

GitHub Security Fundamentals

Best practices, tools, and strategies for creating and collaborating securely

Revision 1.2 – 01/28/24

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.

The screenshot shows the GitHub repository page for 'sec-demo'. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. Below the navigation bar, the repository name 'sec-demo' is shown as 'Public'. To the right of the repository name is a 'Watch' button, a 'Fork' button (which is circled in red), and a 'Star' button. A tooltip above the 'Fork' button says 'Fork your own copy of skillrepos/sec-demo'. Below the repository name, there are sections for 'main' (with 2 branches and 0 tags), a file search bar, and an 'Add file' button. The main content area displays a commit history from 'Brent Laster' with three commits: 'initial add' (models, properties, .gitignore), all made 8 minutes ago. To the right of the commit history is an 'About' section with details like 'Repo for use in GitHub Security Fundamentals workshop', 'MIT license', 'Activity', '0 stars', and '1 watching'.

The screenshot shows the 'Create a new fork' dialog box. It starts with a brief description of what a fork is. Below that, it says 'Required fields are marked with an asterisk (*).'. There are two main input fields: 'Owner *' (set to 'gwstudent') and 'Repository name *' (set to 'sec-demo'). A note says 'sec-demo is available.' Below these, there's a 'Description (optional)' field containing 'Repo for use in GitHub Security Fundamentals workshop'. At the bottom of the form is a checkbox labeled 'Copy the main branch only' with the sub-instruction 'Contribute back to skillrepos/sec-demo by adding your own branch. [Learn more](#)'. A note at the very bottom says '(i) You are creating a fork in your personal account.' At the bottom right is a large green 'Create fork' button. A blue speech bubble with the text 'Uncheck' points to the 'Copy the main branch only' checkbox.

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).)

gwstudent (gwstudent)
Your personal account

Public profile

Name
Name

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email
Select a verified email to display

You have set your email address to private. To toggle email privacy, go to [email settings](#) and uncheck "Keep my email address private."

Bio
Tell us a little bit about yourself

You can @mention other users and organizations to link to them.

Pronouns
Don't specify

URL

gwstudent
Set status
Your profile
Switch account
Your repositories
Your projects
Your organizations
Your enterprises
Your stars
Your sponsors
Your gists
Upgrade
Try Enterprise
Copilot
Feature preview
Settings
GitHub Docs
GitHub Support
Sign out

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.

gwstudent (gwstudent)
Your personal account

SSH keys

New SSH key

There are no SSH keys associated with your account.
Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

New GPG key

There are no GPG keys associated with your account.
Learn how to [generate a GPG key and add it to your account](#).

Vigilant mode

Flag unsigned commits as unverified
This will include any commit attributed to your account but not signed with your GPG or S/MIME key.

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 like 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 (Private vulnerability reporting, Integrations), and finally Code security and analysis. The main content area is titled 'Code security and analysis' and discusses security and analysis features. It has two main sections: 'User' and 'Repositories'. Under 'User', there's a 'Push protection for yourself' section with a 'Beta' label, which says 'Block commits that contain supported secrets across all public repositories on GitHub.' There's also a 'Give feedback' section with a text input field containing 'I didn't find it useful'. Under 'Repositories', there's a 'Private vulnerability reporting' section with a 'Beta' label, which says 'Allow your community to privately report potential security vulnerabilities to maintainers and repository owners.' It includes 'Disable all' and 'Enable all' buttons, and a checkbox for 'Automatically enable for new public repositories'. A status bar at the bottom indicates '1 hour ago'.

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 like 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 (Private vulnerability reporting, Integrations, Applications, Scheduled reminders), and finally Security log. The main content area is titled 'Recent events' and lists several audit logs. The first event is 'gwstudent – codespaces.destroy' by '136.56.54.83' at 'Cary, North Carolina, United States' 1 hour ago. The second event is 'gwstudent – codespaces.destroy' by '136.56.54.83' at 'Cary, North Carolina, United States' 1 hour ago. The third event is 'gwstudent – private_vulnerability_reporting_new_repos.disable' by '136.56.54.83' at 'Cary, North Carolina, United States' 1 hour ago. The fourth event is 'gwstudent – private_vulnerability_reporting_disable' by '136.56.54.83' at 'Cary, North Carolina, United States' 1 hour ago. The fifth event is 'gwstudent – dependabot_alerts_new_repos.enable' by '136.56.54.83' at 'Cary, North Carolina, United States' 1 hour ago. Each event entry shows detailed information such as timestamp, document ID, action, actor, created at, name, operation type, org, owner, public_repo, pull_request_id, repo, repo_id, user, user_agent, and user_id.

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 content area has a sidebar with 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens' sections. Under 'Personal access tokens', 'Fine-grained tokens' is selected and marked as 'Beta'. A search bar at the top right says 'Type / to search'. Below the sidebar, a heading reads 'Fine-grained personal access tokens (Beta)'. A sub-instruction says 'Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#)'. There is a 'Generate new token' button on the right.

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 title is 'New fine-grained personal access token (Beta)'. It says 'Create a fine-grained, repository-scoped token suitable for personal API use and for using'. The 'Token name *' field contains 'Security Fundamentals'. The 'Expiration *' dropdown is set to '30 days' with the note 'The token will expire on Mon, Feb 5 2024'. 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 'Select repositories' button is shown with the repository 'sec-demo' listed in the dropdown. At the bottom, there is a 'Personal access token' summary: '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 features with their current access levels. The 'Contents' item is highlighted, showing a dropdown menu with options: 'Select an access level', 'No access', 'Read-only', and 'Read and write' (which is checked). The other items listed are Actions, Administration, Code scanning alerts, Codespaces, Codespaces lifecycle admin, Codespaces metadata, Codespaces secrets, Commit statuses, and Dependabot alerts.

