ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 10, Issue 10, October 2023

A Visualized Insight into Dependency Risks in CI/CD Pipelines

[1] Shalparni P Y, [2] Dr Nalini M K, [3] Dr R Ashok Kumar, [4] Muralikrishna Nidugala

[1] CNE, BMS College of Engineering, Bengaluru, India.
[2] Department of ISE, BMS College of Engineering, Bengaluru, India
[3] Department of ISE, BMS College of Engineering, Bengaluru, India
[4] Master Technologist, Hewlett Packard Enterprise
Corresponding Author Email: [1] shalparnipy.scn21@bmsce.ac.in, [2] nalini.ise@bmsce.ac.in, [3] ashokkumar.ise@bmsce.ac.in,
[4] muralikrishna.nidugala@hpe.com

Abstract*— Continuous Integration/Continuous Deployment (CI/CD) pipelines have revolutionized software development, providing a streamlined approach to automate the build, testing, and deployment of applications. This abstract explores the integration of CI/CD pipelines with dependency management in the GitHub ecosystem. It examines the significance of this collaboration, the challenges faced, and presents best practices to optimize the development workflow. CI/CD pipelines integrated with dependency management in GitHub offer developers a powerful platform to manage project dependencies efficiently. The automation of dependency updates ensures that software projects stay up-to-date with the latest features and security patches, minimizing the risk of vulnerabilities caused by outdated libraries. By implementing best practices in dependency management. Utilizing package managers like npm, pip, or yarn helps manage dependencies effectively and simplifies the process of installing required packages. Employing version pinning and semantic versioning practices ensures a stable and predictable development environment. Moreover, integrating security tools like Dependabot within the CI/CD pipeline assists in automatically monitoring and updating dependencies, addressing vulnerabilities proactively. By utilizing GitHub's inherent functionalities, like security alerts and vulnerability assessments, valuable insights can be gained regarding potential risks within the project's dependency tree.

Keywords: CI/CD pipeline, Vulnerability, Dependency, CodeQL, SLSA, Dependabot, supply chain management, GitGuardian, Power Bi, Dependency Graph.

I. INTRODUCTION

The Continuous Integration/Continuous Deployment (CI/CD) pipeline has become a pivotal paradigm in contemporary software development, empowering teams to achieve swifter and more efficient application delivery. This paper provides an overview of CI/CD pipelines, their significance, and the challenges they pose, as well as highlighting best practices to optimize their implementation.

CI/CD pipelines automate the process of building, testing, and deploying software changes, allowing developers to integrate code frequently and reliably. By adopting this approach, manual intervention is substantially reduced, collaboration is streamlined, and the delivery of high-quality applications to end-users is ensured. By fostering a culture of continuous improvement, CI/CD pipelines enable software teams to innovate rapidly and respond promptly to market demands.

However, with the increasing adoption of CI/CD pipelines, new challenges have emerged. Security vulnerabilities, such as code injection, unauthorized access, and misconfigurations, have become potential threats to the integrity and availability of software. These vulnerabilities can lead to severe consequences, including data breaches and service disruptions. In response to these challenges, state-of-the-art security techniques have been incorporated into CI/CD pipelines. Tools like CodeQL, SLSA (Supply Chain Levels for Software Artifacts), and dependabot play a vital role in identifying and mitigating security risks, ensuring that the software remains resilient to attacks.

This paper also underscores essential guidelines for organizations to contemplate during the implementation of CI/CD pipelines. It stresses the significance of upholding a well-defined version control strategy, comprehensive testing at each pipeline stage, and the adoption of infrastructure as code to mitigate configuration errors. Furthermore, cultivating a DevSecOps culture that prioritizes security throughout the development lifecycle is vital in safeguarding the integrity of the CI/CD pipeline.

Supply chain management

A supply chain attack, also known as a value-chain or third-party attack, refers to the infiltration of a system through an external partner or provider that has access to the organization's systems and data. This infiltration has led to a significant transformation in the attack surface of enterprises in recent years, as more suppliers and service providers now handle sensitive data than ever before. Consequently, the risks associated with supply chain attacks have reached unprecedented levels due to the emergence of novel attack methods, heightened public awareness of these threats, and increased scrutiny from regulatory authorities.

The integration of CI/CD pipelines with dependency management in GitHub has significantly enhanced software development practices. By adopting best practices and

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 10, Issue 10, October 2023

incorporating automated security measures, developers can efficiently manage dependencies, reduce risks, and ensure the continuous delivery of high-quality, secure software. Embracing these advancements empowers teams to accelerate innovation, minimize vulnerabilities, and deliver reliable software products to end-users.

II. LITERATURE REVIEW

Vulnerabilities in Continuous, Delivery Pipelines A Case Study This paper team members that work with the CD pipeline in different roles do not have a strong security background but are aware of security attributes in general. projects were analyzed using the STRIDE threat analysis approach. Analysis of two industry CD pipelines focusing. Execution of a STRIDE threat analysis (confidentiality, integrity, availability) and mapping of the identified threats based on NIST and OWASP project. Manual vulnerability assessment based on the results of the STRIDE threat analysis [1]. Continuous Security Testing A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. In this paper, approach to integrate three automated dynamic testing techniques into CI/CD pipeline provide an empirical analysis of the introduced identify unique research challenges the DevSecOps communities. The three dynamic application security testing techniques we integrated into a CI/CD pipeline are: Web Application Security Scanning (WAST) using Zed Attack Proxy (ZAP)1, Security API Scanning (SAS) with JMeter 2 and Behaviour Driven Security Testing (BDST) using SeleniumBase automation framework [2] Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. This research aimed at systematically reviewing the state of the art of continuous practices to classify approaches,tools, identify challenges and practices [3]. Software Supply Chain And Devops Security Practices[4]. The paper’s objective is to produce practical and actionable guidelines that meaningfully integrate security practices into development methodologies. The project will also strive to demonstrate the use of current and emerging secure development frameworks, practices, and tools to address cybersecurity challenges.

