

CIS GitLab Benchmark

v1.0.1 - 04-19-2024

Terms of Use

Please see the below link for our current terms of use:

<https://www.cisecurity.org/cis-securesuite/cis-securesuite-membership-terms-of-use/>

Table of Contents

Terms of Use	1
Table of Contents	2
Overview	7
Intended Audience.....	7
Consensus Guidance	8
Typographical Conventions.....	9
Recommendation Definitions.....	10
Title.....	10
Assessment Status.....	10
Automated	10
Manual.....	10
Profile	10
Description.....	10
Rationale Statement	10
Impact Statement.....	11
Audit Procedure.....	11
Remediation Procedure.....	11
Default Value.....	11
References	11
CIS Critical Security Controls® (CIS Controls®)	11
Additional Information.....	11
Profile Definitions	12
Acknowledgements	13
Recommendations	14
1 Source Code	14
1.1 Code Changes.....	15
1.1.1 Ensure any changes to code are tracked in a version control platform (Manual)	16
1.1.2 Ensure any change to code can be traced back to its associated task (Manual)	17
1.1.3 Ensure any change to code receives approval of two strongly authenticated users (Automated)	19
1.1.4 Ensure previous approvals are dismissed when updates are introduced to a code change proposal (Manual)	21
1.1.5 Ensure there are restrictions on who can dismiss code change reviews (Manual).....	23
1.1.6 Ensure code owners are set for extra sensitive code or configuration (Manual)	25
1.1.7 Ensure code owner's review is required when a change affects owned code (Manual) .	27
1.1.8 Ensure inactive branches are periodically reviewed and removed (Manual)	30
1.1.9 Ensure all checks have passed before merging new code (Manual)	32

1.1.10 Ensure open Git branches are up to date before they can be merged into code base (Manual).....	34
1.1.11 Ensure all open comments are resolved before allowing code change merging (Manual).....	36
1.1.12 Ensure verification of signed commits for new changes before merging (Manual)	38
1.1.13 Ensure linear history is required (Manual)	40
1.1.14 Ensure branch protection rules are enforced for administrators (Manual)	42
1.1.15 Ensure pushing or merging of new code is restricted to specific individuals or teams (Manual)	44
1.1.16 Ensure force push code to branches is denied (Manual)	46
1.1.17 Ensure branch deletions are denied (Manual).....	48
1.1.18 Ensure any merging of code is automatically scanned for risks (Manual)	51
1.1.19 Ensure any changes to branch protection rules are audited (Manual)	53
1.1.20 Ensure branch protection is enforced on the default branch (Manual).....	55
1.2 Repository Management	58
1.2.1 Ensure all public repositories contain a SECURITY.md file (Manual)	59
1.2.2 Ensure repository creation is limited to specific members (Manual)	61
1.2.3 Ensure repository deletion is limited to specific users (Manual)	63
1.2.4 Ensure issue deletion is limited to specific users (Manual)	65
1.2.5 Ensure all copies (forks) of code are tracked and accounted for (Manual)	67
1.2.6 Ensure all code projects are tracked for changes in visibility status (Manual)	69
1.2.7 Ensure inactive repositories are reviewed and archived periodically (Manual).....	71
1.3 Contribution Access.....	73
1.3.1 Ensure inactive users are reviewed and removed periodically (Manual)	74
1.3.2 Ensure top-level group creation is limited to specific members (Manual)	76
1.3.3 Ensure minimum number of administrators are set for the organization (Manual)	78
1.3.4 Ensure Multi-Factor Authentication (MFA) is required for contributors of new code (Manual).....	80
1.3.5 Ensure the organization is requiring members to use Multi-Factor Authentication (MFA) (Manual).....	82
1.3.6 Ensure new members are required to be invited using company-approved email (Manual).....	84
1.3.7 Ensure two administrators are set for each repository (Manual)	86
1.3.8 Ensure strict base permissions are set for repositories (Manual)	88
1.3.9 Ensure an organization's identity is confirmed with a "Verified" badge (Manual).....	90
1.3.10 Ensure Source Code Management (SCM) email notifications are restricted to verified domains (Manual)	93
1.3.11 Ensure an organization provides SSH certificates (Manual)	95
1.3.12 Ensure Git access is limited based on IP addresses (Manual)	97
1.3.13 Ensure anomalous code behavior is tracked (Manual)	99
1.4 Third-Party	100
1.4.1 Ensure administrator approval is required for every installed application (Manual)	101
1.4.2 Ensure stale applications are reviewed and inactive ones are removed (Manual)	103
1.4.3 Ensure the access granted to each installed application is limited to the least privilege needed (Manual).....	105
1.4.4 Ensure only secured webhooks are used (Manual)	109
1.5 Code Risks	111
1.5.1 Ensure scanners are in place to identify and prevent sensitive data in code (Manual)	112
1.5.2 Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions (Manual).....	114
1.5.3 Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions (Manual).....	116
1.5.4 Ensure scanners are in place for code vulnerabilities (Manual)	118
1.5.5 Ensure scanners are in place for open-source vulnerabilities in used packages (Manual)	120

1.5.6 Ensure scanners are in place for open-source license issues in used packages (Manual)	122
1.5.7 Ensure scanners are in place for web application runtime security weaknesses (Manual)	124
1.5.8 Ensure scanners are in place for API runtime security weaknesses (Manual)	126
2 Build Pipelines	128
2.1 Build Environment	129
2.1.1 Ensure each pipeline has a single responsibility (Manual)	130
2.1.2 Ensure all aspects of the pipeline infrastructure and configuration are immutable (Manual)	132
2.1.3 Ensure the build environment is logged (Manual)	134
2.1.4 Ensure the creation of the build environment is automated (Manual)	136
2.1.5 Ensure access to build environments is limited (Manual)	138
2.1.6 Ensure users must authenticate to access the build environment (Manual)	140
2.1.7 Ensure build secrets are limited to the minimal necessary scope (Manual)	142
2.1.8 Ensure the build infrastructure is automatically scanned for vulnerabilities (Manual)	144
2.1.9 Ensure default passwords are not used (Manual)	146
2.1.10 Ensure webhooks of the build environment are secured (Manual)	148
2.1.11 Ensure minimum number of administrators are set for the build environment (Manual)	150
2.2 Build Worker	152
2.2.1 Ensure build workers are single-used (Manual)	153
2.2.2 Ensure build worker environments and commands are passed and not pulled (Manual)	155
2.2.3 Ensure the duties of each build worker are segregated (Manual)	157
2.2.4 Ensure build workers have minimal network connectivity (Manual)	160
2.2.5 Ensure run-time security is enforced for build workers (Manual)	162
2.2.6 Ensure build workers are automatically scanned for vulnerabilities (Manual)	163
2.2.7 Ensure build workers' deployment configuration is stored in a version control platform (Manual)	165
2.2.8 Ensure resource consumption of build workers is monitored (Manual)	167
2.3 Pipeline Instructions	169
2.3.1 Ensure all build steps are defined as code (Manual)	170
2.3.2 Ensure steps have clearly defined build stage input and output (Manual)	171
2.3.3 Ensure output is written to a separate, secured storage repository (Manual)	173
2.3.4 Ensure changes to pipeline files are tracked and reviewed (Manual)	175
2.3.5 Ensure access to build process triggering is minimized (Manual)	177
2.3.6 Ensure pipelines are automatically scanned for misconfigurations (Manual)	180
2.3.7 Ensure pipelines are automatically scanned for vulnerabilities (Manual)	181
2.3.8 Ensure scanners are in place to identify and prevent sensitive data in pipeline files (Automated)	182
2.4 Pipeline Integrity	184
2.4.1 Ensure all artifacts on all releases are signed (Manual)	185
2.4.2 Ensure all external dependencies used in the build process are locked (Manual)	186
2.4.3 Ensure dependencies are validated before being used (Manual)	188
2.4.4 Ensure the build pipeline creates reproducible artifacts (Manual)	190
2.4.5 Ensure pipeline steps produce a Software Bill of Materials (SBOM) (Manual)	192
2.4.6 Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced (Manual)	195
3 Dependencies	196
3.1 Third-Party Packages	197
3.1.1 Ensure third-party artifacts and open-source libraries are verified (Manual)	198
3.1.2 Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers (Manual)	199
3.1.3 Ensure signed metadata of the build process is required and verified (Manual)	201
3.1.4 Ensure dependencies are monitored between open-source components (Manual)	202

3.1.5 Ensure trusted package managers and repositories are defined and prioritized (Manual)	204
3.1.6 Ensure a signed Software Bill of Materials (SBOM) of the code is supplied (Manual)	206
3.1.7 Ensure dependencies are pinned to a specific, verified version (Manual)	208
3.1.8 Ensure all packages used are more than 60 days old (Manual)	209
3.2 Validate Packages	211
3.2.1 Ensure an organization-wide dependency usage policy is enforced (Manual)	212
3.2.2 Ensure packages are automatically scanned for known vulnerabilities (Manual)	213
3.2.3 Ensure packages are automatically scanned for license implications (Manual)	215
3.2.4 Ensure packages are automatically scanned for ownership change (Manual)	217
4 Artifacts	218
4.1 Verification	219
4.1.1 Ensure all artifacts are signed by the build pipeline itself (Manual)	220
4.1.2 Ensure artifacts are encrypted before distribution (Manual)	221
4.1.3 Ensure only authorized platforms have decryption capabilities of artifacts (Manual)	222
4.2 Access to Artifacts	224
4.2.1 Ensure the authority to certify artifacts is limited (Manual)	225
4.2.2 Ensure number of permitted users who may upload new artifacts is minimized (Manual)	226
4.2.3 Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA) (Manual)	228
4.2.4 Ensure user management of the package registry is not local (Manual)	230
4.2.5 Ensure anonymous access to artifacts is revoked (Manual)	231
4.2.6 Ensure minimum number of administrators are set for the package registry (Manual)	233
4.3 Package Registries	235
4.3.1 Ensure all signed artifacts are validated upon uploading the package registry (Manual)	236
4.3.2 Ensure all versions of an existing artifact have their signatures validated (Manual)	238
4.3.3 Ensure changes in package registry configuration are audited (Manual)	240
4.3.4 Ensure webhooks of the repository are secured (Manual)	241
4.4 Origin Traceability	242
4.4.1 Ensure artifacts contain information about their origin (Manual)	243
5 Deployment	245
5.1 Deployment Configuration	246
5.1.1 Ensure deployment configuration files are separated from source code (Manual)	247
5.1.2 Ensure changes in deployment configuration are audited (Manual)	248
5.1.3 Ensure scanners are in place to identify and prevent sensitive data in deployment configuration (Manual)	249
5.1.4 Limit access to deployment configurations (Manual)	250
5.1.5 Scan Infrastructure as Code (IaC) (Manual)	251
5.1.6 Ensure deployment configuration manifests are verified (Manual)	253
5.1.7 Ensure deployment configuration manifests are pinned to a specific, verified version (Manual)	254
5.2 Deployment Environment	256
5.2.1 Ensure deployments are automated (Manual)	257
5.2.2 Ensure the deployment environment is reproducible (Manual)	258
5.2.3 Ensure access to production environment is limited (Manual)	259
5.2.4 Ensure default passwords are not used (Manual)	260
Appendix: Summary Table	262
Appendix: CIS Controls v7 IG 1 Mapped Recommendations	272
Appendix: CIS Controls v7 IG 2 Mapped Recommendations	274
Appendix: CIS Controls v7 IG 3 Mapped Recommendations	279

<i>Appendix: CIS Controls v8 IG 1 Mapped Recommendations</i>	<i>284</i>
<i>Appendix: CIS Controls v8 IG 2 Mapped Recommendations</i>	<i>286</i>
<i>Appendix: CIS Controls v8 IG 3 Mapped Recommendations</i>	<i>291</i>
<i>Appendix: Change History</i>	<i>297</i>

Overview

All CIS Benchmarks focus on technical configuration settings used to maintain and/or increase the security of the addressed technology, and they should be used in **conjunction** with other essential cyber hygiene tasks like:

- Monitoring the base operating system for vulnerabilities and quickly updating with the latest security patches
- Monitoring applications and libraries for vulnerabilities and quickly updating with the latest security patches

In the end, the CIS Benchmarks are designed as a key **component** of a comprehensive cybersecurity program.

This document provides prescriptive guidance for establishing a secure configuration posture for securing the Software Supply Chain. To obtain the latest version of this guide, please visit www.cisecurity.org. If you have questions, comments, or have identified ways to improve this guide, please write to us at support@cisecurity.org.

Special Note: The set of configuration files mentioned anywhere throughout this benchmark document may vary according to the deployment tool and the platform. Any reference to a configuration file should be modified according to the actual configuration files used on the specific deployment.

Intended Audience

This document is intended for DevOps and application security administrators, security specialists, auditors, help desk, and platform deployment personnel who plan to develop, deploy, assess, or secure solutions to build and deploy software updates through automated means of DevOps pipelines.

Consensus Guidance

This CIS Benchmark was created using a consensus review process comprised of a global community of subject matter experts. The process combines real world experience with data-based information to create technology specific guidance to assist users to secure their environments. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Each CIS Benchmark undergoes two phases of consensus review. The first phase occurs during initial Benchmark development. During this phase, subject matter experts convene to discuss, create, and test working drafts of the Benchmark. This discussion occurs until consensus has been reached on Benchmark recommendations. The second phase begins after the Benchmark has been published. During this phase, all feedback provided by the Internet community is reviewed by the consensus team for incorporation in the Benchmark. If you are interested in participating in the consensus process, please visit <https://workbench.cisecurity.org/>.

Typographical Conventions

The following typographical conventions are used throughout this guide:

Convention	Meaning
<code>Stylized Monospace font</code>	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
<code>Monospace font</code>	Used for inline code, commands, or examples. Text should be interpreted exactly as presented.
<i><italic font in brackets></i>	Italic texts set in angle brackets denote a variable requiring substitution for a real value.
<i>Italic font</i>	Used to denote the title of a book, article, or other publication.
Note	Additional information or caveats

Recommendation Definitions

The following defines the various components included in a CIS recommendation as applicable. If any of the components are not applicable it will be noted or the component will not be included in the recommendation.

Title

Concise description for the recommendation's intended configuration.

Assessment Status

An assessment status is included for every recommendation. The assessment status indicates whether the given recommendation can be automated or requires manual steps to implement. Both statuses are equally important and are determined and supported as defined below:

Automated

Represents recommendations for which assessment of a technical control can be fully automated and validated to a pass/fail state. Recommendations will include the necessary information to implement automation.

Manual

Represents recommendations for which assessment of a technical control cannot be fully automated and requires all or some manual steps to validate that the configured state is set as expected. The expected state can vary depending on the environment.

Profile

A collection of recommendations for securing a technology or a supporting platform. Most benchmarks include at least a Level 1 and Level 2 Profile. Level 2 extends Level 1 recommendations and is not a standalone profile. The Profile Definitions section in the benchmark provides the definitions as they pertain to the recommendations included for the technology.

Description

Detailed information pertaining to the setting with which the recommendation is concerned. In some cases, the description will include the recommended value.

Rationale Statement

Detailed reasoning for the recommendation to provide the user a clear and concise understanding on the importance of the recommendation.

Impact Statement

Any security, functionality, or operational consequences that can result from following the recommendation.

Audit Procedure

Systematic instructions for determining if the target system complies with the recommendation.

Remediation Procedure

Systematic instructions for applying recommendations to the target system to bring it into compliance according to the recommendation.

Default Value

Default value for the given setting in this recommendation, if known. If not known, either not configured or not defined will be applied.

References

Additional documentation relative to the recommendation.

CIS Critical Security Controls® (CIS Controls®)

The mapping between a recommendation and the CIS Controls is organized by CIS Controls version, Safeguard, and Implementation Group (IG). The Benchmark in its entirety addresses the CIS Controls safeguards of (v7) "5.1 - Establish Secure Configurations" and (v8) "4.1 - Establish and Maintain a Secure Configuration Process" so individual recommendations will not be mapped to these safeguards.

Additional Information

Supplementary information that does not correspond to any other field but may be useful to the user.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1**

Items in this profile intend to:

- be practical and prudent;
- provide a clear security benefit; and
- not inhibit the utility of the technology beyond acceptable means.

- **Level 2**

This profile extends the "Level 1 - Domain Controller" profile. Items in this profile exhibit one or more of the following characteristics:

- are intended for environments or use cases where security is paramount
- acts as defense in depth measure
- may negatively inhibit the utility or performance of the technology

Acknowledgements

This Benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

This benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks to the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

Authors

- Randall Mowen – Center for Internet Security
- Resheet Kosef – Aqua Security
- Eylam Milner – Aqua Security

Editors

- Sara Meadzinger – GitLab
- Ayoub Fandi – GitLab

Contributors

- Stuart Taylor
- Nikhil Verma
- James Scott
- Tony Wilwerding
- Matthew Reagan – Center for Internet Security
- Phil White – Center for Internet Security

Special Thanks to:

- Greg Myers – GitLab
- Nick Malcolm – GitLab

And the entire GitLab team that contributed their knowledge and inputs to make the benchmark a reality.

Recommendations

1 Source Code

This section consists of security recommendations for proper source code management of any application developed by the organization. This is the first phase of the software supply chain, and is considered the single source of truth for the rest of the process.

It is critical to secure both the source code itself, as well as the platform with which it is managed, in order to protect the integrity of a software release. From the developers who commit changes, to the sensitive data or vulnerabilities that could be placed within it, and ultimately to the source code management platform in which it is stored, verification of the integrity of the source code is imperative in order to keep every software update secure.

1.1 Code Changes

This section consists of security recommendations for code changes and how they should be done. It contains recommendations to protect the main branch of the application code. This branch is the most important one, because it contains the actual code that is being delivered to the customer. It should be protected from any mistake or malicious deed in order to keep the software secured.

1.1.1 Ensure any changes to code are tracked in a version control platform (Manual)

Profile Applicability:

- Level 1

Description:

Manage all code projects in a version control platform.

Rationale:

Version control platforms keep track of every modification to code. They represent the cornerstone of code security, as well as allowing for better code collaboration within engineering teams. With granular access management, change tracking, and key signing of code edits, version control platforms are the first step in securing the software supply chain.

Audit:

Ensure that all code activity is managed in a GitLab repository for every micro-service or application developed by your organization.

Remediation:

Upload existing code projects to a GitLab group or instance and create an identity for each active team member who might contribute or need access to it.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.4 Utilize Automated Software Inventory Tools Utilize software inventory tools, when possible, throughout the enterprise to automate the discovery and documentation of installed software.		●	●
v7	2.4 Track Software Inventory Information The software inventory system should track the name, version, publisher, and install date for all software, including operating systems authorized by the organization.		●	●

1.1.2 Ensure any change to code can be traced back to its associated task (Manual)

Profile Applicability:

- Level 1

Description:

Use a task management system to trace any code back to its associated task.

Rationale:

The ability to trace each piece of code back to its associated task simplifies the Agile and DevOps process by enabling transparency of any code changes. This allows faster remediation of bugs and security issues, while also making it harder to push unauthorized code changes to sensitive projects. Additionally, using a task management system simplifies achieving compliance, as it is easier to track each regulation.

Audit:

Ensure every code change can be traced back to its origin task in a task management system.

Remediation:

Use GitLab issues to manage tasks as the starting point for each code change. Whether it is a new feature, bug fix, or security fix - all should originate from a dedicated task (GitLab issue) in your organization's task management system. Tasks (issues) should be linked to Merge Requests, and Merge requests should be linked to Issues.

References:

1. https://docs.gitlab.com/ee/user/project/issues/related_issues.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.3 Ensure any change to code receives approval of two strongly authenticated users (Automated)

Profile Applicability:

- Level 1

Description:

Ensure that every code change is reviewed and approved by two authorized contributors who are both strongly authenticated - using Multi-Factor Authentication (MFA), from the team relevant to the code change.

Rationale:

To prevent malicious or unauthorized code changes, the first layer of protection is the process of code review. This process involves engineer teammates reviewing each other's code for errors, optimizations, and general knowledge-sharing. With proper peer reviews in place, an organization can detect unwanted code changes very early in the process of release. In order to help facilitate code review, companies should employ automation to verify that every code change has been reviewed and approved by at least two team members before it is pushed into the code base. These team members should be from the team that is related to the code change, so it will be a meaningful review.

Impact:

To enforce a code review requirement, verification for a minimum of two reviewers must be put into place. This will ensure new code will not be able to be pushed to the code base before it has received two independent approvals.

Audit:

For every project in use, perform the next steps to verify that two approvals from the specific project team are required to push new code to the code base:

- On the left sidebar, select Search or go to and find your project.
- Select Settings > Merge requests.
- In the Merge request approvals section, in the Approval rules section, next to the rule you want to edit, select Edit.
- Review the field In Approvals required, if the number is 2 or above, you are compliant.

Remediation:

For every project in use, perform the next steps to require two approvals from the specific project team in order to push new code to the code base:

- On the left sidebar, select Search or go to and find your project.
- Select Settings > Merge requests.
- In the Merge request approvals section, in the Approval rules section, next to the rule you want to edit, select Edit.
- Edit the field In Approvals required to ensure the value is 2 or above.
- Select Update approval rule.

Default Value:

0

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/approvals/rules.html#edit-an-approval-rule

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.4 Ensure previous approvals are dismissed when updates are introduced to a code change proposal (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that when a proposed code change is updated, previous approvals are declined, and new approvals are required.

Rationale:

An approval process is necessary when code changes are suggested. Through this approval process, however, changes can still be made to the original proposal even after some approvals have already been given. This means malicious code can find its way into the code base even if the organization has enforced a review policy. To ensure this is not possible, outdated approvals must be declined when changes to the suggestion are introduced.

Impact:

If new code changes are pushed to a specific proposal, all previously accepted code change proposals must be declined.

Audit:

For each code repository in use, perform the next steps to verify that each updated code suggestion declines the previously received approvals:

- On GitLab, navigate to the main page of a repository.
- Navigate to **Settings > Merge Requests**.
- Click **Expand** next to the **Merge Request Approvals** section.
- Verify the repository has merge request approvals configured on all protected branches.
- Verify that "Remove all approvals" is selected under Approval Settings for "When a commit is added".

Remediation:

For each code repository in use, perform the next steps to enforce dismissal of given approvals to code change suggestions if those suggestions were updated:

- On GitLab, navigate to the main page of a repository.
- Navigate to **Settings > Merge Requests**.
- Click **Expand** next to the **Merge Request Approvals** section.
- Configure approval rules and a list of eligible approvers for all protected branches (or all branches).
- Select **Remove all approvals** under "When a commit is added".
- Click **Save changes**

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/approvals/rules.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.5 Ensure there are restrictions on who can dismiss code change reviews (Manual)

Profile Applicability:

- Level 1

Description:

Only trusted users should be allowed to dismiss code change reviews.

Rationale:

Dismissing a code change review permits users to merge new suggested code changes without going through the standard process of approvals. Controlling who can perform this action will prevent malicious actors from simply dismissing the required reviews to code changes and merging malicious or dysfunctional code into the code base.

Impact:

In cases where a code change proposal has been updated since it was last reviewed and the person who reviewed it isn't available for approval, a general collaborator would not be able to merge their code changes until a user with "dismiss review" abilities could dismiss the open review.

Users who are not allowed to dismiss code change reviews will not be permitted to do so, and thus are unable to waive the standard flow of approvals.

Audit:

For each code repository in use, perform the next steps to verify that only trusted users are allowed to dismiss code change reviews: To verify that restrictions are in place around who can dismiss code change reviews, view your branch protection settings for a project. You must have at least the Maintainer role.

1. On the left sidebar, select **Search or go** to and find your project.
2. Select **Settings > Repository**.
3. Expand **Protected branches**.

Remediation:

Prerequisites:

You must have at least the Maintainer role. When granting a group Allowed to merge or Allowed to push and merge permissions on a protected branch, the group must be added to the project. To protect a branch:

1. On the left sidebar, select Search or go to and find your project.
2. Select Settings > Repository.
3. Expand Protected branches.
4. Select Add protected branch.
5. From the Branch dropdown list, select the branch you want to protect.
6. From the Allowed to merge list, select a role that can merge into this branch.
7. From the Allowed to push and merge list, select a role that can push to this branch.

In GitLab Premium and Ultimate, you can also add groups or individual users to Allowed to merge and Allowed to push and merge. Select Protect. The protected branch displays in the list of protected branches.

Default Value:

By default, all users who have write access to the code repository are able to dismiss code change reviews.

References:

1. https://docs.gitlab.com/ee/user/project/protected_branches.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 <u>Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	4.3 <u>Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

1.1.6 Ensure code owners are set for extra sensitive code or configuration (Manual)

Profile Applicability:

- Level 1

Description:

Code owners are trusted users that are responsible for reviewing and managing an important piece of code or configuration. An organization is advised to set code owners for every extremely sensitive code or configuration.

Rationale:

Configuring code owners protects data by verifying that trusted users will notice and review every edit, thus preventing unwanted or malicious changes from potentially compromising sensitive code or configurations.

Impact:

Code owner users will receive notifications for every change that occurs to the code and subsequently added as reviewers of pull requests automatically.

Audit:

In every project, view the Code Owners of a file or directory:

1. On the left sidebar, select Search or go to and find your project.
2. Select Code > Repository.
3. Go to the file or directory you want to see the Code Owners for.
4. Optional. Select a branch or tag.

GitLab shows the Code Owners at the top of the page.

Remediation:

Prerequisite: You must be able to either push to the default branch or create a merge request.

1. Create a CODEOWNERS file in your [preferred location](#).
2. Define some rules in the file following the [Code Owners syntax reference](#). Some suggestions:
 - Configure [All eligible approvers](#) approval rule.
 - [Require Code Owner approval](#) on a protected branch.
3. Commit your changes, and push them up to GitLab.

Default Value:

None

References:

1. <https://docs.gitlab.com/ee/user/project/codeowners/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.8 Define and Maintain Role-Based Access Control Define and maintain role-based access control, through determining and documenting the access rights necessary for each role within the enterprise to successfully carry out its assigned duties. Perform access control reviews of enterprise assets to validate that all privileges are authorized, on a recurring schedule at a minimum annually, or more frequently.			●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

1.1.7 Ensure code owner's review is required when a change affects owned code (Manual)

Profile Applicability:

- Level 1

Description:

Ensure trusted code owners are required to review and approve any code change proposal made to their respective owned areas in the code base.

Rationale:

Configuring code owners ensures that no code, especially code which could prove malicious, will slip into the source code or configuration files of a repository. This allows an organization to mark areas in the code base that are especially sensitive or more prone to an attack. It can also enforce review by specific individuals who are designated as owners to those areas so that they may filter out unauthorized or unwanted changes beforehand.

Impact:

If an organization enforces code owner-based reviews, some code change proposals would not be able to be merged to the codebase before specific, trusted individuals approve them.

Audit:

With merge request approval rules, you can set the minimum number of required approvals by code owners before work can merge into your project.

1. On the left sidebar, select **Search or go to** and find your project.
2. Select **Settings > Repository**.
3. Expand **Protected branches**.
4. Next to the default branch, turn on the toggle under **Code owner approval**.

Remediation:

Prerequisites:

- You must have at least the Maintainer role for the project.
- To add a group as an approver in GitLab.com, you must be a member of the group or the group must be public.

To add a merge request approval rule:

1. On the left sidebar, select **Search or go to** and find your project.
2. Select **Settings > Merge requests**.
3. In the **Merge request approvals** section, in the **Approval rules** section, select **Add approval rule**.
4. Complete the fields:
 - In **Approvals required**, a value of 0 makes [the rule optional](#), and any number greater than 0 creates a required rule. Maximum number of required approvals is 100.
 - From **Add approvers**, select users or groups that are [eligible to approve](#). GitLab suggests approvers based on previous authors of the files changed by the merge request.
5. Select **Add approval rule**. You can add [multiple approval rules](#).

Default Value:

Code owners are not required to review changes by default.

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/approvals/
2. https://docs.gitlab.com/ee/user/project/merge_requests/approvals/rules.html
3. <https://docs.gitlab.com/ee/user/project/codeowners/index.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.8 Ensure inactive branches are periodically reviewed and removed (Manual)

Profile Applicability:

- Level 1

Description:

Keep track of code branches that are inactive for a lengthy period of time and periodically remove them.

Rationale:

Git branches that have been inactive (i.e., no new changes introduced) for a long period of time are enlarging the surface of attack for malicious code injection, sensitive data leaks, and CI pipeline exploitation. They potentially contain outdated dependencies which may leave them highly vulnerable. They are more likely to be improperly managed, and could possibly be accessed by a large number of members of the organization.

Impact:

Removing inactive Git branches means that any code changes they contain would be removed along with them, thus work done in the past might not be accessible after auditing for inactivity.

