A security champion

In the ever-evolving landscape of technology, where innovation and advancement are celebrated, there exists a silent sentinel that stands guard against the looming threats of cyber-attacks and data breaches. This protector is none other than secure coding practices – the unsung hero of software development.

Imagine a world where the code you write is impervious to the malicious intentions of hackers, where user data remains safe, and the integrity of systems remains unassailable. This is the promise of secure coding practices.

At its core, secure coding is not merely a technical obligation but a moral imperative. It embodies the commitment to safeguarding the trust bestowed upon us by users who rely on our software. It is a testament to our dedication to privacy, confidentiality, and the sanctity of information.

The importance of secure coding practices cannot be overstated. They serve as the first line of defense against a barrage of cyber threats lurking in the digital realm. From SQL injection and cross-site scripting to buffer overflows and zero-day exploits, the vulnerabilities inherent in poorly written code provide a gateway for malevolent entities to wreak havoc.

Moreover, the repercussions of insecure code extend far beyond mere inconvenience. They can have profound ramifications on businesses, ranging from financial losses and legal liabilities to irreparable damage to brand reputation. The fallout of a single security breach can be catastrophic, undermining years of hard work and eroding customer trust in an instant.

Yet, amidst the looming specter of cybercrime, there exists a beacon of hope – the conscientious developer armed with the knowledge and discipline of secure coding practices. By adhering to principles such as input validation, least privilege, and defense in depth, we fortify our code against the ever-present threat of exploitation.

But secure coding is not just about erecting walls; it is about fostering a culture of vigilance and responsibility within the development community. It is about instilling a mindset where security is not an afterthought but an integral part of the software development lifecycle.

As developers, we wield immense power – the power to shape the digital world and influence the lives of millions. With this power comes a solemn duty – the duty to wield it responsibly, with integrity and foresight. Secure coding practices are our compass, guiding us on the path of ethical stewardship in the digital age.

-- ChatGPT 3.5. May 4th, 2024.

Task

Phishing is considered the most prevalent cyber threat in the world, and it is estimated that up to 90 per cent of data breaches are linked to successful phishing attacks, making it a major source of stolen credentials and information.

In 2023. a notorious ‘phishing-as-a-service’ (PaaS) platform known as ‘16shop' has been shut down in a global investigation coordinated by Interpol. The platform sold hacking tools to compromise more than 70,000 users in 43 countries and the amount of loss is estimated to be 8 million USD.

It is estimated that 90% of successful cyber-attacks start with email phishing, which continues to be very lucrative for attackers.

According to a recent report by Cloudflare, attackers posed as more than 1,000 different organizations in their brand impersonation attempts. However, in the majority (51.7%) of incidents, they impersonated one of 20 of the largest global brands.

And that's exactly why we need your help.

Your task is to create a solution for tracking phishing events but not any kind of solution, our aim is to have as secure as possible solution which meets best secure coding practices (OWASP secure coding practices, OWASP Top 10, OWASP API Security, CWE Top 25)

Your solution should have:

- User registration process (name, surname, email address, password)
- User login with “forgot password” option
- Only registered users have access
- Phishing event database:
    - Users should have ability to add or edit events
    - Users should not have ability to delete events
    - Event consists of
        - Name
        - Date and time of creation
        - Affected brand
        - Description (up to 1500 characters)
        - Malicious URL used during phishing campaign
        - Malicious domain registration date
        - Malicious domain DNS records (A, NS, MX) – this should be separated entries
        - List of matching keywords (matching keywords can be for example brand name, product name,…)
        - Status (todo, in progress, done)
        - Analyst comments
            - Every comment should have timestamp
            - All comments should be displayed chronologically
            - Commend can be edited but not deleted
- List of all events with search option

- Search option:
    - User can search phishing database using keywords (name, date, affected brand, malicious domain name, keywords)
- API endpoint for searching:
    - Endpoint receives URL as an input data
    - Endpoint will check whether the received URL is already registered in the database of malicious URLs and whether there are other similar URLs in the database for the domain or brand in question.
    - As a result, the endpoint should return in JSON form whether the specified URL was found as well as a list of similar URLs for the specified domain or brand if present in database. The endpoint must be protected by one of the secure authentication methods of your choice.