III. METHODOLOGY

In GitHub, dependencies pertain to external libraries, frameworks, or packages essential for a project's proper functioning. These dependencies are typically integrated to introduce specific functionalities or features without the necessity of developing everything from scratch. They encompass diverse types of software components, including code libraries, plugins, or modules. Managing dependencies is crucial for software development because it allows developers to leverage existing solutions and focus on building the unique aspects of their projects. GitHub provides several ways to handle dependencies, and one of the most common methods is using package managers. Package managers are tools that automate the process of installing, updating, and removing dependencies. They keep track of the versions and dependencies required by a project and ensure that everything is consistent and compatible. Some popular package managers used in GitHub projects include:

- npm (Node Package Manager): Used for JavaScript projects, primarily in the Node.js ecosystem.
- pip: Used for Python projects.

1. Creation of CI/CD pipeline in repository

Creating a CI/CD pipeline for a repository on GitHub involves setting up an automated workflow that performs continuous integration (CI) and continuous delivery/deployment (CD) tasks. The CI/CD pipeline automates the build, test, and deployment processes, enabling faster and more reliable development cycles. Below are the general steps to create a CI/CD pipeline in a GitHub repository:

- Setup GitHub Repository: Create a new repository on GitHub or use an existing one to host your project.
- Version Control: Ensure that your project is well-structured and uses version control, preferably Git, to track changes effectively.
- Opt for a CI/CD Service: Decide on a CI/CD service that seamlessly integrates with GitHub.

Some popular options include:
GitHub Actions (integrated directly into GitHub)
Travis CI
CircleCI
Jenkins (self-hosted)

- Configuration File: In your repository, create a configuration file for the CI/CD pipeline. The file specifies the steps to be executed during the workflow.
- Continuous Integration (CI): The CI part of the pipeline ensures that code changes are continuously integrated into the main codebase. This process involves the following steps:

Setting up the build environment.
Installing dependencies.
Building the application.
Running unit tests and other relevant tests.

- Continuous Delivery/Deployment (CD): The CD component of the pipeline automates the deployment process following a successful CI. This process includes the following steps:

Deployment to a staging environment for additional testing
Deployment to production or any other designated target environment.

- Handling Secrets and Environment Variables: Ensure that any sensitive information, such as API keys or passwords, is properly handled using GitHub's secrets or an external secret management tool.
- Push Pipeline Configuration to Repository: Commit and push the configuration file for your CI/CD

pipeline to your GitHub repository. This triggers the CI/CD service to execute the defined workflow whenever changes are pushed to the repository.

- Monitor and Debug: Monitor the CI/CD pipeline's execution, and if any issues arise, use the logs and debugging tools provided by your CI/CD service to identify and fix the problems.
- Iterate and Improve: Continuously refine and improve your CI/CD pipeline based on feedback and changing requirements.

2. Handling Dependency in a repository

Dependencies are comprised of external libraries, frameworks, or packages that a software project necessitates to operate accurately. These dependencies are integral elements of a project and frequently serve to deliver distinct functionalities, minimize development workload, and ensure the reuse of code. Within GitHub repositories, configuration files are frequently present, detailing the dependencies of the project and the precise versions they require. These files make it easier for developers to set up the development environment and ensure consistency across different installations. Some common dependency management files in GitHub repositories include:

package.json: Used for JavaScript projects managed with npm (Node Package Manager). It lists the project's dependencies and their versions.

requirements.txt: Typically used for Python projects, it includes a list of required Python packages and their versions.

pom.xml: Used for Java projects managed with Apache Maven. The Project Object Model (POM) file lists the project's dependencies and other project-related information.

Gemfile: Used for Ruby projects, it contains a list of Ruby gems (dependencies) and their versions.

composer.json: Used for PHP projects managed with Composer. It specifies the project's PHP dependencies and their versions.

Handling dependencies in a GitHub repository involves properly managing and updating these files, so the project's dependencies are correctly installed and maintained. Developers can use package managers like npm, pip, Maven, or Composer to install and manage these dependencies automatically. It is important to keep dependencies up-to-date to ensure security, stability, and compatibility with other parts of the project. Additionally, GitHub provides various features and tools that help manage dependencies effectively, such as dependency graphs, which visualize a repository's dependencies and their relationships. Regularly reviewing and updating dependencies, along with performing security audits, are essential practices to maintain a healthy and secure codebase in a GitHub repository. This ensures that the project remains robust and can take advantage of the latest features and improvements offered by external libraries and frameworks.

Handling dependencies in a repository is crucial for ensuring that your project can be built, tested, and executed correctly on different systems. Managing dependencies involves specifying the required external libraries, frameworks, or packages that your project relies on. Here are some best practices for handling dependencies in a repository:

- Use a Package Manager:

Most programming languages have package managers that help manage dependencies automatically. Examples include npm (Node.js), pip (Python), Maven (Java), and Composer (PHP). Using a package manager simplifies the process of installing, updating, and removing dependencies.