Audit:

For each project, verify that all existing Git branches are active or have yet to be checked for inactivity by performing the next steps:

- Navigate to the main page of the project.
- In the sidebar select Code > Branches
- Use the navigation at the top of the page to view the Stale branches. The Stale view shows all branches that no one has committed to in the last three months, ordered by the branches with the oldest commits first.
- If the list is empty, you are compliant. If the list is not empty, but there is a valid reason the branches listed are not deleted, you are compliant.

You can perform the next steps to verify that merge request branches are prevented from becoming stale branches by default:

- Navigate to the main page of the project.
- In the left sidebar select Settings > Merge requests
- Verify if 'Enable "Delete source branch" option by default' is selected. If it is not selected, you are more likely to become non-compliant.

Remediation:

For each project, review existing Git branches and remove those which were identified during the audit as being non-compliant by performing the following:

- Navigate to the main page of the project.
- In the sidebar select Code > Branches
- Next to each non-compliant branch select the vertical ellipsis
- Select 'Delete branch'
- Read the warning
- Select 'Yes, delete branch'

You can perform the next steps to reduce the likelihood of a stale branches remaining after a merge request:

- Navigate to the main page of the project.
- In the left sidebar select Settings > Merge requests
- Select 'Enable "Delete source branch" option by default'.

Default Value:

By default, newly opened Git branches would never be removed, regardless of activity or inactivity.

References:

1. <https://docs.gitlab.com/ee/user/project/repository/branches/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

1.1.9 Ensure all checks have passed before merging new code (Manual)

Profile Applicability:

- Level 1

Description:

Before a code change request can be merged to the code base, all predefined checks must successfully pass.

Rationale:

On top of manual reviews of code changes, a code protect should contain a set of prescriptive checks which validate each change. Organizations should enforce those status checks so that changes can only be introduced if all checks have successfully passed. This set of checks should serve as the absolute quality, stability, and security conditions which must be met in order to merge new code to a project.

Impact:

Code changes in which all checks do not pass successfully would not be able to be pushed into the code base of the specific code repository.

Audit:

Within each project's settings, you can see a list of status check services added to the project:

1. In your project, go to **Settings > Merge requests** section.
2. Scroll down to **Status checks**.
3. Ensure that the **Status checks must succeed** checkbox has been selected.

Remediation:

To block the merging of merge requests when checks fail:

1. On the left sidebar, select **Search or go to** and find your project.
2. Select **Settings > Merge requests**.
3. Select the **Status checks must succeed** checkbox.
4. Select **Save changes**.

Default Value:

By default, no checks are defined per project, and thus no enforcement of checks is made.

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/status_checks.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.10 Ensure open Git branches are up to date before they can be merged into code base (Manual)

Profile Applicability:

- Level 1

Description:

Organizations should make sure each suggested code change is in full sync with the existing state of its origin code repository before allowing merging.

Rationale:

Git branches can easily become outdated since the origin code repository is constantly being edited. This means engineers working on separate code branches can accidentally include outdated code with potential security issues which might have already been fixed, overriding the potential solutions for those security issues when merging their own changes.

Impact:

If enforced, outdated branches would not be able to be merged into their origin repository without first being updated to contain any recent changes.

Audit:

For each project, verify that open branches must be updated before merging by performing the following:

- Navigate to the main page of the project
- In the sidebar, select Settings > Merge requests
- Look at the 'Merge method'. If 'Merge commit with semi-linear history' or 'Fast-forward merge' is selected, the project is compliant.

Remediation:

For each project identified as being non-compliant, performing the following:

- Navigate to the main page of the project
- In the sidebar, select Settings > Merge requests
- Under the 'Merge method' select either 'Merge commit with semi-linear history' or 'Fast-forward merge'.

Default Value:

By default, there is no requirement to update a branch before merging it.

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/methods/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.11 Ensure all open comments are resolved before allowing code change merging (Manual)

Profile Applicability:

- Level 2

Description:

Organizations should enforce a "no open comments" policy before allowing code change merging.

Rationale:

In an open code change proposal, reviewers can leave comments containing their questions and suggestions. These comments can also include potential bugs and security issues. Requiring all comments on a code change proposal to be resolved before it can be merged ensures that every concern is properly addressed or acknowledged before the new code changes are introduced to the code base.

Impact:

Code change proposals containing open comments would not be able to be merged into the code base.

Audit:

Ensure that **All threads must be resolved** before changes in a branch can be merged. To review these settings:

1. On the left sidebar, select **Search or go to** and find your project.
2. Select **Settings > Merge requests**.
3. In the **Merge checks** section, check to see that the **All threads must be resolved** checkbox has been selected.

Remediation:

You can prevent merge requests from being merged until all threads are resolved. When this setting is enabled, the **Unresolved threads** counter in a merge request is shown in orange when at least one thread remains unresolved.

1. On the left sidebar, select **Search or go to** and find your project.
2. Select **Settings > Merge requests**.
3. In the **Merge checks** section, select the **All threads must be resolved** checkbox.
4. Select **Save changes**.

Default Value:

By default, code changes with open comments on them are able to be merged into the code base.

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/index.html#prevent-merge-unless-all-threads-are-resolved

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.12 Ensure verification of signed commits for new changes before merging (Manual)

Profile Applicability:

- Level 2

Description:

Ensure every commit in a pull request is signed and verified before merging.

Rationale:

Signing commits, or requiring to sign commits, gives other users confidence about the origin of a specific code change. It ensures that the author of the change is not hidden and is verified by the version control system, thus the change comes from a trusted source.

Impact:

Pull requests with unsigned commits cannot be merged.

Audit:

Identify which projects permit unsigned commits by performing the following steps for each project:

- Navigate to the main page of the project.
- In the sidebar, select Settings > Repository.
- Expand the Push rules section.
- Identify the Select push rules section.
- If 'Reject unsigned commits' is selected the project is compliant.

Remediation:

Ensure only signed commits can be merged for every branch via branch protection rules by performing the following steps for each project:

- Navigate to the main page of the project.
- In the sidebar, select Settings > Repository.
- Expand the Push rules section.
- Under Select push rules select 'Reject unsigned commits'.
- Select 'Save push rules'.

As an administrator you can configure a secure default for new projects by performing the following steps:

- Navigate to the Admin Area.
- In the sidebar, select Push Rules.
- Select 'Reject unsigned commits'.
- Select 'Save push rules'.

References:

1. https://docs.gitlab.com/ee/user/project/repository/push_rules.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.13 Ensure linear history is required (Manual)

Profile Applicability:

- Level 2

Description:

Linear history is the name for Git history where all commits are listed in chronological order, one after another. Such history exists if a pull request is merged either by rebase merge (re-order the commits history) or squash merge (squashes all commits to one). Ensure that linear history is required by requiring the use of rebase or squash merge when merging a pull request.

Rationale:

Enforcing linear history produces a clear record of activity, and as such it offers specific advantages: it is easier to follow, easier to revert a change, and bugs can be found more easily.

Impact:

Pull request cannot be merged except squash or rebase merge.

Audit:

For every project, perform the following steps:

- Navigate to your project
- In the sidebar, select Settings > Repository
- Under 'Merge method', if 'Merge Commit' or 'Merge commit with semi-linear history' is selected then the project is not-compliant.

Remediation:

For every project, perform the following steps:

- Navigate to your project
- In the sidebar, select Settings > Repository
- Under 'Merge method', select 'Fast-forward merge'.

References:

1. https://docs.gitlab.com/ee/user/project/merge_requests/methods/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.14 Ensure branch protection rules are enforced for administrators (Manual)

Profile Applicability:

- Level 1

Description:

Ensure administrators are subject to branch protection rules.

Rationale:

Administrators by default are excluded from any branch protection rules. This means these privileged users (both on the repository and organization levels) are not subject to protections meant to prevent untrusted code insertion, including malicious code. This is extremely important since administrator accounts are often targeted for account hijacking due to their privileged role.

Impact:

Administrator users won't be able to push code directly to the protected branch without being compliant with listed branch protection rules.

Audit:

GitLab administrators can validate this privilege for group owners, enforcing the instance-level protection rule:

1. Select Settings > Repository.
2. On the left sidebar, at the bottom, select Admin Area.
3. Expand the Default branch section.
4. Ensure that the Allow owners to manage default branch protection per group is unchecked.
5. Select Save changes.

Remediation:

GitLab administrators can disable this privilege for group owners, enforcing the instance-level protection rule:

1. On the left sidebar, at the bottom, select Admin Area.
2. Select Settings > Repository.
3. Expand the Default branch section.
4. Uncheck Allow owners to manage default branch protection per group checkbox.
5. Select Save changes.

Default Value:

Administrator accounts are not subject to branch protection rules by default.

References:

1. <https://docs.gitlab.com/ee/user/project/repository/branches/default.html>
2. https://docs.gitlab.com/ee/user/project/protected_branches.html#who-can-modify-a-protected-branch

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.15 Ensure pushing or merging of new code is restricted to specific individuals or teams (Manual)

Profile Applicability:

- Level 2

Description:

Ensure that only trusted users can push or merge new code to protected branches.

Rationale:

Requiring that only trusted users may push or merge new changes reduces the risk of unverified code, especially malicious code, to a protected branch by reducing the number of trusted users who are capable of doing such.

Impact:

Only administrators and trusted users can push or merge to the protected branch.

Audit:

For every code repository in use, ensure only trusted and responsible users can push or merge new code by performing the following:

1. On the left sidebar, select Search or go to and find your project.
2. Select Settings > Repository.
3. Expand Protected branches.
4. If there are no protected branches, the repository is not compliant.
5. For each protected branch, ensure:
 1. The role(s) and user(s) who are 'Allowed to merge' are trusted
 2. The role(s) and user(s) who are 'Allowed to push and merge' are trusted
 3. The 'Allowed to force push' toggle is not selected

Remediation:

Prerequisites: You must have at least the Maintainer role in the group.

For every code repository in use, allow only trusted and responsible users to push or merge new code by performing the following:

1. On the left sidebar, select Search or go to and find your project.
2. Select Settings > Repository.
3. Expand Protected branches.
4. Select Add protected branch.
5. From the Branch dropdown list, select the branch you want to protect.
6. From the Allowed to merge list, select a role that can merge into this branch.
7. From the Allowed to push and merge list, select a role that can push to this branch.

References:

1. https://docs.gitlab.com/ee/administration/merge_requests_approvals.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 <u>Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	4.3 <u>Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

1.1.16 Ensure force push code to branches is denied (Manual)

Profile Applicability:

- Level 1

Description:

The "Force Push" option allows users with "Push" permissions to force their changes directly to the branch without a pull request, and thus should be disabled.

Rationale:

The "Force Push" option allows users to override the existing code with their own code. This can lead to both intentional and unintentional data loss, as well as data infection with malicious code. Disabling the "Force Push" option prohibits users from forcing their changes to the master branch, which ultimately prevents malicious code from entering source code.

Impact:

Users cannot force push to protected branches.

Audit:

For every code repository in use, validate that no one can force push code by performing the following:

- On GitLab, navigate to the main page of the repository.
- Navigate to Settings > Repository .
- Click Expand next to the Protected Branches section.
- Verify that your repository's main (or default) branch is protected.
- Verify that "Allowed to force push" is toggled off.

Remediation:

For each repository in use, block the option to "Force Push" code by performing the following:

- On GitLab, navigate to the main page of the repository.
- Navigate to Settings > Repository .
- Click Expand next to the Protected Branches section.
- Ensure your project's default branch is protected.
- Toggle "Allowed to force push" off.

References:

1. https://docs.gitlab.com/ee/user/project/protected_branches.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.1.17 Ensure branch deletions are denied (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that users with only push access are incapable of deleting a protected branch.

Rationale:

When enabling deletion of a protected branch, any user with at least push access to the repository can delete a branch. This can be potentially dangerous, as a simple human mistake or a hacked account can lead to data loss if a branch is deleted. It is therefore crucial to prevent such incidents by denying protected branch deletion.

Impact:

Protected branches cannot be deleted.

Audit:

For each repository that is being used, verify that protected branches cannot be deleted by performing the following:

View your protected branches by going to the left sidebar and selecting Search or go to and find your project. Select Settings > Repository. Expand Protected branches to view a list of protected branches.

Remediation:

For each repository that is being used, protect a branch in order to block the option to delete branches. To protect a branch for one project:

1. On the left sidebar, select **Search or go to** and find your project.
2. Select **Settings > Repository**.
3. Expand **Protected branches**.
4. Select **Add protected branch**.
5. From the **Branch** dropdown list, select the branch you want to protect.
6. From the **Allowed to merge** list, select a role that can merge into this branch.
7. From the **Allowed to push and merge** list, select a role that can push to this branch.

Group owners can create protected branches at the group level. These settings are inherited by all projects in the group and can't be overridden by project settings. If a specific branch is configured with **Allowed to force push** settings at both the group and project levels, the **Allowed to force push** setting at the *project* level is ignored in favor of the group level setting.

Prerequisites:

- You must have the Owner role in the group.

To protect a branch for all the projects in a group:

1. On the left sidebar, select **Search or go to** and find your group.
2. Select **Settings > Repository**.
3. Expand **Protected branches**.
4. Select **Add protected branch**.
5. In the **Branch** text box, type the branch name or a wildcard. Branch names and wildcards [are case-sensitive](#).
6. From the **Allowed to merge** list, select a role that can merge into this branch.
7. From the **Allowed to push and merge** list, select a role that can push to this branch.
8. Select **Protect**.

References:

1. https://docs.gitlab.com/ee/user/project/protected_branches.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.1.18 Ensure any merging of code is automatically scanned for risks (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that every pull request is required to be scanned for risks.

Rationale:

Scanning pull requests to detect risks allows for early detection of vulnerable code and/or dependencies and helps mitigate potentially malicious code.

Audit:

For each project in use, ensure that every merge request must be scanned for risks by performing the following:

1. On the left sidebar, select **Search or go to** and search for the “go-example-a” project.
2. Go to **Secure > Policies**.
3. Review your list of existing policies. The **Policy Type** column will indicate whether you have enabled a **Scan Execution Policy** for the specified project.
4. Click on the **Name** in order to view the policy details which specifies which scanners run when code is merged.

Remediation:

For each project in use, ensure that every merge request must be scanned for risks by creating a scan execution policy:

1. On the left sidebar, select **Search or go to** and search for the “go-example-a” project.
2. Go to **Secure > Policies**.
3. Select **New policy**.
4. In the **Scan execution policy** section, select **Select policy**.
5. Complete the fields.
 - o **Name:** Enforce secret detection.
 - o **Policy status:** Enabled.
 - o **Actions:** Run a Secret Detection scan.
 - o **Conditions:** Triggers every time a pipeline runs for all branches.
6. Select **Configure with a merge request**. The policy project “go-example-a” security project is created, and a merge request is created.
7. Optional. Review the generated policy YAML in the merge request’s **Changes** tab.
8. Go to the **Overview** tab and select **Merge**.
9. On the left sidebar, select **Search or go to** and search for the “go-example-a” project.
10. Go to **Secure > Policies**.

You now have a scan execution policy that runs a secret detection scan on every MR, for any branch. Test the policy by creating a merge request in project A.

References:

1. https://docs.gitlab.com/ee/user/application_security/index.html#enforce-scan-execution
2. https://docs.gitlab.com/ee/user/application_security/policies/scan-execution-policies.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 Implement Code-Level Security Checks Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	18.7 Apply Static and Dynamic Code Analysis Tools Apply static and dynamic analysis tools to verify that secure coding practices are being adhered to for internally developed software.		●	●

1.1.19 Ensure any changes to branch protection rules are audited (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that changes in the branch protection rules are audited.

Rationale:

Branch protection rules should be configured on every repository. The only users who may change such rules are administrators. In a case of an attack on an administrator account or of human error on the part of an administrator, protection rules could be disabled, and thus decrease source code confidentiality as a result. It is important to track and audit such changes to prevent potential incidents as soon as possible.

Audit:

Ensure that changes in the branch protection rules are audited regularly. You can view audit events from user actions across an entire GitLab instance. To view instance audit events:

1. On the left sidebar, select Search or go to.
2. Select Admin Area.
3. On the left sidebar, select Monitoring > Audit Events.
4. Filter by the following: Event Type protected_branch_updated. This event type is triggered when the setting for protected branches is updated.
5. Ensure every action is reasonable and secure and is investigated if not.

Remediation:

Use the audit log to audit changes in branch protection rules by performing the following:

1. On the left sidebar, select Search or go to.
2. Select Admin Area.
3. On the left sidebar, select Monitoring > Audit Events.
4. Filter by the following: Event Type protected_branch_updated
5. Ensure every action is reasonable and secure and is investigated if not.

References:

1. https://docs.gitlab.com/ee/administration/audit_event_streaming/audit_event_types.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

1.1.20 Ensure branch protection is enforced on the default branch (Manual)

Profile Applicability:

- Level 1

Description:

Enforce branch protection on the default and main branch.

Rationale:

The default or main branch of repositories is considered very important, as it is eventually gets deployed to the production. Therefore it needs protection. By enforcing branch protection rules on this branch, it is secured from unwanted or unauthorized changes. It can also be protected from untested and unreviewed changes and more.

Audit:

The default branch of GitLab repositories are protected by default. This setting can be overridden at the instance, group, or project level.

To verify that branch protection is enabled for the main or default branch at the project level:

1. Navigate to the main page of the GitLab repository.
2. Select **Settings > Repository**.
3. Expand **Protected branches**.
4. Verify that initial default branch protection is applied to the "main" or default branch.

GitLab Group Owners can also perform the following to ensure branch protection is enabled by default for new projects at the group level:

1. Navigate to the main page for your GitLab group.
2. Select **Settings > Repository**.
3. Under **Default branch**, verify that initial default branch protection is applied to the "main" or default branch.

GitLab administrators can perform the following to ensure branch protection is enabled by default for the main or default branch of all new projects at the instance level (self-managed GitLab only):

1. Navigate to **Admin Area**.
2. Select **Settings > Repository**.
3. Under **Default branch**, verify that initial default branch protection is applied to the "main" or default branch.

Remediation:

Perform the following to enforce branch protection on the main or default branch at the project level:

1. Navigate to your project page.
2. Select **Settings > Repository**.
3. Expand **Protected branches**.
4. Select **Add protected branch**.
5. From the **Branch** dropdown list, select the project's main or default branch.
6. Choose the roles who should be **Allowed to merge** and **Allowed to push and merge** for this protected default branch.
7. Select **Protect**.

Perform the following to enforce branch protection on the main or default branch of new projects at the group level:

1. Navigate to the main page for your GitLab group.
2. Select **Settings > Repository**.
3. Expand **Default branch**
4. Enable initial default branch protection for the "main" or default branch of new repositories created in the group.
5. Select **Save changes**.

Perform the following to enforce branch protection on the main branch of new projects at the instance level (self-managed GitLab administrators only):

1. Navigate to **Admin Area**.
2. Select **Settings > Repository**.
3. Expand **Default branch**.
4. Enable initial default branch protection for the "main" or default branch of new repositories created on this GitLab instance.
5. Select **Save changes**.

References:

1. https://docs.gitlab.com/ee/api/protected_branches.html
2. https://docs.gitlab.com/ee/api/group_protected_branches.html (PREMIUM, SaaS only)
3. <https://docs.gitlab.com/ee/api/settings.html> (Instance-level Admin Only)

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.2 Repository Management

This section consists of security recommendations for proper code repository management.

Code repositories are where the application code is stored and organized. It is important to keep code repositories organized and maintained to avoid data loss, data theft and other attacks that may happen unknowingly when a repository is not maintained well. The recommendations of this section are setting guides to do so.

1.2.1 Ensure all public repositories contain a SECURITY.md file (Manual)

Profile Applicability:

- Level 1

Description:

A SECURITY.md file is a security policy file that offers instruction on reporting security vulnerabilities in a project. When someone creates an issue within a specific project, a link to the SECURITY.md file will subsequently be shown.

Rationale:

A SECURITY.md file provides users with crucial security information. It can also serve an important role in project maintenance, encouraging users to think ahead about how to properly handle potential security issues, updates, and general security practices.

Audit:

Verify that each public repository has a SECURITY.md file by performing the following:

- Navigate to the main page of a GitLab repository.
- Verify that the repository has a SECURITY.md file with crucial security information at the top-level.

Remediation:

Ensure that each public repository has a SECURITY.md file by performing the following:

- Navigate to the main page of a repository without a SECURITY.md file in GitLab.
- Create a SECURITY.md file (in the web UI or locally) with security information like supported versions of your project and how to report a vulnerability.
- Commit this file to the repository and push your changes.
- If you open a merge request to add the SECURITY.md file, make sure this change is merged to your repository's default branch.

References:

1. <https://docs.gitlab.com/ee/api/projects.html>
2. https://docs.gitlab.com/ee/api/repository_files.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.2 Establish and Maintain a Process to Accept and Address Software Vulnerabilities</u> Establish and maintain a process to accept and address reports of software vulnerabilities, including providing a means for external entities to report. The process is to include such items as: a vulnerability handling policy that identifies reporting process, responsible party for handling vulnerability reports, and a process for intake, assignment, remediation, and remediation testing. As part of the process, use a vulnerability tracking system that includes severity ratings, and metrics for measuring timing for identification, analysis, and remediation of vulnerabilities. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard. Third-party application developers need to consider this an externally-facing policy that helps to set expectations for outside stakeholders.			
v7	<u>18.8 Establish a Process to Accept and Address Reports of Software Vulnerabilities</u> Establish a process to accept and address reports of software vulnerabilities, including providing a means for external entities to contact your security group.			

1.2.2 Ensure repository creation is limited to specific members (Manual)

Profile Applicability:

- Level 1

Description:

Limit the ability to create repositories to trusted users and teams.

Rationale:

Restricting repository creation to trusted users and teams is recommended in order to keep the organization properly structured, track fewer items, prevent impersonation, and to not overload the version-control system. It will allow administrators easier source code tracking and management capabilities, as they will have fewer repositories to track. The process of detecting potential attacks also becomes far more straightforward, as well, since the easier it is to track the source code, the easier it is to detect malicious acts within it. Additionally, the possibility of a member creating a public repository and sharing the organization's data externally is significantly decreased.

Impact:

Specific users will not be permitted to create repositories.

Audit:

Verify that only trusted users and teams can create repositories by performing the following. As an administrator:

- Navigate to the Admin Area
- In the sidebar, select Settings > General
- Expand the Sign-up restrictions section
- If 'Sign up enabled' is disabled, the instance is compliant.
- If 'Sign-up enabled' and 'Require admin approval for new sign-ups' are selected, and if 'Email confirmation settings' is set to Hard, the instance is compliant.

Remediation:

Ensure that only trusted users and teams can create repositories by performing the following. As an administrator:

- Navigate to the Admin Area
- In the sidebar, select Settings > General
- Expand the Sign-up restrictions section
- (Option 1) Deselect 'Sign-up enabled', OR
- (Option 2) Select 'Sign-up enabled', select 'Require admin approval for new sign-ups' are selected, and under 'Email confirmation settings' select 'Hard'
- Select 'Save changes'

References:

1. https://docs.gitlab.com/ee/administration/settings/sign_up_restrictions.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.2.3 Ensure repository deletion is limited to specific users (Manual)

Profile Applicability:

- Level 1

Description:

Ensure only a limited number of trusted users can delete repositories.

Rationale:

Restricting the ability to delete repositories protects the organization from intentional and unintentional data loss. This ensures that users cannot delete repositories or cause other potential damage — whether by accident or due to their account being hacked — unless they have the correct privileges.

Impact:

Certain users will not be permitted to delete repositories.

Audit:

Verify that only a limited number of trusted users can delete repositories by performing either of the following steps:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Owner/Maintainer role in the list, you are compliant.

Remediation:

Enforce repository deletion by a few trusted and responsible users only by performing either of the following steps:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member.

Default Value:

Only organization owners or members with admin privileges can delete repositories.

References:

1. <https://docs.gitlab.com/ee/user/permissions.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.2.4 Ensure issue deletion is limited to specific users (Manual)

Profile Applicability:

- Level 1

Description:

Ensure only trusted and responsible users can delete issues.

Rationale:

Issues are a way to keep track of things happening in repositories, such as setting new milestones or requesting urgent fixes. Deleting an issue is not a benign activity, as it might harm the development workflow or attempt to hide malicious behavior. Because of this, it should be restricted and allowed only by trusted and responsible users.

Impact:

Certain users will not be permitted to delete issues.

Audit:

Verify that only a limited number of trusted users can delete issues by performing either of the following steps:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Owner/Maintainer role in the list, you are compliant.

Remediation:

Enforce issue deletion by a few trusted and responsible users only by performing either of the following steps:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member.

Default Value:

Only organization owners or members with admin privileges can delete issues.

References:

1. <https://docs.gitlab.com/ee/user/permissions.html#project-features-permissions>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.2.5 Ensure all copies (forks) of code are tracked and accounted for (Manual)

Profile Applicability:

- Level 1

Description:

Track every fork of code and ensure it is accounted for.

Rationale:

A fork is a copy of a repository. On top of being a plain copy, any updates to the original repository itself can be pulled and reflected by the fork under certain conditions. A large number of repository copies (forks) become difficult to manage and properly secure. New and sensitive changes can often be pushed into a critical repository without developer knowledge of an updated copy of the very same repository. If there is no limit on doing this, then it is recommended to track and delete copies of organization repositories as needed.

Impact:

Disabling forks completely may slow down the development process as more actions will be necessary to take in order to fork a repository.

Audit:

Verify that the following steps are done regularly to track and examine forks.

- Navigate to the project home page.
- Find the 'Fork' button, and select the number next to it.
- Examine the forks listed there.

Remediation:

Track forks and examine them by performing the following on a regular basis:

- Navigate to the project home page.
- Find the 'Fork' button, and select the number next to it.
- Examine the forks listed there.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>2.1 Establish and Maintain a Software Inventory</u> Establish and maintain a detailed inventory of all licensed software installed on enterprise assets. The software inventory must document the title, publisher, initial install/use date, and business purpose for each entry; where appropriate, include the Uniform Resource Locator (URL), app store(s), version(s), deployment mechanism, and decommission date. Review and update the software inventory bi-annually, or more frequently.			
v8	<u>3.14 Log Sensitive Data Access</u> Log sensitive data access, including modification and disposal.			
v7	<u>2.4 Track Software Inventory Information</u> The software inventory system should track the name, version, publisher, and install date for all software, including operating systems authorized by the organization.			
v7	<u>14.9 Enforce Detail Logging for Access or Changes to Sensitive Data</u> Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).			

1.2.6 Ensure all code projects are tracked for changes in visibility status (Manual)

Profile Applicability:

- Level 1

Description:

Ensure every change in visibility of projects is tracked.

Rationale:

Visibility of projects determines who can access a project and/or fork it: anyone, designated users, or only members of the organization. If a private project becomes public, this may point to a potential attack, which can ultimately lead to data loss, the leaking of sensitive information, and finally to a supply chain attack. It is crucial to track these changes in order to prevent such incidents.

Audit:

Ensure that every change in project visibility is investigated by performing the following regularly. As an administrator:

- Navigate to the Admin Area.
- In the sidebar, select Monitoring > Audit Events.
- Review the log for Actions with the content 'Changed visibility from Private to Public' or 'Changed visibility from Internal to Public'.
- Ensure every change is reasonable and secure and is investigated if it is not.

Remediation:

Ensure that every change in project visibility is investigated by performing the following regularly. As an administrator:

- Navigate to the Admin Area.
- In the sidebar, select Monitoring > Audit Events.
- Review the log for Actions with the content 'Changed visibility from Private to Public' or 'Changed visibility from Internal to Public'.
- Ensure every change is reasonable and secure and is investigated if it is not.
- (Optional) Use Instance Audit Event Streaming (https://docs.gitlab.com/ee/administration/audit_event_streaming/#instance-streaming-destinations) to send visibility change events to a third party alerting tool. Integrate these alerts in to your change management and/or incident response processes.

References:

1. https://docs.gitlab.com/ee/administration/audit_event_streaming/audit_event_types.html#groups-and-projects

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.1 <u>Establish and Maintain a Software Inventory</u> Establish and maintain a detailed inventory of all licensed software installed on enterprise assets. The software inventory must document the title, publisher, initial install/use date, and business purpose for each entry; where appropriate, include the Uniform Resource Locator (URL), app store(s), version(s), deployment mechanism, and decommission date. Review and update the software inventory bi-annually, or more frequently.	●	●	●
v8	3.14 <u>Log Sensitive Data Access</u> Log sensitive data access, including modification and disposal.			●
v7	2.4 <u>Track Software Inventory Information</u> The software inventory system should track the name, version, publisher, and install date for all software, including operating systems authorized by the organization.		●	●
v7	14.9 <u>Enforce Detail Logging for Access or Changes to Sensitive Data</u> Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).			●

1.2.7 Ensure inactive repositories are reviewed and archived periodically (Manual)

Profile Applicability:

- Level 1

Description:

Track inactive repositories and remove them periodically.

