

GitHub Security Fundamentals

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

Revision 1.4 – 02/10/24

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 interface for the 'sec-demo' repository. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. Below that is a header for the 'sec-demo' repository, which is public. The repository has 1 commit from 'Brent Laster' and 2 branches. On the right side, there's an 'About' section with details like 'Repo for use in GitHub Security Fundamentals workshop', 'MIT license', 'Activity', '0 stars', and '1 watching'. A prominent 'Fork' button is located in the top right corner of the repository card, with a red circle drawn around it to indicate it.

This screenshot shows the 'Create a new fork' dialog box. It includes fields for 'Owner' (set to 'gwstudent') and 'Repository name' (set to 'sec-demo'). A note says 'sec-demo is available.' Below these, there's a description field containing 'Repo for use in GitHub Security Fundamentals workshop'. A checkbox labeled 'Copy the main branch only' is present with the explanatory text: 'Contribute back to skillrepos/sec-demo by adding your own branch. [Learn more](#)'. A note at the bottom states '(?) You are creating a fork in your personal account.' A large 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

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

GPG keys
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 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 notice about push protection and a feedback form. The 'Repositories' section includes a notice about private vulnerability reporting and a list of recent events.

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 list of 'Recent events' with details such as timestamp, document ID, action, actor, actor ID, created at, name, operation type, org, owner, public repo, pull request ID, repo, repo ID, user, user agent, and user ID. Each event entry is hyperlinked and includes a 'More' button.

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 blue border and a 'Beta' badge. A sub-menu shows 'Tokens (classic)'. To the right, the title 'Fine-grained personal access tokens' is followed by a 'Beta' badge and a 'Generate new token' button. Below this, a note says 'Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#)'. There is also a 'Tokens (classic)' link at the bottom of the sidebar.

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' with a 'Beta' badge. The instructions say 'Create a fine-grained, repository-scoped token suitable for personal API use and for using'. The 'Token name' field is filled with 'Security Fundamentals'. The 'Expiration' dropdown is set to '30 days', with a note that 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 'sec-demo' entered into the search bar. 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 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.

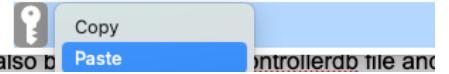


If instead, you are on the command line and prompted for a password, just paste the token in at the prompt. Note that it will not show up on the line, but you can just hit enter afterwards.

```
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.

The screenshot shows the GitHub repository settings for 'sec-demo'. The 'Branch protection rules' section is active, displaying a new rule for the 'dev*' branch. The rule is set to 'Lock branch' and 'Do not allow bypassing the above settings'. A success message 'Branch protection rule created.' is visible at the bottom of the page.

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
```