- Dependency File:

Create a file (e.g., package.json, requirements.txt) in your repository to list all the dependencies and their versions. This file acts as a manifest of required packages, making it easy for other developers to set up the same development environment.

- Lock File:

Some package managers generate a lock file (e.g., package-lock.json, pip-lock.txt) that ensures consistent versions of dependencies across different installations. This file is useful for maintaining a stable build environment.

- Avoid Global Dependencies:

Whenever possible, avoid installing dependencies globally. Instead, keep dependencies local to your project, making it easier to manage versions and avoid conflicts with other projects.

- Versioning:

Be explicit about the versions of dependencies your project requires. Use semantic versioning (e.g., ^1.2.3, ~2.1.0) to specify acceptable version ranges. This helps prevent unexpected updates that could introduce breaking changes.

- Security Audits:

Regularly audit your project's dependencies for security vulnerabilities. Many package managers provide tools for scanning and flagging vulnerable dependencies. Keep your dependencies up to date to minimize security risks.

- Automated Testing:

Include automated tests in your CI/CD pipeline that ensure your project works with the specified dependency versions. Automated testing helps catch compatibility issues early.

- Documentation:

Include clear documentation in your repository's README or documentation files about how to set up the development environment and install the necessary dependencies.

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 10, Issue 10, October 2023

By following these best practices, manage dependencies effectively, leading to a more reliable and maintainable project. Remember to regularly review and update dependencies to benefit from the latest features, bug fixes, and security patches.

3. Using a dependency graph to comprehend vulnerabilities in the files.

The dependency graph in GitHub is a powerful tool that offers a graphical depiction of a repository's dependences. It aids developers in comprehending the connections between the repository's code and the external libraries, frameworks, and packages it relies on. Managing and maintaining projects with such dependencies becomes more accessible through this feature. The dependency graph is accessible for both public and private repositories employing package managers like npm, pip, Maven, Composer, and others. By automatically analyzing the repository's dependency files, such as package.json, requirements.txt, pom.xml, Gemfile, or composer.json, it creates a comprehensive graph based on the declared dependencies.

Key features of the dependency graph in GitHub include:

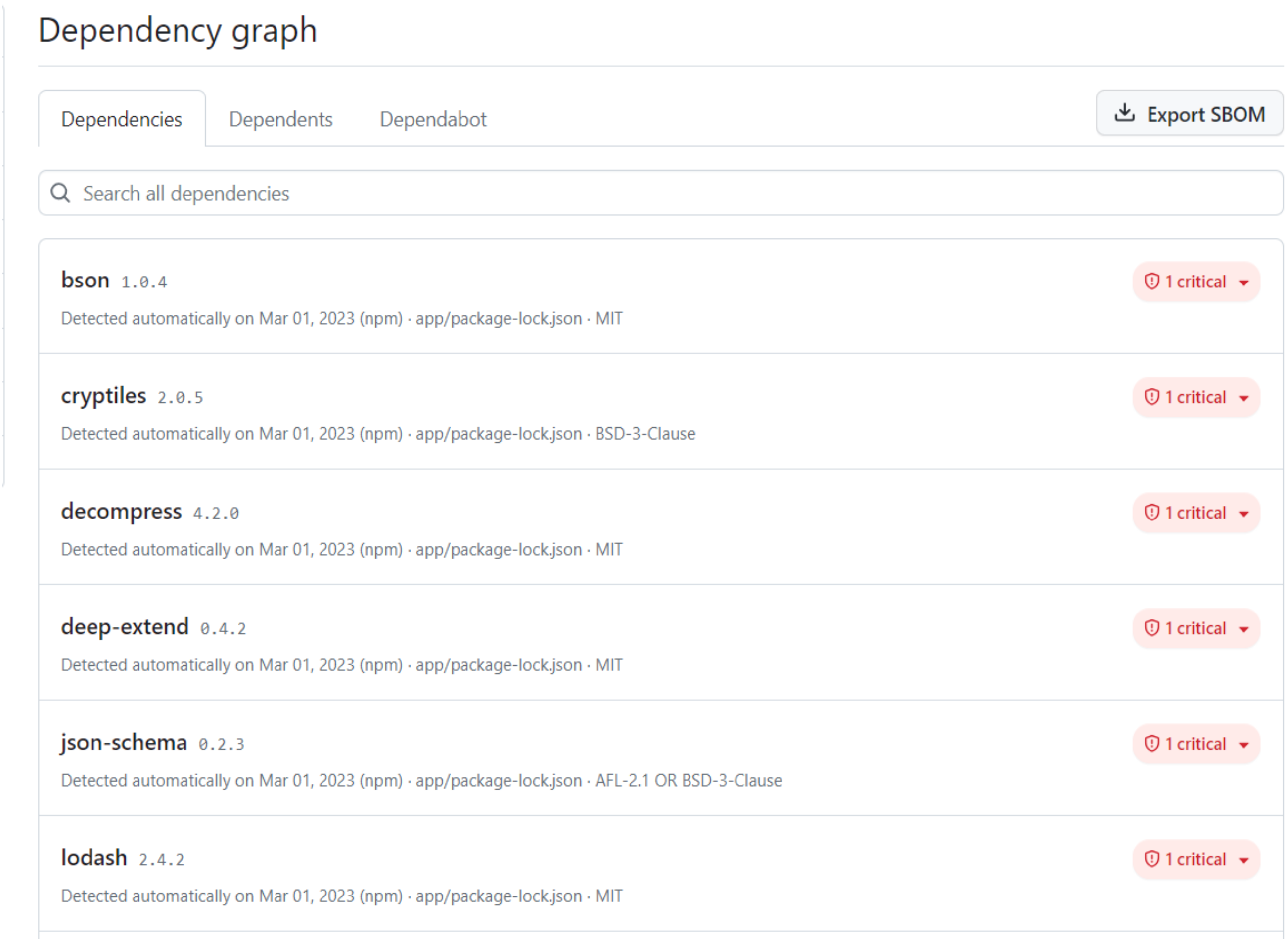
Dependency Visualization: The graph displays the dependencies of the repository's code and how they are interconnected. It provides a clear and easy-to-understand visual representation of the project's dependency hierarchy.

