snyk Cheat Sheet: 10 GitHub Security Best Practices

www.snyk.io

1. Sensitive Data

Never store credentials as code/config in GitHub.
Some good practices:

1. Block sensitive data being pushed to GitHub by git-secrets or its likes as a git pre-commit hook.
2. Break the build using the same tools.
3. Audit for slipped secrets with GitRob or truffleHog.
4. Use ENV variables for secrets in CI/CD and secret managers like Vault in production.

2. Removing Sensitive data

If sensitive data made to a repo after all:

1. Invalidate tokens and passwords.
2. Remove the info and clear the GitHub history (force push rewrite history).
3. Assess impact of leaked private info.

3. Tightly control access

Failures in security are often humans making bad decisions.
Mandate the following practices for your contributors:

1. Require Two-factor authentication for all your GitHub accounts.
2. Never let users share GitHub accounts/passwords.
3. Any laptops/devices with access to your source code must be properly secured.
4. Diligently revoke access from users who are no longer working with you.

Manage team access to data. Give contributors only access to what they need to do their work.

4. Add a SECURITY.md file

You should include a SECURITY.md file that highlights security related information for your project. This should contain:

Disclosure policy.
Define the procedure for what a reporter who finds a security issue needs to do in order to fully disclose the problem safely, including who to contact and how. Consider HackerOne's community edition or simply a 'security@' email.

Security Update policy.
Define how you intend to update users about new security vulnerabilities as they are found.

Security related configuration.
Settings users should consider that would impact the security posture of deploying this project, such as HTTPS, authorisation and many others.

Known security gaps & future enhancements.
Security improvements you haven't gotten to yet.
Inform users those security controls aren't in place, and perhaps suggest they contribute an implementation!

For some great reference examples of SECURITY.md files, look at Apache Storm and TensorFlow.

5. GitHub Apps

Remember these apps are written by third-party developers, not GitHub. Validate:

1. The application access rights.
2. The author/organisation credibility.
3. How good is the app's security posture - a breach of them gives attackers access to your code!

Monitor changes in (ii) and (iii) over time and consider using application access restrictions.

6. Add Security testing to PRs

Use GitHub hooks to check your PRs don't introduce new vulnerabilities

1. SonarCloud - code quality testing.
2. CodeClimate - automated code reviews.
3. Snyk - dependency vuln testing.

7. Use the right GitHub offering

If you don't want anybody to have access to your code (even GitHub), or if regulations require it, use GitHub Enterprise's on-prem offering.

8. Rotate SSH keys and Personal Access Tokens

GitHub access is typically done using SSH keys or personal user tokens (in lieu of a password, because you enabled 2FA!). But what happens if those tokens are stolen and you didn't know?

Be sure to refresh your keys and tokens periodically, mitigating any damage caused by keys that leaked out.

9. Create New Projects with Security in Mind

When creating any project, develop it like an open source project. Don't rely on security by obscurity. You will:

1. Write more defensively when you push code/data knowing anyone could see.
2. Find it easier and safer if you decide to open source the project.

10. Importing Projects

Before importing a project into a public GitHub repo, fully audit the history for sensitive data, and remove it before adding to GitHub.