Rationale:

Inactive repositories (i.e., no new changes introduced for a long period of time) can enlarge the surface of a potential attack or data leak. These repositories are more likely to be improperly managed, and thus could possibly be accessed by a large number of users in an organization.

Impact:

Bug fixes and deployment of necessary changes could prove complicated for archived repositories.

Audit:

Perform the following to ensure that all the projects in the organization are active. For each group:

- Navigate to the group homepage.
- Expand any sub-groups.
- For each project, review the updated date and ensure it has been updated within the last 6 months.

Remediation:

Perform the following to remediate the presence of inactive projects. For each inactive project identified during the audit:

- Navigate to the project homepage.
- In the sidebar, select Settings > General.
- In the 'Advanced' section, select 'Expand'.
- Select 'Archive project'.
- Read the warning.
- Select 'Archive project'.

To automate the deletion of inactive projects, perform the following steps as an Administrator:

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > Repository.
- Expand Repository maintenance.
- In the Inactive project deletion section, select Delete inactive projects.
- Configure the settings.
 - The warning email is sent to users who have the Owner and Maintainer role for the inactive project.
 - The email duration must be less than the Delete project after duration.
- Select Save changes.

References:

1. https://docs.gitlab.com/ee/administration/inactive_project_deletion.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.		●	●
v7	<u>13.2 Remove Sensitive Data or Systems Not Regularly Accessed by Organization</u> Remove sensitive data or systems not regularly accessed by the organization from the network. These systems shall only be used as stand alone systems (disconnected from the network) by the business unit needing to occasionally use the system or completely virtualized and powered off until needed.	●	●	●

1.3 Contribution Access

This section consists of security recommendations for managing access to the application code. This includes managing both internal and external access, administrator accounts, permissions, identification methods, etc. Securing these items is important for software safety because every security constraint on access is an obstacle in the way of attacks.

This section differentiates between the common user account and an admin account. It is important to understand that due to the high permissions of the admin account, it should be used only for administrative work and not for everyday tasks.

1.3.1 Ensure inactive users are reviewed and removed periodically (Manual)

Profile Applicability:

- Level 2

Description:

Track inactive user accounts and periodically remove them.

Rationale:

User accounts that have been inactive for a long period of time are enlarging the surface of attack. Inactive users with high-level privileges are of particular concern, as these accounts are more likely to be targets for attackers. This could potentially allow access to large portions of an organization should such an attack prove successful. It is recommended to remove them as soon as possible in order to prevent this.

Audit:

As an Administrator:

- Navigate to the Admin Area
- In the sidebar, select Users
- Select the 'Deactivated' filter tab
- Identify if any users are present
- Select the 'Active' filter tab
- Sort by 'Oldest updated'
- Use the 'Last activity' column to determine which users are inactive

Remediation:

As an Administrator:

- Navigate to the Admin Area
- In the sidebar, select Users
- Next to each inactive user, select the vertical ellipsis
- Select either 'Block' (recommended), 'Delete user', or 'Delete user and contributions'

Perform the following steps as an Administrator to automatically deactivate dormant users:

- Navigate to the Admin Area
- Select Settings > General.
- Expand the Account and limit section.
- Under Dormant users, check Deactivate dormant users after a period of inactivity.
- Under Days of inactivity before deactivation, enter the number of days before deactivation. Minimum value is 90 days.
- Select Save changes.

References:

1. https://docs.gitlab.com/ee/administration/moderate_users.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.3 <u>Disable Dormant Accounts</u> Delete or disable any dormant accounts after a period of 45 days of inactivity, where supported.			
v7	16.9 <u>Disable Dormant Accounts</u> Automatically disable dormant accounts after a set period of inactivity.			

1.3.2 Ensure top-level group creation is limited to specific members (Manual)

Profile Applicability:

- Level 1

Description:

Limit ability to create teams to trusted and specific users.

Rationale:

The ability to create new teams should be restricted to specific members in order to keep the organization orderly and ensure users have access to only the lowest privilege level necessary. Teams typically inherit permissions from their parent team, thus if base permissions are less restricted and any user has the ability to create a team, a permission leverage could occur in which certain data is made available to users who should not have access to it. Such a situation could potentially lead to the creation of shadow teams by an attacker. Restricting team creation will also reduce additional clutter in the organizational structure, and as a result will make it easier to track changes and anomalies.

Impact:

Only specific users will be able to create new teams.

Audit:

For every organization, ensure that top-level group creation is limited to specific, trusted users by performing the following:

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > General.
- Expand Account and limit.
- Verify that the Allow new users to create top-level groups checkbox is not checked.

Remediation:

For every organization, limit top-level group creation to specific, trusted users by performing the following:

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > General.
- Expand Account and limit.
- Clear the Allow new users to create top-level groups checkbox.

References:

1. https://docs.gitlab.com/ee/administration/settings/account_and_limit_settings.html#prevent-new-users-from-creating-top-level-groups

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

1.3.3 Ensure minimum number of administrators are set for the organization (Manual)

Profile Applicability:

- Level 1

Description:

Ensure the organization has a minimum number of administrators.

Rationale:

Organization administrators have the highest level of permissions, including the ability to add/remove collaborators, create or delete repositories, change branch protection policy, and convert to a publicly-accessible repository. Due to the permissive access granted to an organization administrator, it is highly recommended to keep the number of administrator accounts as minimal as possible.

Audit:

Verify the minimum number of administrators in your project by performing the following:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Owner/Maintainer role in the list, you are compliant.

Remediation:

Set the minimum number of administrators in your project by performing the following:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member.

References:

1. <https://docs.gitlab.com/ee/user/project/members/#filter-and-sort-project-members>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

1.3.4 Ensure Multi-Factor Authentication (MFA) is required for contributors of new code (Manual)

Profile Applicability:

- Level 2

Description:

Require collaborators from outside the organization to use Multi-Factor Authentication (MFA) in addition to a standard user name and password when authenticating to the source code management platform.

Rationale:

By default every user authenticates within the system by password only. If the password of a user is compromised, however, the user account and every repository to which they have access are in danger of data loss, malicious code commits, and data theft. It is therefore recommended that each user has Multi-Factor Authentication enabled. This adds an additional layer of protection to ensure the account remains secure even if the user's password is compromised.

Impact:

A member without enabled Multi-Factor Authentication cannot contribute to the project. They must enable Multi-Factor Authentication before they can contribute any code.

Audit:

For your top-level group, verify that Multi-Factor Authentication is enforced for contributors and is the only way to authenticate, by doing the following:

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > General.
- Expand Sign-in restrictions:
- Check if Enforce two-factor authentication is enabled. If so, you are compliant.

Remediation:

For your top-level group, enforce that Multi-Factor Authentication is enforced for contributors and is the only way to authenticate, by doing the following:

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > General.
- Expand Sign-in restrictions:
- Select Enforce two-factor authentication to enable this feature.

References:

1. https://docs.gitlab.com/ee/security/two_factor_authentication.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.3 Require MFA for Externally-Exposed Applications Require all externally-exposed enterprise or third-party applications to enforce MFA, where supported. Enforcing MFA through a directory service or SSO provider is a satisfactory implementation of this Safeguard.		●	●
v8	6.4 Require MFA for Remote Network Access Require MFA for remote network access.	●	●	●
v8	6.5 Require MFA for Administrative Access Require MFA for all administrative access accounts, where supported, on all enterprise assets, whether managed on-site or through a third-party provider.	●	●	●
v7	16.3 Require Multi-factor Authentication Require multi-factor authentication for all user accounts, on all systems, whether managed onsite or by a third-party provider.		●	●

1.3.5 Ensure the organization is requiring members to use Multi-Factor Authentication (MFA) (Manual)

Profile Applicability:

- Level 2

Description:

Require members of the organization to use Multi-Factor Authentication (MFA) in addition to a standard user name and password when authenticating to the source code management platform.

Rationale:

By default every user authenticates within the system by password only. If the password of a user is compromised, however, the user account and every repository to which they have access are in danger of data loss, malicious code commits, and data theft. It is therefore recommended that each user has Multi-Factor Authentication enabled. This adds an additional layer of protection to ensure the account remains secure even if the user's password is compromised.

Impact:

Members will be removed from the organization if they don't have Multi-Factor Authentication already enabled. If this is the case, it is recommended that an invitation be sent to reinstate the user's access and former privileges. They must enable Multi-Factor Authentication to accept the invitation.

Audit:

For every organization that exists in your GitLab platform, verify that Two-Factor Authentication is enforced and is the only way to authenticate. Administrators can enforce 2FA for all users in two different ways:

- Enforce on next sign in.
- Suggest on next sign in, but allow a grace period before enforcing.

Use the UI:

1. On the left sidebar, select **Search or go to**.
2. Select **Admin Area**.
3. On the left sidebar, select **Settings > General**.
4. Expand the **Sign-in restrictions** section:
5. Verify that **Enforce two-factor authentication** is enabled.

Remediation:

Use the UI:

1. On the left sidebar, select **Search or go to**.
2. Select **Admin Area**.
3. On the left sidebar, select **Settings > General**.
4. Expand the **Sign-in restrictions** section:
 - Select **Enforce two-factor authentication** to enable this feature.
 - In Two-factor grace period, enter a number of hours. If you want to enforce 2FA on next sign-in attempt, enter 0.

Use the API:

Use the [application settings API](#) to modify the following settings:

- `require_two_factor_authentication`, set to TRUE.

References:

1. https://docs.gitlab.com/ee/security/two_factor_authentication.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.3 <u>Require MFA for Externally-Exposed Applications</u> Require all externally-exposed enterprise or third-party applications to enforce MFA, where supported. Enforcing MFA through a directory service or SSO provider is a satisfactory implementation of this Safeguard.		●	●
v8	6.4 <u>Require MFA for Remote Network Access</u> Require MFA for remote network access.	●	●	●
v8	6.5 <u>Require MFA for Administrative Access</u> Require MFA for all administrative access accounts, where supported, on all enterprise assets, whether managed on-site or through a third-party provider.	●	●	●
v7	16.3 <u>Require Multi-factor Authentication</u> Require multi-factor authentication for all user accounts, on all systems, whether managed onsite or by a third-party provider.		●	●

1.3.6 Ensure new members are required to be invited using company-approved email (Manual)

Profile Applicability:

- Level 2

Description:

Existing members of an organization can invite new members to join, however new members must only be invited with their company-approved email.

Rationale:

Ensuring new members of an organization have company-approved email prevents existing members of the organization from inviting arbitrary new users to join. Without this verification, they can invite anyone who is using the organization's version control system or has an active email account, thus allowing outside users (and potential threat actors) to easily gain access to company private code and resources. This practice will subsequently reduce the chance of human error or typos when inviting a new member.

Impact:

Existing members would not be able to invite new users who do not have a company-approved email address.

Audit:

For each group in use, verify for every invitation that the invited email address is company-approved by performing the following:

On the left sidebar, select Search or go to and find your group.

Select Manage > Members.

Members that are not automatically added are displayed on the Invited tab.

Verify that each invitation email is company approved by your company.

Remediation:

For each group, allow only users with company-approved email to be invited. If a user was invited without company-approved email, perform the following:

On the left sidebar, select Search or go to and find your group.

Select Manage > Members.

Members that are not automatically added are displayed on the Invited tab.

Verify that each invitation email is company approved by your company.

To cancel the user's invitation to join your organization, click Cancel invitation.

References:

1. <https://docs.gitlab.com/ee/user/group/#add-users-to-a-group>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.1 <u>Establish an Access Granting Process</u> Establish and follow a process, preferably automated, for granting access to enterprise assets upon new hire, rights grant, or role change of a user.			
v7	14.7 <u>Enforce Access Control to Data through Automated Tools</u> Use an automated tool, such as host-based Data Loss Prevention, to enforce access controls to data even when data is copied off a system.			

1.3.7 Ensure two administrators are set for each repository (Manual)

Profile Applicability:

- Level 2

Description:

Ensure every repository has two users with administrative permissions.

Rationale:

Repository administrators have the highest permissions to said repository. These include the ability to add/remove collaborators, change branch protection policy, and convert to a publicly-accessible repository. Due to the liberal access granted to a repository administrator, it is highly recommended that only two contributors occupy this role.

Impact:

Removing administrative users from a repository would result in them losing high-level access to that repository.

Audit:

For every group, verify there are two administrators by performing the following:

- On the left sidebar, at the bottom, select Admin Area.
- Select Overview > Users.
- List users selecting the Admin tab.
- Verify that there are only 2 members with Admin permission.

Remediation:

For every group in use, set two administrators by performing the following:

- On the left sidebar, at the bottom, select Admin Area.
- Select Overview > Users.
- List users selecting the Admin tab.
- Find the team or person whose you'd like to revoke admin permissions. To edit a user, in the user's row, select Edit.

References:

1. https://docs.gitlab.com/ee/administration/admin_area.html#administering-users

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.3.8 Ensure strict base permissions are set for repositories (Manual)

Profile Applicability:

- Level 1

Description:

Base permissions define the permission level automatically granted to all organization members. Define strict base access permissions for all of the repositories in the organization, including new ones.

Rationale:

Defining strict base permissions is the best practice in every role-based access control (RBAC) system. If the base permission is high — for example, "write" permission — every member of the organization will have "write" permission to every repository in the organization. This will apply regardless of the specific permissions a user might need, which generally differ between organization repositories. The higher the permission, the higher the risk for incidents such as bad code commit or data breach. It is therefore recommended to set the base permissions to the strictest level possible.

Impact:

Users might not be able to access organization repositories or perform some acts as commits. These specific permissions should be granted individually for each user or team, as needed.

Audit:

Verify that strict base permissions are set for the organization groups by doing the following:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
In GitLab, you set the specific role for each new user in your group. Check the roles of your users to determine if it matches the least-privilege principle.

Remediation:

Set strict base permissions for the organization groups with the next steps:

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
In GitLab, you set the specific role for each new user in your group. Ensure the roles for your users match the least-privilege principle.

Default Value:

Read permission

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.7 Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			
v7	<u>18.11 Use Standard Hardening Configuration Templates for Databases</u> For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.			

1.3.9 Ensure an organization's identity is confirmed with a "Verified" badge (Manual)

Profile Applicability:

- Level 2

Description:

Confirm the domains an organization owns with a "Verified" badge.

Rationale:

Verifying the organization's domain gives developers assurance that a given domain is truly the official home for a public organization. Attackers can pretend to be an organization and steal information via a faked/spoof domain, therefore the use of a "Verified" badge instills more confidence and trust between developers and the open-source community.

Audit:

On the left sidebar, select Search or go to and find your top-level group.

Select Settings > Domain Verification.

View your domains and if they are verified or unverified.

Remediation:

Step 1:

- On the left sidebar, select Search or go to and find your top-level group.
- Select Settings > Domain Verification.
- In the upper-right corner, select Add Domain.
- In Domain, enter the domain name.
- In Project, link to a project.
- In Certificate:
 - If you do not have or do not want to use an SSL certificate, leave Automatic certificate management using Let's Encrypt selected.
 - Optional. Turn on the Manually enter certificate information toggle to add an SSL/TLS certificate. You can also add the certificate and key later.
- Select Add Domain.

Step 2:

After you create a new domain, the verification code prompts you. Copy the values from GitLab and paste them in your domain's control panel as a TXT record.

Step 3:

After you have added all the DNS records:

- On the left sidebar, select Search or go to and find your group.
- Select Settings > Domain Verification.
- On the domain table row, Select Retry verification ().

References:

1. https://docs.gitlab.com/ee/user/enterprise_user/#verified-domains-for-groups

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

1.3.10 Ensure Source Code Management (SCM) email notifications are restricted to verified domains (Manual)

Profile Applicability:

- Level 2

Description:

Restrict the Source Code Management (SCM) organization's email notifications to approved domains only.

Rationale:

Restricting Source Code Management email notifications to verified domains only prevents data leaks, as personal emails and custom domains are more prone to account takeover via DNS hijacking or password breach.

Impact:

Only members with approved email would be able to receive Source Code Management notifications.

Audit:

Ensure Source Code Management email notifications are restricted to approved domains only by performing the following:

- On the left sidebar, select Search or go to and find your top-level group.
- Select Settings > Domain Verification.
- When viewing Domain Verification, select the project listed next to the relevant domain.
- Check if access is limited to the relevant domains.

Remediation:

Restrict Source Code Management email notifications to approved domains only by performing the following:

- On the left sidebar, select Search or go to and find your top-level group.
- Select Settings > Domain Verification.
- When viewing Domain Verification, select the project listed next to the relevant domain.
- Ensure access is limited to the relevant domains.

References:

1. https://docs.gitlab.com/ee/user/enterprise_user/#verified-domains-for-groups
2. https://docs.gitlab.com/ee/user/group/access_and_permissions.html#restrict-group-access-by-domain

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

1.3.11 Ensure an organization provides SSH certificates (Manual)

Profile Applicability:

- Level 2

Description:

As an organization, become an SSH Certificate Authority and provide SSH keys for accessing repositories.

Rationale:

There are two ways for remotely working with Source Code Management: via HTTPS, which requires authentication by user/password, or via SSH, which requires the use of SSH keys. SSH authentication is better in terms of security; key creation and distribution, however, must be done in a secure manner. This can be accomplished by implementing SSH certificates, which are used to validate the server's identity. A developer will not be able to connect to a Git server if its key cannot be verified by the SSH Certificate Authority (CA) server. As an organization, one can verify the SSH certificate signature used to authenticate if a CA is defined and used. This ensures that only verified developers can access organization repositories, as their SSH key will be the only one signed by the CA certificate. This reduces the risk of misuse and malicious code commits.

Impact:

Members with unverified keys will not be able to clone organization repositories. Signing, certification, and verification might also slow down the development process.

Audit:

GitLab allows you to restrict the allowed SSH key technology as well as specify the minimum key length for each technology:

1. On the left sidebar, at the bottom, select Admin Area.
2. Select Settings > General .
3. Expand Visibility and access controls
4. If a restriction is imposed on any key type, users cannot upload new SSH keys that don't meet the requirement. Any existing keys that don't meet it are disabled but not removed and users cannot pull or push code using them.

Remediation:

If you do not have an existing SSH key pair, generate a new one:

1. Open a terminal.
2. Run `ssh-keygen -t` followed by the key type and an optional comment. This comment is included in the `.pub` file that's created.
3. Press Enter.
4. Accept the suggested filename and directory, unless you are generating a deploy key or want to save in a specific directory where you store other keys.
5. Specify a passphrase
6. A confirmation is displayed, including information about where your files are stored. A public and private key are generated.
7. Add the public SSH key to your GitLab account and keep the private key secure.

References:

1. <https://docs.gitlab.com/ee/user/ssh.html#generate-an-ssh-key-pair>
2. https://docs.gitlab.com/ee/security/ssh_keys_restrictions.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	12.5 Centralize Network Authentication, Authorization, and Auditing (AAA) Centralize network AAA.			
v8	12.7 Ensure Remote Devices Utilize a VPN and are Connecting to an Enterprise's AAA Infrastructure Require users to authenticate to enterprise-managed VPN and authentication services prior to accessing enterprise resources on end-user devices.			
v7	1.8 Utilize Client Certificates to Authenticate Hardware Assets Use client certificates to authenticate hardware assets connecting to the organization's trusted network.			

1.3.12 Ensure Git access is limited based on IP addresses (Manual)

Profile Applicability:

- Level 2

Description:

Limit Git access based on IP addresses by having a allowlist of IP addresses from which connection is possible.

Rationale:

Allowing access to Git repositories (source code) only from specific IP addresses adds yet another layer of restriction and reduces the risk of unauthorized connection to the organization's assets. This will prevent attackers from accessing Source Code Management (SCM), as they would first need to know the allowed IP addresses to gain access to them.

Impact:

Only members with allowlisted IP addresses will be able to access the organization's Git repositories.

Audit:

To ensure only people from your organization can access particular resources, you can restrict access to groups by IP address. This top-level group setting applies to:

- The GitLab UI, including subgroups, projects, and issues. It does not apply to GitLab Pages. [In GitLab 12.3 and later](#), the API.
- In self-managed installations of GitLab 15.1 and later, you can also configure [globally-allowed IP address ranges](#) at the group level.

To determine whether IP restrictions are in place:

- On the left sidebar, select **Search or go** to and find your group.
- Select **Settings > General**.
- Expand the **Permissions and group features** section.
- In the **Restrict access by IP address** text box, view the list of IP addresses.

Remediation:

To restrict group access by IP address:

1. On the left sidebar, select **Search or go to** and find your group.
2. Select **Settings > General**.
3. Expand the **Permissions and group features** section.
4. In the **Restrict access by IP address** text box, enter a list of IPv4 or IPv6 address ranges in CIDR notation. This list:
 - Has no limit on the number of IP address ranges.
 - Has a size limit of 1 GB.
 - Applies to both SSH or HTTP authorized IP address ranges. You cannot split this list by type of authorization.
5. Select **Save changes**.

References:

1. https://docs.gitlab.com/ee/user/group/access_and_permissions.html#restrict-group-access-by-ip-address

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 Allowlist Authorized Software Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.		●	●
v8	2.6 Allowlist Authorized Libraries Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.		●	●
v7	2.7 Utilize Application Whitelisting Utilize application whitelisting technology on all assets to ensure that only authorized software executes and all unauthorized software is blocked from executing on assets.			●
v7	2.8 Implement Application Whitelisting of Libraries The organization's application whitelisting software must ensure that only authorized software libraries (such as *.dll, *.ocx, *.so, etc) are allowed to load into a system process.			●

1.3.13 Ensure anomalous code behavior is tracked (Manual)

Profile Applicability:

- Level 1

Description:

Track code anomalies.

Rationale:

Carefully analyze any code anomalies within the organization. For example, a code anomaly could be a push made outside of working hours. Such a code push has a higher likelihood of being the result of an attack, as most if not all members of the organization would likely be outside the office. Another example is an activity that exceeds the average activity of a particular user. Tracking and auditing such behaviors creates additional layers of security and can aid in early detection of potential attacks.

Audit:

For every project in use, ensure code anomalies relevant to the organization are promptly investigated.

Remediation:

For every project in use, track and investigate anomalous code behavior and activity.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 Implement Code-Level Security Checks Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	18.7 Apply Static and Dynamic Code Analysis Tools Apply static and dynamic analysis tools to verify that secure coding practices are being adhered to for internally developed software.		●	●

1.4 Third-Party

This section consists of security recommendations for using third-party applications in the code repositories.

Applications are typically automated integrations that improve the workflow of an organization, for example, OAuth applications. Those applications are written by third-party developers and therefore should be reviewed carefully before use. It is important to monitor their use and permissions because unused applications or unnecessary high permissions can enlarge the attack surface.

1.4.1 Ensure administrator approval is required for every installed application (Manual)

Profile Applicability:

- Level 1

Description:

Ensure an administrator approval is required when installing applications.

Rationale:

Applications are typically automated integrations that improve the workflow of an organization. They are written by third-party developers, and therefore should be validated before using in case they're malicious or not trustable. Because administrators are expected to be the most qualified and trusted members of the organization, they should review the applications being installed and decide whether they are both trusted and necessary.

Impact:

Applications will not be installed without administrator approval.

Audit:

Verify that applications are installed only after receiving administrator approval:
You are compliant by default. That is because by default only maintainers and owners can integrate with external applications.

For OAuth Apps, perform the following:

- On the left sidebar, select your avatar.
- Select Edit profile and then select Applications.
- See the Authorized applications section.
- Review the scope level for the authorised applications with your credentials

Remediation:

Require an administrator approval for every installed application:
You are compliant by default. That is because by default only maintainers and owners can integrate with external applications.

For OAuth Apps, perform the following:

- On the left sidebar, select your avatar.
- Select Edit profile and then select Applications.
- See the Authorized applications section.
- Update the scope level for the authorised applications with your credentials

Default Value:

Maintainers are organization owners.

References:

1. https://docs.gitlab.com/ee/integration/oauth_provider.html#create-a-user-owned-application
2. https://docs.gitlab.com/ee/integration/oauth_provider.html#view-all-authorized-applications

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 Allowlist Authorized Software Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.			
v8	2.6 Allowlist Authorized Libraries Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.			
v7	2.7 Utilize Application Whitelisting Utilize application whitelisting technology on all assets to ensure that only authorized software executes and all unauthorized software is blocked from executing on assets.			
v7	2.8 Implement Application Whitelisting of Libraries The organization's application whitelisting software must ensure that only authorized software libraries (such as *.dll, *.ocx, *.so, etc) are allowed to load into a system process.			

1.4.2 Ensure stale applications are reviewed and inactive ones are removed (Manual)

Profile Applicability:

- Level 1

Description:

Ensure stale (inactive) applications are reviewed and removed if no longer in use.

Rationale:

Applications that have been inactive for a long period of time are enlarging the surface of attack for data leaks. They are more likely to be improperly managed, and could possibly be accessed by third-party developers as a tool for collecting internal data of the organization or repository in which they are installed. It is important to remove these inactive applications as soon as possible.

Audit:

Verify that all the applications in the organization are actively used, and remove those that are no longer in use. Ensure that Dependency scanning is enabled, which enables Continuous Vulnerability scanning by default and identifies vulnerabilities applications, even if they are stale.

Remediation:

1. Review all stale applications and periodically remove them.
2. Enable dependency scanning to automatically detect vulnerabilities in stale applications.
3. Add the following to your .gitlab-ci.yml file:

```
include: - template: Security/Dependency-Scanning.gitlab-ci.yml
```

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>2.2 Ensure Authorized Software is Currently Supported</u> Ensure that only currently supported software is designated as authorized in the software inventory for enterprise assets. If software is unsupported, yet necessary for the fulfillment of the enterprise's mission, document an exception detailing mitigating controls and residual risk acceptance. For any unsupported software without an exception documentation, designate as unauthorized. Review the software list to verify software support at least monthly, or more frequently.			
v8	<u>2.4 Utilize Automated Software Inventory Tools</u> Utilize software inventory tools, when possible, throughout the enterprise to automate the discovery and documentation of installed software.			
v7	<u>13.2 Remove Sensitive Data or Systems Not Regularly Accessed by Organization</u> Remove sensitive data or systems not regularly accessed by the organization from the network. These systems shall only be used as stand alone systems (disconnected from the network) by the business unit needing to occasionally use the system or completely virtualized and powered off until needed.			

1.4.3 Ensure the access granted to each installed application is limited to the least privilege needed (Manual)

Profile Applicability:

- Level 1

Description:

Ensure installed application permissions are limited to the lowest privilege level required.

Rationale:

Applications are typically automated integrations that can improve the workflow of an organization. They are written by third-party developers, and therefore should be reviewed carefully before use. It is recommended to use the "least privilege" principle, granting applications the lowest level of permissions required. This may prevent harm from a potentially malicious application with unnecessarily high-level permissions leaking data or modifying source code.

Audit:

Verify that each installed integration and application has the least privilege needed.
For each Project and each Group, perform the following:

- Navigate to the project or group homepage
- In the sidebar, select Settings > Integrations
- Next to every integration, select 'Configure'
- Review the integration's configuration and verify that it is limited to the least privilege needed

For each Group, perform the following:

- Navigate to the project or group homepage
- In the sidebar, select Settings > Applications
 - Next to every Application, select Edit
 - Review the Applications configuration and verify that it is limited to the least privilege needed

As an administrator, perform the following:

- Navigate to the Admin Area
- In the sidebar, select Applications
 - Next to every Application, select Edit
 - Review the Applications configuration and verify that it is limited to the least privilege needed
- In the sidebar, select Settings > Integrations
 - Next to every integration, select 'Configure'
 - Review the integration's configuration and verify that it is limited to the least privilege needed
- In the sidebar, select Overview > Users
 - Select each user's first name
 - On the users detail page, select 'Impersonate'
 - Navigate to their Preferences page
 - In the sidebar, select Applications
 - Next to every Application under 'Authorized applications', review the scopes permitted
 - Select the 'Stop impersonating' icon (next to the impersonated user's avatar)
 - Repeat for each user

Remediation:

Grant permissions to applications by the "least privilege" principle, meaning the lowest possible permission necessary.

For any Integrations identified during the audit as needing modification:

- Next to the integration, select Configure.
- Edit the permissions or settings so that they grant the least possible privileges. For example, restrict the branches it can access, or the features that are enabled.
- (Optionally) Select 'Test settings'
- Select 'Save changes'.

For any Applications identified during the audit as needing modification:

- Next to the application, select 'Edit'.
- Edit the permissions or settings so that they grant the least possible privileges. For example, restrict the API scopes it can use.
- Select 'Save application'.

If any user authorized applications were identified during the audit as having overly permissive scopes, as an administrator perform the following:

- Navigate to the Admin Area
- In the sidebar, select Overview > Users
 - Select the user's first name
 - On the users detail page, select 'Impersonate'
 - Navigate to their Preferences page
 - In the sidebar, select Applications
 - Under 'Authorized applications', re-identify the overly permissive application
 - Select 'Revoke'
 - Select the 'Stop impersonating' icon (next to the impersonated user's avatar)

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.8 <u>Define and Maintain Role-Based Access Control</u> Define and maintain role-based access control, through determining and documenting the access rights necessary for each role within the enterprise to successfully carry out its assigned duties. Perform access control reviews of enterprise assets to validate that all privileges are authorized, on a recurring schedule at a minimum annually, or more frequently.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.4.4 Ensure only secured webhooks are used (Manual)

Profile Applicability:

- Level 1

Description:

Use only secured webhooks in the source code management platform.