Security Alerts: The dependency graph can also show security alerts for known vulnerabilities in the project's dependencies. If any vulnerable dependencies are detected, GitHub will display alerts and recommend updating to a secure version.

Version Information: The graph typically includes version numbers for each dependency, allowing developers to see if the project is using the latest or outdated versions.

Filtering and Search: The dependency graph permits the filtering and searching of distinct dependencies or packages, simplifying the process of locating and concentrating on specific elements. Observations: It offers valuable observations regarding the project's well-being and framework, particularly in relation to external dependencies.

The dependency graph proves to be a valuable asset in upholding project well-being and security, aiding developers in comprehending and proficiently handling dependencies. It fosters enhanced collaboration, well-informed choices, and expedited problem-solving when addressing dependencies within a GitHub repository.



bson 1.0.4
Detected automatically on Mar 01, 2023 (npm) · app/package-lock.json · MIT
1 critical

cryptiles 2.0.5
Detected automatically on Mar 01, 2023 (npm) · app/package-lock.json · BSD-3-Clause
1 critical

decompress 4.2.0
Detected automatically on Mar 01, 2023 (npm) · app/package-lock.json · MIT
1 critical

deep-extend 0.4.2
Detected automatically on Mar 01, 2023 (npm) · app/package-lock.json · MIT
1 critical

json-schema 0.2.3
Detected automatically on Mar 01, 2023 (npm) · app/package-lock.json · AFL-2.1 OR BSD-3-Clause
1 critical

lodash 2.4.2
Detected automatically on Mar 01, 2023 (npm) · app/package-lock.json · MIT
1 critical

Figure 1: List of dependencies in a GitHub project

ISSN (Online) 2456 -1304

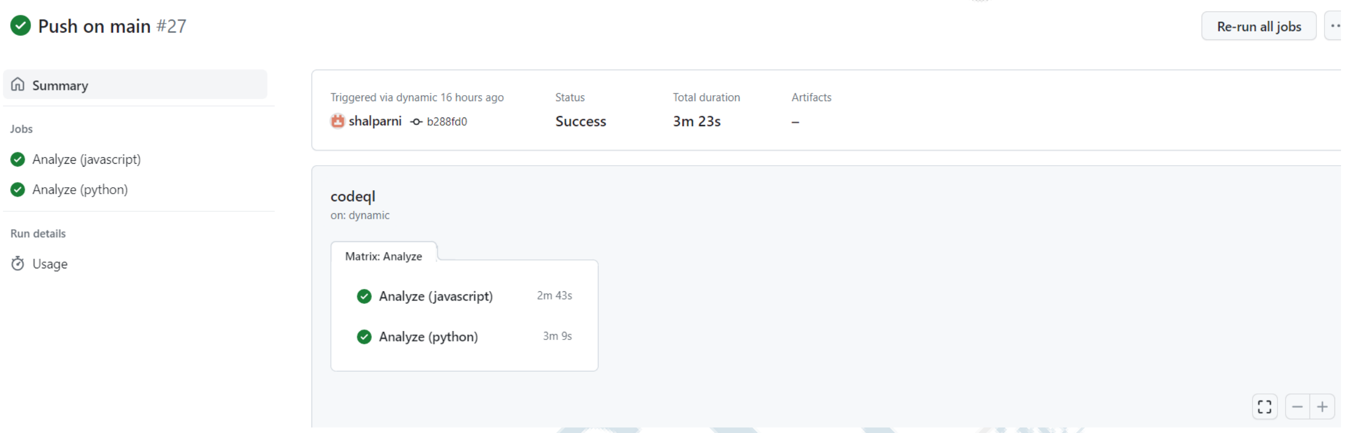
International Journal of Science, Engineering and Management (IJSEM)

Vol 10, Issue 10, October 2023

Dependency graph using Power Bi

Power BI, a robust business analytics service developed by Microsoft, empowers users to craft engaging and interactive reports and dashboards from diverse data sources. Through the utilization of Power BI, enterprises acquire valuable insights from their data, thereby enabling informed decisions based on data. Embraced across various sectors, Power BI functions as a versatile instrument for data analysis, business intelligence, and reporting. It accommodates both individuals seeking self-service analytics and organizations requiring robust data solutions to enhance their business processes. The project name, incidents and the number of dependencies are considered and by using Power Bi Dependency graph is generated.

