# Here is a summary of the module on code scanning:

The module focuses on code scanning and its implementation to ensure code security. The key points include:

* Introduction to code scanning: Code scanning is a feature offered by GitHub Advanced Security that helps identify security vulnerabilities and errors in code.
* Enabling code scanning: Steps are provided to enable code scanning in a repository, allowing alerts to be generated for problematic code.
* Enabling code scanning with third-party analysis: Guidelines are given for incorporating third-party analysis tools into the code scanning process.
* Implementing CodeQL analysis: CodeQL, a powerful code analysis engine, can be integrated into GitHub Actions workflows or third-party continuous integration (CI) tools.
* Configuring code scanning: Details are provided on configuring code scanning by specifying triggering events that initiate code scans.
* Frequency of code scanning workflows: A comparison is made between scheduled code scanning workflows and those triggered by events, helping determine the optimal frequency for scanning.

# Here is a summary of the section on code scanning with CodeQL:

* Code scanning with CodeQL: Code scanning utilizes the CodeQL analysis engine to scan code for security vulnerabilities and coding errors. It is available for all public repositories and private repositories owned by organizations with GitHub Advanced Security enabled.
* Code scanning alerts: When potential vulnerabilities or errors are found, GitHub displays alerts in the repository's Security tab. After fixing the code triggering the alert, GitHub automatically closes the alert.
* Purpose of code scanning: Code scanning helps identify and prioritize existing code issues and prevents the introduction of new problems. Scans can be scheduled or triggered by specific events such as a push to the repository.
* Using CodeQL for code scanning: CodeQL treats code as data, enabling more accurate identification of vulnerabilities. It generates a CodeQL database representing the codebase and runs queries on it to identify problems. CodeQL supports various languages, including C/C++, C#, Go, Java, JavaScript/TypeScript, Python, and Ruby.
* Adding the CodeQL workflow: To set up code scanning with CodeQL, navigate to the repository's Security tab, click "Set up code scanning," and select the CodeQL analysis workflow. Customizations can be made to the workflow, but the default configuration generally suffices.
* Committing the workflow: Choose whether to commit directly to the default branch or create a new branch and start a pull request. Once committed, code scanning will begin analyzing the code each time changes are pushed to the default or protected branches, or when pull requests are raised against the default branch.
* Billing for Actions: Running code scanning with GitHub Actions may affect monthly billing minutes. Additional usage beyond the included limits may incur charges. The specific billing details depend on the type of account and product being used.

Prerequisites for setting up code scanning include having write permissions to the repository, a GitHub enterprise account with GitHub Advanced Security license, and basic familiarity with GitHub administrative settings and GitHub Actions. The module provides guidance on configuring code scanning to enhance code security effectively.

# Enable Code Scanning with Third-Party Tools

* Code scanning alerts generated outside of GitHub can be uploaded to display in the repository.
* Static Analysis Results Interchange Format (SARIF) files are used to upload code scanning alerts from third-party tools.
* SARIF files must be in version 2.1.0 format.
* Code scanning alerts can be uploaded using the code scanning API, CodeQL CLI, or GitHub Actions.
* The code scanning API allows retrieval and update of code scanning alerts and supports SARIF files in application/sarif+json format.
* The CodeQL CLI generates a CodeQL database for code analysis and can upload results in SARIF format to GitHub.
* GitHub Actions workflows can be used to upload SARIF files to a repository.
* The "upload-sarif" action from the github/codeql-action repository is used in the workflow to configure the upload.
* The main input parameter in the workflow is "sarif-file" which specifies the file or directory of SARIF files to be uploaded.
* The upload-sarif action can be triggered by push events or scheduled events.
* SARIF files uploaded through GitHub Actions are processed, and alerts are added to the repository.
* GitHub uses fingerprints in SARIF files to prevent duplicate alerts.
* SARIF file uploads have a maximum limit of 5000 results per upload and a maximum file size of 10 MB for gzip-compressed files.
* SARIF files generated outside the repository can be uploaded using a workflow that runs when commits are pushed.
* The upload-sarif action can be added as a step in a continuous integration (CI) workflow to upload SARIF files generated during CI testing.

Examples:

1. Upload SARIF files generated outside your repository:
   * Create a workflow that runs when commits are pushed or on a schedule.
   * Use the "upload-sarif" action to upload the SARIF file.
   * Specify the path to the SARIF file relative to the repository root.
2. Upload SARIF files generated as part of a CI workflow:
   * Add the "upload-sarif" action as a step in the CI workflow.
   * Generate the SARIF file during the CI workflow, e.g., using a static analysis tool.
   * Specify the path to the SARIF file relative to the repository root.