Rationale:

A webhook is an event listener, attached to critical and sensitive parts of the software delivery process. It is triggered by a list of events (such as a new code being committed), and when triggered, the webhook sends out a notification with some payload to specific internet endpoints. Since the payload of the webhook contains sensitive organization data, it's important all webhooks are directed to an endpoint (URL) protected by SSL verification (HTTPS). This helps ensure that the data sent is delivered to securely without any man-in-the-middle, who could easily access and even alter the payload of the request.

Impact:

Perform the following to ensure all webhooks used are secured (HTTPS):

1. Navigate to your organization or repository and select **Settings**.
2. Select **Webhooks** on the side menu.
3. Verify that each webhook URL starts with 'https'.

Audit:

Perform the following to secure all webhooks.

For each project and for each group:

- Navigate to the project or group
- Select Settings > Webhooks on the side menu.
- Ensure all webhooks starts with 'https'.
- Ensure all webhooks state 'SSL Verification: enabled'

As an Administrator:

- Navigate to the Admin Area
- Select System Hooks on the side menu.
- Ensure all webhooks starts with 'https'.
- Ensure all webhooks state 'SSL Verification: enabled'

Remediation:

Perform the following to secure all webhooks.
For each project and for each group:

- Navigate to the project or group
- Select Settings > Webhooks on the side menu.
- Find any webhooks that start with 'http' and not 'https', or which have 'SSL Verification: disabled'.
- Click Edit.
- Change the payload URL to begin with 'https'
- Select the 'Enable SSL verification' checkbox
- Click Update webhook.

As an Administrator:

- Navigate to the Admin Area
- Select System Hooks on the side menu.
- Find any webhooks that start with 'http' and not 'https', or which have 'SSL Verification: disabled'.
- Click Edit.
- Change the payload URL to begin with 'https'
- Select the 'Enable SSL verification' checkbox
- Click Update webhook.

References:

1. <https://docs.gitlab.com/ee/user/project/integrations/webhooks.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

1.5 Code Risks

This section consists of recommendations for many security code scanners. This includes for example, looking for hardcoded secrets, common misconfigurations that are vulnerable to attack or restrictive licenses. Because an application code has a lot of components, it is important to scan each part that can lead to attack - from secrets to licenses.

1.5.1 Ensure scanners are in place to identify and prevent sensitive data in code (Manual)

Profile Applicability:

- Level 2

Description:

Detect and prevent sensitive data in code, such as confidential ID numbers, passwords, etc.

Rationale:

Having sensitive data in the source code makes it easier for attackers to maliciously use such information. In order to avoid this, designate scanners to identify and prevent the existence of sensitive data in the code.

Audit:

Audit: For every repository in use, verify that scanners are set to identify and prevent the existence of sensitive data in code by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. Review the CI pipeline configuration to verify that [Secret Detection has been enabled](#) on this project.

Remediation:

Remediation: For every repository in use, designate scanners to identify and prevent sensitive data in code by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. [Enable secret detection](#) for this project.

Additional Information:

By January 2023, this feature is supposed to be open to all plans. Until then it is only for enterprise users.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 <u>Implement Code-Level Security Checks</u> Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	18.7 <u>Apply Static and Dynamic Code Analysis Tools</u> Apply static and dynamic analysis tools to verify that secure coding practices are being adhered to for internally developed software.		●	●

1.5.2 Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions (Manual)

Profile Applicability:

- Level 2

Description:

Detect and prevent misconfigurations and insecure instructions in CI pipelines

Rationale:

Detecting and fixing misconfigurations or insecure instructions in CI pipelines decreases the risk for a successful attack through or on the CI pipeline. The more secure the pipeline, the less risk there is for potential exposure of sensitive data, a deployment being compromised, or external access mistakenly being granted to the CI infrastructure or the source code.

Audit:

For every project:

- Identify if one or more CI configuration files exist
- Review CI configuration files for misconfigurations or insecure instructions
- Review whether the absence of any CI configuration is itself a misconfiguration

Remediation:

For every project identified during the Audit as having misconfigured or insecure CI instructions:

- Update the CI instructions
- Consider using Scan Execution Policies at the project or group level to enforce security scans
- Consider using the Compliance Framework feature to enforce pipeline configuration
- Consider procuring and enabling a CI instructions scanning tool to identify and prevent misconfigurations and insecure instructions and scans all CI pipelines.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			

1.5.3 Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions (Manual)

Profile Applicability:

- Level 2

Description:

Detect and prevent misconfigurations or insecure instructions in Infrastructure as Code (IaC) files, such as Terraform files.

Rationale:

Detecting and fixing misconfigurations and/or insecure instructions in IaC (Infrastructure as Code) files decreases the risk for data leak or data theft. It is important to secure IaC instructions in order to prevent further problems of deployment, exposed assets, or improper configurations, which can ultimately lead to easier ways to attack and steal organization data.

Audit:

Audit: For every repository that holds IaC instructions files, verify that a scanning tool such as GitLab's Infrastructure as Code scanning is configured to identify and prevent misconfigurations and insecure instructions.

1. On GitLab, navigate to the main page of the repository.
2. Review the CI pipeline configuration to verify that [IaC scanning has been enabled](#) on this project.

Remediation:

Remediation: For every repository that holds IaC instructions files, configure GitLab's Infrastructure as Code scanning (or another IaC scanning tool) to identify and prevent misconfigurations and insecure instructions.

1. On GitLab, navigate to the main page of the repository.
2. [Enable IaC scanning](#) for this project.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v8	<u>16.7 Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			
v7	<u>18.11 Use Standard Hardening Configuration Templates for Databases</u> For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.			

1.5.4 Ensure scanners are in place for code vulnerabilities (Manual)

Profile Applicability:

- Level 2

Description:

Detect and prevent known open source vulnerabilities in the code.

Rationale:

Open source code blocks are used a lot in developed software. This has its own advantages, but it also has risks. Because the code is open for everyone, it means that attackers can publish or add malicious code to these open-source code blocks, or use their knowledge to find vulnerabilities in an existing code. Detecting and fixing such code vulnerabilities, by SCA (Software Composition Analysis) prevents insecure flaws from reaching production. It gives another opportunity for developers to secure the source code before it is deployed in production, where it is far more exposed and vulnerable to attacks.

Audit:

Audit: For every repository that is in use, verify that a scanning tool is set to identify and prevent code vulnerabilities by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. Review the CI pipeline configuration to verify that [SAST has been configured to run](#) on this project.

Remediation:

Remediation: For every repository that is in use, set a scanning tool to identify and prevent code vulnerabilities by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. Configure SAST to run on this project.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 <u>Implement Code-Level Security Checks</u> Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	18.7 <u>Apply Static and Dynamic Code Analysis Tools</u> Apply static and dynamic analysis tools to verify that secure coding practices are being adhered to for internally developed software.		●	●

1.5.5 Ensure scanners are in place for open-source vulnerabilities in used packages (Manual)

Profile Applicability:

- Level 2

Description:

Detect, prevent and monitor known open-source vulnerabilities in packages that are being used.

Rationale:

Open-source vulnerabilities might exist before one starts to use a package, but they are also discovered over time. New attacks and vulnerabilities are announced every now and then. It is important to keep track of these and to monitor whether the dependencies used are affected by the recent vulnerability. Detecting and fixing those packages' vulnerabilities decreases the attack surface within deployed and running applications that use such packages. It prevents security flaws from reaching the production environment which could eventually lead to a security breach.

Audit:

Audit: For every repository that is in use, verify that a dependency scanning tool is set to detect, prevent, and monitor vulnerabilities in project dependencies by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. Review the CI pipeline configuration to verify that [Dependency Scanning has been configured to run](#) on this project.

Remediation:

Remediation: For every repository that is in use, set a dependency scanning tool to detect, prevent, and monitor vulnerabilities in project packages by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. [Configure Dependency Scanning](#) to run on this project.

Additional Information:

Note: GitLab Dependency Scanning is only available in GitLab Ultimate. If you're not using GitLab Ultimate, consider using a free open-source dependency scanner as part of your CI pipeline.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v8	<u>16.5 Use Up-to-Date and Trusted Third-Party Software Components</u> Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			
v7	<u>18.4 Only Use Up-to-date And Trusted Third-Party Components</u> Only use up-to-date and trusted third-party components for the software developed by the organization.			

1.5.6 Ensure scanners are in place for open-source license issues in used packages (Manual)

Profile Applicability:

- Level 2

Description:

Detect open-source license problems in used dependencies and fix them.

Rationale:

A software license is a legal document that establishes several key conditions between a software company or developer and a user in order to allow the use of software. Software licenses have the potential to create code dependencies. Not following the conditions in the software license can also lead to lawsuits. When using packages with a software license, especially commercial ones (which are the most permissive), it is important to verify what is allowed by that license in order to be protected against lawsuits.

Audit:

Ensure a license scanning tool is set up to identify open-source license problems and that every package you use is scanned by it.

- The Dependency Scanning or Container Scanning CI job must be configured for your project.
- Your project uses at least one of the languages and package managers supported by Gemnasium.
- A successful pipeline was run on the default branch. You should not change the default behavior of allowing the application security jobs to fail.
- Review the output from the scan in Secure > Dependency list.

Remediation:

- The Dependency Scanning or Container Scanning CI job must be configured for your project.
- Your project uses at least one of the languages and package managers supported by Gemnasium.
- A successful pipeline was run on the default branch. You should not change the default behavior of allowing the application security jobs to fail.
- Take necessary actions based on the outcome of the scan in Secure > Dependency list.

References:

1. https://docs.gitlab.com/ee/user/compliance/license_scanning_of_cyclonedx_files/index.html
2. https://docs.gitlab.com/ee/user/application_security/dependency_list/index.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

1.5.7 Ensure scanners are in place for web application runtime security weaknesses (Manual)

Profile Applicability:

- Level 2

Description:

Dynamic Application Security Testing (DAST) runs automated penetration tests to find vulnerabilities in your web applications as they are running. DAST automates a hacker's approach and simulates real-world attacks for critical threats such as cross-site scripting (XSS), SQL injection (SQLi), and cross-site request forgery (CSRF) to uncover vulnerabilities and misconfigurations that other security tools cannot detect.

Rationale:

Audit:

Audit: Ensure Browser-based DAST is set up to identify dynamic vulnerabilities in web applications that cannot be detected by other tools earlier in the SDLC. Ensure that every project that contains a web application is scanned by DAST.

- The DAST template must be configured for your project.
- Review the output from the previous scan on your default branch in Secure > Vulnerability Report.

Remediation:

1. Include the DAST.gitlab-ci.yml template in your .gitlab-ci.yml file.
2. Add a dast stage to your GitLab CI/CD stages configuration.
3. Define the URL to be scanned by DAST by setting the DAST_WEBSITE CI/CD variable.
4. Add any additionally desired CI variables to customize the test.
5. After configuration, the analyzer will run in your pipeline.
6. View the results in the Vulnerability report and remediate the vulnerabilities.

References:

1. https://docs.gitlab.com/ee/user/application_security/dast/browser/configuration/customize_settings.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 <u>Implement Code-Level Security Checks</u> Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	18.7 <u>Apply Static and Dynamic Code Analysis Tools</u> Apply static and dynamic analysis tools to verify that secure coding practices are being adhered to for internally developed software.		●	●

1.5.8 Ensure scanners are in place for API runtime security weaknesses (Manual)

Profile Applicability:

- Level 2

Description:

Dynamic Application Security Testing (DAST) runs automated penetration tests to find vulnerabilities in your APIs as they are running. DAST automates a hacker's approach and simulates real-world attacks for critical threats such as cross-site scripting (XSS), SQL injection (SQLi), and cross-site request forgery (CSRF) to uncover vulnerabilities and misconfigurations that other security tools cannot detect.

Rationale:

Audit:

Audit: Ensure a dynamic API scanning tool is set up to identify API-specific vulnerabilities and that every project that contains an API is scanned by it.

- The DAST-API template must be configured for your project.
- A successful pipeline was run on the default branch. You should not change the default behavior of allowing the application security jobs to fail.
- Review the output from the scan in Secure > Vulnerability Report

Remediation:

1. Include the DAST-API.gitlab-ci.yml template in your .gitlab-ci.yml file.
2. The configuration file has several testing profiles defined with different checks enabled. Select a profile and provide it by adding the DAST_API_PROFILE CI/CD variable to your .gitlab-ci.yml file.
3. Provide the location of the OpenAPI Specification as either a file or URL. Specify the location by adding the DAST_API_OPENAPI variable.
4. The target API instance's base URL is also required. Provide it by using the DAST_API_TARGET_URL variable or an environment_url.txt file.
5. After configuration, the analyzer will run in your pipeline.
6. View the results in the Vulnerability report and remediate the vulnerabilities.

References:

1. https://docs.gitlab.com/ee/user/application_security/dast_api/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 <u>Implement Code-Level Security Checks</u> Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	18.7 <u>Apply Static and Dynamic Code Analysis Tools</u> Apply static and dynamic analysis tools to verify that secure coding practices are being adhered to for internally developed software.		●	●

2 Build Pipelines

This section consists of security recommendations for the management of application build pipelines developed by an organization.

Build pipelines are a set of instructions dedicated to taking raw files of source code and running a series of tasks on them to achieve some final artifact as output. This artifact represents the final form of the recent version of software, which is subsequently packaged for convenient storing, handling, and deploying. Build pipelines are a general name for the environment in which this compilation process takes place, the pipeline files that orchestrate the process, and all sets of instructions related to them.

2.1 Build Environment

This section consists of security recommendations for the build pipelines environment.

Build environment is everything related to the infrastructure of the organization's artifacts build - the orchestrator, the pipeline executor, where the build workers are running, while pipeline is a set of commands that runs in the build environment. Most of the build environment recommendations are relevant for self-hosted build platforms only. For example, instance of CircleCi that is self-hosted.

2.1.1 Ensure each pipeline has a single responsibility (Manual)

Profile Applicability:

- Level 2

Description:

Ensure each pipeline has a single responsibility in the build process.

Rationale:

Build pipelines generally have access to multiple secrets depending on their purposes. There are, for example, secrets of the test environment for the test phase, repository and artifact credentials for the build phase, etc. Limiting access to these credentials/secrets is therefore recommended by dividing pipeline responsibilities, as well as having a dedicated pipeline for each phase with the lowest privilege instead of a single pipeline for all. This will ensure that any potential damage caused by attacks on a workflow will be limited.

Audit:

For each pipeline, ensure it has only one responsibility in the build process.

Remediation:

Divide each multi-responsibility pipeline into multiple pipelines, each having a single responsibility with the least privilege. Additionally, create all new pipelines with a sole purpose going forward.

References:

1. <https://docs.gitlab.com/ee/user/permissions.html#job-permissions>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.6 Securely Manage Enterprise Assets and Software Securely manage enterprise assets and software. Example implementations include managing configuration through version-controlled-infrastructure-as-code and accessing administrative interfaces over secure network protocols, such as Secure Shell (SSH) and Hypertext Transfer Protocol Secure (HTTPS). Do not use insecure management protocols, such as Telnet (Teletype Network) and HTTP, unless operationally essential.			
v8	16.10 Apply Secure Design Principles in Application Architectures Apply secure design principles in application architectures. Secure design principles include the concept of least privilege and enforcing mediation to validate every operation that the user makes, promoting the concept of "never trust user input." Examples include ensuring that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats. Secure design also means minimizing the application infrastructure attack surface, such as turning off unprotected ports and services, removing unnecessary programs and files, and renaming or removing default accounts.			

2.1.2 Ensure all aspects of the pipeline infrastructure and configuration are immutable (Manual)

Profile Applicability:

- Level 1

Description:

Ensure the pipeline orchestrator and its configuration are immutable.

Rationale:

An immutable infrastructure is one that cannot be changed during execution of the pipeline. This can be done, for example, by using Infrastructure as Code for configuring the pipeline and the pipeline environment. Utilizing such infrastructure creates a more predictable environment because updates will require re-deployment to prevent any previous configuration from interfering. Because it is dependent on automation, it is easier to revert changes. Testing code is also simpler because it is based on virtualization. Most importantly, an immutable pipeline infrastructure ensures that a potential attacker seeking to compromise the build environment itself would not be able to do so if the orchestrator, its configuration, and any other component cannot be changed. Verifying that all aspects of the pipeline infrastructure and configuration are immutable therefore keeps them safe from malicious tampering attempts.

Audit:

Verify that the pipeline orchestrator, its configuration, and all other aspects of the build environment are immutable.

Remediation:

Use an immutable pipeline orchestrator and ensure that its configuration and all other aspects of the built environment are immutable, as well.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v8	<u>18.1 Establish and Maintain a Penetration Testing Program</u> Establish and maintain a penetration testing program appropriate to the size, complexity, and maturity of the enterprise. Penetration testing program characteristics include scope, such as network, web application, Application Programming Interface (API), hosted services, and physical premise controls; frequency; limitations, such as acceptable hours, and excluded attack types; point of contact information; remediation, such as how findings will be routed internally; and retrospective requirements.			

2.1.3 Ensure the build environment is logged (Manual)

Profile Applicability:

- Level 1

Description:

Keep build logs of the build environment detailing configuration and all activity within it. Also, consider to store them in a centralized organizational log store.

Rationale:

Logging the environment is important for two primary reasons: one, for debugging and investigating the environment in case of a bug or security incident; and two, for reproducing the environment easily when needed. Storing these logs in a centralized organizational log store allows the organization to generate useful insights and identify anomalies in the build process faster.

Audit:

Verify that the build environment is logged and stored in a centralized organizational log store.

When a build environment is created, all information related to the environment is logged as part of the job logs.

Depending on the stage in the build environment lifecycle, the path to retrieve the logs will differ:

- When a job is running: `#{ROOT_PATH}/gitlab-ci/builds/#{YYYY_mm}/#{project_id}/#{job_id}.log`
- After a job is finished: `#{ROOT_PATH}/gitlab-rails/shared/artifacts/#{disk_hash}/#{YYYY_mm_dd}/#{job_id}/#{job_artifact_id}/job.log`
- After a log is archived:
`#{bucket_name}/#{disk_hash}/#{YYYY_mm_dd}/#{job_id}/#{job_artifact_id}/job.log`

Remediation:

Keep logs of the build environment. Also, store the logs in a centralized organizational log store.

This is automatically done by GitLab and can be retrieved in the paths mentioned in the audit section.

References:

1. https://docs.gitlab.com/ee/administration/job_logs.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.14 <u>Log Sensitive Data Access</u> Log sensitive data access, including modification and disposal.			●
v7	14.9 <u>Enforce Detail Logging for Access or Changes to Sensitive Data</u> Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).			●

2.1.4 Ensure the creation of the build environment is automated (Manual)

Profile Applicability:

- Level 1

Description:

Automate the creation of the build environment.

Rationale:

Automating the deployment of the build environment reduces the risk for human mistakes — such as a wrong configuration or exposure of sensitive data — because it requires less human interaction and intervention. It also eases re-deployment of the environment. It is best to automate with Infrastructure as Code because it offers more control over changes made to the environment creation configuration and stores to a version control platform.

Audit:

Verify that the deployment of the build environment is automated and can be easily redeployed.

In GitLab, build environments are automatically created for each CI/CD pipeline. To verify that a build environment has been automatically created, do the following:

- On the left sidebar, select Search or go to and find your project.
- Select Operate > Environments.
- Review the environments.

Remediation:

Automate the deployment of the build environment.

In GitLab, build environments are automatically created for each CI/CD pipeline. To automate a deployment of the build environment, you need to create a CI/CD pipeline using the `gitlab-ci.yml` file.

References:

1. <https://docs.gitlab.com/ee/ci/environments/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

2.1.5 Ensure access to build environments is limited (Manual)

Profile Applicability:

- Level 1

Description:

Restrict access to the build environment (orchestrator, pipeline executor, their environment, etc.) to trusted and qualified users only.

Rationale:

A build environment contains sensitive data such as environment variables, secrets, and the source code itself. Any user that has access to this environment can make changes to the build process, including changes to the code within it. Restricting access to the build environment to trusted and qualified users only will reduce the risk for mistakes such as exposure of secrets or misconfiguration. Limiting access also reduces the number of accounts that are vulnerable to hijacking in order to potentially harm the build environment.

Impact:

Reducing the number of users who have access to the build process means those users would lose their ability to make direct changes to that process.

Audit:

Verify each build environment is accessible only to known and authorized users. In GitLab, viewing environments in private projects is limited to Reporter roles at least and you must have at least the Developer role to create a new environment

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Reporter/Developer role or above in the list, you are compliant.

Remediation:

Restrict access to the build environment to trusted and qualified users.
In GitLab, viewing environments in private projects is limited to Reporter roles at least and you must have at least the Developer role to create a new environment

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member

References:

1. <https://docs.gitlab.com/ee/ci/environments/#environment-permissions>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

2.1.6 Ensure users must authenticate to access the build environment (Manual)

Profile Applicability:

- Level 1

Description:

Require users to login in to access the build environment - where the orchestrator, the pipeline executor, where the build workers are running, etc.

Rationale:

Requiring users to authenticate and disabling anonymous access to the build environment allows organization to track every action on that environment, good or bad, to its actor. This will help recognizing attack and its attacker because the authentication is required.

Impact:

Anonymous users won't be able to access the build environment.

Audit:

Ensure authentication is required to access the build environment.

In GitLab, viewing environments in private projects is limited to Reporter roles at least and you must have at least the Developer role to create a new environment. Both require prior authentication.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Reporter/Developer role or above in the list, you are compliant.

Remediation:

Require authentication to access the build environment and disable anonymous access. In GitLab, viewing environments in private projects is limited to Reporter roles at least and you must have at least the Developer role to create a new environment. Both require prior authentication.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member

References:

1. <https://docs.gitlab.com/ee/ci/environments/#environment-permissions>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 Allowlist Authorized Software Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.			
v7	2.7 Utilize Application Whitelisting Utilize application whitelisting technology on all assets to ensure that only authorized software executes and all unauthorized software is blocked from executing on assets.			

2.1.7 Ensure build secrets are limited to the minimal necessary scope (Manual)

Profile Applicability:

- Level 2

Description:

Build tools providers offer a secure way to store secrets that should be used during the build process. These secrets will often be credentials used to access other tools, for example for pulling code or for uploading artifacts. Access to these secrets can be defined on various scopes. To protect these critical assets it is important to choose the most restrictive scope necessary.

Rationale:

Allowing over permissive access to these secrets may affect on their exposure. For example if a secret is defined in an organization level, and users can create new repositories, there is a scenario where a user can create a new repo and run a controlled build just to exfiltrate these secrets.

Impact:

Increased risk of exposure of build related secrets.

Audit:

In the gitlab-ci.yml file, review the secrets permission scope defined in either a file or in an external secrets manager.

Remediation:

In the gitlab-ci.yml file, review the secrets defined in either a file or in an external secrets manager and change over permissive scopes to more restrictive ones based on the required access.

References:

1. <https://docs.gitlab.com/ee/ci/yaml/index.html#secrets>
2. <https://docs.gitlab.com/ee/ci/secrets/index.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

2.1.8 Ensure the build infrastructure is automatically scanned for vulnerabilities (Manual)

Profile Applicability:

- Level 1

Description:

Scan the build infrastructure and its dependencies for vulnerabilities. It is recommended that this be done automatically.

Rationale:

Automatic scanning for vulnerabilities detects known vulnerabilities in the tooling used by the build infrastructure and its dependencies. These vulnerabilities can lead to a potentially massive breach if not handled as fast as possible, as attackers might also be aware of such vulnerabilities.

Audit:

Verify that your build infrastructure is reviewed for vulnerabilities.

GitLab Runner is designed to run user-controlled scripts. To reduce the attack surface if a job is malicious, you can consider running them in their own network segment. This would provide network separation from other infrastructure and services.

Remediation:

Ensure security hardening for your build infrastructure.

For a cloud environment, this could include:

- Configuring runner virtual machines in their own network segment
- Blocking SSH access from the Internet to runner virtual machines
- Restricting traffic between runner virtual machines
- Filtering access to cloud provider metadata endpoints

For static host runner, whether bare-metal or virtual machine, you should implement security best practices for the host operating system:

Malicious code executed in the context of a CI job could compromise the host, so security protocols can help mitigate the impact. Other points to keep in mind include securing or removing files such as SSH keys from the host system that may enable an attacker to access other endpoints in the environment.

References:

1. <https://docs.gitlab.com/runner/security/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			

2.1.9 Ensure default passwords are not used (Manual)

Profile Applicability:

- Level 1

Description:

Do not use default passwords of build tools and components.

Rationale:

Sometimes build tools and components are provided with default passwords for the first login. This password is intended to be used only on the first login and should be changed immediately after. Using the default password substantially increases the attack risk. It is especially important to ensure that default passwords are not used in build tools and components.

Audit:

GitLab's default root password depends on the installation method, and when the installation occurred:

1. When deploying a GitLab instance using the official AWS AMI, the root password to the instance is the EC2 Instance ID
2. Most installation methods allow a non-default password to be provided as configuration
3. Prior to 14.0 the default password was 5iveL!fe
4. Otherwise the default password is unique and randomly generated.

Attempt to log in as root using a suspected default password to audit whether it has changed.

For any other external build tools, ensure the password used is not the default one.

Remediation:

GitLab's root password can be changed by an administrator using the UI, the "gitlab:password:reset" rake task, or by using the Rails console.

For each build tool with a default password, change to a unique cryptographically secure pseudorandom password.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.2 <u>Use Unique Passwords</u> Use unique passwords for all enterprise assets. Best practice implementation includes, at a minimum, an 8-character password for accounts using MFA and a 14-character password for accounts not using MFA.			
v7	4.2 <u>Change Default Passwords</u> Before deploying any new asset, change all default passwords to have values consistent with administrative level accounts.			

2.1.10 Ensure webhooks of the build environment are secured (Manual)

Profile Applicability:

- Level 1

Description:

Use secured webhooks of the build environment.

Rationale:

Webhooks are used for triggering an HTTP request based on an action made in the platform. Typically, build environment feature webhooks for a pipeline trigger based on source code event. Since webhooks are an HTTP POST request, they can be malformed if not secured over SSL. To prevent a potential hack and compromise of the webhook or to the environment or web server excepting the request, use only secured webhooks.

Audit:

For each webhook in use, ensure it is secured (HTTPS) via the following process:

- In your project or group, on the left sidebar, select Settings > Webhooks.
- For each webhook, click Edit.
- Verify if the Enable SSL verification checkbox is checked.
- Select Add webhook.

Remediation:

For each webhook in use, change it to secured (over HTTPS).

- In your project or group, on the left sidebar, select Settings > Webhooks.
- For each webhook, click Edit.
- Ensure the Enable SSL verification checkbox is checked.
- Select Save.

References:

1. <https://docs.gitlab.com/ee/user/project/integrations/webhooks.html#configure-a-webhook-in-gitlab>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	12.6 <u>Use of Secure Network Management and Communication Protocols</u> Use secure network management and communication protocols (e.g., 802.1X, Wi-Fi Protected Access 2 (WPA2) Enterprise or greater).			

2.1.11 Ensure minimum number of administrators are set for the build environment (Manual)

Profile Applicability:

- Level 1

Description:

Ensure the build environment has a minimum number of administrators.

Rationale:

Build environment administrators have the highest level of permissions, including the ability to add/remove users, create or delete pipelines, control build workers, change build trigger permissions and more. Due to the permissive access granted to a build environment administrator, it is highly recommended to keep the number of administrator accounts as minimal as possible.

Audit:

Verify that the build environment has only the minimum number of administrators.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Owner/Maintainer role in the list, you are compliant.

Remediation:

Set the minimum number of administrators in the build environment.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

2.2 Build Worker

This section consists of security recommendations for build workers management and use.

Build workers are often called Runners. They are the infrastructure on which the pipeline runs. Build workers are considered sensitive because usually they have access to multiple, if not all, software supply chain components. One worker can run code checkout with source code management access, run tests, and push to the registry which requires access to it. Also, some of the pipeline commands running in a build worker can be vulnerable to attack and enlarge the attack surface. Because of all of that, it is especially important to ensure that the build workers are protected.

2.2.1 Ensure build workers are single-used (Manual)

Profile Applicability:

- Level 1

Description:

Use a clean instance of build worker for every pipeline run.

Rationale:

Using a clean instance of build worker for every pipeline run eliminates the risks of data theft, data integrity breaches, and unavailability. It limits the pipeline's access to data stored on the file system from previous runs, and the cache is volatile. This prevents malicious changes from affecting other pipelines or the Continuous Integration/Continuous Delivery system itself.

Impact:

Data and cache will not be saved in different pipeline runs.

Audit:

Ensure that every pipeline that is being run has its own clean, new runner. In GitLab, each of your jobs runs in a newly provisioned VM, which is dedicated to the specific job.

The VM is active only for the duration of the job and immediately deleted.

This means that any changes that your job makes to the virtual machine will not be available to a subsequent job.

The virtual machine where your job runs has sudo access with no password.

Remediation:

Create a clean build worker for every pipeline that is being run, or use build platform-hosted runners, as they typically offer a clean instance for every run.

In GitLab, use Runner SaaS to ensure all of these settings are available by default and that each job runs in a dedicated VM.

References:

1. <https://docs.gitlab.com/ee/ci/runners/index.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.2.2 Ensure build worker environments and commands are passed and not pulled (Manual)

Profile Applicability:

- Level 1

Description:

A worker's environment can be passed (for example, a pod in a Kubernetes cluster in which an environment variable is passed to it). It also can be pulled, like a virtual machine that is installing a package. Ensure that the environment and commands are passed to the workers and not pulled from it.

Rationale:

Passing an environment means additional configuration happens in the build time phase and not in run time. It will also pass locally and not remotely. Passing a worker environment, instead of pulling it from an outer source, reduces the possibility for an attacker to gain access and potentially pull malicious code into it. By passing locally and not pulling from remote, there is also less chance of an attack based on the remote connection, such as a man-in-the-middle or malicious scripts that can run from remote. This therefore prevents possible infection of the build worker.

Audit:

For each build worker, ensure its environment and commands are passed and not pulled.

1. **Review `.gitlab-ci.yml` and Runner Configurations:** Check your project's `.gitlab-ci.yml` file and any associated Runner configurations to ensure that all necessary environment variables, commands, and configurations are explicitly defined and passed to the Runner. This involves reviewing the job definitions to confirm they do not rely on external sources for configuration at runtime.
2. **Examine Runner Execution Environment:** Verify that the execution environment of the Runner (whether it's a Docker container, a Kubernetes pod, or a virtual machine) receives its configuration from the `.gitlab-ci.yml` file or the GitLab project settings directly, without pulling from external sources during the job execution.
3. **Check for External Dependencies:** Ensure that any external dependencies, such as third-party libraries or tools, are explicitly defined and version-controlled within the project repository or through secure, trusted registries. Avoid configurations that allow the Runner to dynamically fetch or update these dependencies during the build process without strict version controls.

Remediation:

For each build worker, pass its environment and commands to it instead of pulling it.

1. **Update .gitlab-ci.yml:** Amend your .gitlab-ci.yml file to include all necessary environment variables and configurations directly within the file or through secure, project-level settings in GitLab. Avoid dynamic fetching of configurations from external sources during runtime.
2. **Secure Runner Environment:** Configure your GitLab Runner environments (Docker, Kubernetes, VMs, etc.) to use pre-defined images and configurations that do not require pulling additional settings or scripts during execution. Use trusted, version-controlled base images and scripts.
3. **Utilize Secure Variables and Templates:** Leverage GitLab's features for secure variables and include templates for common configurations to ensure that environment settings and commands are passed securely and consistently across all projects.

References:

1. <https://docs.gitlab.com/ee/ci/#the-gitlab-ciyaml-file>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.2.3 Ensure the duties of each build worker are segregated (Manual)

Profile Applicability:

- Level 1

Description:

Separate responsibilities in the build workflow, such as testing, compiling, pushing artifacts, etc., to different build workers so that each worker will have a single duty.

Rationale:

Separating duties and allocating them to many workers makes it easier to verify each step in the build process and ensure there is no corruption. It also limits the effect of an attack on a build worker, as such an attack would be less critical if the worker has less access and duties that are subject to harm.

Audit:

To ensure separation of duties, you can have runner only running jobs for a specific project or a specific group.

For a group:

- On the left sidebar, select Search or go to and find your group.
- Select Settings > CI/CD.
- Expand the Runners section.
- Review the Runners related to your specific group.

For a project:

- On the left sidebar, select Search or go to and find your project.
- Select Settings > CI/CD.
- Expand the Runners section.
- Review the Runners related to your specific project.

Remediation:

To ensure separation of duties, create runners dedicated to a single group or project.
For a group:

- On the left sidebar, select Search or go to and find your group.
- Select Build > Runners.
- Select New group runner.
- Select the operating system where GitLab Runner is installed.
- In the Tags section, in the Tags field, enter the job tags to specify jobs the runner can run. If there are no job tags for this runner, select Run untagged.
- Optional. In the Runner description field, add a runner description that displays in GitLab.
- Optional. In the Configuration section, add additional configurations.
- Select Create runner.
- Follow the on-screen instructions to register the runner from the command line.
When prompted by the command line:
 - For the GitLab instance URL, use the URL for your GitLab instance. For example, if your project is hosted on `gitlab.example.com/yourname/yourproject`, your GitLab instance URL is <https://gitlab.example.com>.
 - For the executor, enter the type of executor. The executor is the environment where the runner executes the job.

For a project:

- On the left sidebar, select Search or go to and find your project.
- Select Settings > CI/CD.
- Expand the Runners section.
- Select New project runner.
- Select the operating system where GitLab Runner is installed.
- In the Tags section, in the Tags field, enter the job tags to specify jobs the runner can run. If there are no job tags for this runner, select Run untagged.
- Optional. In the Runner description field, add a description for the runner that displays in GitLab.
- Optional. In the Configuration section, add additional configurations.
- Select Create runner.
- Follow the on-screen instructions to register the runner from the command line.
When prompted by the command line:
 - For the GitLab instance URL, use the URL for your GitLab instance. For example, if your project is hosted on `gitlab.example.com/yourname/yourproject`, your GitLab instance URL is <https://gitlab.example.com>.
 - For the executor, enter the type of executor. The executor is the environment where the runner executes the job.

References:

1. https://docs.gitlab.com/ee/ci/runners/runners_scope.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.2.4 Ensure build workers have minimal network connectivity (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that build workers have minimal network connectivity.

Rationale:

Restricting the network connectivity of build workers decreases the possibility that an attacker would be capable of entering the organization from the outside. If the build workers are connected to the public internet without any restriction, it is far simpler for attackers to compromise them. Limiting network connectivity between build workers also protects the organization in case an attacker was successful and subsequently attempts to spread the attack to other components of the environment.

Impact:

Developers will not have connectivity to every resource they might need from the outside. Workers will also only be able to exchange data through shareable storage.

Audit:

Verify that build workers, environment, and any other components have only the required minimum of network connectivity.

Review these configuration measures for your self-managed runners:

- Configuring runner virtual machines in their own network segment
- Blocking SSH access from the Internet to runner virtual machines
- Restricting traffic between runner virtual machines
- Filtering access to cloud provider metadata endpoints

Remediation:

Limit the network connectivity of build workers, environment, and any other components to the necessary minimum.

Ensure these configuration measures are in place for your self-managed runners:

- Configuring runner virtual machines in their own network segment
- Blocking SSH access from the Internet to runner virtual machines
- Restricting traffic between runner virtual machines
- Filtering access to cloud provider metadata endpoints

References:

1. <https://docs.gitlab.com/runner/security/#network-segmentation>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>13.5 Manage Access Control for Remote Assets</u> Manage access control for assets remotely connecting to enterprise resources. Determine amount of access to enterprise resources based on: up-to-date anti-malware software installed, configuration compliance with the enterprise's secure configuration process, and ensuring the operating system and applications are up-to-date.			
v7	<u>12.12 Manage All Devices Remotely Logging into Internal Network</u> Scan all enterprise devices remotely logging into the organization's network prior to accessing the network to ensure that each of the organization's security policies has been enforced in the same manner as local network devices.			

2.2.5 Ensure run-time security is enforced for build workers (Manual)

Profile Applicability:

- Level 1

Description:

Add traces to build workers' operating systems and installed applications so that in run time, collected events can be analyzed to detect suspicious behavior patterns and malware.

Rationale:

Build workers are exposed to data exfiltration attacks, code injection attacks, and more while running. It is important to secure them from such attacks by enforcing run-time security on the build worker itself. This will identify attempted attacks in real time and prevent them.

Audit:

Verify that a run-time security solution is enforced on every active build worker.

Remediation:

Deploy and enforce a run-time security solution on build workers.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.2.6 Ensure build workers are automatically scanned for vulnerabilities (Manual)

Profile Applicability:

- Level 1

Description:

Scan build workers for vulnerabilities. It is recommended that this be done automatically.

Rationale:

Automatic scanning for vulnerabilities detects known weaknesses in environmental sources in use, such as docker images or kernel versions. Such vulnerabilities can lead to a massive breach if these environments are not replaced as fast as possible, since attackers also know about these vulnerabilities and often try to take advantage of them. Setting automatic scanning which scans environmental sources ensures that if any new vulnerability is revealed, it can be replaced quickly and easily. This protects the worker from being exposed to attacks.

Audit:

If you are using a static host for a runner, whether bare-metal or virtual machine, you should implement security best practices for the host operating system.

Malicious code executed in the context of a CI job could compromise the host, so security protocols can help mitigate the impact. Other points to keep in mind include securing or removing files such as SSH keys from the host system that may enable an attacker to access other endpoints in the environment.

Remediation:

For each build worker, automatically scan its environmental sources, such as docker image, for vulnerabilities.

- Create a new project.
- Add a Dockerfile file to the project. This Dockerfile contains minimal configuration required to create a Docker image.
- Create pipeline configuration for the new project to create a Docker image from the Dockerfile, build and push a Docker image to the container registry, and then scan the Docker image for vulnerabilities.
- Check for reported vulnerabilities.
- Update the Docker image and scan the updated image.

References:

1. https://docs.gitlab.com/ee/tutorials/container_scanning/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v7	<u>3.2 Perform Authenticated Vulnerability Scanning</u> Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.			

2.2.7 Ensure build workers' deployment configuration is stored in a version control platform (Manual)

Profile Applicability:

- Level 1

Description:

Store the deployment configuration of build workers in a version control platform, such as GitLab.

Rationale:

Build workers are a sensitive part of the build phase. They generally have access to the code repository, the Continuous Integration platform, the deployment platform, etc. This means that an attacker gaining access to a build worker may compromise other platforms in the organization and cause a major incident. One thing that can protect workers is to ensure that their deployment configuration is safe and well-configured. Storing the deployment configuration in version control enables more observability of these configurations because everything is catalogued in a single place. It adds another layer of security, as every change will be reviewed and noticed, and thus malicious changes will theoretically occur less. In the case of a mistake, bug, or security incident, it also offers an easier way to "revert" back to a safe version or add a "hot fix" quickly.

Impact:

Changes in deployment configuration may only be applied by declaration in the version control platform. This could potentially slow down the development process.

Audit:

Verify that the deployment configuration of build workers is stored in a version control platform.

- On the left sidebar, select Code > Repository.
- Check if a .gitlab-ci.yml file is at the root of your repository.

Remediation:

Document and store every deployment configuration of build workers in a version control platform.

- On the left sidebar, select Code > Repository.
- Above the file list, select the branch you want to commit to. If you're not sure, leave master or main. Then select the plus icon (+) and New file:
- For the Filename, type .gitlab-ci.yml.
- Select Commit changes.

References:

1. https://docs.gitlab.com/ee/ci/quick_start/index.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 <u>Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 <u>Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

2.2.8 Ensure resource consumption of build workers is monitored (Manual)

Profile Applicability:

- Level 1

Description:

Monitor the resource consumption of build workers and set alerts for high consumption that can lead to resource exhaustion.

Rationale:

Resource exhaustion is when machine resources or services are highly consumed until exhausted. Resource exhaustion may lead to DOS (Denial of Service). When such a situation happens to build workers, it slows down and even stops the build process, which harms the production of artifacts and the organization's ability to deliver software on schedule. To prevent that, it is recommended to monitor resources consumption in the build workers and set alerts to notify when they are highly consumed. That way resource exhaustion can be acknowledged and prevented at an early stage.

Audit:

Verify that there is monitoring of resources consumption for each build worker. GitLab Runner is instrumented with native Prometheus metrics, which can be exposed via an embedded HTTP server on the /metrics path. The server - if enabled - can be scraped by the Prometheus monitoring system or accessed with any other HTTP client. This is the list of available metrics:

- #HELP gitlab_runner_api_request_statuses_total The total number of api requests, partitioned by runner, endpoint and status.
- #HELP gitlab_runner_autoscaling_machine_creation_duration_seconds Histogram of machine creation time.
- #HELP gitlab_runner_autoscaling_machine_states The current number of machines per state in this provider.
- #HELP gitlab_runner_concurrent The current value of concurrent setting
- #HELP gitlab_runner_errors_total The number of caught errors.
- #HELP gitlab_runner_limit The current value of limit setting
- #HELP gitlab_runner_request_concurrency The current number of concurrent requests for a new job
- #HELP gitlab_runner_request_concurrency_exceeded_total Count of excess requests above the configured request_concurrency limit
- #HELP gitlab_runner_version_info A metric with a constant '1' value labeled by different build stats fields.

Remediation:

Set resources consumption monitoring for each build worker.

To learn how to set up a Prometheus server to scrape this HTTP endpoint and make use of the collected metrics, see Prometheus's Getting started guide. Once this is done, the following information will be exposed:

The exposed information includes:

- Runner business logic metrics (e.g., the number of currently running jobs)
- Go-specific process metrics (garbage collection stats, goroutines, memstats, etc.)
- general process metrics (memory usage, CPU usage, file descriptor usage, etc.)
- build version information

References:

1. <https://docs.gitlab.com/runner/monitoring/index.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.3 Pipeline Instructions

This section consists of security recommendations for pipeline instructions and commands.

Pipeline instructions are dedicated to taking raw files of source code and running a series of tasks on them to achieve some final artifact as output. They are most of the time written by third-party developers so they should be treated carefully and can also be vulnerable to attack in certain situations. Pipeline instructions files are considered very sensitive, and it is important to secure all their aspects - instructions, access, etc.

2.3.1 Ensure all build steps are defined as code (Manual)

Profile Applicability:

- Level 1

Description:

Use pipeline as code for build pipelines and their defined steps.

Rationale:

Storing pipeline instructions as code in a version control system means automation of the build steps and less room for human error, which could potentially lead to a security breach. Additionally, It creates the ability to revert back to a previous pipeline configuration in order to pinpoint the affected change should a malicious incident occur.

Audit:

Verify that all build steps are defined as code and stored in a version control system.

- On the left sidebar, select Code > Repository.
- Check if a .gitlab-ci.yml file is at the root of your repository

Remediation:

Convert pipeline instructions into code-based syntax and upload them to the organization's version control platform.

- On the left sidebar, select Code > Repository.
- Above the file list, select the branch you want to commit to. If you're not sure, leave master or main. Then select the plus icon (+) and New file:
- For the Filename, type .gitlab-ci.yml.
- Select Commit changes.

References:

1. <https://docs.gitlab.com/ee/ci/pipelines/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.3.2 Ensure steps have clearly defined build stage input and output (Manual)

Profile Applicability:

- Level 1

Description:

Define clear expected input and output for each build stage.

Rationale:

In order to have more control over data flow in the build pipeline, clearly define the input and output of the pipeline steps. If anything malicious happens during the build stage, it will be recognized more easily and stand out as an anomaly.

Audit:

For each build stage, verify that the expected input and output are clearly defined.

- On the left sidebar, select Code > Repository.
- Review the .gitlab-ci.yml file to check if the build stage job input and output are clearly defined.

Remediation:

For each build stage, clearly define what is expected for input and output.

- On the left sidebar, select Code > Repository.
- Ensure the .gitlab-ci.yml file has build stage job input and output clearly defined.

References:

1. <https://docs.gitlab.com/ee/ci/pipelines/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

2.3.3 Ensure output is written to a separate, secured storage repository (Manual)

Profile Applicability:

- Level 1

Description:

Write pipeline output artifacts to a secured storage repository.

Rationale:

To maintain output artifacts securely and reduce the potential surface for attack, store such artifacts separately in secure storage. This separation enforces the Single Responsibility Principle by ensuring the orchestration platform will not be the same as the artifact storage, which reduces the potential harm of an attack. Using the same security considerations as the input (for example, the source code) will protect artifacts stored and will make it harder for a malicious actor to successfully execute an attack.

Audit:

For each pipeline that produces output artifacts, ensure that they're written to a secured storage repository.

Review where job output artifacts are stored, either locally or in object storage. By default artifacts are stored locally in:

- Linux package (Omnibus): `/var/opt/gitlab/gitlab-rails/shared/artifacts`
- Self-compiled (source): `/home/git/gitlab/shared/artifacts`

Remediation:

For each pipeline that produces output artifacts, write them to a secured storage repository.

One approach is to activate object storage and use an encrypted S3 bucket (or similar).

Once the storage is configured, these are the steps to activate it for job artifacts:

Linux package (Omnibus):

- Configure the object storage.
- Migrate the artifacts: `sudo gitlab-rake gitlab:artifacts:migrate`
- Verify that there are no files on disk in the artifacts directory: `sudo find /var/opt/gitlab/gitlab-rails/shared/artifacts -type f | grep -v tmp | wc -l`

Self-compiled (source):

- Configure the object storage.
- Migrate the artifacts: `sudo -u git -H bundle exec rake gitlab:artifacts:migrate RAILS_ENV=production`
- Verify that there are no files on disk in the artifacts directory: `sudo find /home/git/gitlab/shared/artifacts -type f | grep -v tmp | wc -l`

References:

1. https://docs.gitlab.com/ee/administration/job_artifacts.html?tab=Self-compiled+%28source%29#migrating-to-object-storage

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.12 Segment Data Processing and Storage Based on Sensitivity Segment data processing and storage based on the sensitivity of the data. Do not process sensitive data on enterprise assets intended for lower sensitivity data.		●	●

2.3.4 Ensure changes to pipeline files are tracked and reviewed (Manual)

Profile Applicability:

- Level 1

Description:

Track and review changes to pipeline files.

Rationale:

Pipeline files are sensitive files. They have the ability to access sensitive data and control the build process, thus it is just as important to review changes to pipeline files as it is to verify source code. Malicious actors can potentially add harmful code to these files, which may lead to sensitive data exposure and hijacking of the build environment or artifacts.

Audit:

For each pipeline file, ensure changes to it are being tracked and reviewed.

- On the left sidebar, select Code > Repository.
- Check if a .gitlab-ci.yml file is at the root of your repository
- If it exists, all changes will be tracked via Git.
- In the upper-right corner, select History to review the history of the gitlab-ci.yml file

Remediation:

For each pipeline file, track changes to it and review them.

- On the left sidebar, select Code > Repository.
- Above the file list, select the branch you want to commit to. If you're not sure, leave master or main. Then select the plus icon (+) and New file:
- For the Filename, type .gitlab-ci.yml.
- Select Commit changes.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.14 <u>Log Sensitive Data Access</u> Log sensitive data access, including modification and disposal.			●
v7	14.9 <u>Enforce Detail Logging for Access or Changes to Sensitive Data</u> Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).			●

2.3.5 Ensure access to build process triggering is minimized (Manual)

Profile Applicability:

- Level 1

Description:

Restrict access to pipeline triggers.

Rationale:

Build pipelines are used for multiple reasons. Some are very sensitive, such as pipelines which deploy to production. In order to protect the environment from malicious acts or human mistakes, such as a developer deploying a bug to production, it is important to apply the Least Privilege principle to pipeline triggering. This principle requires restrictions placed on which users can run which pipeline. It allows for sensitive pipelines to only be run by administrators, who are generally the most trusted and skilled members of the organization.

Audit:

For every pipeline in use, verify only the necessary users have permission to trigger it.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members
- At the top of the member list, from the dropdown list, sort by Max role the members have in the group
- Review the members to ensure relevant members are allowed to trigger pipelines in protected environments.

Remediation:

For every pipeline in use, grant only the necessary users permission to trigger it.

- On the left sidebar, select Search or go to and find your project.
- Select Settings > CI/CD.
- Expand Protected environments.
- Select Protect an environment.
- From the Environment list, select the environment you want to protect.
- In the Allowed to deploy list, select the role, users, or groups you want to give deploy access to. Keep in mind that:
 - There are two roles to choose from:
 - Maintainers: Allows access to all of the project's users with the Maintainer role.
 - Developers: Allows access to all of the project's users with the Maintainer and Developer role.
 - You can only select groups that are already invited to the project.
 - Users must have at least the Developer role to appear in the Allowed to deploy list.
- In the Approvers list, select the role, users, or groups you want to give deploy access to. Keep in mind that:
 - There are two roles to choose from:
 - Maintainers: Allows access to all of the project's users with the Maintainer role.
 - Developers: Allows access to all of the project's users with the Maintainer and Developer role.
 - You can only select groups that are already invited to the project.
 - Users must have at least the Developer role to appear in the Approvers list.
- In the Approval rules section:
 - Ensure that this number is less than or equal to the number of members in the rule.
 - See Deployment Approvals for more information about this feature.
- Select Protect.

References:

1. https://docs.gitlab.com/ee/ci/environments/protected_environments.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

2.3.6 Ensure pipelines are automatically scanned for misconfigurations (Manual)

Profile Applicability:

- Level 1

Description:

Scan the pipeline for misconfigurations. It is recommended that this be performed automatically.

Rationale:

Automatic scans for misconfigurations detect human mistakes and misconfigured tasks. This protects the environment from backdoors caused by such mistakes, which create easier access for attackers. For example, a task that mistakenly configures credentials to persist on the disk makes it easier for an attacker to steal them. This type of incident can be prevented by auto-scanning.

Audit:

For each pipeline, verify that it is automatically scanned for misconfigurations.

Remediation:

For each pipeline, set automated misconfiguration scanning.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 Establish and Maintain a Secure Configuration Process Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.			

2.3.7 Ensure pipelines are automatically scanned for vulnerabilities (Manual)

Profile Applicability:

- Level 1

Description:

Scan pipelines for vulnerabilities. It is recommended that this be implemented automatically.

Rationale:

Automatic scanning for vulnerabilities detects known vulnerabilities in pipeline instructions and components, allowing faster patching in case one is found. These vulnerabilities can lead to a potentially massive breach if not handled as fast as possible, as attackers might also be aware of such vulnerabilities.

Audit:

For each pipeline, verify that it is automatically scanned for vulnerabilities.

Remediation:

For each pipeline, set automated vulnerability scanning.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	7.5 <u>Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v7	3.1 <u>Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			

2.3.8 Ensure scanners are in place to identify and prevent sensitive data in pipeline files (Automated)

Profile Applicability:

- Level 2

Description:

Detect and prevent sensitive data, such as confidential ID numbers, passwords, etc., in pipelines.

Rationale:

Sensitive data in pipeline configuration, such as cloud provider credentials or repository credentials, create vulnerabilities with which malicious actors could steal such information if they gain access to a pipeline. In order to mitigate this, set scanners that will identify and prevent the existence of sensitive data in the pipeline.

Audit:

For every pipeline that is in use, verify that scanners are set to identify and prevent the existence of sensitive data within it.

- On the left sidebar, select Search or go to and find your project.
- Select Build > Pipeline editor.
- Ensure the following lines are present in your gitlab-ci.yml.

```
include:  
- template: Jobs/Secret-Detection.gitlab-ci.yml
```

Remediation:

For every pipeline that is in use, set scanners that will identify and prevent sensitive data within it.

- On the left sidebar, select Search or go to and find your project.
- Select Build > Pipeline editor.
- Copy and paste the following to the bottom of the .gitlab-ci.yml file. If an include line already exists, add only the template line below it.

```
include:  
- template: Jobs/Secret-Detection.gitlab-ci.yml
```

- Select the Validate tab, then select Validate pipeline. The message Simulation completed successfully indicates the file is valid.
- Select the Edit tab.

- Optional. In the Commit message text box, customize the commit message. In the Branch text box, enter the name of the default branch.
- Select Commit changes.

References:

1. https://docs.gitlab.com/ee/user/application_security/secret_detection/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.14 <u>Log Sensitive Data Access</u> Log sensitive data access, including modification and disposal.			●
v7	14.5 <u>Utilize an Active Discovery Tool to Identify Sensitive Data</u> Utilize an active discovery tool to identify all sensitive information stored, processed, or transmitted by the organization's technology systems, including those located onsite or at a remote service provider and update the organization's sensitive information inventory.			●

2.4 Pipeline Integrity

This section consists of security recommendations for keeping pipeline integrity.

Integrity means ensuring that the pipelines, the dependencies they use, and their artifacts are all authentic and what they intended to be. Securing the pipeline integrity is to verify that every change and process running during the build pipeline run is what it is supposed to be. One way to do that for example is to lock each dependency to a certain secured version. It is important to insist on securing that because this is the way to set trust with the customer.

2.4.1 Ensure all artifacts on all releases are signed (Manual)

Profile Applicability:

- Level 1

Description:

Sign all artifacts in all releases with user or organization keys.

Rationale:

Signing artifacts is used to validate both their integrity and security. Organizations signal that artifacts may be trusted and they themselves produced them by ensuring that every artifact is properly signed. The presence of this signature also makes potentially malicious activity far more difficult.

Audit:

Ensure every artifact in every release is signed.

Remediation:

For every artifact in every release, verify that all are properly signed.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.4.2 Ensure all external dependencies used in the build process are locked (Manual)

Profile Applicability:

- Level 1

Description:

External dependencies may be public packages needed in the pipeline, or perhaps the public image being used for the build worker. Lock these external dependencies in every build pipeline.

Rationale:

External dependencies are sources of code that aren't under organizational control. They might be intentionally or unintentionally infected with malicious code or have known vulnerabilities, which could result in sensitive data exposure, data harvesting, or the erosion of trust in an organization. Locking each external dependency to a specific, safe version gives more control and less chance for risk.

Audit:

Ensure every external dependency being used in pipelines is locked.

- Go to Code > Repository in your project
- Review the following files at the root of your repository:
 - Gemfiles
 - package.jsons
 - go.sum
- Review the versions of the external dependencies

Remediation:

For all external dependencies being used in pipelines, verify they are locked.

- Go to Code > Repository in your project
- Review the following files at the root of your repository:
 - Gemfiles
 - package.jsons
 - go.sum
- Ensure the version of the external dependencies corresponds to your internal policies

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.5 Use Up-to-Date and Trusted Third-Party Software Components</u> Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.			
v7	<u>18.4 Only Use Up-to-date And Trusted Third-Party Components</u> Only use up-to-date and trusted third-party components for the software developed by the organization.			

2.4.3 Ensure dependencies are validated before being used (Manual)

Profile Applicability:

- Level 1

Description:

Validate every dependency of the pipeline before use.

Rationale:

To ensure that a dependency used in a pipeline is trusted and has not been infected by malicious actor (for example, the codecov incident), validate dependencies before using them. This can be accomplished by comparing the checksum of the dependency to its checksum in a trusted source. If a difference arises, this is a sign that an unknown actor has interfered and may have added malevolent code. If this dependency is used, it will infect the environment, which could end in a massive breach and leave the organization exposed to data leaks, etc.

Audit:

For every dependency used in every pipeline, ensure it has been validated.

- On the left sidebar, select Code > Repository.
- Check if a .gitlab-ci.yml file is at the root of your repository
- Verify the following job is included in your .gitlab-ci.yml file:

```
include:  
- template: Jobs/Dependency-Scanning.gitlab-ci.yml
```

- Go to Build > Pipelines and confirm that the latest pipeline completed successfully.

Remediation:

For every dependency used in every pipeline, validate each one.

- On the left sidebar, select Code > Repository.
- Check if a .gitlab-ci.yml file is at the root of your repository
- Include the following job is included in your .gitlab-ci.yml file:

```
include:
- template: Jobs/Dependency-Scanning.gitlab-ci.yml
```

- Go to Build > Pipelines and confirm that the latest pipeline completed successfully. In the pipeline, dependency scanning runs and the vulnerabilities are detected automatically.
- Go to Secure > Vulnerability report.
- Select each of the vulnerabilities by selecting the checkbox in each row.
- Review the recommended solution for each vulnerability and investigate further if needed.
- From the Set status dropdown list select the relevant option and select Change status.

References:

1. https://docs.gitlab.com/ee/tutorials/dependency_scanning.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.5 <u>Use Up-to-Date and Trusted Third-Party Software Components</u> Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.			
v7	18.4 <u>Only Use Up-to-date And Trusted Third-Party Components</u> Only use up-to-date and trusted third-party components for the software developed by the organization.			

2.4.4 Ensure the build pipeline creates reproducible artifacts (Manual)

Profile Applicability:

- Level 1

Description:

Verify that the build pipeline creates reproducible artifacts, meaning that an artifact of the build pipeline is the same in every run when given the same input.

Rationale:

A reproducible build is a build that produces the same artifact when given the same input data. Ensuring that the build pipeline produces the same artifact when given the same input helps verify that no change has been made to the artifact. This action allows an organization to trust that its artifacts are built only from safe code that has been reviewed and tested and has not been tainted or changed abruptly.

Audit:

Ensure that build pipelines create reproducible artifacts.

- On the left sidebar, select Build > Artifacts.
- Review the artifacts associated to your build pipelines

Remediation:

Create build pipelines that produce the same artifact given the same input (for example, artifacts that do not rely on timestamps).

- On the left sidebar, select Code > Repository.
- Check if a `.gitlab-ci.yml` file is at the root of your repository
- To create job artifacts, use the artifacts keyword in your `.gitlab-ci.yml` file, as in this example:

```
pdf:
  script: xelatex mycv.tex
  artifacts:
    paths:
      - mycv.pdf
```

In this example, a job named pdf calls the xelatex command to build a PDF file from the LaTeX source file, mycv.tex.

The paths keyword determines which files to add to the job artifacts. All paths to files and directories are relative to the repository where the job was created.

References:

1. https://docs.gitlab.com/ee/ci/jobs/job_artifacts.html#create-job-artifacts

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.4.5 Ensure pipeline steps produce a Software Bill of Materials (SBOM) (Manual)

Profile Applicability:

- Level 1

Description:

SBOM (Software Bill of Materials) is a file that specifies each component of software or a build process. Generate an SBOM after each run of a pipeline.

Rationale:

Generating a Software Bill of Materials after each run of a pipeline will validate the integrity and security of that pipeline. Recording every step or component role in the pipeline ensures that no malicious acts have been committed during the pipeline's run.

Audit:

For each pipeline, ensure it produces a Software Bill of Materials on every run.

- Check if your gitlab-ci.yml file contains the following:

```
include:  
  - template: Jobs/Dependency-Scanning.gitlab-ci.yml
```

If your project has the following structure:

```
.  
├── ruby-project/  
│   └── Gemfile.lock  
├── ruby-project-2/  
│   └── Gemfile.lock  
├── php-project/  
│   └── composer.lock  
└── go-project/  
    └── go.sum
```

Then the Gemnasium scanner generates the following CycloneDX SBOMs:

```
.  
├── ruby-project/  
│   ├── Gemfile.lock  
│   └── gl-sbom-gem-bundler.cdx.json  
├── ruby-project-2/  
│   ├── Gemfile.lock  
│   └── gl-sbom-gem-bundler.cdx.json  
├── php-project/  
│   ├── composer.lock  
│   └── gl-sbom-packagist-composer.cdx.json  
└── go-project/  
    ├── go.sum  
    └── gl-sbom-go-go.cdx.json
```

- Review if these files exists in your environment

Remediation:

For each pipeline, configure it to produce a Software Bill of Materials on every run.

- On the left sidebar, select Search or go to and find your project.
- Select Build > Pipeline editor.
- If no .gitlab-ci.yml file exists, select Configure pipeline, then delete the example content.
- Copy and paste the following to the bottom of the .gitlab-ci.yml file. If an include line already exists, add only the template line below it.

```
include:  
- template: Jobs/Dependency-Scanning.gitlab-ci.yml
```

- Select the Validate tab, then select Validate pipeline. The message Simulation completed successfully confirms the file is valid.
- Select the Edit tab.
- Complete the fields. Do not use the default branch for the Branch field.
- Select the Start a new merge request with these changes checkbox, then select Commit changes.
- Complete the fields according to your standard workflow, then select Create merge request.
- Review and edit the merge request according to your standard workflow, then select Merge.

The CycloneDX SBOMs are:

- Named gl-sbom--.cdx.json.
- Available as job artifacts of the dependency scanning job.
- Saved in the same directory as the detected lock or build files.

References:

1. https://docs.gitlab.com/ee/user/application_security/dependency_scanning/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

2.4.6 Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced (Manual)

Profile Applicability:

- Level 1

Description:

SBOM (Software Bill of Materials) is a file that specifies each component of software or a build process. It should be generated after every pipeline run. After it is generated, it must then be signed.

Rationale:

Software Bill of Materials (SBOM) is a file used to validate the integrity and security of a build pipeline. Signing it ensures that no one tampered with the file when it was delivered. Such interference can happen if someone tries to hide unusual activity. Validating the SBOM signature can detect this activity and prevent much greater incident.

Audit:

For each pipeline, ensure it signs the Software Bill of Materials it produces on every run.

Remediation:

For each pipeline, configure it to sign its produced Software Bill of Materials on every run.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

3 Dependencies

This section consists of security recommendations for the management of various dependencies introduced as part of the software build and release process. These are comprised of anything that goes into application code or is used by build pipelines themselves.

Dependencies are a huge part of the software supply chain, as they are integrated in a lot of important phases. They are often written by third-party developers and might be vulnerable to certain attacks, for example the log4j attack. Because of that it is particularly important to secure them and their use in the supply chain.

3.1 Third-Party Packages

This section consists of security recommendations for the use and management of third-party dependencies and packages. As a consumer of various third-party packages, you need to ensure certain conditions exist to trust them and use them safely. Using third-party packages affects not only the software, but also its costumers, so it is important to carefully examine each one of these packages.

3.1.1 Ensure third-party artifacts and open-source libraries are verified (Manual)

Profile Applicability:

- Level 1

Description:

Ensure third-party artifacts and open-source libraries in use are trusted and verified.

Rationale:

Verify third-party artifacts used in code are trusted and have not been infected by a malicious actor before use. This can be accomplished, for example, by comparing the checksum of the dependency to its checksum in a trusted source. If a difference arises, this may be a sign that someone interfered and added malicious code. If this dependency is used, it will infect the environment and could end in a massive breach, leaving the organization exposed to data leaks and more.

Audit:

Ensure third-party artifacts and open-source libraries are verified.

Remediation:

Set third-party artifacts and open-source libraries to be verified.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.5 Use Up-to-Date and Trusted Third-Party Software Components</u> Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.			
v7	<u>18.4 Only Use Up-to-date And Trusted Third-Party Components</u> Only use up-to-date and trusted third-party components for the software developed by the organization.			

3.1.2 Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers (Manual)

Profile Applicability:

- Level 1

Description:

A Software Bill Of Materials (SBOM) is a file that specifies each component of software or a build process. Require an SBOM from every third-party provider.

Rationale:

A Software Bill of Materials (SBOM) for every third-party artifact helps to ensure an artifact is safe to use and fully compliant. This file lists all important metadata, especially all the dependencies of an artifact, and allows for verification of each dependency. If one of the dependencies/artifacts are attacked or has a new vulnerability (for example, the "SolarWinds" or even "log4j" attacks), it is easier to detect what has been affected by this incident because dependencies in use are listed in the SBOM file.

Audit:

For every third-party dependency in use, ensure it has a Software Bill of Materials.

Remediation:

For every third-party dependency in use, require a Software Bill of Materials from its supplier.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>2.1 Establish and Maintain a Software Inventory</u> Establish and maintain a detailed inventory of all licensed software installed on enterprise assets. The software inventory must document the title, publisher, initial install/use date, and business purpose for each entry; where appropriate, include the Uniform Resource Locator (URL), app store(s), version(s), deployment mechanism, and decommission date. Review and update the software inventory bi-annually, or more frequently.	●	●	●
v8	<u>16.5 Use Up-to-Date and Trusted Third-Party Software Components</u> Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.		●	●
v7	<u>2.4 Track Software Inventory Information</u> The software inventory system should track the name, version, publisher, and install date for all software, including operating systems authorized by the organization.		●	●
v7	<u>18.4 Only Use Up-to-date And Trusted Third-Party Components</u> Only use up-to-date and trusted third-party components for the software developed by the organization.		●	●

3.1.3 Ensure signed metadata of the build process is required and verified (Manual)

Profile Applicability:

- Level 1

Description:

Require and verify signed metadata of the build process for all dependencies in use.

Rationale:

The metadata of a build process lists every action that took place during an artifact build. It is used to ensure that an artifact has not been compromised during the build, that no malicious code was injected into it, and that no nefarious dependencies were added during the build phase. This creates trust between user and vendor that the software supplied is exactly the software that was promised. Signing this metadata adds a checksum to ensure there have been no revisions since its creation, as this checksum changes when the metadata is altered. Verification of proper metadata signature with Certificate Authority confirms that the signature was produced by a trusted entity.

Audit:

For each artifact used, ensure it was supplied with verified and signed metadata of its build process. The signature should be the organizational signature and should be verifiable by common Certificate Authority servers.

Remediation:

For each artifact in use, require and verify signed metadata of the build process.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

3.1.4 Ensure dependencies are monitored between open-source components (Manual)

Profile Applicability:

- Level 1

Description:

Monitor, or ask software suppliers to monitor, dependencies between open-source components in use.

Rationale:

Monitoring dependencies between open-source components helps to detect if software has fallen victim to attack on a common open-source component. Swift detection can aid in quick application of a fix. It also helps find potential compliance problems with components usage. Some dependencies might not be compatible with the organization's policies, and other dependencies might have a license that is not compatible with how the organization uses this specific dependency. If dependencies are monitored, such situations can be detected and mitigated sooner, potentially deterring malicious attacks.

Audit:

For each project, ensure that dependency scanning and container scanning are enabled in order to monitor dependencies.

1. On GitLab, navigate to the main page of the repository.
2. Review the CI pipeline configuration to verify that Dependency Scanning and that Container Scanning have been configured to run on this project.

Remediation:

For every repository that is in use, set a dependency scanning and container scanning tools to detect, prevent, and monitor vulnerabilities in project packages and container images by performing the following:

1. On GitLab, navigate to the main page of the repository.
2. Configure Dependency Scanning and Container Scanning to run on this project

References:

1. https://docs.gitlab.com/ee/user/application_security/dependency_scanning/#configuration
2. https://docs.gitlab.com/ee/user/application_security/container_scanning/#configuration

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>2.4 Utilize Automated Software Inventory Tools</u> Utilize software inventory tools, when possible, throughout the enterprise to automate the discovery and documentation of installed software.			
v7	<u>2.3 Utilize Software Inventory Tools</u> Utilize software inventory tools throughout the organization to automate the documentation of all software on business systems.			
v7	<u>2.4 Track Software Inventory Information</u> The software inventory system should track the name, version, publisher, and install date for all software, including operating systems authorized by the organization.			

3.1.5 Ensure trusted package managers and repositories are defined and prioritized (Manual)

Profile Applicability:

- Level 1

Description:

Prioritize trusted package registries over others when pulling a package.

Rationale:

When pulling a package by name, the package manager might look for it in several package registries, some of which may be untrusted or badly configured. If the package is pulled from such a registry, there is a higher likelihood that it could prove malicious. In order to avoid this, configure packages to be pulled from trusted package registries.

Audit:

For each package registry in use, ensure it is trusted. The GitLab package registry is enabled by default.

Remediation:

For each package to be downloaded, configure it to be downloaded from a trusted source. To view your GitLab package registry and its contents, click on "Deploy" --> "Package Registry."

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 <u>Allowlist Authorized Software</u> Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.			
v8	2.6 <u>Allowlist Authorized Libraries</u> Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.			
v8	2.7 <u>Allowlist Authorized Scripts</u> Use technical controls, such as digital signatures and version control, to ensure that only authorized scripts, such as specific .ps1, .py, etc., files, are allowed to execute. Block unauthorized scripts from executing. Reassess bi-annually, or more frequently.			
v7	2.7 <u>Utilize Application Whitelisting</u> Utilize application whitelisting technology on all assets to ensure that only authorized software executes and all unauthorized software is blocked from executing on assets.			
v7	2.8 <u>Implement Application Whitelisting of Libraries</u> The organization's application whitelisting software must ensure that only authorized software libraries (such as *.dll, *.ocx, *.so, etc) are allowed to load into a system process.			
v7	2.9 <u>Implement Application Whitelisting of Scripts</u> The organization's application whitelisting software must ensure that only authorized, digitally signed scripts (such as *.ps1, *.py, macros, etc) are allowed to run on a system.			

3.1.6 Ensure a signed Software Bill of Materials (SBOM) of the code is supplied (Manual)

Profile Applicability:

- Level 1

Description:

A Software Bill of Materials (SBOM) is a file that specifies each component of software or a build process. When using a dependency, demand its SBOM and ensure it is signed for validation purposes.

Rationale:

A Software Bill of Materials (SBOM) creates trust between its provider and its users by ensuring that the software supplied is the software described, without any potential interference in between. Signing an SBOM creates a checksum for it, which will change if the SBOM's content was changed. With that checksum, a software user can be certain nothing had happened to it during the supply chain, engendering trust in the software. When there is no such trust in the software, the risk surface is increased because one cannot know if the software is potentially vulnerable. Demanding a signed SBOM and validating it decreases that risk.

Audit:

For every artifact supplied, ensure it has a validated, signed Software Bill of Materials. To view the SBOMs associated with your projects, Navigate to Secure --> Dependency list
You may export the SBOM in .json format.

Remediation:

For every artifact supplied, require and verify a signed Software Bill of Materials from its supplier.

1. To create an SBOM, you must run a dependency scan.
2. Enable dependency scanning, and an SBOM will be automatically generated for your project.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>2.1 Establish and Maintain a Software Inventory</u> Establish and maintain a detailed inventory of all licensed software installed on enterprise assets. The software inventory must document the title, publisher, initial install/use date, and business purpose for each entry; where appropriate, include the Uniform Resource Locator (URL), app store(s), version(s), deployment mechanism, and decommission date. Review and update the software inventory bi-annually, or more frequently.			
v8	<u>2.2 Ensure Authorized Software is Currently Supported</u> Ensure that only currently supported software is designated as authorized in the software inventory for enterprise assets. If software is unsupported, yet necessary for the fulfillment of the enterprise's mission, document an exception detailing mitigating controls and residual risk acceptance. For any unsupported software without an exception documentation, designate as unauthorized. Review the software list to verify software support at least monthly, or more frequently.			
v7	<u>2.1 Maintain Inventory of Authorized Software</u> Maintain an up-to-date list of all authorized software that is required in the enterprise for any business purpose on any business system.			
v7	<u>2.2 Ensure Software is Supported by Vendor</u> Ensure that only software applications or operating systems currently supported by the software's vendor are added to the organization's authorized software inventory. Unsupported software should be tagged as unsupported in the inventory system.			

3.1.7 Ensure dependencies are pinned to a specific, verified version (Manual)

Profile Applicability:

- Level 1

Description:

Pin dependencies to a specific version. Avoid using the "latest" tag or broad version.

Rationale:

When using a wildcard version of a package, or the "latest" tag, the risk of encountering a new, potentially malicious package increases. The "latest" tag pulls the last package pushed to the registry. This means that if an attacker pushes a new, malicious package successfully to the registry, the next user who pulls the "latest" will pull it and risk attack. This same rule applies to a wildcard version - assuming one is using version v1.*, it will install the latest version of the major version 1, meaning that if an attacker can push a malicious package with that same version, those using it will be subject to possible attack. By using a secure, verified version, use is restricted to this version only and no other may be pulled, decreasing the risk for any malicious package.

Audit:

For every dependency in use, ensure it is pinned to a specific version.

Remediation:

For every dependency in use, pin to a specific version.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.5 Use Up-to-Date and Trusted Third-Party Software Components Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.			
v7	18.4 Only Use Up-to-date And Trusted Third-Party Components Only use up-to-date and trusted third-party components for the software developed by the organization.			

3.1.8 Ensure all packages used are more than 60 days old (Manual)

Profile Applicability:

- Level 2

Description:

Use packages that are more than 60 days old.

Rationale:

Third-party packages are a major risk since an organization cannot control their source code, and there is always the possibility these packages could be malicious. It is therefore good practice to remain cautious with any third-party or open-source package, especially new ones, until they can be verified that they are safe to use. Avoiding a new package allows the organization to fully examine it, its maintainer, and its behavior, and gives enough time to determine whether or not to use it.

Impact:

Developers may not use packages that are less than 60 days old.

Audit:

For every package used, ensure it is more than 60 days old.

Remediation:

If a package used is less than 60 days old, stop using it and find another solution.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>2.5 Allowlist Authorized Software</u> Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.			
v8	<u>2.6 Allowlist Authorized Libraries</u> Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.			
v8	<u>2.7 Allowlist Authorized Scripts</u> Use technical controls, such as digital signatures and version control, to ensure that only authorized scripts, such as specific .ps1, .py, etc., files, are allowed to execute. Block unauthorized scripts from executing. Reassess bi-annually, or more frequently.			
v8	<u>16.5 Use Up-to-Date and Trusted Third-Party Software Components</u> Use up-to-date and trusted third-party software components. When possible, choose established and proven frameworks and libraries that provide adequate security. Acquire these components from trusted sources or evaluate the software for vulnerabilities before use.			
v7	<u>2.7 Utilize Application Whitelisting</u> Utilize application whitelisting technology on all assets to ensure that only authorized software executes and all unauthorized software is blocked from executing on assets.			
v7	<u>2.8 Implement Application Whitelisting of Libraries</u> The organization's application whitelisting software must ensure that only authorized software libraries (such as *.dll, *.ocx, *.so, etc) are allowed to load into a system process.			
v7	<u>2.9 Implement Application Whitelisting of Scripts</u> The organization's application whitelisting software must ensure that only authorized, digitally signed scripts (such as *.ps1, *.py, macros, etc) are allowed to run on a system.			

3.2 Validate Packages

This section consists of security recommendations for managing package validations and checks. Third-party packages and dependencies might put the organization in danger, not only by being vulnerable to attacks, but also by being improperly used and harming license conditions. To protect the software supply chain from these dangers, it is important to validate packages and understand how and if to use them. This section's recommendations cover this topic.

3.2.1 Ensure an organization-wide dependency usage policy is enforced (Manual)

Profile Applicability:

- Level 1

Description:

Enforce a policy for dependency usage across the organization. For example, disallow the use of packages less than 60 days old.

Rationale:

Enforcing a policy for dependency usage in an organization helps to manage dependencies across the organization and ensure that all usage is compliant with security policy. If, for example, the policy limits the package managers that can be used, enforcing it will make sure that every dependency is installed only from these package managers, and limit the risk of installing from any untrusted source.

Audit:

Verify that a policy for dependency usage is enforced across the organization.

Remediation:

Enforce policies for dependency usage across the organization.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

3.2.2 *Ensure packages are automatically scanned for known vulnerabilities (Manual)*

Profile Applicability:

- Level 1

Description:

Automatically scan every package for vulnerabilities.

Rationale:

Automatic scanning for vulnerabilities detects known vulnerabilities in packages and dependencies in use, allowing faster patching when one is found. Such vulnerabilities can lead to a massive breach if not handled as fast as possible, as attackers will also know about those vulnerabilities and swiftly try to take advantage of them. Scanning packages regularly for vulnerabilities can also verify usage compliance with the organization's security policy.

Audit:

Once Dependency Scanning is enabled for your project, continuous scanning of packages for dependency vulnerabilities is enabled.

Remediation:

Enable Dependency scanning in order to automatically scan packages for vulnerabilities. To enable the analyzer, either:

- Enable Auto DevOps, which includes dependency scanning.
- Edit the .gitlab-ci.yml file manually. Use this method if your .gitlab-ci.yml file is complex.
- Use a preconfigured merge request.
- Create a scan execution policy that enforces dependency scanning.

References:

1. https://docs.gitlab.com/ee/user/application_security/continuous_vulnerability_scanning/
2. https://docs.gitlab.com/ee/user/application_security/dependency_scanning/index.html#enabling-the-analyzer

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			

3.2.3 Ensure packages are automatically scanned for license implications (Manual)

Profile Applicability:

- Level 1

Description:

A software license is a document that provides legal conditions and guidelines for the use and distribution of software, usually defined by the author. It is recommended to scan for any legal implications automatically.

Rationale:

When using packages with software licenses, especially commercial ones which tend to be the strictest, it is important to verify that the use of the package meets the conditions of the license. If the use of the package violates the licensing agreement, it exposes the organization to possible lawsuits. Scanning used packages for such license implications leads to faster detection and quicker fixes of such violations, and also reduces the risk for a lawsuit.

Audit:

Ensure license implication rules are configured and are scanned automatically. Once Dependency Scanning is enabled for your project, continuous license scanning is enabled.

Remediation:

Enable Dependency scanning in order to automatically scan packages for license implications. To enable the analyzer, either:

- Enable Auto DevOps, which includes dependency scanning.
- Edit the `.gitlab-ci.yml` file manually. Use this method if your `.gitlab-ci.yml` file is complex.
- Use a preconfigured merge request.
- Create a scan execution policy that enforces dependency scanning.

References:

1. https://docs.gitlab.com/ee/user/compliance/license_scanning_of_cyclonedx_files/index.html
2. https://docs.gitlab.com/ee/user/application_security/dependency_scanning/index.html#enabling-the-analyzer

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

3.2.4 Ensure packages are automatically scanned for ownership change (Manual)

Profile Applicability:

- Level 1

Description:

Scan every package automatically for ownership change.

Rationale:

A change in package ownership is not a regular action. In some cases it can lead to a massive problem (for example, the "event-stream" incident). Open-source contributors are not always trusted, since by its very nature everyone can contribute. This means malicious actors can become contributors as well. Package maintainers might transfer their ownership to someone they do not know if maintaining the package is too much for them, in some cases without the other user's knowledge. This has led to known security breaches in the past. It is best to be aware of such activity as soon as it happens and to carefully examine the situation before continuing using the package in order to determine its safety.

Audit:

Ensure automatic scanning of packages for ownership change is set.

Remediation:

Set automatic scanning of packages for ownership change.

References:

1. <https://blog.npmjs.org/post/182828408610/the-security-risks-of-changing-package-owners.html>
2. <https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

4 Artifacts

This section consists of security recommendations for the management of artifacts produced by build pipelines, as well as ones used by the application in the build process itself.

Artifacts are packaged versions of software. They are stored in package registries (or artifact managers) and require securing from the moment they are created, through the time they are copied and updated, and up to deployment to their relevant environment.

4.1 Verification

This section consists of security recommendations for managing verification of artifacts.

When build artifacts are being pushed to the registry, a lot of different attacks can happen: a malicious artifact with the same name can be pushed, the artifact can be stolen over the network or if the registry is hacked, etc. It is important to secure artifacts by ensuring various verification methods, listed in the recommendations in this section, are available.

4.1.1 Ensure all artifacts are signed by the build pipeline itself (Manual)

Profile Applicability:

- Level 2

Description:

Configure the build pipeline to sign every artifact it produces and verify that each artifact has the appropriate signature.

Rationale:

A cryptographic signature can be used to verify artifact authenticity. The signature created with a certain key is unique and not reversible, thus making it unique to the author. This means that an attacker tampering with a signed artifact will be noticed immediately using a simple verification step because the signature will change. Signing artifacts by the build pipeline that produces them ensures the integrity of those artifacts.

Audit:

Verify that the build pipeline signs every new artifact it produces and all artifacts are signed.

There are many different signing tools or options each have their own method or commands to verify that the code or package created is signed.

Remediation:

Sign every artifact produced with the build pipeline that created it. Configure the build pipeline to sign each artifact.

Default Value:

Artifacts are not signed by Default.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			
v7	0.0 <u>Explicitly Not Mapped</u> Explicitly Not Mapped			

4.1.2 Ensure artifacts are encrypted before distribution (Manual)

Profile Applicability:

- Level 2

Description:

Encrypt artifacts before they are distributed and ensure only trusted platforms have decryption capabilities.

Rationale:

Build artifacts might contain sensitive data such as production configurations. In order to protect them and decrease the risk for breach, it is recommended to encrypt them before delivery. Encryption makes data unreadable, so even if attackers gain access to these artifacts, they won't be able to harvest sensitive data from them without the decryption key.

Audit:

Ensure every artifact is encrypted before it is delivered.

Remediation:

Encrypt every artifact before distribution.

Default Value:

Artifacts do not get encrypted by default.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 <u>Encrypt Sensitive Data at Rest</u> Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.			
v7	14.8 <u>Encrypt Sensitive Information at Rest</u> Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.			

4.1.3 Ensure only authorized platforms have decryption capabilities of artifacts (Manual)

Profile Applicability:

- Level 2

Description:

Grant decryption capabilities of artifacts only to trusted and authorized platforms.

Rationale:

Build artifacts might contain sensitive data such as production configuration. To protect them and decrease the risk of a breach, it is recommended to encrypt them before delivery. This will make them unreadable for every unauthorized user who doesn't have the decryption key. By implementing this, the decryption capabilities become overly sensitive in order to prevent a data leak or theft. Ensuring that only trusted and authorized platforms can decrypt the organization's packages decreases the possibility for an attacker to gain access to the critical data in artifacts.

Audit:

Ensure only trusted and authorized platforms have decryption capabilities of the organization's artifacts.

Remediation:

Grant decryption capabilities of the organization's artifacts only for trusted and authorized platforms.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 <u>Allowlist Authorized Software</u> Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.			
v8	2.6 <u>Allowlist Authorized Libraries</u> Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.			
v8	2.7 <u>Allowlist Authorized Scripts</u> Use technical controls, such as digital signatures and version control, to ensure that only authorized scripts, such as specific .ps1, .py, etc., files, are allowed to execute. Block unauthorized scripts from executing. Reassess bi-annually, or more frequently.			

4.2 Access to Artifacts

This section consists of security recommendations for access management of artifacts.

Artifacts are often stored in registries, some external and some internal. Those registries have user entities that control access and permissions. Artifacts are considered sensitive, because they are being delivered to the customer, and are prone to many attacks: data theft, dependency confusion, malicious packages and more. That's why their access management should be restrictive and careful.

4.2.1 Ensure the authority to certify artifacts is limited (Manual)

Profile Applicability:

- Level 1

Description:

Software certification is used to verify the safety of certain software usage and to establish trust between the supplier and the consumer. Any artifact can be certified. Limit the authority to certify different artifacts.

Rationale:

Artifact certification is a powerful tool in establishing trust. Clients use a software certificate to verify that the artifact is safe to use according to their security policies. Because of this, certifying artifacts is considered sensitive. If an artifact is for debugging or internal use, or if it were compromised, the organization would not want certification. An attacker gaining access to both certificate authority and the artifact registry might also be able to certify its own artifact and cause a major breach. To prevent these issues, limit which artifacts can be certified by which platform so there will be minimal access to certification.

Audit:

Ensure only certain artifacts can be certified by certain parties.

Remediation:

Limit which artifact can be certified by which authority.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

4.2.2 Ensure number of permitted users who may upload new artifacts is minimized (Manual)

Profile Applicability:

- Level 1

Description:

Minimize ability to upload artifacts to the lowest number of trusted users possible.

Rationale:

Artifacts might contain sensitive data. Even the simplest mistake can also lead to trust issues with customers and harm the integrity of the product. To decrease these risks, allow only trusted and qualified users to upload new artifacts. Those users are less likely to make mistakes. Having the lowest number of such users possible will also decrease the risk of hacked user accounts, which could lead to a massive breach or artifact compromising.

Audit:

Ensure only trusted and qualified users can upload new artifacts, and that their number is the lowest possible.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- If there are minimum number of members with Owner/Maintainer role in the list, you are compliant.

Remediation:

Allow only trusted and qualified users to upload new artifacts and limit them in number.

- On the left sidebar, select Search or go to and find your project.
- Select Manage > Members.
- At the top of the member list, from the dropdown list, select Max role the members have in the group and descending order.
- Next to the project member you want to remove, select Remove member.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

4.2.3 Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA) (Manual)

Profile Applicability:

- Level 2

Description:

Enforce Multi-Factor Authentication (MFA) for user access to the package registry.

Rationale:

By default, every user authenticates to the system by password only. If a user's password is compromised, the user account and all its related packages are in danger of data theft and malicious builds. It is therefore recommended that each user enables Multi-Factor Authentication. This additional step guarantees that the account stays secure even if the user's password is compromised, as it adds another layer of authentication.

Audit:

For each package registry in use, verify that Multi-Factor Authentication is enforced and is the only way to authenticate.

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > General.
- Expand Sign-in restrictions:
- Check if Enforce two-factor authentication is enabled. If so, you are compliant.

Remediation:

For each package registry in use, enforce Multi-Factor Authentication as the only way to authenticate.

- On the left sidebar, at the bottom, select Admin Area.
- Select Settings > General.
- Expand Sign-in restrictions:
- Select Enforce two-factor authentication to enable this feature.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.3 <u>Require MFA for Externally-Exposed Applications</u> Require all externally-exposed enterprise or third-party applications to enforce MFA, where supported. Enforcing MFA through a directory service or SSO provider is a satisfactory implementation of this Safeguard.			
v7	4.5 <u>Use Multifactor Authentication For All Administrative Access</u> Use multi-factor authentication and encrypted channels for all administrative account access.			

4.2.4 Ensure user management of the package registry is not local (Manual)

Profile Applicability:

- Level 1

Description:

Manage users and their access to the package registry with an external authentication server and not with the package registry itself.

Rationale:

Some package registries offer a tool for user management, aside from the main Lightweight Directory Access Protocol (LDAP) or Active Directory (AD) server of the organization. That tool usually offers simple authentication and role-based permissions, which might not be granular enough. Having multiple user management tools in the organization could result in confusion and privilege escalation, as there will be more to manage. To avoid a situation where users escalate their privileges because someone missed them, manage user access to the package registry via the main authentication server and not locally on the package registry.

Audit:

For each package registry, verify that its user access is not managed locally, but instead with the main authentication server of the organization.

Remediation:

For each package registry, use the main authentication server of the organization for user management and do not manage locally.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

4.2.5 Ensure anonymous access to artifacts is revoked (Manual)

Profile Applicability:

- Level 1

Description:

For GitLab projects anonymous access is not available. Verify that all that require access controls are Private or Internal.

Rationale:

Disable the option to view artifacts as an anonymous user in order to protect private artifacts from being exposed.

Impact:

Only logged and authorized users will be able to access artifacts.

Audit:

Reviewing a project's visibility for artifacts:

- On the left sidebar, select Search or go to and find your project.
- Select Settings > General.
- Expand Visibility, project features, permissions.
- From the Project visibility dropdown list, review the current selection.

Remediation:

Changing a project's visibility for artifacts:

- On the left sidebar, select Search or go to and find your project.
- Select Settings > General.
- Expand Visibility, project features, permissions.
- From the Project visibility dropdown list, select an option. The visibility setting for a project must be at least as restrictive as the visibility of its parent group.
- Select Save changes.

References:

1. https://docs.gitlab.com/ee/user/public_access.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.5 <u>Allowlist Authorized Software</u> Use technical controls, such as application allowlisting, to ensure that only authorized software can execute or be accessed. Reassess bi-annually, or more frequently.			
v8	2.6 <u>Allowlist Authorized Libraries</u> Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

4.2.6 Ensure minimum number of administrators are set for the package registry (Manual)

Profile Applicability:

- Level 1

Description:

Ensure the package registry has a minimum number of administrators.

Rationale:

Package registry admins have the ability to add/remove users, repositories, packages. Due to the permissive access granted to an admin, it is highly recommended to keep the number of administrator accounts as minimal as possible.

Impact:

Administrator privileges are required to provide and maintain a secure and stable platform but allowing extraneous administrator accounts can create a vulnerability.

Audit:

Verify that your package registry has only the minimum number of administrators. For each project that you administer on GitLab, you can see every team or person with access to the project. Project-level permissions determine actions such as downloading, pushing, or deleting packages.

Remediation:

Set the minimum number of administrators in your package registry. To accomplish this: For each project that you administer on GitLab, you can see an overview of every team or person with access to the repository. Provide access to the appropriate people or teams.

References:

1. https://docs.gitlab.com/ee/user/packages/package_registry

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

4.3 Package Registries

This section consists of security recommendations for management of package registries and artifacts that are stored in them.

Package registries are where the organization artifacts are stored. To keep an artifact safe, you must keep the registry where it is stored safe too. furthermore, you need to ensure that every artifact that reaches the registry is safe to use and doesn't put the registry in danger.

4.3.1 Ensure all signed artifacts are validated upon uploading the package registry (Manual)

Profile Applicability:

- Level 1

Description:

Validate artifact signatures before uploading to the package registry.

Rationale:

Cryptographic signature is a tool to verify artifact authenticity. Every artifact is supposed to be signed by its creator in order to confirm that it was not compromised before reaching the client. Validating an artifact signature before delivering it is another level of protection which ensures the signature has not been changed, meaning no one tried or succeeded in tampering with the artifact. This creates trust between the supplier and the client.

Audit:

Ensure every artifact in the package registry has been validated with its signature.

1. On the left sidebar, select Search or go to and find your project.
 - To review commits for a project, select Code > Commits.
 - To review commits for a merge request, select Code > Merge requests, then select your merge request.
2. Select Commits.
3. Identify the commit you want to review. Signed commits show either a Verified or Unverified badge, depending on the verification status of the signature. Unsigned commits do not display a badge.

Remediation:

Validate every artifact with its signature before uploading it to the package registry. It is recommended to do so automatically.

Default Value:

Artifacts are not scanned by default.

References:

1. https://docs.gitlab.com/ee/user/project/repository/signed_commits/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			
v7	<u>3.2 Perform Authenticated Vulnerability Scanning</u> Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.			

4.3.2 Ensure all versions of an existing artifact have their signatures validated (Manual)

Profile Applicability:

- Level 1

Description:

Validate the signature of all versions of an existing artifact.

Rationale:

In order to be certain a version of an existing and trusted artifact is not malicious or delivered by someone looking to interfere with the supply chain, it is a good practice to validate the signatures of each version. Doing so decreases the risk of using a compromised artifact, which might lead to a breach.

Audit:

For each artifact, ensure that all of its versions are signed and validated before it is uploaded or used.

Ensure every artifact in the package registry has been validated with its signature.

Ensure every artifact in the package registry has been validated with its signature.

1. On the left sidebar, select Search or go to and find your project.
 - To review commits for a project, select Code > Commits.
 - To review commits for a merge request, select Code > Merge requests, then select your merge request.
2. Select Commits.
3. Identify the commit you want to review. Signed commits show either a Verified or Unverified badge, depending on the verification status of the signature. Unsigned commits do not display a badge.

Remediation:

For each artifact, sign and validate each version before uploading or using the artifact.

References:

1. https://docs.gitlab.com/ee/user/project/repository/signed_commits/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>7.6 Perform Automated Vulnerability Scans of Externally-Exposed Enterprise Assets</u> Perform automated vulnerability scans of externally-exposed enterprise assets using a SCAP-compliant vulnerability scanning tool. Perform scans on a monthly, or more frequent, basis.			
v7	<u>3.1 Run Automated Vulnerability Scanning Tools</u> Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			
v7	<u>3.2 Perform Authenticated Vulnerability Scanning</u> Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.			

4.3.3 Ensure changes in package registry configuration are audited (Manual)

Profile Applicability:

- Level 1

Description:

Audit changes of the package registry configuration.

Rationale:

The package registry is a crucial component in the software supply chain. It stores artifacts with potentially sensitive data that will eventually be deployed and used in production. Every change made to the package registry configuration must be examined carefully to ensure no exposure of the registry's sensitive data. This examination also ensures no malicious actors have performed modifications to a stored artifact. Auditing the configuration and its changes helps in decreasing such risks.

Audit:

Verify that all changes to the packages registry configuration are audited.

Remediation:

Audit the changes to the package registry configuration.

References:

1. https://docs.gitlab.com/ee/administration/audit_event_types.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.		●	●
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.		●	●

4.3.4 Ensure webhooks of the repository are secured (Manual)

Profile Applicability:

- Level 1

Description:

Use secured webhooks to reduce the possibility of malicious payloads.

Rationale:

Webhooks are used for triggering an HTTP request based on an action made in the platform. Typically, package registries feature webhooks when a package receives an update. Since webhooks are an HTTP POST request, they can be malformed if not secured over SSL. To prevent a potential hack and compromise of the webhook or to the registry or web server excepting the request, use only secured webhooks.

Impact:

Reduces the payloads that the web hook can listen for and receive.

Audit:

Remediation:

For each webhook in use, change it to secured (over HTTPS).

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			
v8	12.6 <u>Use of Secure Network Management and Communication Protocols</u> Use secure network management and communication protocols (e.g., 802.1X, Wi-Fi Protected Access 2 (WPA2) Enterprise or greater).			
v7	14.4 <u>Encrypt All Sensitive Information in Transit</u> Encrypt all sensitive information in transit.			

4.4 Origin Traceability

This section consists of security recommendations for managing the traceability of artifacts. This means ensuring that both the organization and its customers know where this artifact came from, for example with an SBOM (Software Bill Of Materials), and also verifying that it came from the registry it was supposed.

4.4.1 Ensure artifacts contain information about their origin (Manual)

Profile Applicability:

- Level 1

Description:

When delivering artifacts, ensure they have information about their origin. This may be done by providing a Software Bill of Manufacture (SBOM) or some metadata files.

Rationale:

Information about artifact origin can be used for verification purposes. Having this kind of information allows the user to decide if the organization supplying the artifact is trusted. In a case of potential vulnerability or version update, this can be used to verify that the organization issuing it is the actual origin and not someone else. If users need to report problems with the artifact, they will have an address to contact as well.

Audit:

For each artifact, ensure it has information about its origin.

Remediation:

For each artifact supplied, supply information about its origin. For each artifact in use, ask for information about its origin.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>1.1 Establish and Maintain Detailed Enterprise Asset Inventory</u> Establish and maintain an accurate, detailed, and up-to-date inventory of all enterprise assets with the potential to store or process data, to include: end-user devices (including portable and mobile), network devices, non-computing/IoT devices, and servers. Ensure the inventory records the network address (if static), hardware address, machine name, enterprise asset owner, department for each asset, and whether the asset has been approved to connect to the network. For mobile end-user devices, MDM type tools can support this process, where appropriate. This inventory includes assets connected to the infrastructure physically, virtually, remotely, and those within cloud environments. Additionally, it includes assets that are regularly connected to the enterprise's network infrastructure, even if they are not under control of the enterprise. Review and update the inventory of all enterprise assets bi-annually, or more frequently.			
v7	<u>1.5 Maintain Asset Inventory Information</u> Ensure that the hardware asset inventory records the network address, hardware address, machine name, data asset owner, and department for each asset and whether the hardware asset has been approved to connect to the network.			

5 Deployment

This section consists of security recommendations for management of the release process, the application deployment, and the configuration files that comes with it.

This is the final phase of the software supply chain. After that, the client already uses the application, and it is running in production. This phase contains the deployment orchestrator, the deployment configuration, the manifest files, and the deployment environment. It is important to secure all of these to deliver the software to the client safely.

5.1 Deployment Configuration

This section consists of security recommendations for the management of the deployment configuration. This consists of the files, instructions, and access management of the deployment configuration. Usually, the configuration files are stored in a version control system, so they need to be protected in it as well.

5.1.1 Ensure deployment configuration files are separated from source code (Manual)

Profile Applicability:

- Level 2

Description:

Deployment configurations are often stored in a version control system. Separate deployment configuration files from source code repositories.

Rationale:

Deployment configuration manifests are often stored in version control systems. Storing them in dedicated repositories, separately from source code repositories, has several benefits. First, it adds order to both maintenance and version control history. This makes it easier to track code or manifest changes, as well as spot any malicious code or misconfigurations. Second, it helps achieve the Least Privilege principle. Because access can be configured differently for each repository, fewer users will have access to this configuration, which is typically sensitive.

Audit:

Ensure each deployment configuration file is stored separately from source code.

Remediation:

Store each deployment configuration file in a dedicated repository separately from source code.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.1.2 Ensure changes in deployment configuration are audited (Manual)

Profile Applicability:

- Level 1

Description:

Audit and track changes made in deployment configuration.

Rationale:

Deployment configuration is sensitive in nature. The tiniest mistake can lead to downtime or bugs in production, which consequently may have a direct effect on both product integrity and customer trust. Misconfigurations might also be used by malicious actors to attack the production platform. Because of this, every change in the configuration needs a review and possible "revert" in case of a mistake or malicious change. Auditing every change and tracking them helps detect and fix such incidents more quickly.

Audit:

For each deployment configuration, ensure changes made to it are audited and tracked.

Remediation:

For each deployment configuration, track and audit changes made to it.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

5.1.3 Ensure scanners are in place to identify and prevent sensitive data in deployment configuration (Manual)

Profile Applicability:

- Level 1

Description:

Detect and prevent sensitive data – such as confidential ID numbers, passwords, etc. – in deployment configurations.

Rationale:

Sensitive data in deployment configurations might create a major incident if an attacker gains access to it, as this can cause data loss and theft. It is important to keep sensitive data safe and to not expose it in the configuration. In order to prevent a possible exposure, set scanners that will identify and prevent such data in deployment configurations.

Audit:

For each deployment configuration file, verify that scanners are set to identify and prevent the existence of sensitive data within it.

Remediation:

For each deployment configuration file, set scanners to identify and prevent sensitive data within it.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.12 Implement Code-Level Security Checks Apply static and dynamic analysis tools within the application life cycle to verify that secure coding practices are being followed.			●
v7	3.1 Run Automated Vulnerability Scanning Tools Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.		●	●

5.1.4 Limit access to deployment configurations (Manual)

Profile Applicability:

- Level 1

Description:

Restrict access to the deployment configuration to trusted and qualified users only.

Rationale:

Deployment configurations are sensitive in nature. The tiniest mistake can lead to downtime or bugs in production, which can have a direct effect on the product's integrity and customer trust. Misconfigurations might also be used by malicious actors to attack the production platform. To avoid such harm as much as possible, ensure only trusted and qualified users have access to such configurations. This will also reduce the number of accounts that might affect the environment in case of an attack.

Impact:

Reducing the number of users who have access to the deployment configuration means those users would lose their ability to make direct changes to that configuration.

Audit:

Verify each deployment configuration is accessible only to known and authorized users.

Remediation:

Restrict access to the deployment configuration to trusted and qualified users.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

5.1.5 Scan Infrastructure as Code (IaC) (Manual)

Profile Applicability:

- Level 2

Description:

Detect and prevent misconfigurations or insecure instructions in Infrastructure as Code (IaC) files, such as Terraform files.

Rationale:

Infrastructure as Code (IaC) files are used for production environment and application deployment. These are sensitive parts of the software supply chain because they are always in touch with customers, and thus might affect their opinion of or trust in the product. Attackers often target these environments. Detecting and fixing misconfigurations and/or insecure instructions in IaC files decreases the risk for data leak or data theft. It is important to secure IaC instructions in order to prevent further problems of deployment, exposed assets, or improper configurations, which might ultimately lead to easier ways to attack and steal organization data.

Audit:

For every Infrastructure as Code (IaC) instructions file, verify that scanners are set to identify and prevent misconfigurations and insecure instructions.

Remediation:

For every Infrastructure as Code (IaC) instructions file, set scanners to identify and prevent misconfigurations and insecure instructions.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>7.5 Perform Automated Vulnerability Scans of Internal Enterprise Assets</u> Perform automated vulnerability scans of internal enterprise assets on a quarterly, or more frequent, basis. Conduct both authenticated and unauthenticated scans, using a SCAP-compliant vulnerability scanning tool.			
v8	<u>16.7 Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			

Controls Version	Control	IG 1	IG 2	IG 3
v7	3.1 Run Automated Vulnerability Scanning Tools Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.			
v7	18.11 Use Standard Hardening Configuration Templates for Databases For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.			

5.1.6 Ensure deployment configuration manifests are verified (Manual)

Profile Applicability:

- Level 1

Description:

Verify the deployment configuration manifests.

Rationale:

To ensure that the configuration manifests used are trusted and have not been infected by malicious actors before arriving at the platform, it is important to verify the manifests. This may be done by comparing the checksum of the manifest file to its checksum in a trusted source. If a difference arises, this is a sign that an unknown actor has interfered and may have added malicious instructions. If this manifest is used, it might harm the environment and application deployment, which could end in a massive breach and leave the organization exposed to data leaks, etc.

Audit:

For each deployment configuration manifest in use, ensure it has been verified.

Remediation:

Verify each deployment configuration manifest in use.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 Establish and Maintain a Secure Configuration Process Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.			

5.1.7 Ensure deployment configuration manifests are pinned to a specific, verified version (Manual)

Profile Applicability:

- Level 1

Description:

Deployment configuration is often stored in a version control system and is pulled from there. Pin the configuration used to a specific, verified version or commit Secure Hash Algorithm (SHA). Avoid referring configuration without its version tag specified.

Rationale:

Deployment configuration manifests are often stored in version control systems and pulled from there either by automation platforms, for example Ansible, or GitOps platforms, such as ArgoCD. When a manifest is pulled from a version control system without tag or commit Secure Hash Algorithm (SHA) specified, it is pulled from the HEAD revision, which is equal to the 'latest' tag, and pulls the last change made. This increases the risk of encountering a new, potentially malicious configuration. If an attacker pushes malicious configuration to the version control system, the next user who pulls the HEAD revision will pull it and risk attack. To avoid that risk, use a version tag of verified version or a commit SHA of a trusted commit, which will ensure this is the only version pulled.

Impact:

Changes in deployment configuration will not be pulled unless their version tag or commit Secure Hash Algorithm (SHA) is specified. This might slow down the deployment process.

Audit:

For every deployment configuration manifest in use, ensure it is pinned to a specific version or commit Secure Hash Algorithm (SHA).

Remediation:

For every deployment configuration manifest in use, pin to a specific version or commit Secure Hash Algorithm (SHA).

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 <u>Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 <u>Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

5.2 Deployment Environment

This section consists of security recommendations for the management of the deployment environment.

The deployment environment is the orchestrator and the production environment where the application is deployed. It directly affects the customer experience and trust in a product, which has serious effects on the organization itself. Securing it varies from access management to automation.

5.2.1 Ensure deployments are automated (Manual)

Profile Applicability:

- Level 1

Description:

Automate deployments of production environment and application.

Rationale:

Automating the deployments of both production environment and applications reduces the risk for human mistakes — such as a wrong configuration or exposure of sensitive data — because it requires less human interaction or intervention. It also eases redeployment of the environment. It is best to automate with Infrastructure as Code (IaC) because it offers more control over changes made to the environment creation configuration and stores to a version control platform.

Audit:

For each deployment process, ensure it is automated.

Remediation:

Automate each deployment process of the production environment and application.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.1 Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>18.1 Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.			

5.2.2 Ensure the deployment environment is reproducible (Manual)

Profile Applicability:

- Level 1

Description:

Verify that the deployment environment – the orchestrator and the production environment where the application is deployed – is reproducible. This means that the environment stays the same in each deployment if the configuration has not changed.

Rationale:

A reproducible build is a build that produces the same artifact when given the same input data, and in this case the same environment. Ensuring that the same environment is produced when given the same input helps verify that no change has been made to it. This action allows an organization to trust that its deployment environment is built only from safe code and configuration that has been reviewed and tested and has not been tainted or changed abruptly.

Audit:

Verify that the deployment/production environment is reproducible.

Remediation:

Adjust the process that deploys the deployment/production environment to build the same environment each time when the configuration has not changed.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 <u>Establish and Maintain a Secure Application Development Process</u> Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 <u>Establish Secure Coding Practices</u> Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.2.3 Ensure access to production environment is limited (Manual)

Profile Applicability:

- Level 1

Description:

Restrict access to the production environment to a few trusted and qualified users only.

Rationale:

The production environment is an extremely sensitive one. It directly affects the customer experience and trust in a product, which has serious effects on the organization itself. Because of this sensitive nature, it is important to restrict access to the production environment to only a few trusted and qualified users. This will reduce the risk of mistakes such as exposure of secrets or misconfiguration. This restriction also reduces the number of accounts that are vulnerable to hijacking in order to potentially harm the production environment.

Impact:

Reducing the number of users who have access to the production environment means those users would lose their ability to make direct changes to that environment.

Audit:

Verify that the production environment is accessible only to trusted and qualified users.

Remediation:

Restrict access to the production environment to trusted and qualified users.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.8 Separate Production and Non-Production Systems</u> Maintain separate environments for production and non-production systems.		●	●
v7	<u>18.9 Separate Production and Non-Production Systems</u> Maintain separate environments for production and nonproduction systems. Developers should not have unmonitored access to production environments.		●	●

5.2.4 Ensure default passwords are not used (Manual)

Profile Applicability:

- Level 1

Description:

Do not use default passwords of deployment tools and components.

Rationale:

Many deployment tools and components are provided with default passwords for the first login. This password is intended to be used only on the first login and should be changed immediately after. Using the default password substantially increases the attack risk. It is very important to ensure that default passwords are not used in deployment tools and components.

Audit:

GitLab's default root password depends on the installation method, and when the installation occurred:

1. When deploying a GitLab instance using the official AWS AMI, the root password to the instance is the EC2 Instance ID
2. Most installation methods allow a non-default password to be provided as configuration
3. Prior to 14.0 the default password was 5iveL!fe
4. Otherwise the default password is unique and randomly generated.

Attempt to log in as root using a suspected default password to audit whether it has changed.

For any other external build tools, ensure the password used is not the default one.

Remediation:

GitLab's root password can be changed by an administrator using the UI, the "gitlab:password:reset" rake task, or by using the Rails console.

For each build tool with a default password, change to a unique cryptographically secure pseudorandom password.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.2 <u>Use Unique Passwords</u> Use unique passwords for all enterprise assets. Best practice implementation includes, at a minimum, an 8-character password for accounts using MFA and a 14-character password for accounts not using MFA.			
v7	4.2 <u>Change Default Passwords</u> Before deploying any new asset, change all default passwords to have values consistent with administrative level accounts.			
v7	4.4 <u>Use Unique Passwords</u> Where multi-factor authentication is not supported (such as local administrator, root, or service accounts), accounts will use passwords that are unique to that system.			

Appendix: Summary Table

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1	Source Code		
1.1	Code Changes		
1.1.1	Ensure any changes to code are tracked in a version control platform (Manual)	☐	☐
1.1.2	Ensure any change to code can be traced back to its associated task (Manual)	☐	☐
1.1.3	Ensure any change to code receives approval of two strongly authenticated users (Automated)	☐	☐
1.1.4	Ensure previous approvals are dismissed when updates are introduced to a code change proposal (Manual)	☐	☐
1.1.5	Ensure there are restrictions on who can dismiss code change reviews (Manual)	☐	☐
1.1.6	Ensure code owners are set for extra sensitive code or configuration (Manual)	☐	☐
1.1.7	Ensure code owner's review is required when a change affects owned code (Manual)	☐	☐
1.1.8	Ensure inactive branches are periodically reviewed and removed (Manual)	☐	☐
1.1.9	Ensure all checks have passed before merging new code (Manual)	☐	☐
1.1.10	Ensure open Git branches are up to date before they can be merged into code base (Manual)	☐	☐
1.1.11	Ensure all open comments are resolved before allowing code change merging (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1.1.12	Ensure verification of signed commits for new changes before merging (Manual)	☐	☐
1.1.13	Ensure linear history is required (Manual)	☐	☐
1.1.14	Ensure branch protection rules are enforced for administrators (Manual)	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams (Manual)	☐	☐
1.1.16	Ensure force push code to branches is denied (Manual)	☐	☐
1.1.17	Ensure branch deletions are denied (Manual)	☐	☐
1.1.18	Ensure any merging of code is automatically scanned for risks (Manual)	☐	☐
1.1.19	Ensure any changes to branch protection rules are audited (Manual)	☐	☐
1.1.20	Ensure branch protection is enforced on the default branch (Manual)	☐	☐
1.2	Repository Management		
1.2.1	Ensure all public repositories contain a SECURITY.md file (Manual)	☐	☐
1.2.2	Ensure repository creation is limited to specific members (Manual)	☐	☐
1.2.3	Ensure repository deletion is limited to specific users (Manual)	☐	☐
1.2.4	Ensure issue deletion is limited to specific users (Manual)	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1.2.6	Ensure all code projects are tracked for changes in visibility status (Manual)	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically (Manual)	☐	☐
1.3	Contribution Access		
1.3.1	Ensure inactive users are reviewed and removed periodically (Manual)	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members (Manual)	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization (Manual)	☐	☐
1.3.4	Ensure Multi-Factor Authentication (MFA) is required for contributors of new code (Manual)	☐	☐
1.3.5	Ensure the organization is requiring members to use Multi-Factor Authentication (MFA) (Manual)	☐	☐
1.3.6	Ensure new members are required to be invited using company-approved email (Manual)	☐	☐
1.3.7	Ensure two administrators are set for each repository (Manual)	☐	☐
1.3.8	Ensure strict base permissions are set for repositories (Manual)	☐	☐
1.3.9	Ensure an organization's identity is confirmed with a "Verified" badge (Manual)	☐	☐
1.3.10	Ensure Source Code Management (SCM) email notifications are restricted to verified domains (Manual)	☐	☐
1.3.11	Ensure an organization provides SSH certificates (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1.3.12	Ensure Git access is limited based on IP addresses (Manual)	☐	☐
1.3.13	Ensure anomalous code behavior is tracked (Manual)	☐	☐
1.4	Third-Party		
1.4.1	Ensure administrator approval is required for every installed application (Manual)	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed (Manual)	☐	☐
1.4.3	Ensure the access granted to each installed application is limited to the least privilege needed (Manual)	☐	☐
1.4.4	Ensure only secured webhooks are used (Manual)	☐	☐
1.5	Code Risks		
1.5.1	Ensure scanners are in place to identify and prevent sensitive data in code (Manual)	☐	☐
1.5.2	Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions (Manual)	☐	☐
1.5.3	Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions (Manual)	☐	☐
1.5.4	Ensure scanners are in place for code vulnerabilities (Manual)	☐	☐
1.5.5	Ensure scanners are in place for open-source vulnerabilities in used packages (Manual)	☐	☐
1.5.6	Ensure scanners are in place for open-source license issues in used packages (Manual)	☐	☐
1.5.7	Ensure scanners are in place for web application runtime security weaknesses (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1.5.8	Ensure scanners are in place for API runtime security weaknesses (Manual)	☐	☐
2	Build Pipelines		
2.1	Build Environment		
2.1.1	Ensure each pipeline has a single responsibility (Manual)	☐	☐
2.1.2	Ensure all aspects of the pipeline infrastructure and configuration are immutable (Manual)	☐	☐
2.1.3	Ensure the build environment is logged (Manual)	☐	☐
2.1.4	Ensure the creation of the build environment is automated (Manual)	☐	☐
2.1.5	Ensure access to build environments is limited (Manual)	☐	☐
2.1.6	Ensure users must authenticate to access the build environment (Manual)	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope (Manual)	☐	☐
2.1.8	Ensure the build infrastructure is automatically scanned for vulnerabilities (Manual)	☐	☐
2.1.9	Ensure default passwords are not used (Manual)	☐	☐
2.1.10	Ensure webhooks of the build environment are secured (Manual)	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment (Manual)	☐	☐
2.2	Build Worker		
2.2.1	Ensure build workers are single-used (Manual)	☐	☐
2.2.2	Ensure build worker environments and commands are passed and not pulled (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
2.2.3	Ensure the duties of each build worker are segregated (Manual)	☐	☐
2.2.4	Ensure build workers have minimal network connectivity (Manual)	☐	☐
2.2.5	Ensure run-time security is enforced for build workers (Manual)	☐	☐
2.2.6	Ensure build workers are automatically scanned for vulnerabilities (Manual)	☐	☐
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform (Manual)	☐	☐
2.2.8	Ensure resource consumption of build workers is monitored (Manual)	☐	☐
2.3	Pipeline Instructions		
2.3.1	Ensure all build steps are defined as code (Manual)	☐	☐
2.3.2	Ensure steps have clearly defined build stage input and output (Manual)	☐	☐
2.3.3	Ensure output is written to a separate, secured storage repository (Manual)	☐	☐
2.3.4	Ensure changes to pipeline files are tracked and reviewed (Manual)	☐	☐
2.3.5	Ensure access to build process triggering is minimized (Manual)	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations (Manual)	☐	☐
2.3.7	Ensure pipelines are automatically scanned for vulnerabilities (Manual)	☐	☐
2.3.8	Ensure scanners are in place to identify and prevent sensitive data in pipeline files (Automated)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
2.4	Pipeline Integrity		
2.4.1	Ensure all artifacts on all releases are signed (Manual)	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked (Manual)	☐	☐
2.4.3	Ensure dependencies are validated before being used (Manual)	☐	☐
2.4.4	Ensure the build pipeline creates reproducible artifacts (Manual)	☐	☐
2.4.5	Ensure pipeline steps produce a Software Bill of Materials (SBOM) (Manual)	☐	☐
2.4.6	Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced (Manual)	☐	☐
3	Dependencies		
3.1	Third-Party Packages		
3.1.1	Ensure third-party artifacts and open-source libraries are verified (Manual)	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers (Manual)	☐	☐
3.1.3	Ensure signed metadata of the build process is required and verified (Manual)	☐	☐
3.1.4	Ensure dependencies are monitored between open-source components (Manual)	☐	☐
3.1.5	Ensure trusted package managers and repositories are defined and prioritized (Manual)	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
3.1.7	Ensure dependencies are pinned to a specific, verified version (Manual)	☐	☐
3.1.8	Ensure all packages used are more than 60 days old (Manual)	☐	☐
3.2	Validate Packages		
3.2.1	Ensure an organization-wide dependency usage policy is enforced (Manual)	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities (Manual)	☐	☐
3.2.3	Ensure packages are automatically scanned for license implications (Manual)	☐	☐
3.2.4	Ensure packages are automatically scanned for ownership change (Manual)	☐	☐
4	Artifacts		
4.1	Verification		
4.1.1	Ensure all artifacts are signed by the build pipeline itself (Manual)	☐	☐
4.1.2	Ensure artifacts are encrypted before distribution (Manual)	☐	☐
4.1.3	Ensure only authorized platforms have decryption capabilities of artifacts (Manual)	☐	☐
4.2	Access to Artifacts		
4.2.1	Ensure the authority to certify artifacts is limited (Manual)	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized (Manual)	☐	☐
4.2.3	Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA) (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
4.2.4	Ensure user management of the package registry is not local (Manual)	☐	☐
4.2.5	Ensure anonymous access to artifacts is revoked (Manual)	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry (Manual)	☐	☐
4.3	Package Registries		
4.3.1	Ensure all signed artifacts are validated upon uploading the package registry (Manual)	☐	☐
4.3.2	Ensure all versions of an existing artifact have their signatures validated (Manual)	☐	☐
4.3.3	Ensure changes in package registry configuration are audited (Manual)	☐	☐
4.3.4	Ensure webhooks of the repository are secured (Manual)	☐	☐
4.4	Origin Traceability		
4.4.1	Ensure artifacts contain information about their origin (Manual)	☐	☐
5	Deployment		
5.1	Deployment Configuration		
5.1.1	Ensure deployment configuration files are separated from source code (Manual)	☐	☐
5.1.2	Ensure changes in deployment configuration are audited (Manual)	☐	☐
5.1.3	Ensure scanners are in place to identify and prevent sensitive data in deployment configuration (Manual)	☐	☐
5.1.4	Limit access to deployment configurations (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
5.1.5	Scan Infrastructure as Code (IaC) (Manual)	☐	☐
5.1.6	Ensure deployment configuration manifests are verified (Manual)	☐	☐
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version (Manual)	☐	☐
5.2	Deployment Environment		
5.2.1	Ensure deployments are automated (Manual)	☐	☐
5.2.2	Ensure the deployment environment is reproducible (Manual)	☐	☐
5.2.3	Ensure access to production environment is limited (Manual)	☐	☐
5.2.4	Ensure default passwords are not used (Manual)	☐	☐

Appendix: CIS Controls v7 IG 1 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
1.1.5	Ensure there are restrictions on who can dismiss code change reviews	☐	☐
1.1.6	Ensure code owners are set for extra sensitive code or configuration	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐
1.2.2	Ensure repository creation is limited to specific members	☐	☐
1.2.3	Ensure repository deletion is limited to specific users	☐	☐
1.2.4	Ensure issue deletion is limited to specific users	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically	☐	☐
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
1.3.7	Ensure two administrators are set for each repository	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐
1.4.3	Ensure the access granted to each installed application is limited to the least privilege needed	☐	☐
2.1.5	Ensure access to build environments is limited	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform	☐	☐
2.3.5	Ensure access to build process triggering is minimized	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.2.5	Ensure anonymous access to artifacts is revoked	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐
5.1.4	Limit access to deployment configurations	☐	☐
5.1.6	Ensure deployment configuration manifests are verified	☐	☐
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version	☐	☐
5.2.4	Ensure default passwords are not used	☐	☐

Appendix: CIS Controls v7 IG 2 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
1.1.1	Ensure any changes to code are tracked in a version control platform	☐	☐
1.1.2	Ensure any change to code can be traced back to its associated task	☐	☐
1.1.3	Ensure any change to code receives approval of two strongly authenticated users	☐	☐
1.1.4	Ensure previous approvals are dismissed when updates are introduced to a code change proposal	☐	☐
1.1.5	Ensure there are restrictions on who can dismiss code change reviews	☐	☐
1.1.6	Ensure code owners are set for extra sensitive code or configuration	☐	☐
1.1.7	Ensure code owner's review is required when a change affects owned code	☐	☐
1.1.8	Ensure inactive branches are periodically reviewed and removed	☐	☐
1.1.9	Ensure all checks have passed before merging new code	☐	☐
1.1.10	Ensure open Git branches are up to date before they can be merged into code base	☐	☐
1.1.11	Ensure all open comments are resolved before allowing code change merging	☐	☐
1.1.12	Ensure verification of signed commits for new changes before merging	☐	☐
1.1.13	Ensure linear history is required	☐	☐
1.1.14	Ensure branch protection rules are enforced for administrators	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams	☐	☐
1.1.16	Ensure force push code to branches is denied	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.1.18	Ensure any merging of code is automatically scanned for risks	☐	☐
1.1.19	Ensure any changes to branch protection rules are audited	☐	☐
1.1.20	Ensure branch protection is enforced on the default branch	☐	☐
1.2.1	Ensure all public repositories contain a SECURITY.md file	☐	☐
1.2.2	Ensure repository creation is limited to specific members	☐	☐
1.2.3	Ensure repository deletion is limited to specific users	☐	☐
1.2.4	Ensure issue deletion is limited to specific users	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
1.2.6	Ensure all code projects are tracked for changes in visibility status	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically	☐	☐
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
1.3.4	Ensure Multi-Factor Authentication (MFA) is required for contributors of new code	☐	☐
1.3.5	Ensure the organization is requiring members to use Multi-Factor Authentication (MFA)	☐	☐
1.3.7	Ensure two administrators are set for each repository	☐	☐
1.3.8	Ensure strict base permissions are set for repositories	☐	☐
1.3.9	Ensure an organization's identity is confirmed with a "Verified" badge	☐	☐
1.3.13	Ensure anomalous code behavior is tracked	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.4.3	Ensure the access granted to each installed application is limited to the least privilege needed	☐	☐
1.5.1	Ensure scanners are in place to identify and prevent sensitive data in code	☐	☐
1.5.2	Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions	☐	☐
1.5.3	Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions	☐	☐
1.5.4	Ensure scanners are in place for code vulnerabilities	☐	☐
1.5.5	Ensure scanners are in place for open-source vulnerabilities in used packages	☐	☐
1.5.7	Ensure scanners are in place for web application runtime security weaknesses	☐	☐
1.5.8	Ensure scanners are in place for API runtime security weaknesses	☐	☐
2.1.4	Ensure the creation of the build environment is automated	☐	☐
2.1.5	Ensure access to build environments is limited	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.8	Ensure the build infrastructure is automatically scanned for vulnerabilities	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.6	Ensure build workers are automatically scanned for vulnerabilities	☐	☐
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform	☐	☐
2.3.2	Ensure steps have clearly defined build stage input and output	☐	☐
2.3.5	Ensure access to build process triggering is minimized	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.3.7	Ensure pipelines are automatically scanned for vulnerabilities	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
3.1.1	Ensure third-party artifacts and open-source libraries are verified	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.4	Ensure dependencies are monitored between open-source components	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied	☐	☐
3.1.7	Ensure dependencies are pinned to a specific, verified version	☐	☐
3.2.1	Ensure an organization-wide dependency usage policy is enforced	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized	☐	☐
4.2.3	Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA)	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.2.5	Ensure anonymous access to artifacts is revoked	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐
4.3.1	Ensure all signed artifacts are validated upon uploading the package registry	☐	☐
4.3.2	Ensure all versions of an existing artifact have their signatures validated	☐	☐
4.3.3	Ensure changes in package registry configuration are audited	☐	☐

Recommendation		Set Correctly	
		Yes	No
4.3.4	Ensure webhooks of the repository are secured	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
5.1.1	Ensure deployment configuration files are separated from source code	☐	☐
5.1.2	Ensure changes in deployment configuration are audited	☐	☐
5.1.3	Ensure scanners are in place to identify and prevent sensitive data in deployment configuration	☐	☐
5.1.4	Limit access to deployment configurations	☐	☐
5.1.5	Scan Infrastructure as Code (IaC)	☐	☐
5.1.6	Ensure deployment configuration manifests are verified	☐	☐
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version	☐	☐
5.2.1	Ensure deployments are automated	☐	☐
5.2.2	Ensure the deployment environment is reproducible	☐	☐
5.2.3	Ensure access to production environment is limited	☐	☐
5.2.4	Ensure default passwords are not used	☐	☐

Appendix: CIS Controls v7 IG 3 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
1.1.1	Ensure any changes to code are tracked in a version control platform	☐	☐
1.1.2	Ensure any change to code can be traced back to its associated task	☐	☐
1.1.3	Ensure any change to code receives approval of two strongly authenticated users	☐	☐
1.1.4	Ensure previous approvals are dismissed when updates are introduced to a code change proposal	☐	☐
1.1.5	Ensure there are restrictions on who can dismiss code change reviews	☐	☐
1.1.6	Ensure code owners are set for extra sensitive code or configuration	☐	☐
1.1.7	Ensure code owner's review is required when a change affects owned code	☐	☐
1.1.8	Ensure inactive branches are periodically reviewed and removed	☐	☐
1.1.9	Ensure all checks have passed before merging new code	☐	☐
1.1.10	Ensure open Git branches are up to date before they can be merged into code base	☐	☐
1.1.11	Ensure all open comments are resolved before allowing code change merging	☐	☐
1.1.12	Ensure verification of signed commits for new changes before merging	☐	☐
1.1.13	Ensure linear history is required	☐	☐
1.1.14	Ensure branch protection rules are enforced for administrators	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams	☐	☐
1.1.16	Ensure force push code to branches is denied	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.1.18	Ensure any merging of code is automatically scanned for risks	☐	☐
1.1.19	Ensure any changes to branch protection rules are audited	☐	☐
1.1.20	Ensure branch protection is enforced on the default branch	☐	☐
1.2.1	Ensure all public repositories contain a SECURITY.md file	☐	☐
1.2.2	Ensure repository creation is limited to specific members	☐	☐
1.2.3	Ensure repository deletion is limited to specific users	☐	☐
1.2.4	Ensure issue deletion is limited to specific users	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
1.2.6	Ensure all code projects are tracked for changes in visibility status	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically	☐	☐
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
1.3.4	Ensure Multi-Factor Authentication (MFA) is required for contributors of new code	☐	☐
1.3.5	Ensure the organization is requiring members to use Multi-Factor Authentication (MFA)	☐	☐
1.3.6	Ensure new members are required to be invited using company-approved email	☐	☐
1.3.7	Ensure two administrators are set for each repository	☐	☐
1.3.8	Ensure strict base permissions are set for repositories	☐	☐
1.3.9	Ensure an organization's identity is confirmed with a "Verified" badge	☐	☐
1.3.11	Ensure an organization provides SSH certificates	☐	☐
1.3.12	Ensure Git access is limited based on IP addresses	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.3.13	Ensure anomalous code behavior is tracked	☐	☐
1.4.1	Ensure administrator approval is required for every installed application	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐
1.4.3	Ensure the access granted to each installed application is limited to the least privilege needed	☐	☐
1.5.1	Ensure scanners are in place to identify and prevent sensitive data in code	☐	☐
1.5.2	Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions	☐	☐
1.5.3	Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions	☐	☐
1.5.4	Ensure scanners are in place for code vulnerabilities	☐	☐
1.5.5	Ensure scanners are in place for open-source vulnerabilities in used packages	☐	☐
1.5.7	Ensure scanners are in place for web application runtime security weaknesses	☐	☐
1.5.8	Ensure scanners are in place for API runtime security weaknesses	☐	☐
2.1.3	Ensure the build environment is logged	☐	☐
2.1.4	Ensure the creation of the build environment is automated	☐	☐
2.1.5	Ensure access to build environments is limited	☐	☐
2.1.6	Ensure users must authenticate to access the build environment	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.8	Ensure the build infrastructure is automatically scanned for vulnerabilities	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.4	Ensure build workers have minimal network connectivity	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.2.6	Ensure build workers are automatically scanned for vulnerabilities	☐	☐
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform	☐	☐
2.3.2	Ensure steps have clearly defined build stage input and output	☐	☐
2.3.4	Ensure changes to pipeline files are tracked and reviewed	☐	☐
2.3.5	Ensure access to build process triggering is minimized	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations	☐	☐
2.3.7	Ensure pipelines are automatically scanned for vulnerabilities	☐	☐
2.3.8	Ensure scanners are in place to identify and prevent sensitive data in pipeline files	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
3.1.1	Ensure third-party artifacts and open-source libraries are verified	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.4	Ensure dependencies are monitored between open-source components	☐	☐
3.1.5	Ensure trusted package managers and repositories are defined and prioritized	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied	☐	☐
3.1.7	Ensure dependencies are pinned to a specific, verified version	☐	☐
3.1.8	Ensure all packages used are more than 60 days old	☐	☐
3.2.1	Ensure an organization-wide dependency usage policy is enforced	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities	☐	☐

Recommendation		Set Correctly	
		Yes	No
4.1.2	Ensure artifacts are encrypted before distribution	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized	☐	☐
4.2.3	Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA)	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.2.5	Ensure anonymous access to artifacts is revoked	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐
4.3.1	Ensure all signed artifacts are validated upon uploading the package registry	☐	☐
4.3.2	Ensure all versions of an existing artifact have their signatures validated	☐	☐
4.3.3	Ensure changes in package registry configuration are audited	☐	☐
4.3.4	Ensure webhooks of the repository are secured	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
5.1.1	Ensure deployment configuration files are separated from source code	☐	☐
5.1.2	Ensure changes in deployment configuration are audited	☐	☐
5.1.3	Ensure scanners are in place to identify and prevent sensitive data in deployment configuration	☐	☐
5.1.4	Limit access to deployment configurations	☐	☐
5.1.5	Scan Infrastructure as Code (IaC)	☐	☐
5.1.6	Ensure deployment configuration manifests are verified	☐	☐
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version	☐	☐
5.2.1	Ensure deployments are automated	☐	☐
5.2.2	Ensure the deployment environment is reproducible	☐	☐
5.2.3	Ensure access to production environment is limited	☐	☐
5.2.4	Ensure default passwords are not used	☐	☐

Appendix: CIS Controls v8 IG 1 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
1.1.5	Ensure there are restrictions on who can dismiss code change reviews	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐
1.2.2	Ensure repository creation is limited to specific members	☐	☐
1.2.3	Ensure repository deletion is limited to specific users	☐	☐
1.2.4	Ensure issue deletion is limited to specific users	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
1.2.6	Ensure all code projects are tracked for changes in visibility status	☐	☐
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
1.3.4	Ensure Multi-Factor Authentication (MFA) is required for contributors of new code	☐	☐
1.3.5	Ensure the organization is requiring members to use Multi-Factor Authentication (MFA)	☐	☐
1.3.6	Ensure new members are required to be invited using company-approved email	☐	☐
1.3.7	Ensure two administrators are set for each repository	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐
2.1.1	Ensure each pipeline has a single responsibility	☐	☐
2.1.5	Ensure access to build environments is limited	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform	☐	☐
2.3.5	Ensure access to build process triggering is minimized	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
5.1.4	Limit access to deployment configurations	☐	☐
5.1.6	Ensure deployment configuration manifests are verified	☐	☐
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version	☐	☐
5.2.4	Ensure default passwords are not used	☐	☐

Appendix: CIS Controls v8 IG 2 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
1.1.1	Ensure any changes to code are tracked in a version control platform	☐	☐
1.1.2	Ensure any change to code can be traced back to its associated task	☐	☐
1.1.3	Ensure any change to code receives approval of two strongly authenticated users	☐	☐
1.1.4	Ensure previous approvals are dismissed when updates are introduced to a code change proposal	☐	☐
1.1.5	Ensure there are restrictions on who can dismiss code change reviews	☐	☐
1.1.7	Ensure code owner's review is required when a change affects owned code	☐	☐
1.1.8	Ensure inactive branches are periodically reviewed and removed	☐	☐
1.1.9	Ensure all checks have passed before merging new code	☐	☐
1.1.10	Ensure open Git branches are up to date before they can be merged into code base	☐	☐
1.1.11	Ensure all open comments are resolved before allowing code change merging	☐	☐
1.1.12	Ensure verification of signed commits for new changes before merging	☐	☐
1.1.13	Ensure linear history is required	☐	☐
1.1.14	Ensure branch protection rules are enforced for administrators	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams	☐	☐
1.1.16	Ensure force push code to branches is denied	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐
1.1.19	Ensure any changes to branch protection rules are audited	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.1.20	Ensure branch protection is enforced on the default branch	☐	☐
1.2.1	Ensure all public repositories contain a SECURITY.md file	☐	☐
1.2.2	Ensure repository creation is limited to specific members	☐	☐
1.2.3	Ensure repository deletion is limited to specific users	☐	☐
1.2.4	Ensure issue deletion is limited to specific users	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
1.2.6	Ensure all code projects are tracked for changes in visibility status	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically	☐	☐
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
1.3.4	Ensure Multi-Factor Authentication (MFA) is required for contributors of new code	☐	☐
1.3.5	Ensure the organization is requiring members to use Multi-Factor Authentication (MFA)	☐	☐
1.3.6	Ensure new members are required to be invited using company-approved email	☐	☐
1.3.7	Ensure two administrators are set for each repository	☐	☐
1.3.8	Ensure strict base permissions are set for repositories	☐	☐
1.3.9	Ensure an organization's identity is confirmed with a "Verified" badge	☐	☐
1.3.11	Ensure an organization provides SSH certificates	☐	☐
1.3.12	Ensure Git access is limited based on IP addresses	☐	☐
1.4.1	Ensure administrator approval is required for every installed application	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.5.2	Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions	☐	☐
1.5.3	Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions	☐	☐
1.5.5	Ensure scanners are in place for open-source vulnerabilities in used packages	☐	☐
2.1.1	Ensure each pipeline has a single responsibility	☐	☐
2.1.2	Ensure all aspects of the pipeline infrastructure and configuration are immutable	☐	☐
2.1.4	Ensure the creation of the build environment is automated	☐	☐
2.1.5	Ensure access to build environments is limited	☐	☐
2.1.6	Ensure users must authenticate to access the build environment	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.8	Ensure the build infrastructure is automatically scanned for vulnerabilities	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐
2.1.10	Ensure webhooks of the build environment are secured	☐	☐
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.4	Ensure build workers have minimal network connectivity	☐	☐
2.2.6	Ensure build workers are automatically scanned for vulnerabilities	☐	☐
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform	☐	☐
2.3.2	Ensure steps have clearly defined build stage input and output	☐	☐
2.3.3	Ensure output is written to a separate, secured storage repository	☐	☐
2.3.5	Ensure access to build process triggering is minimized	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.3.7	Ensure pipelines are automatically scanned for vulnerabilities	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
3.1.1	Ensure third-party artifacts and open-source libraries are verified	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.4	Ensure dependencies are monitored between open-source components	☐	☐
3.1.5	Ensure trusted package managers and repositories are defined and prioritized	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied	☐	☐
3.1.7	Ensure dependencies are pinned to a specific, verified version	☐	☐
3.1.8	Ensure all packages used are more than 60 days old	☐	☐
3.2.1	Ensure an organization-wide dependency usage policy is enforced	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities	☐	☐
4.1.2	Ensure artifacts are encrypted before distribution	☐	☐
4.1.3	Ensure only authorized platforms have decryption capabilities of artifacts	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized	☐	☐
4.2.3	Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA)	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.2.5	Ensure anonymous access to artifacts is revoked	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐

Recommendation		Set Correctly	
		Yes	No
4.3.1	Ensure all signed artifacts are validated upon uploading the package registry	☐	☐
4.3.2	Ensure all versions of an existing artifact have their signatures validated	☐	☐
4.3.3	Ensure changes in package registry configuration are audited	☐	☐
4.3.4	Ensure webhooks of the repository are secured	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
5.1.1	Ensure deployment configuration files are separated from source code	☐	☐
5.1.2	Ensure changes in deployment configuration are audited	☐	☐
5.1.4	Limit access to deployment configurations	☐	☐
5.1.5	Scan Infrastructure as Code (IaC)	☐	☐
5.1.6	Ensure deployment configuration manifests are verified	☐	☐
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version	☐	☐
5.2.1	Ensure deployments are automated	☐	☐
5.2.2	Ensure the deployment environment is reproducible	☐	☐
5.2.3	Ensure access to production environment is limited	☐	☐
5.2.4	Ensure default passwords are not used	☐	☐

Appendix: CIS Controls v8 IG 3 Mapped Recommendations

Recommendation		Set Correctly	
		Yes	No
1.1.1	Ensure any changes to code are tracked in a version control platform	☐	☐
1.1.2	Ensure any change to code can be traced back to its associated task	☐	☐
1.1.3	Ensure any change to code receives approval of two strongly authenticated users	☐	☐
1.1.4	Ensure previous approvals are dismissed when updates are introduced to a code change proposal	☐	☐
1.1.5	Ensure there are restrictions on who can dismiss code change reviews	☐	☐
1.1.6	Ensure code owners are set for extra sensitive code or configuration	☐	☐
1.1.7	Ensure code owner's review is required when a change affects owned code	☐	☐
1.1.8	Ensure inactive branches are periodically reviewed and removed	☐	☐
1.1.9	Ensure all checks have passed before merging new code	☐	☐
1.1.10	Ensure open Git branches are up to date before they can be merged into code base	☐	☐
1.1.11	Ensure all open comments are resolved before allowing code change merging	☐	☐
1.1.12	Ensure verification of signed commits for new changes before merging	☐	☐
1.1.13	Ensure linear history is required	☐	☐
1.1.14	Ensure branch protection rules are enforced for administrators	☐	☐
1.1.15	Ensure pushing or merging of new code is restricted to specific individuals or teams	☐	☐
1.1.16	Ensure force push code to branches is denied	☐	☐
1.1.17	Ensure branch deletions are denied	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.1.18	Ensure any merging of code is automatically scanned for risks	☐	☐
1.1.19	Ensure any changes to branch protection rules are audited	☐	☐
1.1.20	Ensure branch protection is enforced on the default branch	☐	☐
1.2.1	Ensure all public repositories contain a SECURITY.md file	☐	☐
1.2.2	Ensure repository creation is limited to specific members	☐	☐
1.2.3	Ensure repository deletion is limited to specific users	☐	☐
1.2.4	Ensure issue deletion is limited to specific users	☐	☐
1.2.5	Ensure all copies (forks) of code are tracked and accounted for	☐	☐
1.2.6	Ensure all code projects are tracked for changes in visibility status	☐	☐
1.2.7	Ensure inactive repositories are reviewed and archived periodically	☐	☐
1.3.1	Ensure inactive users are reviewed and removed periodically	☐	☐
1.3.2	Ensure top-level group creation is limited to specific members	☐	☐
1.3.3	Ensure minimum number of administrators are set for the organization	☐	☐
1.3.4	Ensure Multi-Factor Authentication (MFA) is required for contributors of new code	☐	☐
1.3.5	Ensure the organization is requiring members to use Multi-Factor Authentication (MFA)	☐	☐
1.3.6	Ensure new members are required to be invited using company-approved email	☐	☐
1.3.7	Ensure two administrators are set for each repository	☐	☐
1.3.8	Ensure strict base permissions are set for repositories	☐	☐
1.3.9	Ensure an organization's identity is confirmed with a "Verified" badge	☐	☐
1.3.11	Ensure an organization provides SSH certificates	☐	☐
1.3.12	Ensure Git access is limited based on IP addresses	☐	☐

Recommendation		Set Correctly	
		Yes	No
1.3.13	Ensure anomalous code behavior is tracked	☐	☐
1.4.1	Ensure administrator approval is required for every installed application	☐	☐
1.4.2	Ensure stale applications are reviewed and inactive ones are removed	☐	☐
1.4.3	Ensure the access granted to each installed application is limited to the least privilege needed	☐	☐
1.5.1	Ensure scanners are in place to identify and prevent sensitive data in code	☐	☐
1.5.2	Ensure scanners are in place to secure Continuous Integration (CI) pipeline instructions	☐	☐
1.5.3	Ensure scanners are in place to secure Infrastructure as Code (IaC) instructions	☐	☐
1.5.4	Ensure scanners are in place for code vulnerabilities	☐	☐
1.5.5	Ensure scanners are in place for open-source vulnerabilities in used packages	☐	☐
1.5.7	Ensure scanners are in place for web application runtime security weaknesses	☐	☐
1.5.8	Ensure scanners are in place for API runtime security weaknesses	☐	☐
2.1.1	Ensure each pipeline has a single responsibility	☐	☐
2.1.2	Ensure all aspects of the pipeline infrastructure and configuration are immutable	☐	☐
2.1.3	Ensure the build environment is logged	☐	☐
2.1.4	Ensure the creation of the build environment is automated	☐	☐
2.1.5	Ensure access to build environments is limited	☐	☐
2.1.6	Ensure users must authenticate to access the build environment	☐	☐
2.1.7	Ensure build secrets are limited to the minimal necessary scope	☐	☐
2.1.8	Ensure the build infrastructure is automatically scanned for vulnerabilities	☐	☐
2.1.9	Ensure default passwords are not used	☐	☐
2.1.10	Ensure webhooks of the build environment are secured	☐	☐

Recommendation		Set Correctly	
		Yes	No
2.1.11	Ensure minimum number of administrators are set for the build environment	☐	☐
2.2.4	Ensure build workers have minimal network connectivity	☐	☐
2.2.6	Ensure build workers are automatically scanned for vulnerabilities	☐	☐
2.2.7	Ensure build workers' deployment configuration is stored in a version control platform	☐	☐
2.3.2	Ensure steps have clearly defined build stage input and output	☐	☐
2.3.3	Ensure output is written to a separate, secured storage repository	☐	☐
2.3.4	Ensure changes to pipeline files are tracked and reviewed	☐	☐
2.3.5	Ensure access to build process triggering is minimized	☐	☐
2.3.6	Ensure pipelines are automatically scanned for misconfigurations	☐	☐
2.3.7	Ensure pipelines are automatically scanned for vulnerabilities	☐	☐
2.3.8	Ensure scanners are in place to identify and prevent sensitive data in pipeline files	☐	☐
2.4.2	Ensure all external dependencies used in the build process are locked	☐	☐
2.4.3	Ensure dependencies are validated before being used	☐	☐
3.1.1	Ensure third-party artifacts and open-source libraries are verified	☐	☐
3.1.2	Ensure Software Bill of Materials (SBOM) is required from all third-party suppliers	☐	☐
3.1.4	Ensure dependencies are monitored between open-source components	☐	☐
3.1.5	Ensure trusted package managers and repositories are defined and prioritized	☐	☐
3.1.6	Ensure a signed Software Bill of Materials (SBOM) of the code is supplied	☐	☐
3.1.7	Ensure dependencies are pinned to a specific, verified version	☐	☐

Recommendation		Set Correctly	
		Yes	No
3.1.8	Ensure all packages used are more than 60 days old	☐	☐
3.2.1	Ensure an organization-wide dependency usage policy is enforced	☐	☐
3.2.2	Ensure packages are automatically scanned for known vulnerabilities	☐	☐
4.1.2	Ensure artifacts are encrypted before distribution	☐	☐
4.1.3	Ensure only authorized platforms have decryption capabilities of artifacts	☐	☐
4.2.1	Ensure the authority to certify artifacts is limited	☐	☐
4.2.2	Ensure number of permitted users who may upload new artifacts is minimized	☐	☐
4.2.3	Ensure user access to the package registry utilizes Multi-Factor Authentication (MFA)	☐	☐
4.2.4	Ensure user management of the package registry is not local	☐	☐
4.2.5	Ensure anonymous access to artifacts is revoked	☐	☐
4.2.6	Ensure minimum number of administrators are set for the package registry	☐	☐
4.3.1	Ensure all signed artifacts are validated upon uploading the package registry	☐	☐
4.3.2	Ensure all versions of an existing artifact have their signatures validated	☐	☐
4.3.3	Ensure changes in package registry configuration are audited	☐	☐
4.3.4	Ensure webhooks of the repository are secured	☐	☐
4.4.1	Ensure artifacts contain information about their origin	☐	☐
5.1.1	Ensure deployment configuration files are separated from source code	☐	☐
5.1.2	Ensure changes in deployment configuration are audited	☐	☐
5.1.3	Ensure scanners are in place to identify and prevent sensitive data in deployment configuration	☐	☐
5.1.4	Limit access to deployment configurations	☐	☐
5.1.5	Scan Infrastructure as Code (IaC)	☐	☐
5.1.6	Ensure deployment configuration manifests are verified	☐	☐

Recommendation		Set Correctly	
		Yes	No
5.1.7	Ensure deployment configuration manifests are pinned to a specific, verified version	☐	☐
5.2.1	Ensure deployments are automated	☐	☐
5.2.2	Ensure the deployment environment is reproducible	☐	☐
5.2.3	Ensure access to production environment is limited	☐	☐
5.2.4	Ensure default passwords are not used	☐	☐

Appendix: Change History

Date	Version	Changes for this version
Apr 19, 2024	1.0.1	Minor text and formatting changes throughout the document. Significant Audit/Remediation changes to the following Recommendations: • 1.3.5 Ensure the organization is requiring members to use Multi-Factor Authentication (MFA) • 4.1.1 Ensure all artifacts are signed by the build pipeline itself • 4.3.1 Ensure all signed artifacts are validated upon uploading the package registry • 4.3.2 Ensure all versions of an existing artifact have their signatures validated • 4.3.3 Ensure changes in package registry configuration are audited