IV. RESULTS



Push on main #27

Re-run all jobs

Summary

Jobs

Analyze (javascript)

Analyze (python)

Run details

Usage

Triggered via dynamic 16 hours ago: shalparni b288fd0

Status: Success

Total duration: 3m 23s

Artifacts: –

codeql
on: dynamic

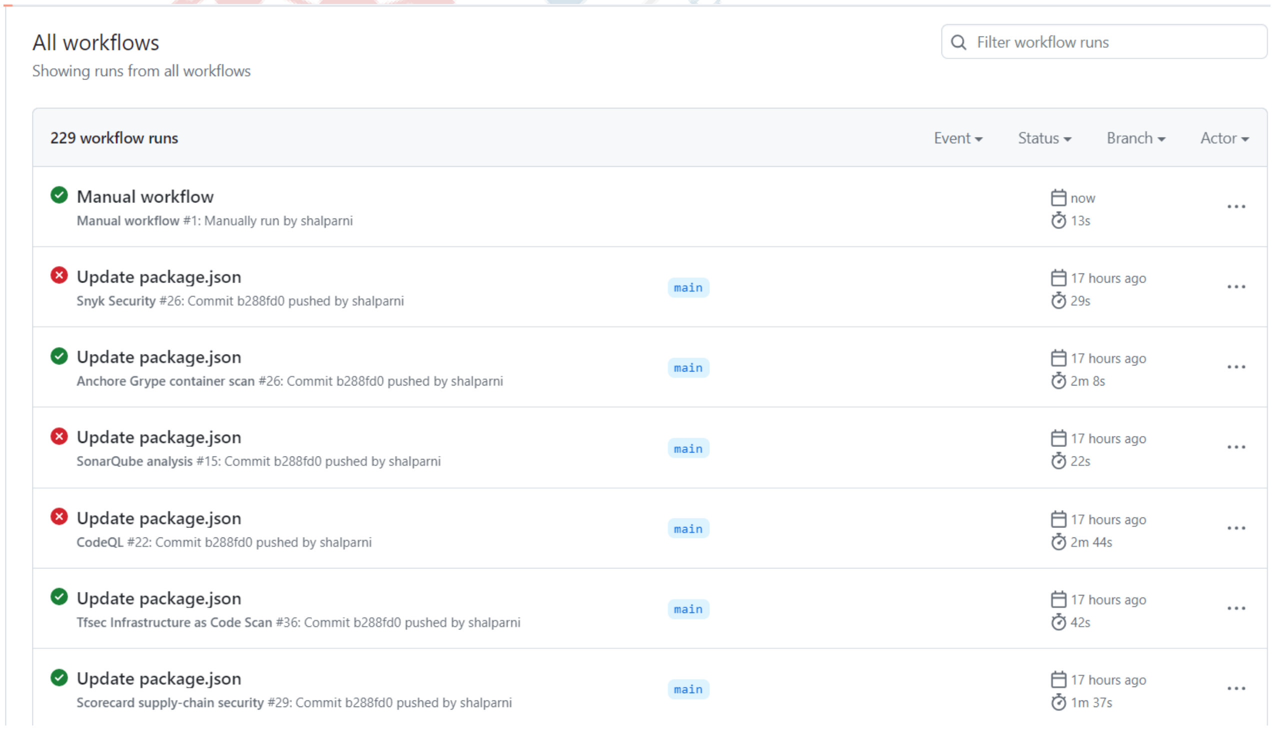
Matrix: Analyze

Analyze (javascript) 2m 43s

Analyze (python) 3m 9s

Figure 2: CI/CD Pipeline in Github

Dependency Graph in GitHub provides a visual representation of the dependencies present in a repository. However, the availability and exact results of the Dependency Graph may vary depending on the repository's configuration, package managers used, and the specific dependencies declared. Once the Dependency Graph is generated, a visual representation of project's dependencies and how they are connected. It will display a tree-like structure, indicating the hierarchy of dependencies, and may also show any security vulnerabilities detected in those dependencies. Remember that the results in the Dependency Graph are based on the package manager and dependencies declared in your repository's configuration files.



All workflows

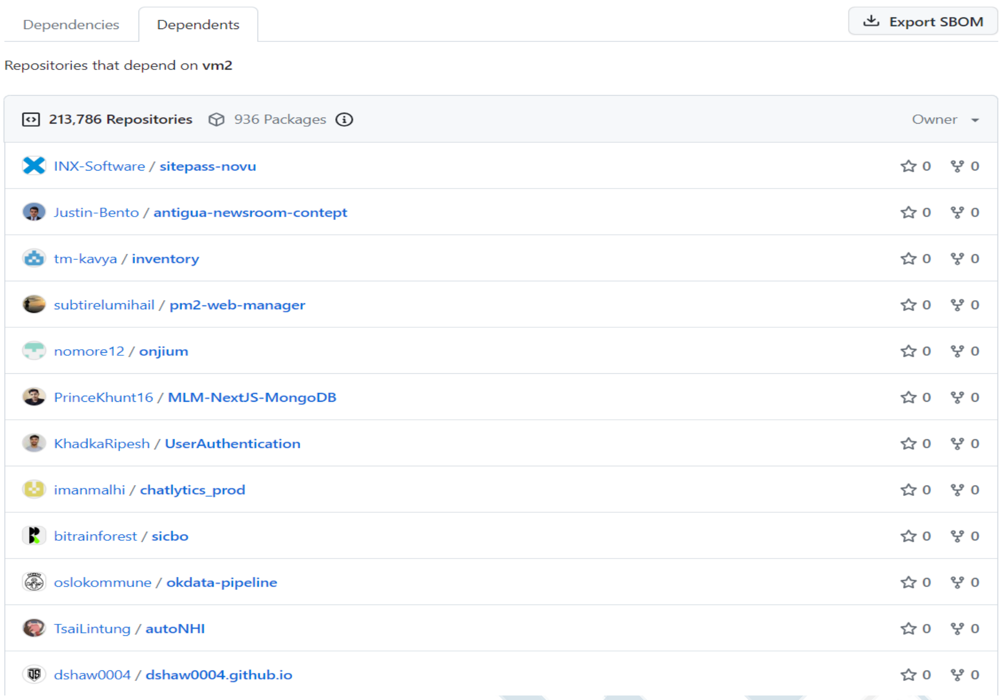
Showing runs from all workflows

Filter workflow runs

229 workflow runs — Event, Status, Branch, Actor

Workflow run	Branch	Time	Duration
✓ Manual workflow — Manual workflow #1: Manually run by shalparni		now	13s
✗ Update package.json — Snyk Security #26: Commit b288fd0 pushed by shalparni	main	17 hours ago	29s
✓ Update package.json — Anchore Grype container scan #26: Commit b288fd0 pushed by shalparni	main	17 hours ago	2m 8s
✗ Update package.json — SonarQube analysis #15: Commit b288fd0 pushed by shalparni	main	17 hours ago	22s
✗ Update package.json — CodeQL #22: Commit b288fd0 pushed by shalparni	main	17 hours ago	2m 44s
✓ Update package.json — Tfsec Infrastructure as Code Scan #36: Commit b288fd0 pushed by shalparni	main	17 hours ago	42s
✓ Update package.json — Scorecard supply-chain security #29: Commit b288fd0 pushed by shalparni	main	17 hours ago	1m 37s

Figure 3: Workflow of the project



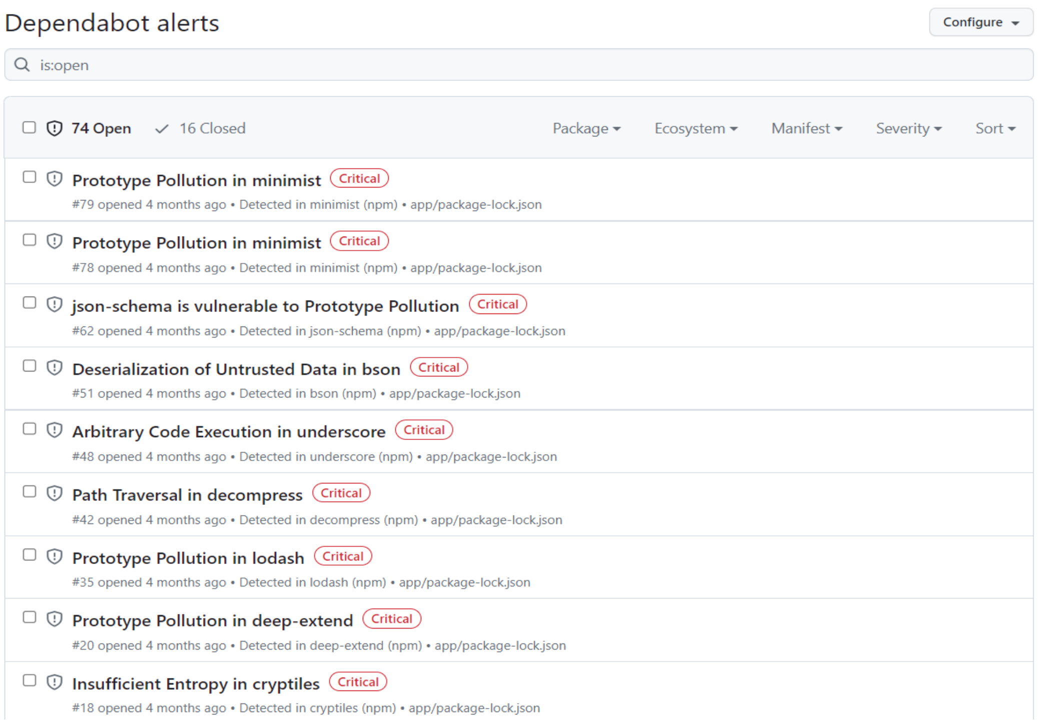
Dependencies | Dependents | Export SBOM

Repositories that depend on **vm2**

213,786 Repositories · 936 Packages · Owner

Repository	Stars	Forks
INX-Software / sitepass-novu	0	0
Justin-Bento / antigua-newsroom-contept	0	0
tm-kavya / inventory	0	0
subtirelumihail / pm2-web-manager	0	0
nomore12 / onjium	0	0
PrinceKhunt16 / MLM-NextJS-MongoDB	0	0
KhadkaRipesh / UserAuthentication	0	0
imanmalhi / chatlytics_prod	0	0
bitrainforest / sicbo	0	0
oslokommune / okdata-pipeline	0	0
TsaiLintung / autoNHI	0	0
dshaw0004 / dshaw0004.github.io	0	0

Figure 4: Dependency in a project and the dependents



Dependabot alerts

is:open

74 Open · 16 Closed · Package · Ecosystem · Manifest · Severity · Sort

- Prototype Pollution in minimist (Critical) — #79 opened 4 months ago • Detected in minimist (npm) • app/package-lock.json
- Prototype Pollution in minimist (Critical) — #78 opened 4 months ago • Detected in minimist (npm) • app/package-lock.json
- json-schema is vulnerable to Prototype Pollution (Critical) — #62 opened 4 months ago • Detected in json-schema (npm) • app/package-lock.json
- Deserialization of Untrusted Data in bson (Critical) — #51 opened 4 months ago • Detected in bson (npm) • app/package-lock.json
- Arbitrary Code Execution in underscore (Critical) — #48 opened 4 months ago • Detected in underscore (npm) • app/package-lock.json
- Path Traversal in decompress (Critical) — #42 opened 4 months ago • Detected in decompress (npm) • app/package-lock.json
- Prototype Pollution in lodash (Critical) — #35 opened 4 months ago • Detected in lodash (npm) • app/package-lock.json
- Prototype Pollution in deep-extend (Critical) — #20 opened 4 months ago • Detected in deep-extend (npm) • app/package-lock.json
- Insufficient Entropy in cryptiles (Critical) — #18 opened 4 months ago • Detected in cryptiles (npm) • app/package-lock.json

Figure 5: Dependabot alerts in GitHub

Git Guardian

GitGuardian's focus is to detect and safeguard against leaks of sensitive information, including API keys, credentials, passwords, and confidential data that might accidentally surface in a repository's source code or configuration files. These leaks can present substantial security risks, potentially resulting in unauthorized access, data breaches, and other cybersecurity incidents.

The key features and capabilities offered by GitGuardian encompass:

Sensitive Data Detection: GitGuardian diligently scans a repository's code and configuration files to pinpoint potential instances of sensitive information. It employs pattern matching and machine learning algorithms to identify known patterns of sensitive data, such as API keys, access tokens, and database passwords.

ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)

Vol 10, Issue 10, October 2023

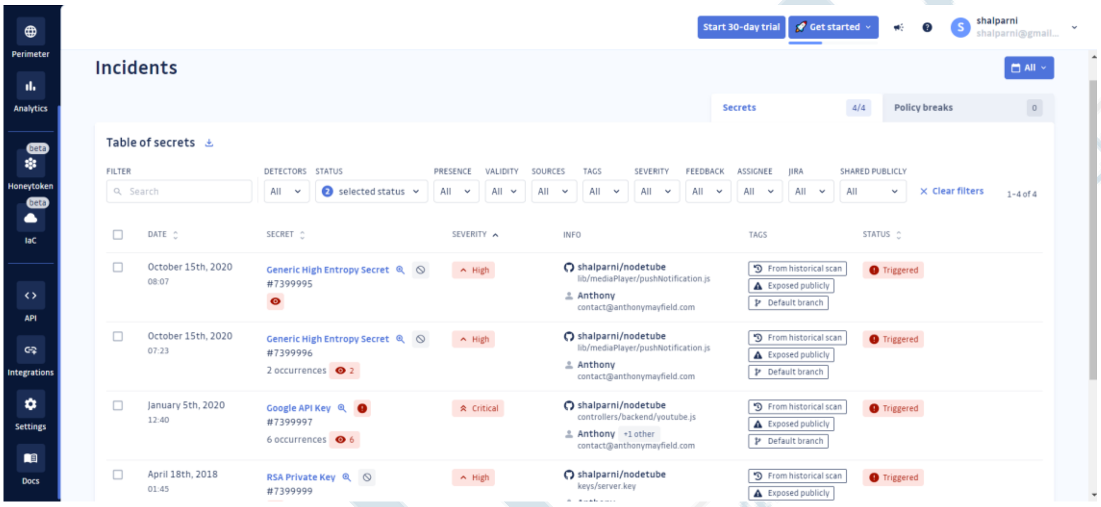
Automated Scanning: With continuous monitoring of repositories, GitGuardian automatically conducts real-time scans of newly pushed code, pull requests, and commits. This proactive approach aids in promptly detecting any sensitive data leaks, effectively preventing security breaches.

Alerts and Notifications: When GitGuardian identifies sensitive data in a repository, it generates alerts and notifications, promptly informing the repository owners or designated security personnel. This enables swift remediation of the exposed information.

Integration with CI/CD Pipelines: GitGuardian seamlessly integrates with continuous integration/continuous deployment (CI/CD) pipelines to ensure inadvertent leaks of sensitive data do not occur during development and deployment processes.

Compliance and Policy Enforcement: GitGuardian supports security policy enforcement, ensuring that sensitive data remains inaccessible within repositories. It assists organizations in adhering to data protection regulations and industry best practices.

Support for Multiple Platforms: While GitGuardian is primarily tailored for Git repositories, it extends its support to various version control platforms, code repositories, and collaboration tools.



Incidents

Table of secrets

FILTER: Search | DETECTORS: All | STATUS: 2 selected status | PRESENCE: All | VALIDITY: All | SOURCES: All | TAGS: All | SEVERITY: All | FEEDBACK: All | ASSIGNEE: All | JIRA: All | SHARED PUBLICLY: All | Clear filters | 1-4 of 4

DATE	SECRET	SEVERITY	INFO	TAGS	STATUS
October 15th, 2020 08:07	Generic High Entropy Secret #7399995	High	shalparni/nodetube lib/mediaPlayer/pushNotification.js Anthony contact@anthonymayfield.com	From historical scan; Exposed publicly; Default branch	Triggered
October 15th, 2020 07:23	Generic High Entropy Secret #7399996 2 occurrences 2	High	shalparni/nodetube lib/mediaPlayer/pushNotification.js Anthony contact@anthonymayfield.com	From historical scan; Exposed publicly; Default branch	Triggered
January 5th, 2020 12:40	Google API Key #7399997 6 occurrences 6	Critical	shalparni/nodetube controllers/backend/youtube.js Anthony +1 other contact@anthonymayfield.com	From historical scan; Exposed publicly; Default branch	Triggered
April 18th, 2018 01:45	RSA Private Key #7399999	High	shalparni/nodetube keys/server.key	From historical scan; Exposed publicly	Triggered

Figure 6: Git guardian dependency incidents

Power Bi

Power BI, a potent business analytics service created by Microsoft, enables users to create captivating and interactive reports and dashboards from various data sources. Through Power BI, organizations unlock valuable insights from their data, facilitating data-driven decision-making. Embraced across industries, Power BI stands as a versatile tool for data analysis, business intelligence, and reporting, catering to both individuals seeking self-service analytics and organizations requiring robust data solutions for their business operations.

