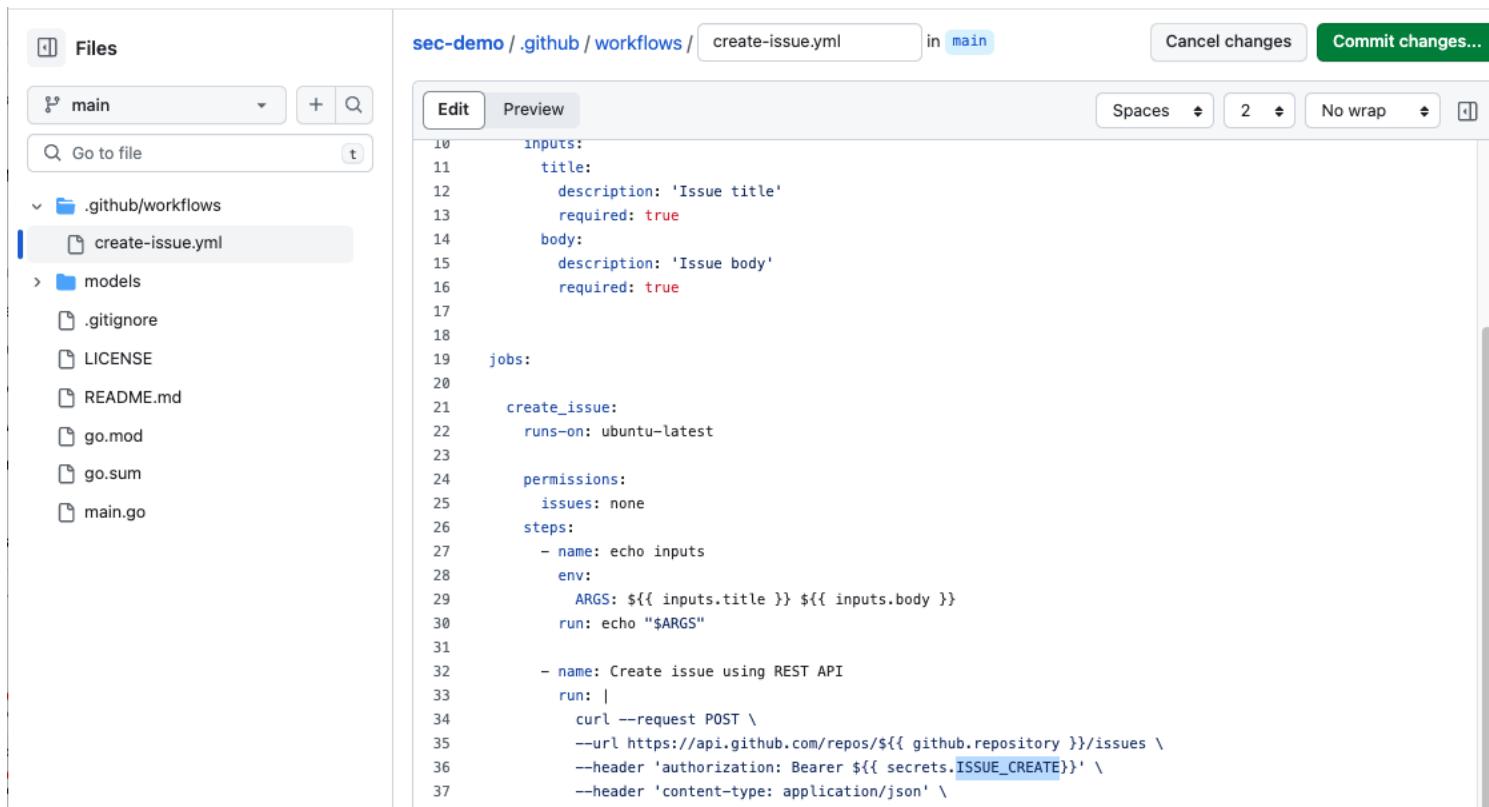
Code security and analysis Deploy keys Secrets and variables 2 Actions Codespaces

9. On the screen for the new secret, enter a name like ISSUE_CREATE in the **Name** field. In the **Secret** field, paste the personal access token you previously generated. Then click on the **Add secret** button.