Bonus task

Shift left security

We encourage you to use shift left security approach during development of your solution by using:

- Linters (e.g. SonarLint) as an IDE plugin or otherwise
- Static application security testing - SAST (e.g. Snyk, Semgrep, …) as an IDE plugin or otherwise
- Software Composition Analysis - SCA (e.g. Snyk, Semgrep, …) as an IDE plugin or otherwise
- Dynamic application security testing – DAST (e.g. OWASP ZAP, Burp Community edition,…)

Scoring

A maximum of 100 points can be achieved.

Distribution of points:

- **Main task: 40 points**
    - 10 points - Methodology:
        - version control usage (e.g. Git)
        - task management tools usage (e.g. Jira, GitHub Project, Trello,)
        - teamwork (e.g. good discussions, not one person did all commits)
    - 10 points – Automatization:
        - Utilization of openly accessible resources for domain registration date
        - Utilization of openly accessible resources for DNS records
    - 10 points – Search algorithm:
        - Accuracy of search results
        - Performance
        - Option to search by multiple parameters
    - 10 points - UI/UX:

        - Fluent and responsive UI
        - Accessibility features
        - Consistency
- **Secure coding: 40 points***
    - 15 points – Secure coding best practices:
        - compliance with security best practices (OWASP secure coding practices, OWASP Top 10, OWASP API Security, CWE Top 25)
    - 15 points - Presence of security vulnerabilities**
    - 10 points - Secrets management (e.g. DB credentials, passwords, API tokens,…)
- **Bonus task: 20 points**:
    - You should document how did you utilize shift left security approach during development of your solution (e.g. write a short report with screenshots of tools used during development). If the jury finds your solution for bonus task to be the best, you will be presenting it (up to 10 min.) for all participants during the final ceremony.