# Here's a summary on configuring code scanning:

* Code scanning configuration allows you to customize how GitHub scans your project for vulnerabilities and errors.
* To configure code scanning, you need to add a GitHub Actions workflow to your repository, typically named codeql-analysis.yml.
* You can edit the workflow file by opening the workflow editor, making the desired edits, and committing the changes.
* Adjusting the frequency of code scanning is a common configuration option. You can schedule scans or trigger them based on specific events such as pushes or pull requests.
* By default, the CodeQL analysis workflow scans code on push events to the default branch and protected branches, and on pull request events targeted against the default branch.
* Pull request scans can be more efficient by scanning the merge commit rather than the head commit.
* You can define the severities (e.g., Error, Critical, High) that cause a pull request check failure, indicating code quality issues that block merging.
* To avoid unnecessary scans on specific pull requests, you can use the on:pull\_request:paths-ignore or on:pull\_request:paths options in the workflow file.
* The scanning schedule can be adjusted by modifying the cron value in the workflow file, allowing you to specify when and how often the code is scanned.

Example workflow configuration:

yaml

on:

push:

branches: [main, protected]

pull\_request:

branches: [main]

schedule:

- cron: '20 14 \* \* 1'

This configuration scans code on every push to the default branch and protected branch, every pull request to the default branch, and schedules a weekly scan on the default branch every Monday at 14:20 UTC.

# What is SARIF?

SARIF stands for Static Analysis Results Interchange Format. It is an open standard developed by the OASIS (Organization for the Advancement of Structured Information Standards) consortium. SARIF defines a standardized output file format that allows static analysis tools to share and exchange their analysis results.

Static analysis tools are used in software development to analyze source code or compiled binaries and identify potential issues, bugs, or security vulnerabilities. SARIF provides a common format for these tools to report their findings, making it easier to consume and process the results in a consistent manner.

By using SARIF, developers can integrate multiple static analysis tools into their development workflows and have a unified view of the results. SARIF files can be ingested by various tools and platforms, allowing developers to analyze and track code quality, security vulnerabilities, and other issues across different environments.

SARIF supports a rich set of information, including details about the analysis tool, rules used, specific results, and related metadata. It allows for the inclusion of location information within the source code, making it easier to navigate to the problematic areas and understand the reported issues.

GitHub, a popular software development platform, has also adopted SARIF for its code scanning feature. It allows developers to upload SARIF files generated by their static analysis tools and view the results directly within their GitHub repositories.

Overall, SARIF improves interoperability and facilitates the exchange of static analysis results between different tools and platforms, enhancing the overall software development and code quality assurance process.

* SARIF (Static Analysis Results Interchange Format) is used to share results from static analysis tools.
* Code scanning on GitHub supports a subset of the SARIF 2.1.0 JSON schema.
* Uploading SARIF files allows displaying code scanning alerts in your GitHub repository.
* SARIF files should be in the SARIF 2.1.0 version and can be uploaded from third-party static code analysis engines.
* Multiple SARIF files can be uploaded for the same commit, and each file can have a specified category.
* Results from different runs are matched using fingerprints to prevent duplicate alerts.
* Consistent filepaths are necessary for proper fingerprint computation and to avoid duplicate alerts.
* Fingerprint data can be included in SARIF files, or GitHub attempts to populate it from source files.
* SARIF files contain rules and results, where rules define the analysis rules and results contain details of alerts.
* The root for source files can be specified to convert absolute URIs to relative URIs.
* SARIF files can be validated for compatibility using the Microsoft SARIF validator.
* There are limits on the size and number of entries for SARIF files in code scanning.
* Certain SARIF properties are supported by code scanning for optimal display of analysis results on GitHub.

Top of Form

Bottom of Form

# Summary

* Enable and configure code scanning for your repository.
* Code scanning can be done with the integrated GitHub CodeQL action or third-party tools.
* Configure code scanning to be scheduled or triggered by specific events.
* Code scanning saves time and helps ensure code quality by identifying errors and security vulnerabilities.
* Manual verification of the code base is time-consuming and prone to mistakes, making code scanning a valuable tool.
* Code scanning alerts you of any problems and provides a centralized location to review and address the identified issues.