Action	Access Level
Actions	No access
Administration	No access
Code scanning alerts	No access
Codespaces	No access
Codespaces lifecycle admin	No access
Codespaces metadata	No access
Codespaces secrets	No access
Commit statuses	No access
Contents	Read and write
Dependabot alerts	No access
Dependabot secrets	No access

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 Developer Settings interface. The URL is `github.com/settings/tokens?type=beta`. On the left, there's a sidebar with options: GitHub Apps, OAuth Apps, Personal access tokens (selected), and Tokens (classic). Under Personal access tokens, 'Fine-grained tokens' is selected, indicated by a green 'Beta' badge. The main area is titled 'Fine-grained personal access tokens (Beta)'. It contains a note: 'These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS.' Below this is a token card for 'github_pat_11AHIKLH0WbtM3dsCC5mM_ijGxYCR1ITC7odA8i', which was generated on Mon, Feb 5 2024. The card includes a copy icon and a delete button labeled 'Delete'.

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.

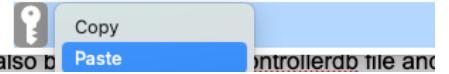


If instead, you are on the command line and prompted for a password, just paste the token in at the prompt. Note that it will not show up on the line, but you can just hit enter afterwards.

```
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. Go to the **Settings** tab for the repository, then select **Branches** on the left. In the new screen, click on the **Add 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.

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

Branch protection rules

Add rule

dev* Currently applies to 1 branch Edit Delete

Access

Collaborators

Moderation options

Code and automation

Branches

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
```

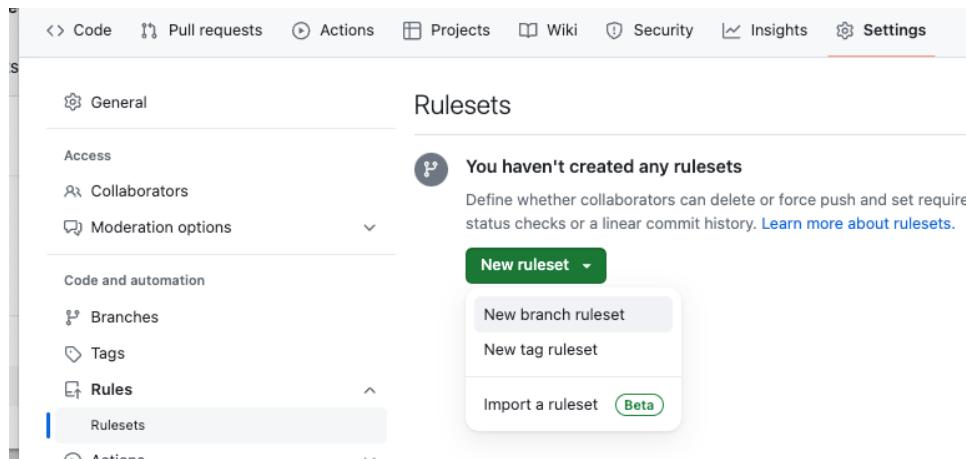
```
$ git commit -am "Update README"
```

```
$ 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*".

The screenshot shows the GitHub Settings interface for creating a new ruleset. The left sidebar has sections like General, Access, Collaborators, Moderation options, Code and automation (Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages), Security (Code security and analysis, Deploy keys, Secrets and variables), Integrations, and GitHub Apps. The main area is titled 'Rulesets / New branch ruleset'. It includes a 'Protect your most important branches' section, a 'Ruleset Name' input field containing 'Protect main', an 'Enforcement status' dropdown set to 'Active', a 'Bypass list' section with a note that it's empty, and a 'Target branches' section where 'Default' is selected. There are buttons for '+ Add bypass' and '+ Add target'.

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.

The screenshot shows the 'Additional settings' section of the branch protection configuration. It lists several checkboxes: 'Restrict deletions' (checked), 'Require linear history' (unchecked), 'Require deployments to succeed' (unchecked), 'Require signed commits' (unchecked), 'Require a pull request before merging' (checked), 'Require status checks to pass' (unchecked), and 'Block force pushes' (checked). A 'Create' button is at the bottom.

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, Rules, and Rulesets. The Rulesets section is currently selected, indicated by a blue bar at the bottom. In the main area, there's a list titled 'Rulesets' with one item: 'Protect main' (3 rules targeting 1 branch). A green button labeled 'New ruleset' is visible in the top right corner.

- 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

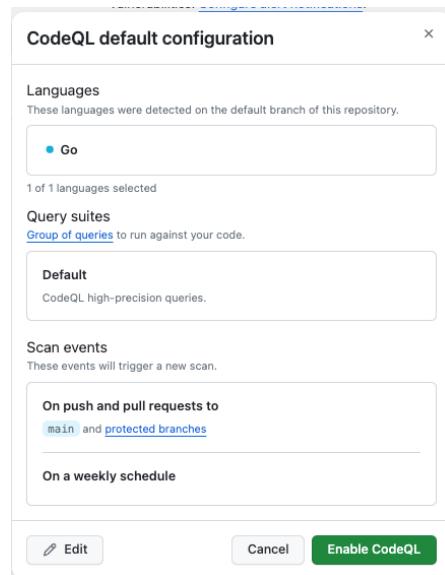
Purpose: In this lab, we'll see how to enable code scanning on our repository

1. 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 'Deploy keys' and 'Secrets and variables'. The main area is titled 'Code security and analysis' and contains sections for 'Private vulnerability reporting' (Beta), 'Dependency graph', 'Dependabot', 'Dependabot alerts', 'Dependabot security updates', 'Grouped security updates' (Beta), 'Dependabot version updates', and 'Code scanning'. Each section has an 'Enable' button. At the bottom, there's a 'Tools' section.

2. 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' (with a 'Set up' button), 'Other tools', 'Protection rules', and 'Pull request check failure'. The 'Default' configuration is selected for CodeQL analysis, which is described as automatically finding the best configuration for the repository. For 'Pull request check failure', the option 'High or higher / Only errors' is selected.



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

3. 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.

The screenshot shows the GitHub Actions page. On the left, the 'Actions' tab is selected, showing the 'All workflows' section with 'CodeQL' listed. On the right, the 'All workflows' table shows one workflow run for 'CodeQL Setup' which completed 10 minutes ago. Below this, the 'CodeQL Setup #1' job details are shown, listing the steps: Set up job, Checkout repository, Initialize CodeQL, Configure, Setup Go, Autobuild, Perform CodeQL Analysis, Record Successful outcome, Post Perform CodeQL Analysis, and Post Initialize CodeQL.

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 lists several items:

- Security policy • Disabled**: Define how users should report security vulnerabilities for this repository. Button: Set up a security policy.
- Security advisories • Enabled**: View or disclose security advisories for this repository. Button: View security advisories.
- Private vulnerability reporting • Disabled**: Allow users to privately report potential security vulnerabilities. Button: Enable vulnerability reporting.
- Dependabot alerts • Disabled**: Get notified when one of your dependencies has a vulnerability. Button: Enable Dependabot alerts.
- Code scanning alerts • Enabled**: Automatically detect common vulnerability and coding errors. Button: View alerts (circled in red).
- Secret scanning alerts • Disabled**: Get notified when a secret is pushed to this repository. Button: Enable in settings.

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 status message: **All tools are working as expected**. Buttons: Tool status (1) and + Add tool.
- A search bar: **is:open branch:main**.
- A summary: **3 Open** (checkbox) and **0 Closed** (checkbox). Buttons: Language, Tool, Branch, Rule, Severity, Sort.
- Three alert entries, each with a checkbox and a "Database query built from user-controlled sources" message. Each entry also has a "High" severity indicator and a timestamp: "#3 opened 5 minutes ago · Detected by CodeQL in models/models.go:76", "#2 opened 5 minutes ago · Detected by CodeQL in models/models.go:57", and "#1 opened 5 minutes ago · Detected by CodeQL in models/models.go:38". Each entry also has a "main" tag.

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 repository's security page with a specific alert highlighted. The alert is for a database query built from user-controlled sources, identified by the file models/models.go:76. The code snippet shows a direct string concatenation:

```
73 // the query, you should be using a parameterized query.
74 func ReadQuery(r string) ([]Book, error) {
75     // Fix: rows, err := DB.Query("SELECT * FROM books WHERE read = ?", r)
76     rows, err := DB.Query(fmt.Sprintf("SELECT * FROM books WHERE read = '%s'", r))
```

A callout box highlights the problematic line: `rows, err := DB.Query(fmt.Sprintf("SELECT * FROM books WHERE read = '%s'", r))`.

The alert details include:

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

The alert also provides a tool breakdown:

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

Description: 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.

Timeline:

- First detected in commit 6 minutes ago
- initial add
- 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 should be a yellow bar noting that there were recent pushes to *dev* and showing a button to **Compare & pull request**. (If you don't see this, you can initiate a new pull request via the **Pull requests** menu at the top.) Click on that button.

The screenshot shows the GitHub repository page for sec-demo. A yellow banner at the top indicates that the dev branch had recent pushes 38 minutes ago. A green button labeled "Compare & pull request" is visible next to the banner.

The repository details shown include:

- sec-demo** (Public)
- Forked from skillrepos/sec-demo
- main branch
- 2 Branches
- 0 Tags
- Go to file search bar
- Code dropdown menu
- This branch is up to date with skillrepos/sec-demo:main
- Contribute and Sync fork buttons

8. 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, the main heading is "Open a pull request". A sub-instruction says "Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more".

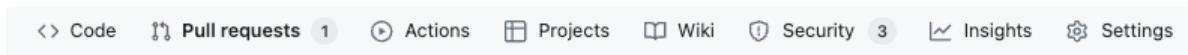
Below the heading, there are dropdown menus for "base: main" and "compare: dev". A green checkmark indicates "Able to merge. These branches can be automatically merged".

The "Add a title" field contains the text "Dev".

The "Add a description" section includes a CKEditor toolbar with buttons for bold, italic, underline, and other rich text options. A text area below the toolbar has the placeholder "Add your description here...".

At the bottom, there are two buttons: "Markdown is supported" and "Paste, drop, or click to add files". To the right, a large green "Create pull request" button is visible.

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

- o fixes for models 60509a9
- o Add README file e1d195b

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

Some checks haven't completed yet Hide all checks
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 Hide all checks
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 options like Actions, New workflow, CodeQL, Management, Caches, and Runners. 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 are sections for Overview, Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, and Code scanning (which is selected). In the main area, under "Code scanning", there is a summary: "All tools are working as expected" and a search bar with the query "is:closed branch:main". Below that, it says "Clear current search query, filters, and sorts". There are three closed alerts listed:

- 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)

A "ProTip!" message at the bottom suggests running CodeQL locally using Visual Studio Code.