The screenshot shows the GitHub Actions secrets creation interface. On the left, there's a sidebar with various navigation options: General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, and Pages. The 'Actions' section is expanded. The main area is titled 'Actions secrets / New secret'. It has two input fields: 'Name *' (containing 'ISSUE_CREATE') and 'Secret *' (containing a long personal access token). At the bottom is a green 'Add secret' button.

10. Now, we need to change the workflow to use the new token we create via the secret instead of the secret with the GitHub token. Edit the file `.github/workflows/create-issue.yml` and change the line (probably around line 35 or 36)

```
--header 'authorization: Bearer ${{ secrets.GITHUB_TOKEN}}' \
to
--header 'authorization: Bearer ${{ secrets.ISSUE_CREATE}}' \
```

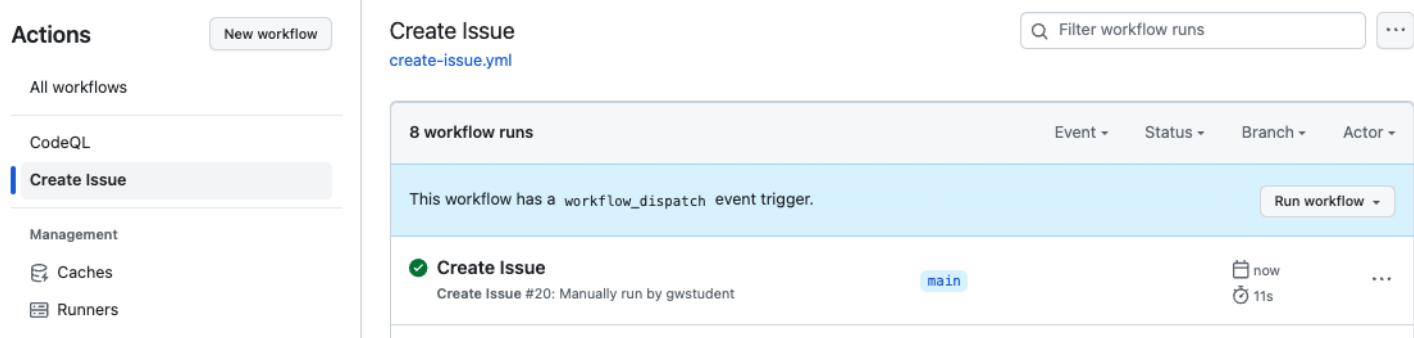


```

10   inputs:
11     title:
12       description: 'Issue title'
13       required: true
14     body:
15       description: 'Issue body'
16       required: true
17
18
19   jobs:
20     create_issue:
21       runs-on: ubuntu-latest
22
23
24     permissions:
25       issues: none
26     steps:
27       - name: echo inputs
28         env:
29           ARGS: ${{ inputs.title }} ${{ inputs.body }}
30         run: echo "$ARGS"
31
32       - name: Create issue using REST API
33         run:
34           curl --request POST \
35             --url https://api.github.com/repos/${{ github.repository }}/issues \
36             --header 'authorization: Bearer ${{ secrets.ISSUE_CREATE }}' \
37             --header 'content-type: application/json' \

```

11. Commit the changes back to the main branch (bypassing the rules as before). Then run the workflow as before, from the **Actions** menu (*Create Issue* under All workflows). You should see that even though we took away the permissions, those were only for the GitHub token. The PAT allows the workflow to run as expected.



Event	Status	Branch	Actor
Workflow run	Success	main	gwstudent

END OF LAB

That's all - THANKS