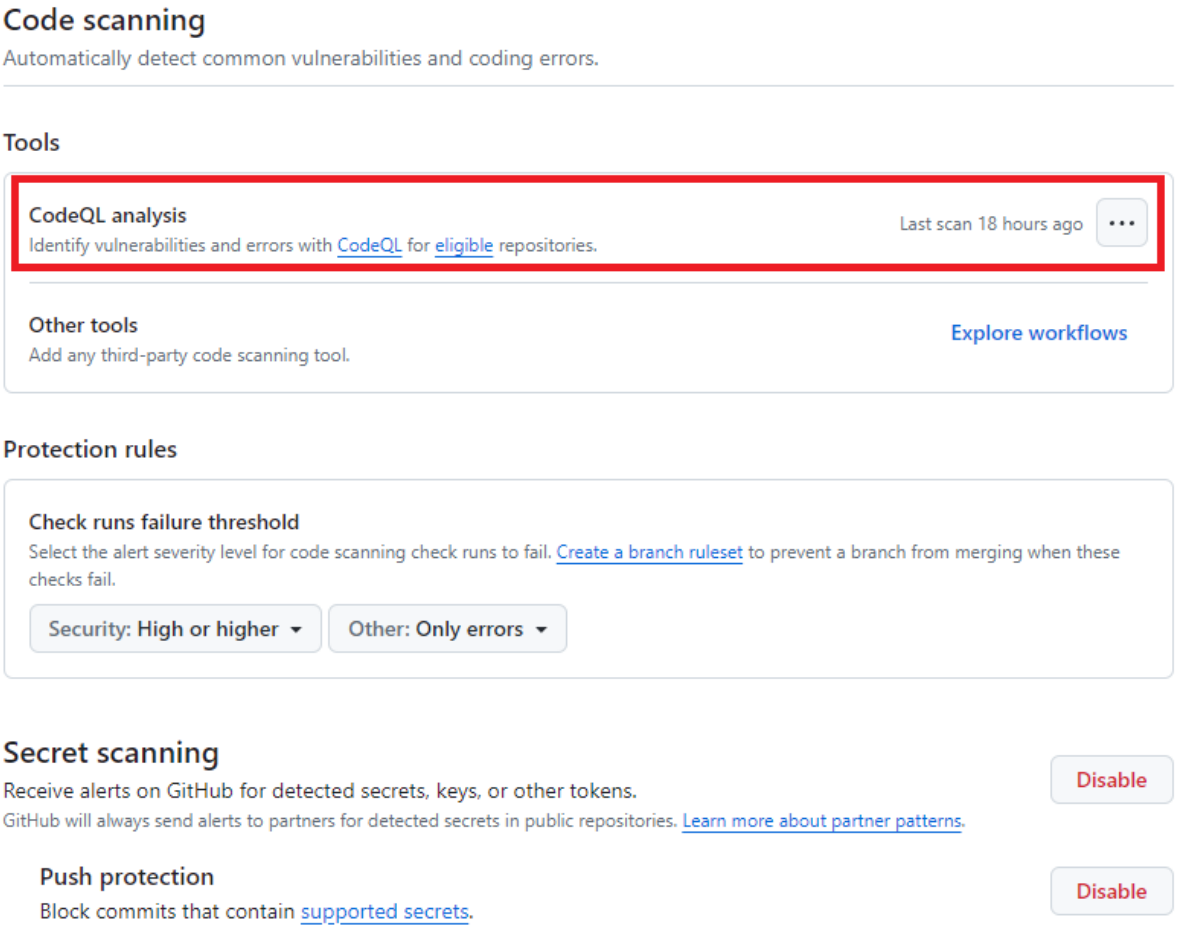
* In order to fairly evaluate all solutions, please upload your solution to your GitHub account and enable security tools (SAST, SCA and secrets detection) according to the “GitHub security tools” guide below. When choosing technology stack for your solution do mind supported languages and frameworks.

** It is important not to have any security vulnerabilities, especially not ones that are rated as critical or high. If you do happen to have some security vulnerabilities, we expect that you provide your analysis why they are not resolved and how you are planning to mitigate them.