This screenshot shows a detailed view of a specific alert. At the top, it says "Code scanning alerts / #3" and "Database query built from user-controlled sources". The alert is labeled as "Fixed" and was last updated "3 minutes ago". The code snippet shown is from "models/models.go:76" and contains a function that reads a string and performs a database query without proper sanitization. A note below the code states: "This query depends on a user-provided value." The alert has a severity of "High", affected branches of "main", and a CWE ID of "CWE-89". It also includes a "Show paths" button and a history section showing its detection and fix across multiple commits and pull requests.

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, we can also have GitHub scan for secrets that may be exposed in our repository files. To enable that functionality, 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, click the **Enable** button.

The screenshot shows the 'Code security and analysis' settings page for a GitHub repository. The 'Secret scanning' section is highlighted. It contains a description: 'Receive alerts on GitHub for detected secrets, keys, or other tokens.' Below this is a note: 'GitHub will always send alerts to partners for detected secrets in public repositories. [Learn more about partner patterns.](#)' At the bottom of the section is a large 'Enable' button, which is circled in red.

- You can also choose to Enable the option to *Block commits that contain supported secrets*.

The screenshot shows the 'Code security and analysis' settings page. The 'Push protection' section is highlighted. It contains a description: 'Block commits that contain [supported secrets](#).' At the bottom of the section is a large 'Enable' button, which is circled in red.

- 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**.

This screenshot shows a GitHub repository interface. At the top, it displays 'main' branch, 3 branches, and 0 tags. A search bar says 'Go to file'. To the right are buttons for 'Add file' and 'Code'. A tooltip for the 'Code' button shows options: '+ Create new file' and 'Upload files'. Below this, a message says 'This branch is 3 commits ahead of skillrepos/sec-demo:main'. A merge pull request from 'gwstudent' is shown, titled 'Merge pull request #1 from gwstudent/dev', with a green checkmark and the commit hash '895af97 · 10 hours ago'. It has 4 commits.

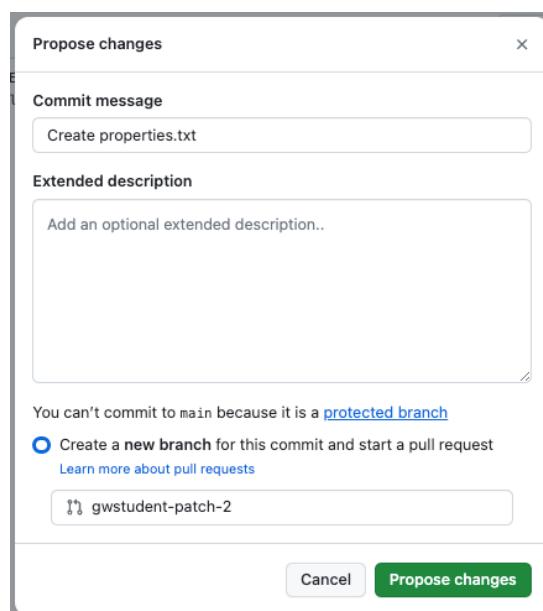
4. In the new file screen, name the file properties/properties.txt and copy and paste the following content into it.

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

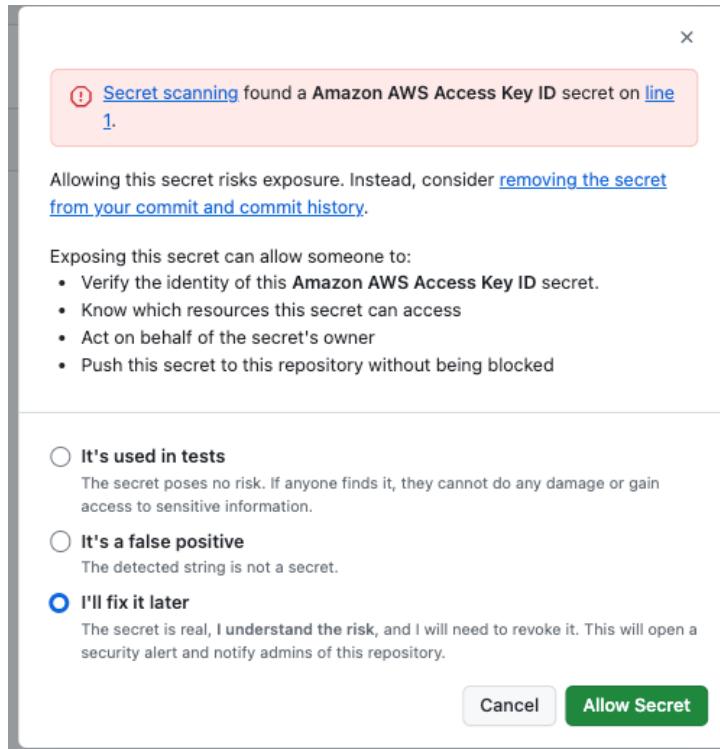
This screenshot shows the GitHub 'Edit' screen for a new file named 'properties.txt' in the 'sec-demo' repository. The file contains the AWS access key ID and secret access key. The 'Commit changes...' button is highlighted in green.

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

5. 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.

Code Pull requests Actions Projects Wiki Security Insights Settings

✓ Secret allowed. You can now commit these changes.

sec-demo / properties / properties.txt in main

Cancel changes **Commit changes...**

Edit Preview Code 55% faster with GitHub Copilot

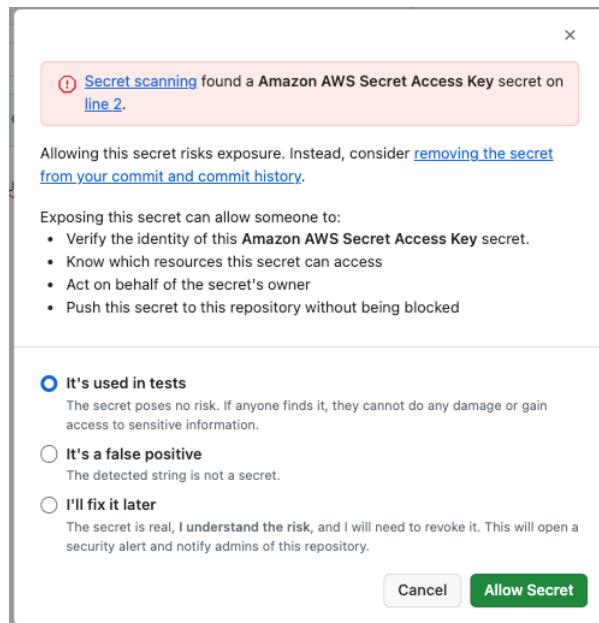
Spaces 2 No wrap

```

1 AWS_ACCESS_KEY_ID="AKIAZBQE445JPKTEAHQD"
2 AWS_SECRET_ACCESS_KEY="wt6lVzza0QFx/U34PU8ErikMbnKiu+bv9jheR0h/D"
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.



9. 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.
10. Now, if you click on the Security tab and the Secret Scanning selection on the side, you'll see the one secret we said we would fix later.

Reporting

Policy

Advisories

Vulnerability alerts

Dependabot

Code scanning

Secret scanning 1

Secret scanning alerts

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:1

11. 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 interface for managing secret scanning alerts. On the left, there's a sidebar with links like Overview, Reporting, Policy, and Secret scanning. The Secret scanning link is highlighted. The main area is titled "Secret scanning alerts". A search bar at the top says "is:open". Below it, a modal window is open, showing "1 selected" and a checkbox checked. The modal title is "Select a close reason". Inside, there are five options: "Revoked" (selected), "Used in tests", "False positive", and "Won't fix". Each option has a brief description. Below the modal is a "Comment" section with a text input field and a placeholder "Add a comment". At the bottom right of the modal are "Cancel" and "Close alert" buttons.

12. 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.

The screenshot shows the GitHub interface for managing secret scanning alerts. The sidebar has the "Secret scanning" link highlighted. The main area is titled "Secret scanning alerts". A search bar at the top says "is:closed". Below it, a table lists two closed alerts. The first alert is for an "Amazon AWS Secret Access Key" and the second for an "Amazon AWS Access Key ID". Both alerts mention they were closed as used in tests. The number "2 Closed" is circled in red in the table header.

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 of the dialog is a large red-outlined **Enable** button. To the right of the dialog, there is another **Enable** button. The entire modal and its surrounding area are highlighted with a red circle.

- 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.

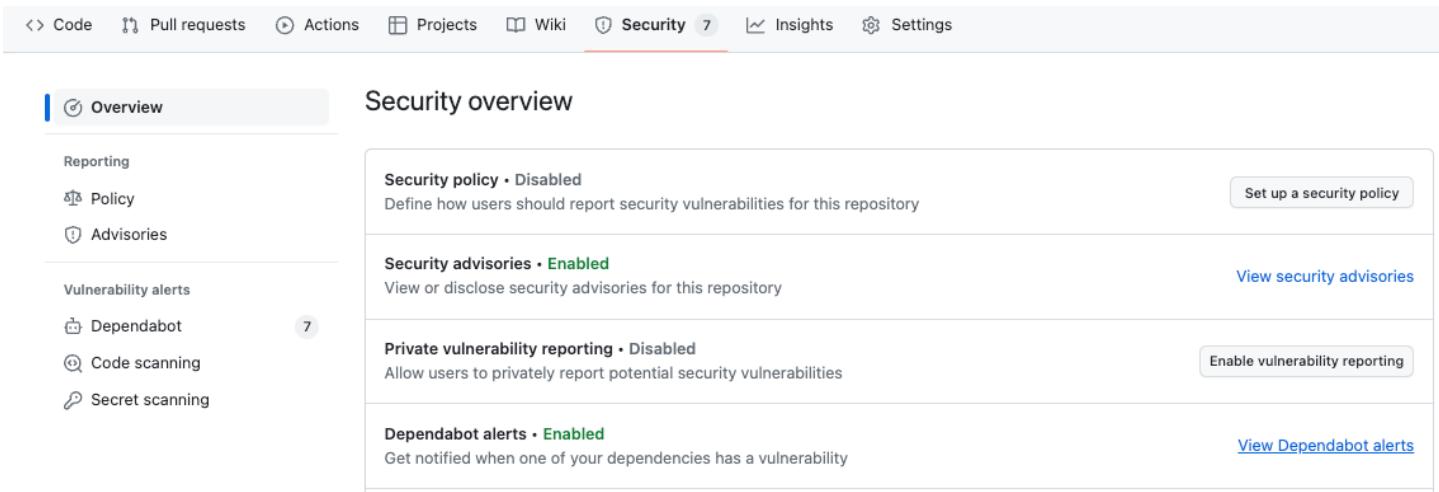
The screenshot shows the GitHub Settings page for a repository. On the left sidebar, under Security, the **Code security and analysis** tab is selected. In the main content area, the **Dependabot alerts** section is visible. It shows a message: "Receive alerts for vulnerabilities that affect your dependencies and manually generate Dependabot pull requests to resolve these vulnerabilities." Below this is a **Disable** button. Under the **Dependabot rules** section, it says "Create your own custom rules and manage alert presets." and "1 rule enabled". A gear icon is also present. The entire Dependabot alerts section is highlighted with a red circle.

- 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. At the top, the **Insights** tab is selected. On the left sidebar, under Pulse, the **Dependency graph** item is selected. The main content area displays the **Dependency graph**. It includes tabs for **Dependencies**, **Dependents**, and **Dependabot**. There is a search bar labeled "Search all dependencies". Below the tabs, three dependency entries are listed:

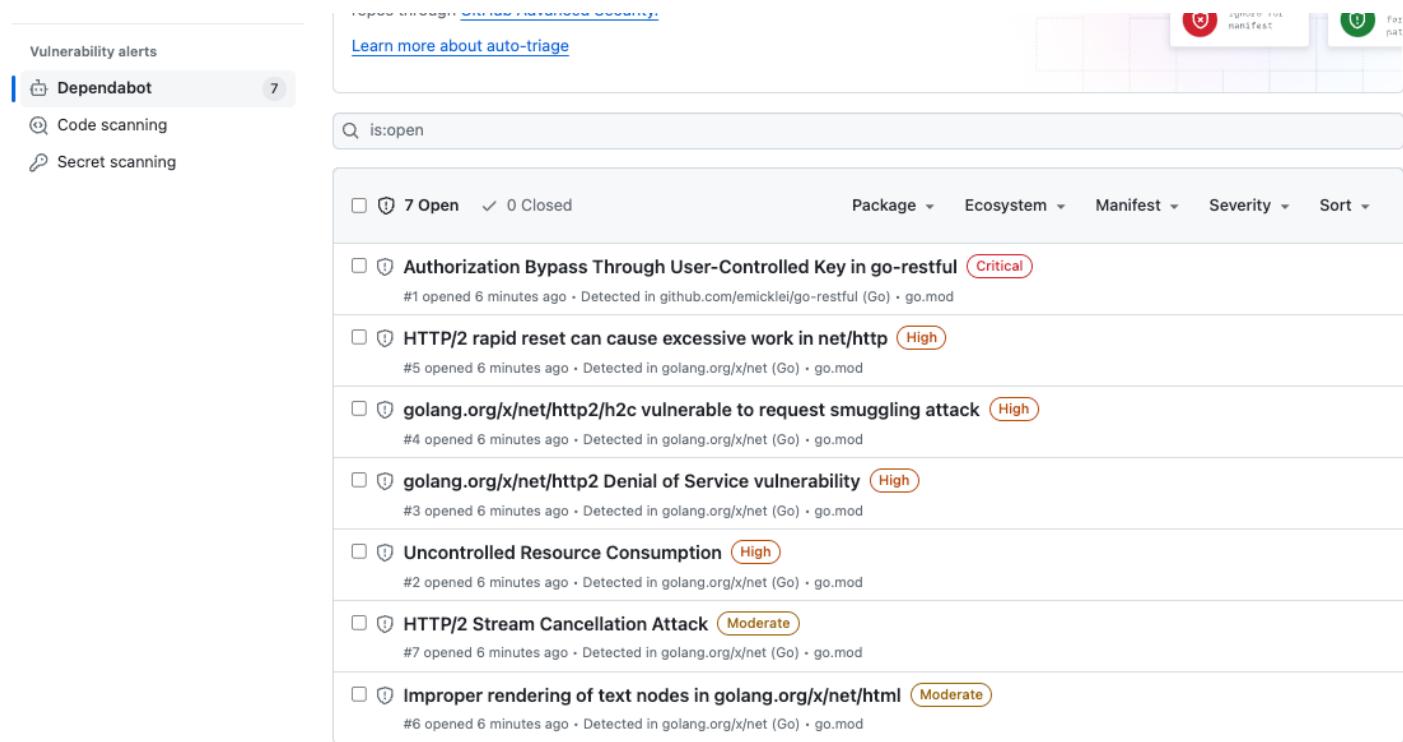
- github.com/emicklei/go-restful** 2.9.5+incompatible
Detected automatically on Jan 10, 2024 (Go modules) · go.mod
Status: 1 critical (red)
- golang.org/x/net** 0.0.0-20220127200216-cd36cc0744dd
Detected automatically on Jan 10, 2024 (Go modules) · go.mod
Status: 4 high (orange)
- github.com/mattn/go-sqlite3** 1.14.8
Detected automatically on Jan 10, 2024 (Go modules) · go.mod

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 for Overview, Reporting, Policy, and more. 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 has a link to "View Dependabot alerts".

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 includes links for Vulnerability alerts, Dependabot (with 7 alerts), Code scanning, and Secret scanning. The main area displays 7 open vulnerabilities, each with a checkbox, a shield icon, and a severity level (Critical, High, Moderate). The vulnerabilities listed are:

- Authorization Bypass Through User-Controlled Key in go-restful (Critical)
- HTTP/2 rapid reset can cause excessive work in net/http (High)
- golang.org/x/net/http2/h2c vulnerable to request smuggling attack (High)
- golang.org/x/net/http2 Denial of Service vulnerability (High)
- Uncontrolled Resource Consumption (High)
- HTTP/2 Stream Cancellation Attack (Moderate)
- 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.

7. 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 a GitHub Dependabot alert for a pull request. The alert indicates that the dependency 'github.com/emicklei/go-restful' has been updated from version 2.9.5+incompatible to 2.16.0+incompatible. It notes that merging this pull request would fix one Dependabot alert in the 'go.mod' file. A green 'Review security update' button is prominently displayed.

8. 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 for a security update. The pull request title is 'Bump github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible #2'. It includes a note that merging the pull request will resolve a critical severity Dependabot alert. The 'Files changed' tab is selected, showing the 'go.mod' and 'go.sum' files. The 'go.mod' file shows the removal of the 'github.com/mattn/go-sqlite3 v1.14.8' dependency. The 'go.sum' file shows the removal of the corresponding checksum entries.

9. 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 two GitHub pages. The top page is a pull request details page for a repository named 'sec-demo'. It displays a commit from 'dependabot' that updated 'github.com/emicklei/go-restful' from version 2.9.5+incompatible to 2.16.0+incompatible. The pull request has passed all checks and can be merged automatically. The bottom page is the merge history for the same pull request, showing it was merged by 'gwstudent' into the 'main' branch. A comment from 'dependabot' is visible, stating that the pull request resolved a Dependabot alert.

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 GitHub Settings page for a repository. On the left, there's a sidebar with options like 'Codepaces', 'Pages', 'Security', 'Integrations', 'GitHub Apps', and 'Email notifications'. The main area is titled 'Social preview' and 'Features'. Under 'Features', there are three sections: 'Wikis' (checked), 'Restrict editing to collaborators only' (checked), and 'Issues' (unchecked). A blue callout bubble points to the 'Issues' checkbox with the text 'Check this box'.

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 interface with the 'Code' tab selected. On the left, the repository structure is visible, including a 'main' branch and a '.github/workflows' folder containing 'create-issue.yml'. The right panel displays the contents of 'create-issue.yml' with syntax highlighting. The code is as follows:

```

1 name: create-issue
2
3 # Controls when the workflow will run
4
5 on:
6   # Allows you to call this manually from the Actions tab

```

- 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 interface with the 'Actions' tab selected. On the left, the 'Create Issue' workflow is selected under the 'All workflows' section. The right panel shows the 'Create Issue' workflow details. It indicates '0 workflow runs' and notes that it has a 'workflow_dispatch' event trigger. A configuration dialog is open on the right, titled 'Run workflow'. It includes fields for 'Use workflow from' (set to 'Branch: main'), 'Issue title *' (containing 'This is a title'), 'Issue body *' (containing 'This is the body text.'), and a 'Run workflow' button.

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 page with the following details:

- Filters:** is:issue is:open
- Labels:** 10
- Milestones:** 0
- New issue** button
- Status:** 6 Open, 0 Closed
- Issue List:**
 - Title:** This is a title
 - Details:** #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 page with the following details:

- Workflow Status:** Workflow run was successfully requested.
- Actions Tab:**
 - Create Issue** is selected.
 - Workflow Configuration:**
 - Name:** Create Issue
 - Description:** create-issue.yml
 - Triggers:** workflow_dispatch
 - Run workflow** button
 - Workflow Run Details:**
 - Event:** workflow_dispatch
 - Status:** Pending
 - Branch:** main
 - Actor:** gwstud
 - Issue Body:** `ls -la`
 - Run workflow** button

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. On the left, a sidebar lists various actions: Actions (selected), New workflow, All workflows, CodeQL, Create Issue (highlighted with a blue bar), Management, Caches, and Runners. The main area displays '2 workflow runs'. The first run, 'Create Issue #12', was triggered by a workflow_dispatch event and completed successfully 1 minute ago. The second run, 'Create Issue #13', was triggered manually by gwstudent and completed successfully 12s ago. A red circle highlights the 'Create Issue' job in the second run's list.

Then click on the job name in the next screen.

The screenshot shows the GitHub issue details page for 'Create Issue #13'. The top navigation bar includes Code, Issues (9), Pull requests, Discussions, Actions (selected), Projects, Wiki, and a three-dot menu. Below the title, there are links for Summary, Jobs (selected), create_issue, Run details, Usage, and Workflow file. The 'create_issue' job is selected in the sidebar. The job summary table shows: Manually triggered 2 minutes ago, Status: Success, Total duration: 12s, and Artifacts: -. The workflow file 'create-issue.yml' is displayed, showing the 'on: workflow_dispatch' trigger and the 'create_issue' job. The 'echo inputs' step is highlighted with a red circle.

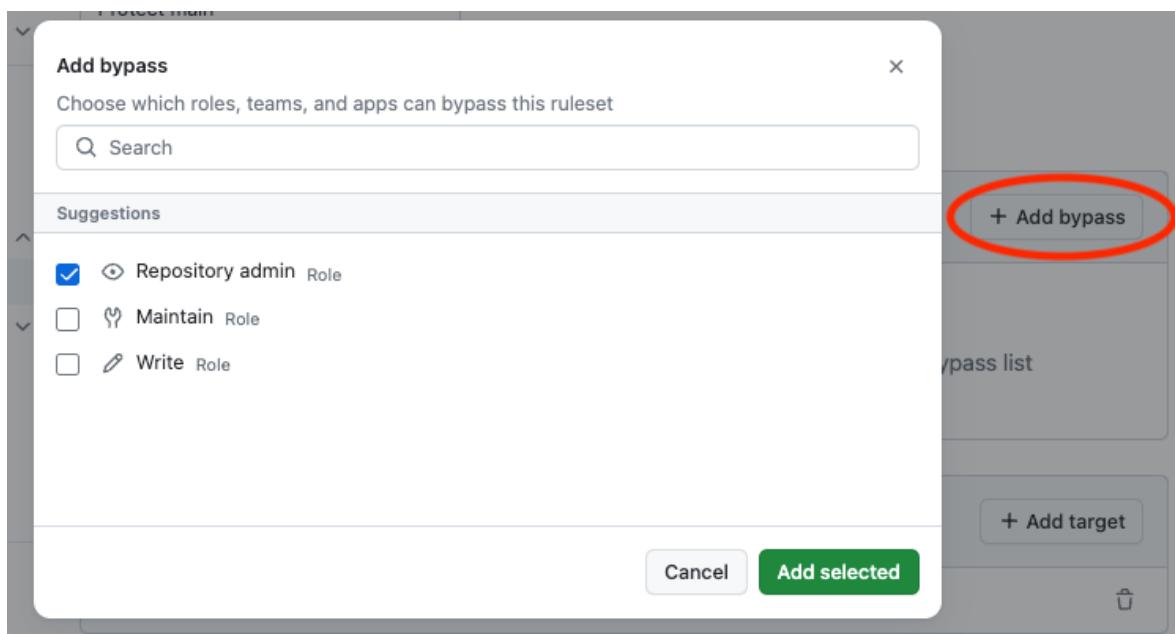
Then click to expand the *echo inputs* step in the log. Notice that the text we put in as the body of the issue was actually executed and run as a command.

The screenshot shows the GitHub issue details page for 'Create Issue #13', focusing on the log for the 'create_issue' job. The top navigation bar is identical to the previous screenshot. The sidebar shows 'create_issue' selected. The log for the 'create_issue' job is displayed, with the 'echo inputs' step expanded. The expanded log shows the command 'run echo This is a title `ls -la`' and its output, which includes 'This is a title' and a directory listing. A large red oval highlights the expanded 'echo inputs' step and its output.

6. 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 **Rulesets** on the left, and click on the ruleset to the right that you previously setup.

The screenshot shows the GitHub repository settings page for a repository named 'Protect main'. On the left, there is a sidebar with various settings categories: General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, and Rulesets. The Rulesets category is currently selected, indicated by a blue bar at the bottom of the sidebar. On the right, under the 'Rulesets' heading, there is a card for 'Protect main' which says '3 rules • targeting 1 branch'. Below this card, there are sections for 'Bypass list' and 'Targets'.

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.

7. 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 interface with the 'Code' tab selected. On the left, the 'Files' sidebar shows a tree view with 'main', '.github', and 'models' folders. The 'create-issue.yml' file under '.github' is selected and highlighted in grey. The main area displays the contents of 'create-issue.yml':

```
1
2   name: Create Issue
3
```

A tooltip 'Edit this file' is shown over the edit icon in the toolbar.

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

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

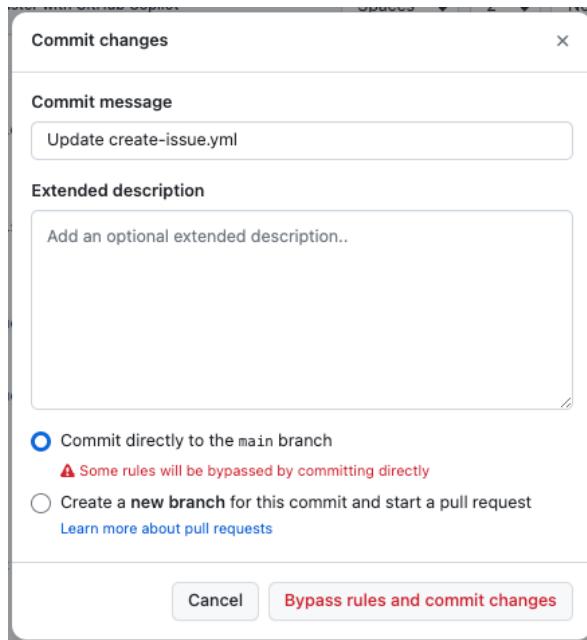
Be sure that it lines up as shown in the screenshot below.

The screenshot shows the GitHub interface with the 'Edit' tab selected for the 'create-issue.yml' file. The code editor shows the following YAML configuration:

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

The 'env:' section has been modified to include 'ARGS'.

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, 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.

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 interface for a workflow named 'create_issue'. The workflow has completed successfully. The log shows two steps: 'Set up job' (0s) and 'echo inputs' (1s). The 'echo inputs' step includes the command 'Run echo "\$ARGS"' and the output 'ls -la' This is the body text.'

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.

The screenshot shows the GitHub repository interface with the '.github/workflows/create-issue.yml' file open for editing. The file content is as follows:

```

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

```

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 sidebar, 'Create Issue' is selected under the 'Actions' category. The main area displays four workflow runs, each with a green checkmark and the label 'Create Issue'. The most recent run was manually triggered by 'gwstudent' on the 'main' branch. A modal window is open on the right, titled 'Run workflow', containing fields for 'Issue title' (set to 'This is a title') and 'Issue body' (set to 'This is the body text'). Below these fields is a large green 'Run workflow' button. At the bottom of the modal, a timestamp indicates it was run 1 hour ago.

3. This run will fail because the token does not have permissions. You can see this by drilling into the logs for the job.

The screenshot shows the GitHub Actions job logs for 'Create Issue #19'. The left sidebar lists 'Summary', 'Jobs', and 'create_issue'. The 'create_issue' job is selected, showing its details. The log output for the 'create_issue' job shows a failure at step 18, where a curl command to create an issue using the REST API returns an error code 403. The log ends with a success message for the final 'Complete job' step.

```

create_issue
failed now in 1s

> ✓ Set up job
0s

> ✓ echo inputs
0s

< ✘ Create issue using REST API
0s

1  ▶ Run curl --request POST \
2    % Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
3                               Dload Upload Total Spent   Left Speed
4
5     0      0      0      0      0      0      0 --:--:-- --:--:-- --:--:--   0
6  35  222      0      0  100    79      0  467 --:--:-- --:--:-- --:--:--  470
7 curl: (22) The requested URL returned error: 403
8 Error: Process completed with exit code 22.

> ✓ Complete job
0s

```

4. Let's create a 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**. Then click on **Generate new token**.

Fine-grained personal access tokens (Beta)

These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS.

Security Fundamentals	Last used within the last week	Delete
Expires on Fri, Feb 9 2024.		

5. As before, give it a name, and under **Repository access**, select **Only select repositories**, and filter it to your **sec-demo** repo.

Personal access tokens (Beta)

Token name *

Workflow

Expiration *

30 days Sat, Feb 10 2024

Description

What is this token for?

Resource owner

gwstudent

Repository access

Public Repositories (read-only)

All repositories

This applies to all current and future repositories owned by the resource owner.
Also includes public repositories (read-only).

Only select repositories

Select at least one repository. Max 50 repositories.
Also includes public repositories (read-only).

Select repositories

sec-demo

gwstudent/sec-demo
Repo for use in GitHub Security Fundamentals workshop

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

Environments	Access: No access
Issues	Access: Read and write
Merge queues	Access: No access
Metadata	Access: Read and write

Select an access level

No access

Read-only

Read and write

7. Click on the **Generate token** button at the bottom and copy your token value.

The screenshot shows the GitHub 'Developer Settings' page under 'Personal access tokens'. A single token is listed with the ID 'github_pat_11AHIKLHQ0YiW24Yshon6b_ZYUUnaYRTRUe80hUQI'. It includes a copy icon and a note that it expires on Saturday, February 10, 2024.

8. To use this new 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**.

The screenshot shows the GitHub repository 'gwstudent / sec-demo' settings. The 'Actions' tab is selected in the sidebar (circled with red number 2). In the main area, the 'Environment secrets' section is shown with a note that no secrets exist. The 'Repository secrets' section is also shown with a note that no secrets exist and a green 'New repository secret' button (circled with red number 3).

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 configuration interface. On the left, there's a sidebar with various settings like General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, and Pages. The 'General' tab is selected. On the right, the main area is titled 'Actions secrets / New secret'. It has two fields: 'Name *' containing 'ISSUE_CREATE' and 'Secret *' containing a long GitHub Personal Access Token. A green 'Add secret' button is at the bottom.

- 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 36)

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

The screenshot shows a GitHub repository interface with a code editor. The repository path is `sec-demo/.github/workflows`, and the file is `create-issue.yml`. The code editor has tabs for "Edit" and "Preview". The "Edit" tab is active, showing the following YAML configuration:

```
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
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. Then run the workflow as before. 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.

Actions

New workflow

All workflows

CodeQL

Create Issue

Management

Caches

Runners

Create Issue

create-issue.yml

8 workflow runs

Event ▾ Status ▾ Branch ▾ Actor ▾

This workflow has a `workflow_dispatch` event trigger.

Run workflow ▾

Create Issue

Create Issue #20: Manually run by gwstudent

main now 11s

END OF LAB

That's all - THANKS