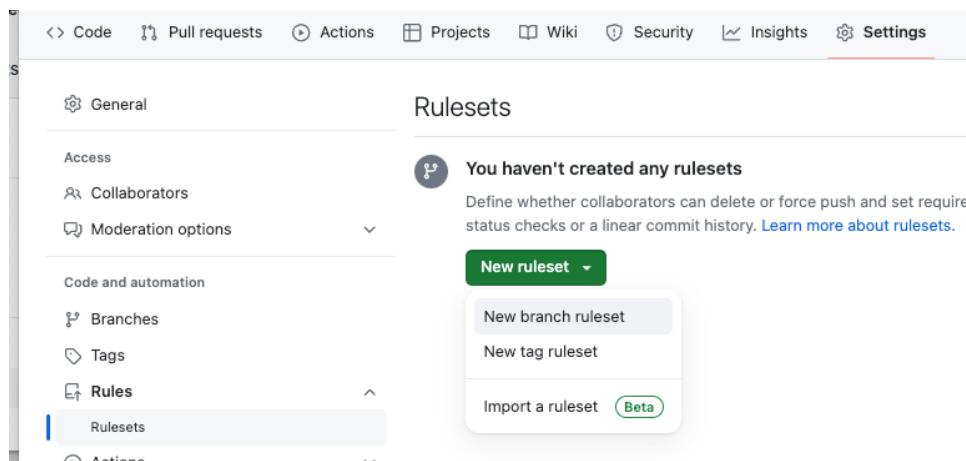
```
$ git commit -am "Update README"
```

```
$ 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 about exempting roles, teams, or apps, 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

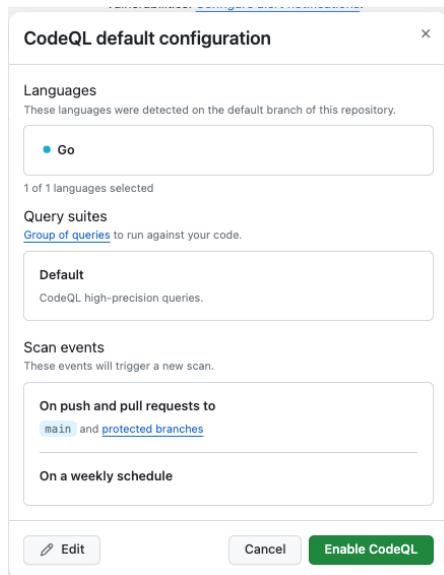
Purpose: In this lab, we'll see how to enable code scanning on our repository

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 'Private vulnerability reporting' (Beta), 'Dependency graph', 'Dependabot' (with sub-sections for Dependabot alerts, Dependabot security updates, Grouped security updates, and Dependabot version updates), and 'Code scanning' (with sub-sections for Tools, CodeQL analysis, Other tools, Protection rules, and Pull request check failure). Each section has an 'Enable' button.

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. For Pull request check failure, the 'High or higher / Only errors' option is selected.



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

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 Actions tab shows the "All workflows" section. A new workflow named "CodeQL" is listed. The "CodeQL" workflow has one run labeled "CodeQL Setup".

The "CodeQL Setup #1" workflow run details are shown. The steps are: 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), and Post Initialize CodeQL (1s). Total duration: 3m 27s.

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.

Security overview

- Reporting**
- Policy**
- Advisories**
- Vulnerability alerts**
- Dependabot**
- Code scanning** 3
- Secret scanning**

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

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

Code scanning

- Reporting**
- Policy**
- Advisories**
- Vulnerability alerts**
- Dependabot**
- Code scanning** 3
- Secret scanning**

All tools are working as expected		Tool status 1	+ Add tool
<input type="text"/> is:open branch:main			
<input type="checkbox"/>	3 Open ✓ 0 Closed	Language Tool Branch Rule Severity Sort	
<input type="checkbox"/>	Database query built from user-controlled sources High	#3 opened 5 minutes ago • Detected by CodeQL in models/models.go:76	main
<input type="checkbox"/>	Database query built from user-controlled sources High	#2 opened 5 minutes ago • Detected by CodeQL in models/models.go:57	main
<input type="checkbox"/>	Database query built from user-controlled sources High	#1 opened 5 minutes ago • Detected by CodeQL in models/models.go:38	main

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 are as follows:

- 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.

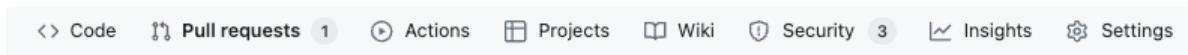
The repository details shown include:

- 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, a banner says "Open a pull request" and "Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more". A message indicates "Able to merge. These branches can be automatically merged." Below the banner, there are fields for "base: main" and "compare: dev". The "Add a title" field contains "Dev". The "Add a description" section has a CKEditor toolbar and a text area placeholder "Add your description here...". At the bottom, it says "Markdown is supported" and "Paste, drop, or click to add files", followed by a large green "Create pull request" button.

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's a summary: "All tools are working as expected" and a search bar with "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:

```

models/models.go:76
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 = ?")
76     rows, err := DB.Query(fmt.Sprintf("SELECT * FROM books WHERE read = '%s'", r))
    This query depends on a user-provided value.

```

Below the code, there's a "Show paths" button. A table provides details about the tool, rule ID, and query. It also notes that the alert was first detected in commit 29 minutes ago and was fixed in branch main 3 minutes ago. A merge pull request from gwstudent/dev was merged, and the issue was verified.

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 'Security' settings page for a GitHub repository. The 'Code security and analysis' tab is selected. At the bottom of the page, under the 'Secret scanning' section, there is an 'Enable' button which is highlighted with a red circle.

- Also select the button to **Enable** the **Push protection** option to *Block commits that contain supported secrets*.

The screenshot shows the 'Security' settings page for a GitHub repository. The 'Code security and analysis' tab is selected. Under the 'Push protection' section, there is an 'Enable' button which is highlighted with a red circle.

- 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 branch is 3 commits ahead of [skillrepos/sec-demo:main](#).

gwstudent Merge pull request #1 from gwstudent/dev 895af97 · 10 hours ago

- 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.

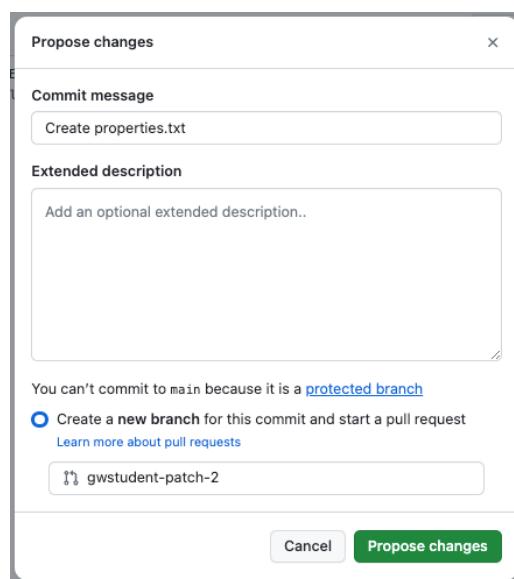
AWS_ACCESS_KEY_ID="AKIAZBQE445JKPTEAHQD"
AWS_SECRET_ACCESS_KEY="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"

sec-demo / properties / properties.txt in **main**

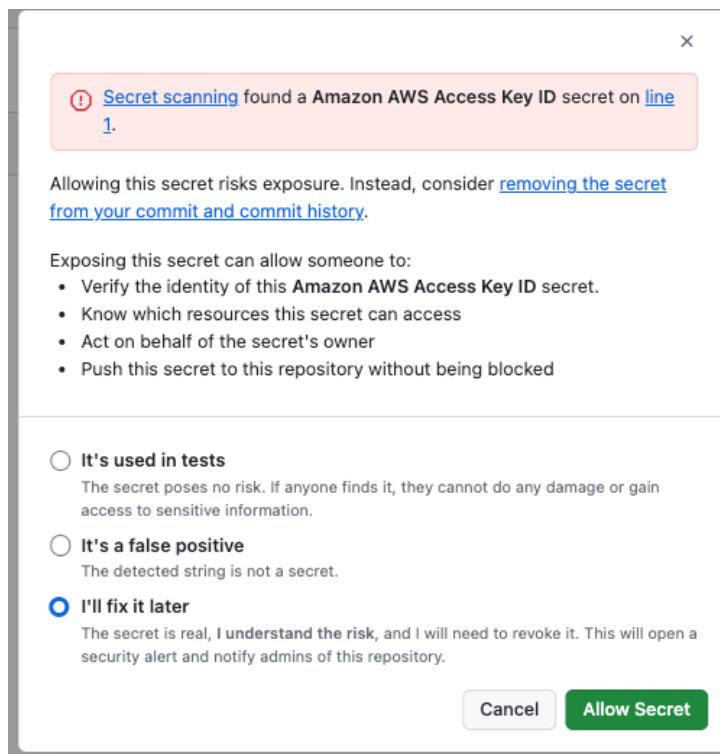
Commit changes...

```
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.

Code Pull requests Actions Projects Wiki Security Insights Settings

✓ Secret allowed. You can now commit these changes.

sec-demo / properties / properties.txt in main

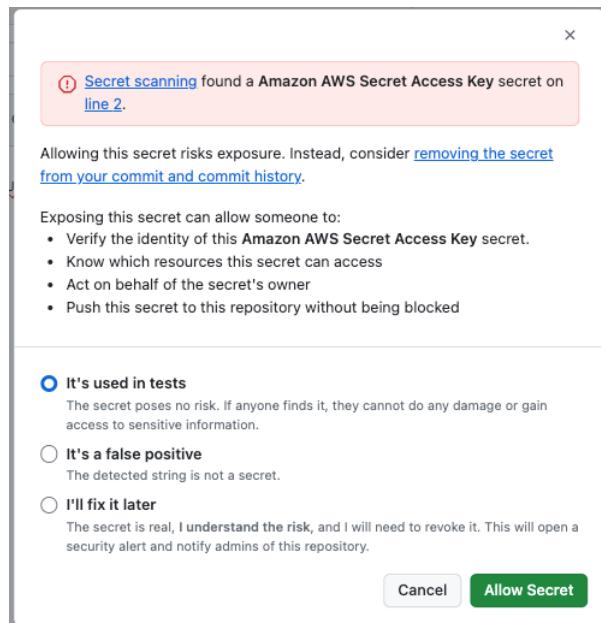
Cancel changes Commit changes...

Edit Preview Code 55% faster with GitHub Copilot

Spaces 2 No wrap

```
1 AWS_ACCESS_KEY_ID="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.



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, 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

Code Pull requests Actions Projects Wiki Security 1 Insights Settings

Overview 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

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 Secret scanning alerts interface. On the left sidebar, under the 'Secret scanning' section, there is a single alert. In the main area, a modal dialog is open titled 'Close as'. It contains a search bar with 'is:open' and a checkbox labeled '1 selected'. Below this, a list of close reasons is shown, with 'Revoked' selected. A note below it says 'This secret has been revoked'. Other options include 'Used in tests' (note: 'This secret is not in production code'), 'False positive' (note: 'This alert is not valid'), and 'Won't fix' (note: 'This alert is not relevant'). There is also a 'Comment' input field with 'Add a comment' placeholder and a 'Close alert' button at the bottom right.

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 Secret scanning alerts interface again. The sidebar shows '2 Closed' alerts. The main area displays two closed alerts: 'Amazon AWS Secret Access Key' (closed as used in tests) and 'Amazon AWS Access Key ID' (closed as revoked). A red circle highlights the '2 Closed' link in the sidebar.

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 next to the rule count. The entire **Dependabot rules** 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
- golang.org/x/net** 0.0.0-20220127200216-cd36cc0744dd
Detected automatically on Jan 10, 2024 (Go modules) · go.mod
- github.com/mattn/go-sqlite3** 1.14.8
Detected automatically on Jan 10, 2024 (Go modules) · go.mod

 Each entry has a red-outlined badge indicating severity: "1 critical" for the first, "4 high" for the second, and "0 medium" for the third. The entire dependency list area is highlighted with a red circle.

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 notifications). The main area is titled "Security overview". It shows four 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 shows Dependabot has 7 alerts. The main area lists 7 open vulnerabilities, each with a checkbox, a shield icon, and a severity level (Critical, High, Moderate, or Low). The first alert is "Authorization Bypass Through User-Controlled Key in go-restful" (Critical).

Severity	Vulnerability Description
Critical	Authorization Bypass Through User-Controlled Key in go-restful
High	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
Moderate	HTTP/2 Stream Cancellation Attack
Moderate	Improper rendering of text nodes in golang.org/x/net/html

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

Dependabot alerts / #1

Authorization Bypass Through User-Controlled Key in go-restful #1

Open Opened 4 minutes ago on [github.com/emicklei/go-restful \(Go\) · go.mod](#)

Upgrade [github.com/emicklei/go-restful](#) to fix 1 Dependabot alert in [go.mod](#)

Upgrade [github.com/emicklei/go-restful](#) to version 2.16.0 or later. For example:

```
require github.com/emicklei/go-restful v2.16.0
```

Create Dependabot security update

Severity	Critical	9.1 / 10
CVSS base metrics		
Attack vector	Network	
Attack complexity	Low	
Privileges required	None	
User interaction	None	
Scope	Unchanged	
Confidentiality	High	
Integrity	High	
Availability	None	

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

Package Affected versions Patched version

github.com/emicklei/go-restful (Go)	< 2.16.0	2.16.0 Patch
---	----------	------------------------------

Authorization Bypass Through User-Controlled Key in GitHub repository emicklei/go-restful prior to v3.8.0.

- 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.

Dependabot alerts / #1

Authorization Bypass Through User-Controlled Key in go-restful #1

Open Opened 10 minutes ago on [github.com/emicklei/go-restful \(Go\) · go.mod](#)

Bump [github.com/emicklei/go-restful](#) from 2.9.5+incompatible to 2.16.0+incompatible

Merging this pull request would fix 1 Dependabot alert on [github.com/emicklei/go-restful](#) in [go.mod](#).

Review security update

- 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 for a repository. The title of the pull request is "Bump github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible #2". A green button labeled "Open" is visible. Below the title, a message from "dependabot" says it wants to merge 1 commit from "dependabot/go_modules/github.com/emicklei/go-restful-2.16.0+incompatible". A note below 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 has a diff showing the addition of a dependency on "github.com/mattn/go-sqlite3 v1.14.8". The go.sum file also shows a dependency update for "github.com/mattn/go-sqlite3". The bottom right corner of the interface shows a status bar with "+3 -9" and a progress bar.

- 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 it has been merged. The title is now "Bump github.com/emicklei/go-restful from 2.9.5+incompatible to 2.16.0+incompatible #2". The "Merge pull request" button is now greyed out. A green bar at the top indicates that the pull request resolved a Dependabot alert. The "Conversation" tab is active, showing a message from "dependabot (bot)" adding the "dependencies" label 4 minutes ago. The "Checks" section shows "All checks have passed" and "This branch has no conflicts with the base branch". The bottom right corner shows a green bar with the text "This pull request resolved a Dependabot alert on [github.com/emicklei/go-restful](#)".

This screenshot is identical to the one above, showing the merged state of the pull request and the resolution of the Dependabot alert. The "Conversation" tab is active, and the green bar at the top indicates the alert was resolved.

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 repository code view for the `.github/workflows/create-issue.yml` file. 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. In the main area, there is a 'Create Issue' card for 'create-issue.yml'. It displays '0 workflow runs' and a note: 'This workflow has a workflow_dispatch event trigger.' Below this is a large icon of a circular pipeline. To the right, a modal window is open with the following fields:

- Use workflow from:** Branch: main
- Issue title ***: This is a title
- Issue body ***: This is the body text.

A green 'Run workflow' button is at the bottom of the modal.

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 tab. At the top, there are filters: 'Filters' set to 'is:issue is:open', 'Labels' (10), 'Milestones' (0), and a 'New issue' button. Below this, it shows '6 Open' issues. One specific issue is highlighted:

#10 opened 27 minutes ago by github-actions bot

This issue has the title 'This is a title' and the body '#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 message at the top states "Workflow run was successfully requested." 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 columns for Event, Status, Branch, and Actor. A note says "This workflow has a workflow_dispatch event trigger." Below this, a row for "Create Issue" is shown with a green checkmark icon. A modal window is open, prompting the user to "Use workflow from Branch: main". The "Issue title *" field contains "This is a title". The "Issue body *" field contains "`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.

This screenshot shows the same GitHub Actions interface after a workflow run. The "Actions" section now shows "2 workflow runs". The first run, "Create Issue #13: Manually run by gwstudent" (Branch: main), was run "1 minute ago" and took "12s". The second run is partially visible below it. The first run's row is circled in red.

Then click on the job name in the next screen.

This screenshot shows the detailed view for the "create_issue" job of the first workflow run. The top navigation bar is identical to the previous screens. The left sidebar shows "Summary", "Jobs" (with "create_issue" selected), "Run details", "Usage", and "Workflow file". The main area displays the "create-issue.yml" configuration, which triggers on "workflow_dispatch". The "create_issue" job is highlighted with a red circle. Its status is "Success" and it completed "12s" ago. A "Re-run all jobs" button is at the top right of this card.

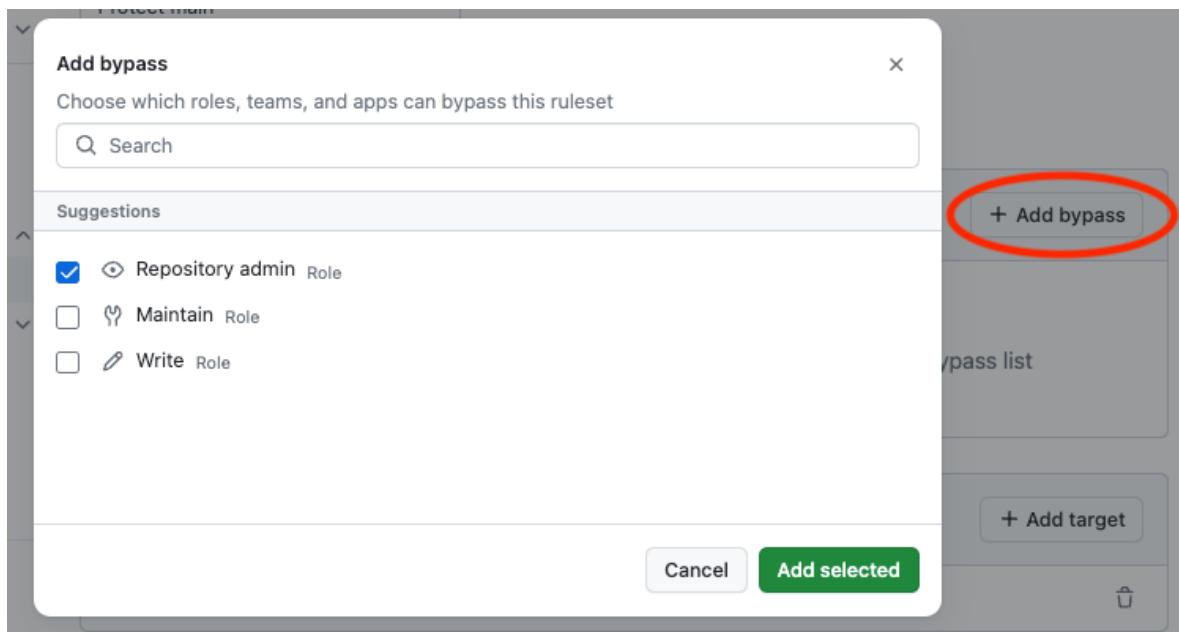
Then click to expand the *echo inputs* step in the log. Notice that the text we put in as the body of the issue was actually executed and run as a command.

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", "create_issue" (which is selected and highlighted in blue), "Run details", "Usage", and "Workflow file". The main area displays the logs for the "create_issue" job. The logs show the "Set up job" step followed by the "echo inputs" step. The output of the "echo inputs" step is expanded, showing two lines of text: "Run echo This is a title `ls -la`" and "This is a title total 8 drwxr-xr-x 2 runner docker 4096 Jan 11 03:34 .". A red oval highlights this expanded section of the log output.

- 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" (which is selected and highlighted in blue). On the right, under the "Rulesets" heading, there's a list titled "Protect main" which contains "3 rules · targeting 1 branch". Below this, there's a button labeled "+ Add bypass".

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 interface with the "Code" tab selected. On the left, the "Files" sidebar shows a tree structure with "main" and ".github/workflows" expanded, and "create-issue.yml" selected. The main area displays the contents of "create-issue.yml":