GitHub security tools

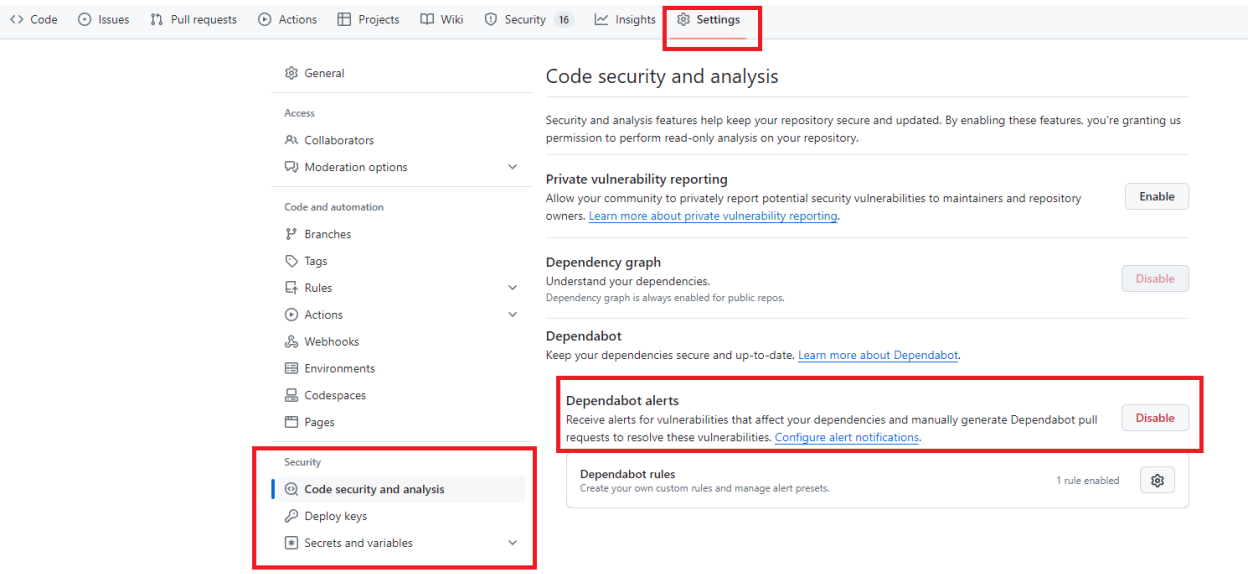
SAST

- On GitHub.com, navigate to the main page of the repository.
- Under your repository name, click **Settings**. If you cannot see the "**Settings**" tab, select the dropdown menu, then click **Settings**.
- In the "**Security**" section of the sidebar, click **Code security and analysis**.
- In the "**Code scanning**" section, select **Set up** , then click **Default**.



SCA

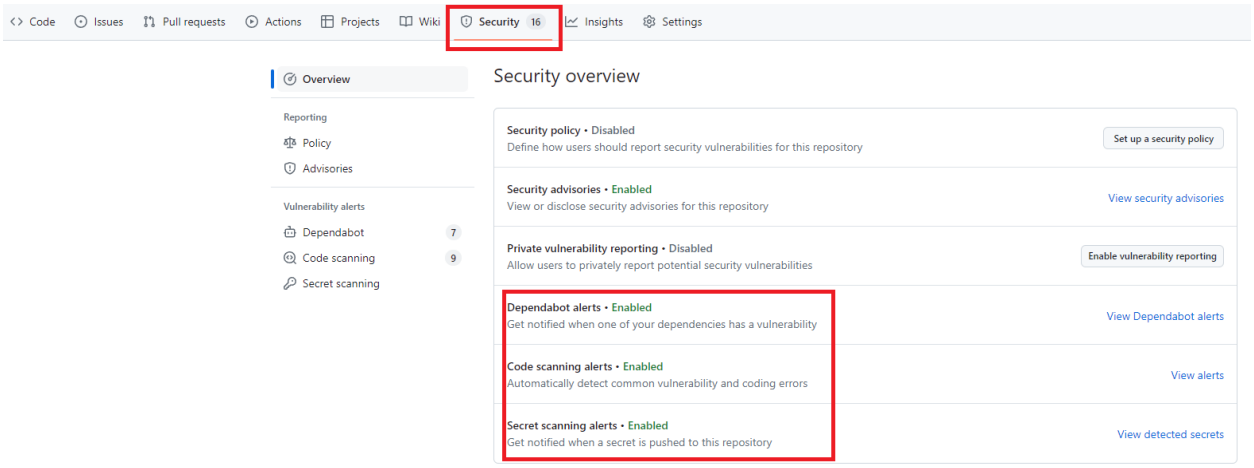
- On GitHub.com, navigate to the main page of the repository.
- Under your repository name, click **Settings**. If you cannot see the "**Settings**" tab, select the dropdown menu, then click **Settings**.
- In the "**Security**" section of the sidebar, click **Code security and analysis**.
- Under "Code security and analysis", to the right of Dependabot alerts, click Enable to enable alerts.



Secrets scanning

- On GitHub.com, navigate to the main page of the repository.
- Under your repository name, click **Settings**. If you cannot see the "**Settings**" tab, select the dropdown menu, then click **Settings**.
- In the "**Security**" section of the sidebar, click **Code security and analysis**.
- Scroll down to the bottom of the page, and click **Enable** for secret scanning. If you see a Disable button, it means that secret scanning is already enabled for the repository.

Check that everything is setup correctly:



How it will look like if you have any vulnerabilities detected: