GitLab

GitLab Technical Papers

NIST SSDF 1.1

How can we help, and how do we align?

Disclaimer

Information in this document is relevant for GitLab Ultimate customers. When features only apply to the SaaS or Self-Managed environments, this will be identified.

Introduction

Supply chain security has always been a focal point for organizations across the globe. How can we ensure that companies outside of our purview have adequate security? If their security fails, can it have knock-on effects on my organization?

This becomes ever more critical when the supply chain is composed of hundreds if not thousands of external parties. With a strong third-party risk management program and sufficient resources, an organization can have a robust supply chain management practice. The true challenge lies elsewhere: the supply chain, but for software.

Think about having thousands of suppliers embedded right into your source code, which means directly in front of your users. This is the state of most organizations today as open source software is commonly included in today's applications in order to speed time to market or to create a competitive advantage.

This is why the White House is committed to help improve American cybersecurity as well as to secure American supply chains after numerous high profile cyberattacks had a substantial impact on US infrastructure.

On September 14, 2022, the Office of Management and Budget (OMB) issued Memorandum M-22-18 recommending that federal agencies vet software they adopt or procure has been developed in accordance with the NIST SSDF guidance.

What is the NIST SSDF?

The SSDF, also known as (SP 800-218), was built on best practices gathered from experts at BSA, OWASP and SAFECode and is intended to help greatly reduce software security vulnerabilities shipped in production software.

This guidance, currently in version 1.1, will be presented succinctly in this document. The goal is to work through the four main areas highlighted by the SSDF and explain how both GitLab the product and GitLab, Inc the company are committed to securing software for our customers.

A distinctive element of the SSDF is the *notional implementation examples*. Their role is to provide practical examples regarding how to meet the requirements of the guideline. The examples will be leveraged in order to map to GitLab product features and security controls that GitLab applies internally.

As such, the SSDF is outcome-based. Current controls may be audited against the SSDF practices to determine where gaps may exist (gap-assessment). The examples provide relevant guidance for implementation of each practice.

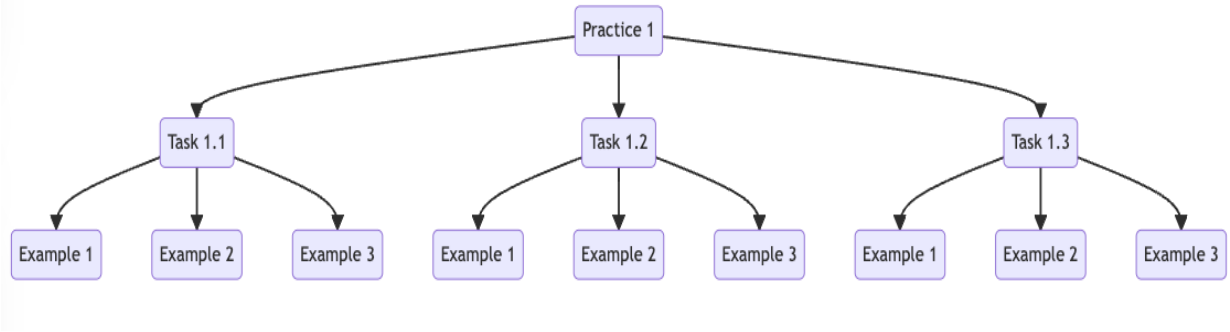

Figure 1: Structure of the SSDF Standard

As visible in Figure 1, the SSDF standard follows a three step process:

- **Practices** are sections of the SSDF detailing one area of focus
- **Tasks** divide the different practices into actionable chunks
- **Examples** demonstrate methods to apply tasks

The SSDF is divided into four key practices that provide a holistic view of software supply chain security:

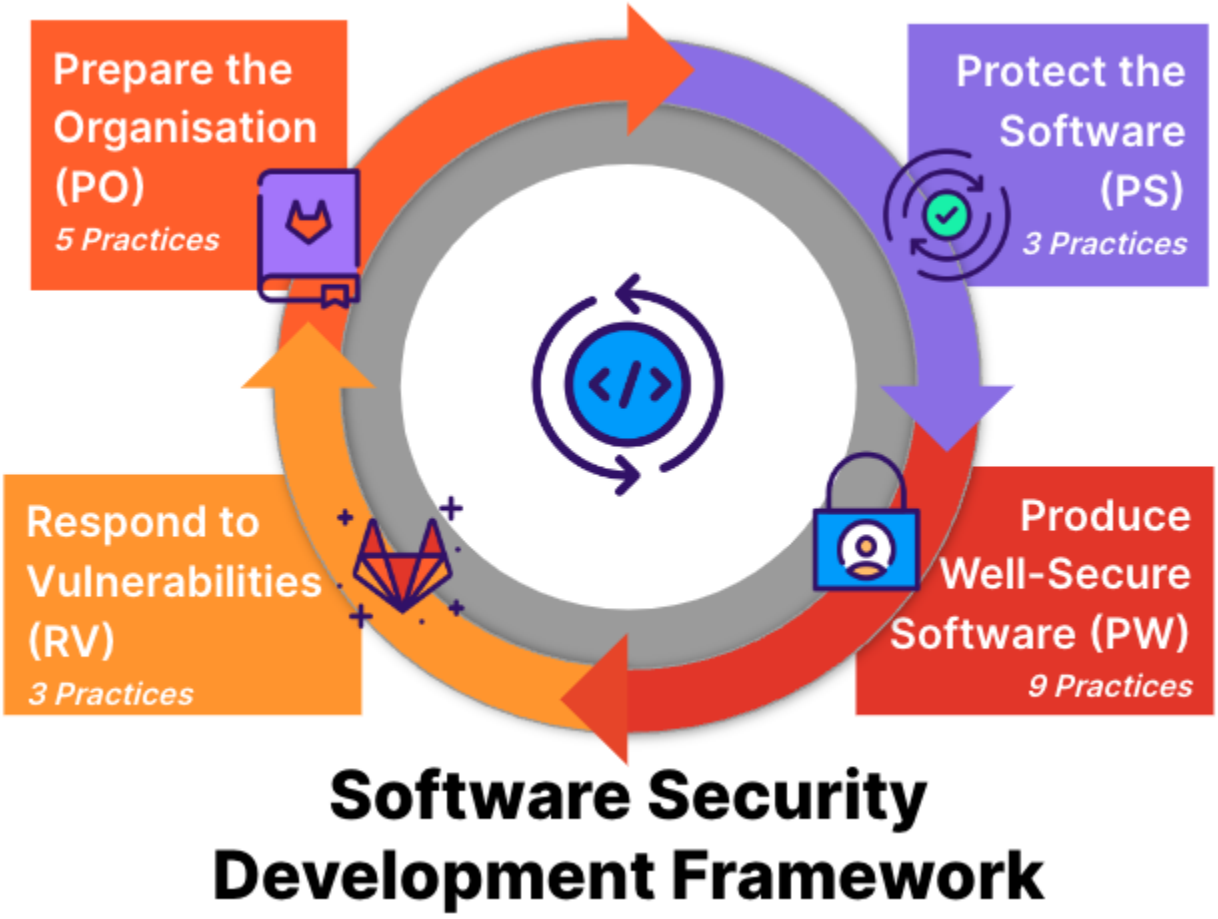

Figure 2: Components of the SSDF

GitLab's Role

As an organization committed to help customers mature in DevSecOps adoption, we welcome frameworks such as the SSDF, which enable organizations to benchmark their software security practices against industry standards.

GitLab's product position is unique as a platform provider tailored to help our customers towards securing their security supply chain. As a vendor, GitLab is a central part of customer supply chains and as such, GitLab follows SSDF guidance internally.

GitLab supports customers on **two fronts**:

- Ensuring that GitLab as a product gives customers the necessary visibility over their supply chain and provides the right security toolchain to properly protect the software factory (**GitLab**)
- Protecting GitLab, Inc as a company, so that, as a vendor, we can be an integral part of our customers' supply chains and uphold the most rigorous security standards, which includes leveraging our own product's security capabilities (**GitLab, Inc**)

The more an organization complies with SSDF 1.1, the more their supply chain will be secure as a result of creating a strong baseline across their software ecosystem.

SSDF Guidelines

The upcoming sections will succinctly walk through the different categories of the SSDF and describe how GitLab and GitLab, Inc align with these guidelines.

For *GitLab, the product*, GitLab product features will be highlighted which enable customers to align to best practice guidance.

For the *GitLab, Inc the company*, the focus is on how GitLab internally aligns with the different practices while providing links to additional documentation.

Prepare the Organisation (PO)

5 Practices

Ensure that the organization's people, processes, and technology are prepared to perform secure software development at the organization level and, in some cases, for individual development groups or projects.

GitLab, the product

Features	Description	SSDF tasks
Merge request approvals and push rules	Configure approvals for merge requests based on specific rules.	PO.3.2, PO.3.3, PO.4.1, PO.4.2
Security Dashboards	Use Security Dashboards to view vulnerabilities trends identified by security scanners.	PO.3.1, PO.3.2, PO.3.3, PO.4.1, PO.4.2. PO.5.2
Audit events and audit reports	Use audit events to track important events, including who performed a related action and when. Reports allow a centralized view over audit events for evidence gathering.	PO.1.1, PO.3.3, PO.4.2, PO.5.2
MFA, LDAP and SSO	Proper access control, authentication and authorization mechanisms to properly secure access to the source code repository.	PO.4.2, PO.5.1, PO.5.2

Granular roles and permissions	Role Based Access Control and granular permissions ensure least privilege and need-to-know principles are enforced.	PO.2.1, PO.2.2, PO.4.2, PO.5.1

Feature spotlight: Merge request approvals

Changes to a project repository typically start with a merge request.

If the default branch is protected, commits must be done through a merge request. Configuring merge request settings with approval rules ensures that changes are properly approved prior to merging code or deploying to production.

Within the merge request approval settings, the number of approvals required and who is allowed to approve merge requests can be specified.

In addition, there are a number of approval settings that further enforce segregation of duties within change management:

- Prevent approval by author: When enabled, the merge request author cannot also provide one of the required approvals.
- Prevent approvals by users who add commits: When enabled, users who have committed to a merge request cannot also approve it.
- Prevent editing approval rules in merge requests: When enabled, users cannot override the project's approval rules on merge requests.
- Require user password to approve: When enabled, users must first authenticate with a password prior to submitting approval.
- Remove all approvals when commits are added to the source branch: When enabled, this removes all existing approvals on a merge request when more changes are added to it.

GitLab, the company

PO.1: Define Security Requirements for Software Development

- Internally, GitLab developed a list of requirements for securely developing and deploying software:
  - Security Requirements for Development and Deployment | GitLab

PO.2: Implement Roles and Responsibilities

- Set responsibilities and roles for the application security review:
  - Application Security Review Process | GitLab

PO.3: Implement Supporting Toolchains

- Reduce human effort and improve accuracy of SDLC by leveraging the native GitLab security scanning capabilities:
  - Secure your application | GitLab

PO.4: Define and Use Criteria for Software Security Checks

- Leveraging the features offered by GitLab
  - GitLab PO Features
- Following a Scaled Agile approach to project management
  - Scaled Agile and GitLab

PO.5: Implement and Maintain Secure Environments for Software Development

- Enforce SSO and MFA via Okta
  Okta | GitLab
- Separation of different infrastructure environments
  Infrastructure Environments | GitLab
- Zero Trust Architecture
  Security at GitLab
- Encryption at rest for all data stored in the GitLab SaaS infrastructure
  Encryption Policy | GitLab

Protect the Software (PS)

3 Practices

Protect all components of the software from tampering and unauthorized access.

GitLab, the product

Features	Description	SSDF tasks
Source Code Management	Git-based repository with version control allowing automatic scanning and built-in CI/CD	PS.1.1
Software Bill of Materials (SBOM)	An inventory of all constituent components and software dependencies involved in the development and delivery of an application	PS.2.1, PS.3.2
Commit Signatures	A digital signature providing non-repudiation and authenticity regarding who committed code to the repository.	PS.1.1, PS.3.1
Code Reviews	Ensuring guidelines are followed to properly review code when assigned. This is necessary to ensure the separation of duties is effective.	PS.1.1

Feature spotlight: Source Code Management

With Source Code Management, software development work is managed through a single source of truth. This enables users to centrally review, track and approve code changes using merge requests. SCM is powered by Git-based repositories which decentralizes source code management and enables developers to work locally.

Version control, in addition to continuous feedback loops, ensures iteration is at the forefront of how code is shipped to production. All of this is powered by automation which reduces human error and ensures the process is always followed as it was designed.

Developers can review, comment and improve on each other's code. This increases visibility, maximizes productivity and shortens time to production.

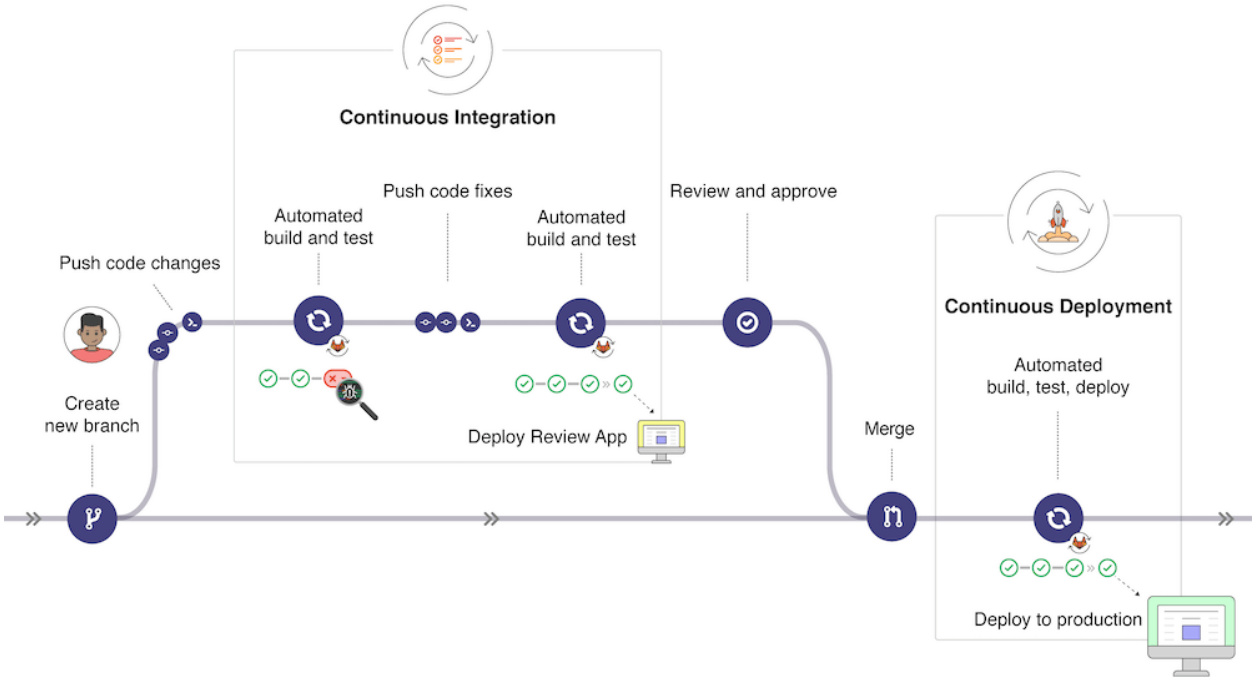

Figure 3: GitLab workflow

GitLab, the company

PS.1: Protect All Forms of Code from Unauthorized Access and Tampering

- All code at GitLab is managed using Source Code Management
  GitLab's source code repositories

PS.2: Provide a Mechanism for Verifying Software Release Integrity

- Hashes for all of GitLab release packages are readily available
  gitlab/gitlab-ee - Packages

PS.3: Archive and Protect Each Software Release

- Strong Access Control Program with Least-Privilege
  Access Management Policy | GitLab

Produce Well-Secure Software (PW)

9 Practices

Produce well-secured software with minimal security vulnerabilities in its releases.

GitLab, the product

Features	Description	SSDF tasks
Vulnerability Report	The Vulnerability Report provides information about vulnerabilities from scans of the default branch. It contains cumulative results of all successful jobs, regardless of whether the pipeline was successful.	PW.1.1, PW.7.2
Static Security Scanning	Using GitLab CI/CD, Static Application Security Testing (SAST) can be used to check source code for known vulnerabilities. The analyzers output JSON-formatted reports as job artifacts.	PW.5.1, PW.8.1
Dynamic Security Scanning	DAST examines applications for vulnerabilities in deployed environments. An application may become exposed to new types of attacks once deployed. For example: misconfigurations of an application server or incorrect assumptions about security controls.	PW.5.1, PW.6.1, PW.8.2
Dependency Scanning	The Dependency Scanning feature can automatically find security vulnerabilities in software dependencies during development and testing of applications.	PW.4.1, PW.4.4, PW.6.1

	For example, dependency scanning identifies when an external (open source) library is adopted that is known to contain vulnerabilities.	
Compliance frameworks and pipelines	Default compliance frameworks can be enabled for all groups. A default framework is applied to all the new projects that are created. Compliance pipelines can be created to be applied to specific projects and branches.	PW.1.2, PW.1.3, PW.6.2, PW.7.1, PW.8.1, PW.9.2

Feature spotlight: Security Scanning

GitLab has a wide breadth of Security Scanning capabilities adapted to each stage of the development process. As detailed in Figure 4, the Commit, Build, Test and Deploy stages Scanners are available to help produce reliable, secure code.

As highlighted in the Get started with GitLab application security page, it is suggested to update the default branch .gitlab-ci.yml file to ensure scanners are enabled for all merge requests.

As each team's needs are different, guidelines are available in the page mentioned above as well as the documentation entries for all the specific scans.

Commit

- IaC Scanning
- SAST
- Secret Detection
- License Scanning

Build

- Fuzz Testing
- Dependency Scanning
- Container Scanning

Test

- API Security
- DAST

Deploy

- Operational Container Scanning

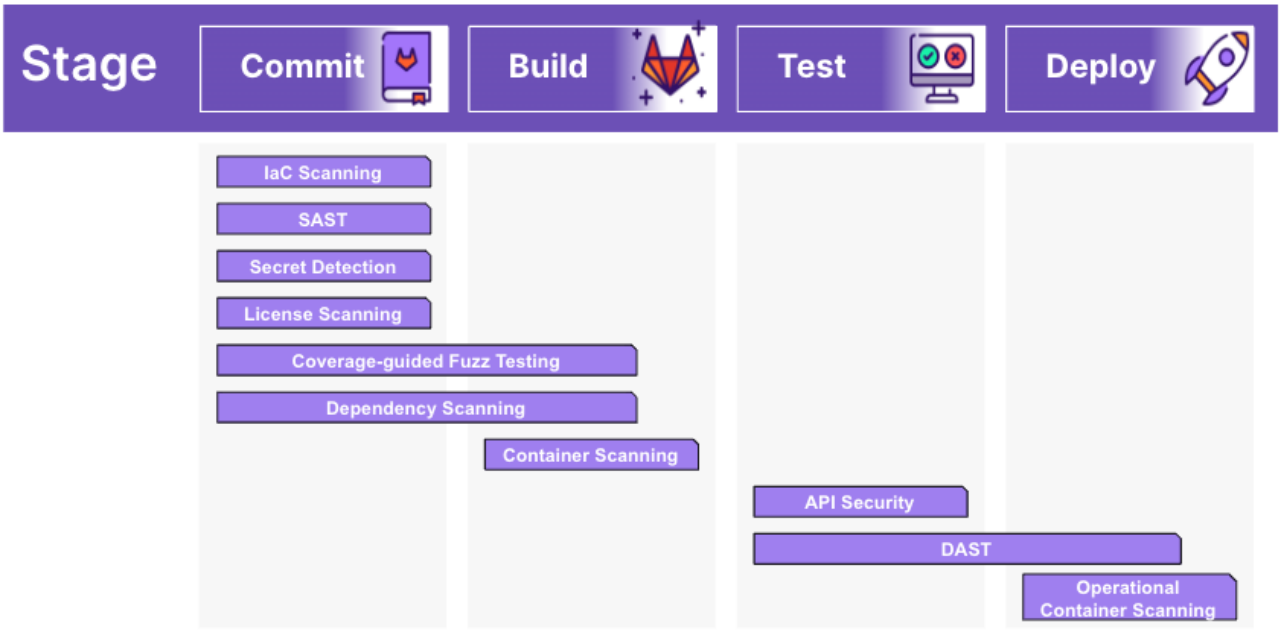

Figure 4: GitLab Security Scanning Capabilities

GitLab, the company

PW.1: Design Software to Meet Security Requirements and Mitigate Security Risks

- Developers collaborate with Application Security to produce Threat Models to inform of potential risks with new features and relevant mitigations
  Threat Modeling | GitLab

PW.2: Review the Software Design to Verify Compliance with Security Requirements and Risk Information

- The Application Security team reviews and proactively discovers vulnerabilities in GitLab applications
  Application Security Review Process | GitLab

PW.4: Reuse Existing, Well-Secured Software When Feasible Instead of Duplicating Functionality

- GitLab validates any Open Source Software added into its application by following a detailed security review process

  Please review the *Securing GitLab's Supply Chain* Technical Paper available in the Community CAP

PW.5: Create Source Code by Adhering to Secure Coding Practices

- GitLab developers follow Secure Coding practices and guidelines for developing and testing and reviewing code securely

  Security Requirements for Development and Deployment | GitLab
  Code Review Guidelines | GitLab

PW.6: Configure the Compilation, Interpreter, and Build Processes to Improve Executable Security

- GitLab Runner technology, which are continuously updated (SaaS), is used within a highly controlled environment and can be deployed via configuration-as-code

  GitLab Runner

PW.7: Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements

- Code reviews and application security analysis are performed on merge requests in addition to automated security scans

  Application Security Review Process | GitLab
  Code Review Guidelines | GitLab
  Secure your application | GitLab

PW.8: Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements

- Relevant tests such as Fuzz Testing and DAST are automatically performed on merge requests before deployment to the production environment
  Security Scanners per Stage

PW.9: Configure Software to Have Secure Settings by Default

- GitLab has a default baseline for all merge requests written as a compliance framework and added to the default branch .gitlab-ci.yml file
  Compliance frameworks | GitLab

Respond to Vulnerabilities (RV)

3 Practices

Identify residual vulnerabilities in software releases and respond appropriately to address those vulnerabilities and prevent similar vulnerabilities from occurring in the future.

GitLab, the product

Features	Description	SSDF tasks
Vulnerability Page	For all vulnerabilities in a project, the page contains details of each vulnerability such as when it was detected, its current status or linked issues. Users can perform numerous actions such as: Change the vulnerability's status, Create an issue, Link issues to the vulnerability, Resolve the vulnerability, or View specific security training.	RV.1.1, RV.1.3, RV.2.1, RV.3.1, RV.3.2, RV.3.3
On-demand DAST scanning	An on-demand DAST scan runs outside the DevOps life cycle. Changes in a repository don't trigger the scan. It must be started manually or scheduled to run (weekly, monthly, etc.)	RV.1.2
Vulnerability Reporting	This reporting provides information about vulnerabilities from scans of the default branch. It contains results of all jobs, information about the total numbers of	RV.2.2, RV.3.3

	vulnerabilities by severity, and details for each vulnerability identified.	

Feature spotlight: Security Training based on Vulnerabilities

Security training helps developers learn how to fix vulnerabilities. Developers can view security training from selected educational providers, relevant to the detected vulnerability.

The vulnerability page may include a training link relevant to the detected vulnerability if security training is enabled. The availability of training depends on whether the enabled training vendor has content matching the particular vulnerability.

Training content is requested based on the vulnerability identifiers. Vulnerabilities with a CWE are most likely to return a training result. This enables developers to learn from past vulnerabilities to avoid reproducing the same vulnerability in the future.

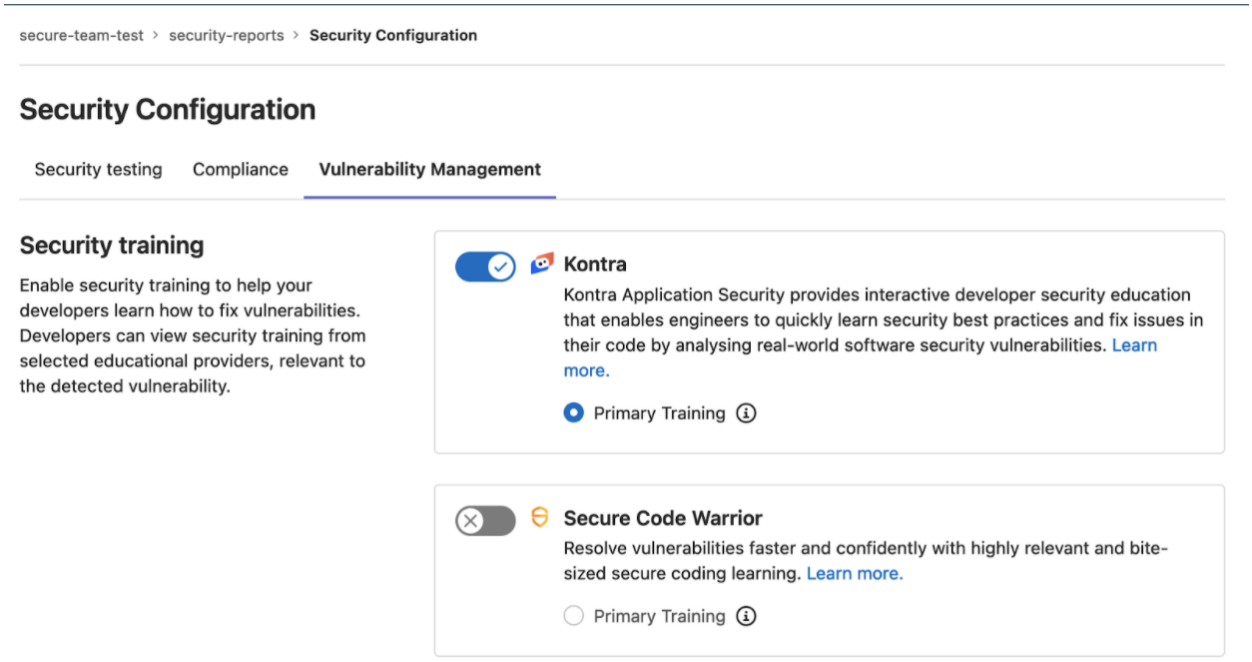

Figure 5: Security Training Vulnerability interface

GitLab, the company

RV.1: Identify and Confirm Vulnerabilities on an Ongoing Basis

- GitLab maintains its own vulnerability database and updates its internal scanners according to newly discovered vulnerabilities
  GitLab Advisory Database

- GitLab has a strong Vulnerability Disclosure Process and leverages HackerOne for its Bug Bounty Program
  Responsible Disclosure Policy | GitLab
  HackerOne Process | GitLab

RV.2: Assess, Prioritize, and Remediate Vulnerabilities

- GitLab's vulnerability management program details the process for assessing, prioritizing and remediating vulnerabilities as well as current SLAs
  Vulnerability Management Overview | GitLab

RV.3: Analyze Vulnerabilities to Identify Their Root Causes

- The GitLab Application Security team performs Root Cause Analysis on critical vulnerabilities and record lessons learned using Issue Management
  Root Cause Analysis for Critical Vulnerabilities | GitLab

Conclusion

The objective of this Technical Paper was to showcase how GitLab's capabilities can help align to the SSDF guidelines. At the same time, as part of your supply chain, GitLab follows the SSDF Guidance as highlighted in *GitLab, the company* sections.

For more information around the security measures implemented at GitLab, please review the Security Handbook.

For more details around the security features available in the product, please review the GitLab Documentation.

We are committed to help you on your DevSecOps journey.