```
name: Create Issue
```

A red box highlights the "Edit this file" button in the top right corner 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"
```

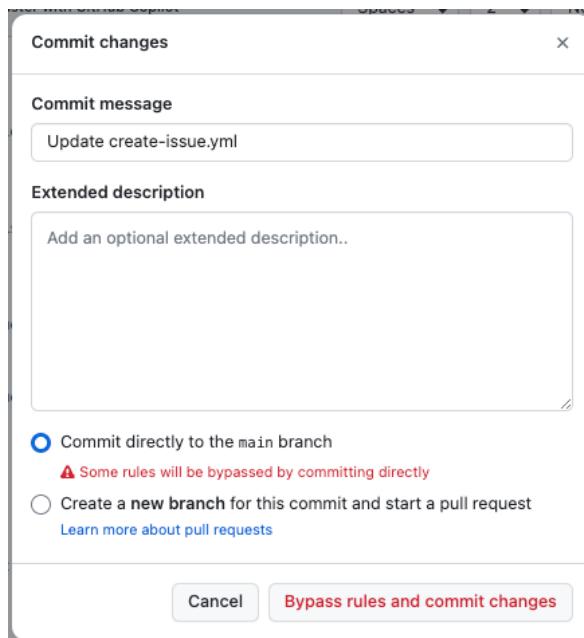
Be sure that it lines up as shown in the next screenshot.

```

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 second run, 'Create Issue #2', has a status of 'In Progress'. A modal window is open over the second run, titled 'Run workflow'. Inside the modal, there are fields for 'Issue title' (set to 'This is a title') and 'Issue body' (set to 'ls -la'). A green 'Run workflow' button is at the bottom of the modal. The GitHub UI header includes 'Code', 'Issues 2', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', 'Settings', and a search bar for 'Filter workflow runs'.

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

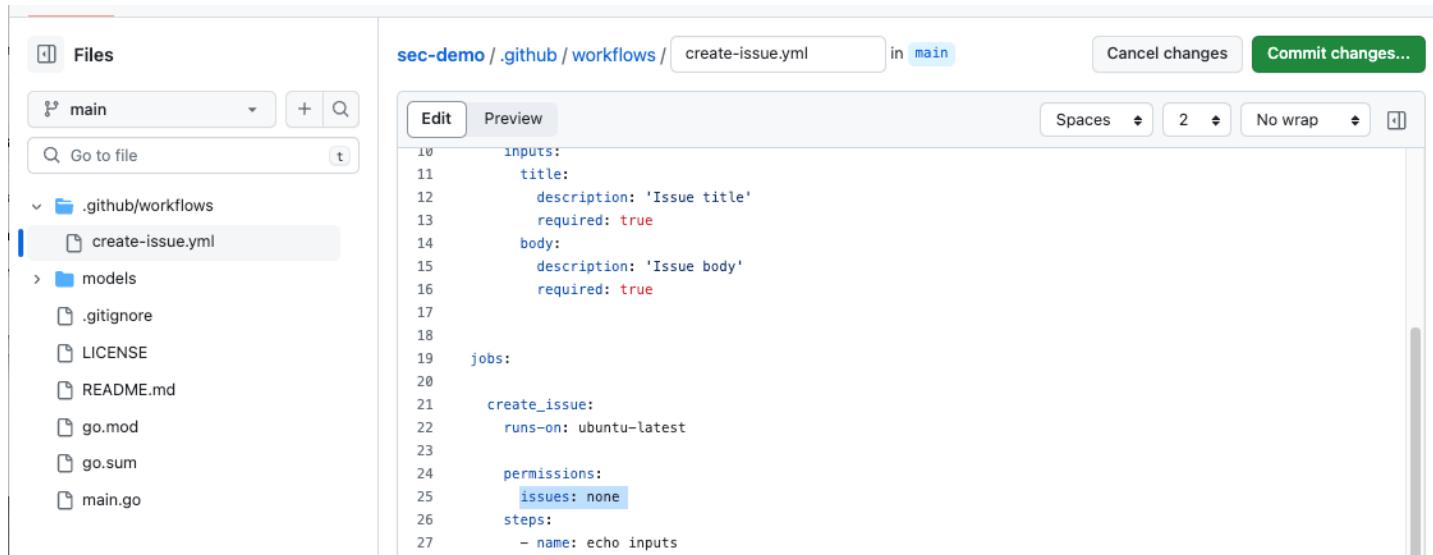
The screenshot shows the GitHub Actions log for 'Create Issue #3'. The log details the execution of the 'create_issue' job. It shows the job succeeded in 1 second. The log output shows the command 'ls -la' was run but did not execute, only being echoed. The GitHub UI header includes 'Code', 'Issues 3', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', 'Settings', and a 'Re-run all jobs' button.

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.



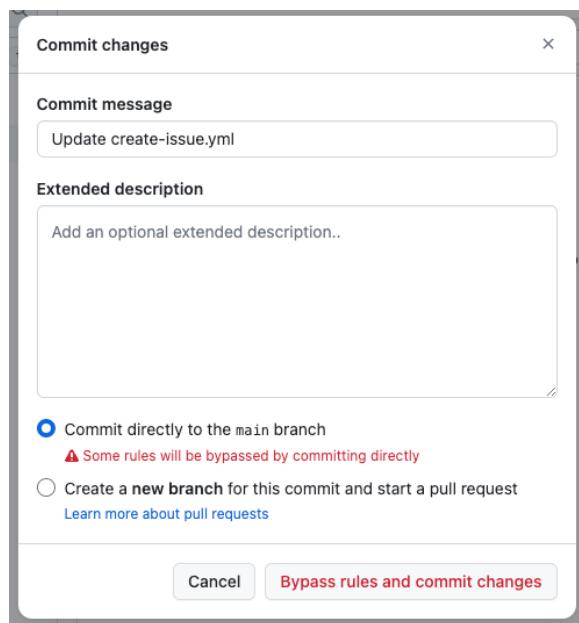
```

Files          sec-demo / .github / workflows / create-issue.yml in main
Cancel changes Commit changes...
main
+ | Go to file t
.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. The workflow has four successful runs listed under '6 workflow runs'. Each run is triggered by a 'workflow_dispatch' event and was manually run by 'gwstudent'. The most recent run was completed 1 hour ago. On the right side, there is a form to run the workflow again, with fields for 'Issue title' and 'Issue body' filled with placeholder text. A 'Run workflow' button is at the bottom of the form.

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 'Create Issue #19'. The job failed now in 1s. The log output shows the execution of a curl command to create an issue using the REST API. Step 18, which contains the error message 'curl: (22) The requested URL returned error: 403', is highlighted in red. Other steps like 'Set up job' and 'echo inputs' are also visible in the log.

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. Below the token name, it says "Expires on Mon, Mar 11 2024". To the right, there are buttons for "Generate new token", "Delete", and "Last used within the last week".

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

The screenshot shows the details of the "Security Fundamentals" token. The left sidebar is identical to the previous screenshot. The main area shows the token name, description ("Token for security fundamentals workshop"), creation date ("Created today"), expiration date ("Expires on Mon, Mar 11 2024"), and an "Edit" button. Below this, there's a section for "Access on gwstudent" with its own "Edit" button, which is circled in red. There's also a "Regenerate token" button. At the bottom, there's 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 "Read our [permissions documentation](#) for information about specific permissions". Below this is a "Repository permissions" section with a note that "2 Selected" and "Repository permissions permit access to repositories and related resources". To the right of this section is a red-circled arrow icon pointing to the right, which typically indicates a collapse or expand action.

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

Actions secrets and variables

Access Collaborators Moderation options

Code and automation Branches Tags Rules Actions Webhooks Environments Codespaces Pages

Secrets Variables

Environment secrets

This repository has no environment secrets. Manage environment secrets

Repository secrets

This repository has no secrets. **New repository secret** 3

Secrets and variables 2

Code security and analysis Deploy keys 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 main area is titled "Actions secrets / New secret". It has two input fields: "Name *" (set to "ISSUE_CREATE") and "Secret *" (containing a long GitHub 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}}' \
```

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 (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.

The screenshot shows the GitHub Actions interface. On the left sidebar, under the 'Actions' heading, there are several items: 'New workflow', 'All workflows', 'CodeQL', 'Create Issue' (which is highlighted with a blue bar), 'Management', 'Caches', and 'Runners'. In the main area, the title 'Create Issue' is displayed above the file 'create-issue.yml'. A search bar at the top right says 'Filter workflow runs'. Below the title, it shows '8 workflow runs'. There is a dropdown menu for 'Event', 'Status', 'Branch', and 'Actor'. A message states 'This workflow has a workflow_dispatch event trigger.' At the bottom, there is a card for the latest run: 'Create Issue #20: Manually run by gwstudent' (status: 'main', run time: 'now' 11s ago).

END OF LAB

That's all - THANKS