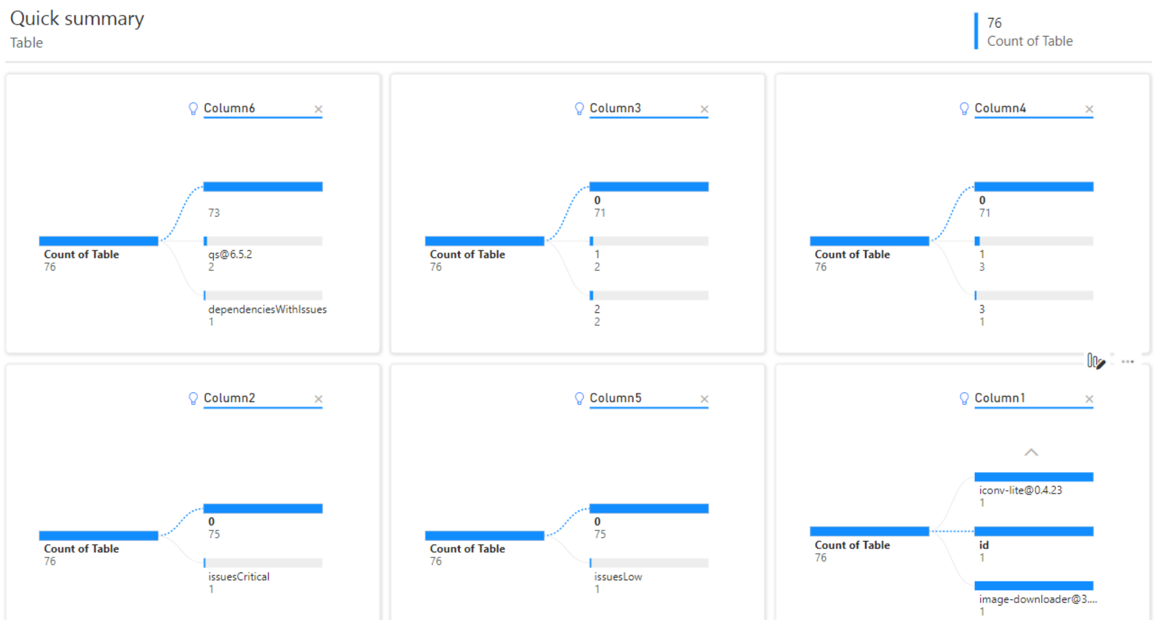


Figure 7: Tree representation of the dependencies using Power Bi tool

V. CONCLUSION

The Dependency Graph in GitHub offers a valuable and informative feature that visually represents a repository's dependencies. It aids developers and organizations in comprehending how the project's code relies on external libraries, frameworks, or packages. By presenting a clear and user-friendly view of the dependency hierarchy, the Dependency Graph simplifies dependency management, fostering a stronger and more efficient codebase. Notably, this feature plays a crucial role in identifying and rectifying potential security vulnerabilities by highlighting outdated or vulnerable dependencies. By enabling effective tracking and updating of dependencies, the Dependency Graph ensures a secure and up-to-date development environment.

Furthermore, the Dependency Graph seamlessly integrates with continuous monitoring and automated scanning, swiftly detecting any changes in dependencies to prevent issues and expedite problem resolution. This promotes collaboration and empowers developers to make well-informed decisions based on the latest insights regarding their project's dependencies.

In summary, the Dependency Graph serves as an invaluable tool supporting software development practices, empowering users to create more reliable, maintainable, and secure applications within their GitHub repositories. As GitHub and its features evolve, the Dependency Graph will continue to be a fundamental component in managing dependencies and fostering efficiency in the development process.

REFERENCES

[1] "Christina Paule, Thomas F. Dullmann, and Andr ¨ e van Hoorn, Vulnerabilities in Continuous Delivery Pipelines? A Case Study- 2019 IEEE"

[2] "Thorsten Rangnau, Remco v. Buijtenen, Frank Fransen Fatih Turkmen: Testing Continuous Security Testing:A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines-2020 IEEE"

[3] "Mojtaba Shahin, Muhammad Ali Babar, Liming Zhu-Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices-2017IEEE"

[4] "Murugiah Souppaya Michael Ogata Paul Watrobsk: software supply chain and devops security practices 2022 NIST"

[5] "Sergejs Bobrovskis, Aleksejs Jurenoks: A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment"

[6] "Michael Färber: Analyzing the GitHub Repositories 2022-researchgate"

[7] "Charanjot Singh, Nikita Seth Gaba, Manjot Kaur: Comparison of Different CI/CD Tools Integrated with Cloud Platform, 2019-IEEE"

[8] "Abdul Malik 1,2, Muhammad Shumail Naveed: Analysis of Code Vulnerabilities in Repositories of GitHub and RosettaCode: A Comparative Study, 2022 IJIST"

[9] "R. He, H. He, Y. Zhang and M. Zhou, "Automating Dependency Updates in Practice: An Exploratory Study on GitHub Dependabot, 2023 IEEE".