## 0.0 Explicitly Not Mapped

Explicitly Not Mapped

# 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 costumer. It should be protected from any mistake or malicious deed in order to keep the software secured.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 1.1.1 | Ensure any changes to code are tracked in a version control platform (Manual) | Manage all code projects in a version control platform. | 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. |
| 1.1.2 | Ensure any change to code can be traced back to its associated task (Manual) | Use a task management system to trace any code back to its associated task. | 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. |
| 1.1.5 | Ensure there are restrictions on who can dismiss code change reviews (Manual) | Only trusted users should be allowed to dismiss code change reviews. | 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. |
| 1.1.6 | Ensure code owners are set for extra sensitive code or configuration (Manual) | 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. | 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: o Configure All eligible approvers approval rule. o Require Code Owner approval on a protected branch. 3. Commit your changes, and push them up to GitLab. |
| 1.1.7 | Ensure code owner's review is required when a change affects owned code (Manual) | Ensure trusted code owners are required to review and approve any code change proposal made to their respective owned areas in the code base. | 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: o 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. o 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. |
| 1.1.8 | Ensure inactive branches are periodically reviewed and removed (Manual) | Keep track of code branches that are inactive for a lengthy period of time and periodically remove them. | 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'. |
| 1.1.9 | Ensure all checks have passed before merging new code (Manual) | Before a code change request can be merged to the code base, all predefined checks must successfully pass. | 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. |
| 1.1.12 | Ensure verification of signed commits for new changes before merging (Manual) | Ensure every commit in a pull request is signed and verified before merging. | Ensure only signed commits can be merged for every branch via branch protection rules by performing the following steps for each project: In the sidebar, select Settings > Repository. • Navigate to the main page of the project. • • 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'. |
| 1.1.13 | Ensure linear history is required (Manual) | 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. | For every project, perform the following steps: • Navigate to your project • • Under 'Merge method', select 'Fast-forward merge'. In the sidebar, select Settings > Repository |
| 1.1.14 | Ensure branch protection rules are enforced for administrators (Manual) | Ensure administrators are subject to branch protection rules. | 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. |
| 1.1.16 | Ensure force push code to branches is denied (Manual) | 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. | 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. |
| 1.1.17 | Ensure branch deletions are denied (Manual) | Ensure that users with only push access are incapable of deleting a protected branch. | 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. |
| 1.1.18 | Ensure any merging of code is automatically scanned for risks (Manual) | Ensure that every pull request is required to be scanned for risks. | 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. |
| 1.1.19 | Ensure any changes to branch protection rules are audited (Manual) | Ensure that changes in the branch protection rules are audited. | 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. |
| 1.1.20 | Ensure branch protection is enforced on the default branch (Manual) | Enforce branch protection on the default and main branch. | 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. |

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 1.2.1 | Ensure all public repositories contain a SECURITY.md file (Manual) | 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. | 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. |
| 1.2.2 | Ensure repository creation is limited to specific members (Manual) | Limit the ability to create repositories to trusted users and teams. | 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 signups' are selected, and under 'Email confirmation settings' select 'Hard' • Select 'Save changes' |
| 1.2.3 | Ensure repository deletion is limited to specific users (Manual) | Ensure only a limited number of trusted users can delete repositories. | 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. |
| 1.2.4 | Ensure issue deletion is limited to specific users (Manual) | Ensure only trusted and responsible users can delete issues. | 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. |
| 1.2.5 | Ensure all copies (forks) of code are tracked and accounted for (Manual) | Track every fork of code and ensure it is accounted for. | 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. |
| 1.2.6 | Ensure all code projects are tracked for changes in visibility status (Manual) | Ensure every change in visibility of projects is tracked. | Ensure that every change in project visibility is investigated by performing the following regularly. As an administrator: • Navigate to the Admin Area. • • Review the log for Actions with the content 'Changed visibility from Private to In the sidebar, select Monitoring > Audit Events. 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/#instancestreaming-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. |
| 1.2.7 | Ensure inactive repositories are reviewed and archived periodically (Manual) | Track inactive repositories and remove them periodically. | Perform the following to remediate the presence of inactive projects. For each inactive project identified during the audit: In the sidebar, select Settings > General. In the 'Advanced' section, select 'Expand'. • Navigate to the project homepage. • • • 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. • • Configure the settings. In the Inactive project deletion section, select Delete inactive projects. o The warning email is sent to users who have the Owner and Maintainer role for the inactive project. o The email duration must be less than the Delete project after duration. • Select Save changes. |

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 1.3.1 | Ensure inactive users are reviewed and removed periodically (Manual) | Track inactive user accounts and periodically remove them. | 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. |
| 1.3.2 | Ensure top-level group creation is limited to specific members (Manual) | Limit ability to create teams to trusted and specific users. | 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. |
| 1.3.3 | Ensure minimum number of administrators are set for the organization (Manual) | Ensure the organization has a minimum number of administrators. | 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. |
| 1.3.7 | Ensure two administrators are set for each repository (Manual) | Ensure every repository has two users with administrative permissions. | 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. |
| 1.3.8 | Ensure strict base permissions are set for repositories (Manual) | 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. | 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. |
| 1.3.9 | Ensure an organization’s identity is confirmed with a “Verified” badge (Manual) | Confirm the domains an organization owns with a "Verified" badge. | 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: o If you do not have or do not want to use an SSL certificate, leave Automatic certificate management using Let’s Encrypt selected. o 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 (). |
| 1.3.11 | Ensure an organization provides SSH certificates (Manual) | As an organization, become an SSH Certificate Authority and provide SSH keys for accessing repositories. | 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. |
| 1.3.12 | Ensure Git access is limited based on IP addresses (Manual) | Limit Git access based on IP addresses by having a allowlist of IP addresses from which connection is possible. | 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: o Has no limit on the number of IP address ranges. o Has a size limit of 1 GB. o Applies to both SSH or HTTP authorized IP address ranges. You cannot split this list by type of authorization. 5. Select Save changes. |
| 1.3.13 | Ensure anomalous code behavior is tracked (Manual) | Track code anomalies. | For every project in use, track and investigate anomalous code behavior and activity. |

## 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 thirdparty 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 1.4.1 | Ensure administrator approval is required for every installed application (Manual) | Ensure an administrator approval is required when installing applications. | 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 |
| 1.4.2 | Ensure stale applications are reviewed and inactive ones are removed (Manual) | Ensure stale (inactive) applications are reviewed and removed if no longer in use. | 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 |
| 1.4.4 | Ensure only secured webhooks are used (Manual) | Use only secured webhooks in the source code management platform. | 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. |

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 1.5.1 | Ensure scanners are in place to identify and prevent sensitive data in code (Manual) | Detect and prevent sensitive data in code, such as confidential ID numbers, passwords, etc. | 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. |
| 1.5.4 | Ensure scanners are in place for code vulnerabilities (Manual) | Detect and prevent known open source vulnerabilities in the code. | 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. |
| 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) |  |  |
| 1.5.8 | Ensure scanners are in place for API runtime security weaknesses (Manual) | 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. | 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. |

## 1.8 Utilize Client Certificates to Authenticate Hardware

Assets Use client certificates to authenticate hardware assets connecting to the organization's trusted network.  
● ●  
● ●  
●

# 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 Establish and Maintain a Software Inventory

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 biannually, or more frequently.  
● ● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 2.1.1 | Ensure each pipeline has a single responsibility (Manual) | Ensure each pipeline has a single responsibility in the build process. | 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. |
| 2.1.3 | Ensure the build environment is logged (Manual) | 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. | 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. |
| 2.1.4 | Ensure the creation of the build environment is automated (Manual) | Automate the creation of the build environment. | 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. |
| 2.1.5 | Ensure access to build environments is limited (Manual) | Restrict access to the build environment (orchestrator, pipeline executor, their environment, etc.) to trusted and qualified users only. | 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 |
| 2.1.6 | Ensure users must authenticate to access the build environment (Manual) | Require users to login in to access the build environment - where the orchestrator, the pipeline executer, where the build workers are running, etc. | 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 |
| 2.1.7 | Ensure build secrets are limited to the minimal necessary scope (Manual) | 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. | 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. |
| 2.1.8 | Ensure the build infrastructure is automatically scanned for vulnerabilities (Manual) | Scan the build infrastructure and its dependencies for vulnerabilities. It is recommended that this be done automatically. | 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. |
| 2.1.9 | Ensure default passwords are not used (Manual) | Do not use default passwords of build tools and components. | 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. |
| 2.1.10 | Ensure webhooks of the build environment are secured (Manual) | Use secured webhooks of the build environment. | 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. |
| 2.1.11 | Ensure minimum number of administrators are set for the build environment (Manual) |  |  |

## 2.2 Ensure Authorized Software is Currently Supported

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.  
● ● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 2.2.1 | Ensure build workers are single-used (Manual) | Use a clean instance of build worker for every pipeline run. | 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. |
| 2.2.2 | Ensure build worker environments and commands are passed and not pulled (Manual) |  |  |
| 2.2.3 | Ensure the duties of each build worker are segregated (Manual) | 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. | 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: o 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. o 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: o 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. o For the executor, enter the type of executor. The executor is the environment where the runner executes the job. |
| 2.2.4 | Ensure build workers have minimal network connectivity (Manual) | Ensure that build workers have minimal network connectivity. | 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 |
| 2.2.5 | Ensure run-time security is enforced for build workers (Manual) | 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. | Deploy and enforce a run-time security solution on build workers. |
| 2.2.6 | Ensure build workers are automatically scanned for vulnerabilities (Manual) | Scan build workers for vulnerabilities. It is recommended that this be done automatically. | 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. |
| 2.2.8 | Ensure resource consumption of build workers is monitored (Manual) | Monitor the resource consumption of build workers and set alerts for high consumption that can lead to resource exhaustion. | 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 |

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 2.3.1 | Ensure all build steps are defined as code (Manual) | Use pipeline as code for build pipelines and their defined steps. | 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. |
| 2.3.2 | Ensure steps have clearly defined build stage input and output (Manual) | Define clear expected input and output for each build stage. | 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. |
| 2.3.3 | Ensure output is written to a separate, secured storage repository (Manual) | Write pipeline output artifacts to a secured storage repository. | 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 |
| 2.3.4 | Ensure changes to pipeline files are tracked and reviewed (Manual) | Track and review changes to pipeline files. | 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. |
| 2.3.5 | Ensure access to build process triggering is minimized (Manual) | Restrict access to pipeline triggers. | 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: o 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. o You can only select groups that are already invited to the project. o 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: o 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. o You can only select groups that are already invited to the project. o Users must have at least the Developer role to appear in the Approvers list. • In the Approval rules section: o Ensure that this number is less than or equal to the number of members in the rule. o See Deployment Approvals for more information about this feature. • Select Protect. |
| 2.3.6 | Ensure pipelines are automatically scanned for misconfigurations (Manual) | Scan the pipeline for misconfigurations. It is recommended that this be performed automatically. | For each pipeline, set automated misconfiguration scanning. |
| 2.3.7 | Ensure pipelines are automatically scanned for vulnerabilities (Manual) | Scan pipelines for vulnerabilities. It is recommended that this be implemented automatically. | For each pipeline, set automated vulnerability scanning. |

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 2.4.1 | Ensure all artifacts on all releases are signed (Manual) | Sign all artifacts in all releases with user or organization keys. | For every artifact in every release, verify that all are properly signed. |
| 2.4.2 | Ensure all external dependencies used in the build process are locked (Manual) | 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. | 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: o Gemfiles o package.jsons o go.sum • Ensure the version of the external dependencies corresponds to your internal policies |
| 2.4.3 | Ensure dependencies are validated before being used (Manual) | Validate every dependency of the pipeline before use. | 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. |
| 2.4.4 | Ensure the build pipeline creates reproducible artifacts (Manual) | 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. | 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. |
| 2.4.5 | Ensure pipeline steps produce a Software Bill of Materials (SBOM) (Manual) | 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. | 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. |
| 2.4.6 | Ensure pipeline steps sign the Software Bill of Materials (SBOM) produced (Manual) | 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. | For each pipeline, configure it to sign its produced Software Bill of Materials on every run. |

## 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.

## 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 biannually, or more frequently.

## 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.

## 2.8 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.  
● ●  
● ●  
●  
●

## 2.9 Implement Application Whitelisting of Scripts

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 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 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.  
● ●  
● ●  
● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 3.1.1 | Ensure third-party artifacts and open-source libraries are verified (Manual) | Ensure third-party artifacts and open-source libraries in use are trusted and verified. | Set third-party artifacts and open-source libraries to be verified. |
| 3.1.3 | Ensure signed metadata of the build process is required and verified (Manual) | Require and verify signed metadata of the build process for all dependencies in use. | For each artifact in use, require and verify signed metadata of the build process. |
| 3.1.4 | Ensure dependencies are monitored between open-source components (Manual) | Monitor, or ask software suppliers to monitor, dependencies between open-source components in use. | 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 |
| 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) | 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. | 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. |
| 3.1.7 | Ensure dependencies are pinned to a specific, verified version (Manual) | Pin dependencies to a specific version. Avoid using the "latest" tag or broad version. | For every dependency in use, pin to a specific version. |
| 3.1.8 | Ensure all packages used are more than 60 days old (Manual) | Use packages that are more than 60 days old. | If a package used is less than 60 days old, stop using it and find another solution. |

## 3.2 Perform Authenticated Vulnerability Scanning

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.  
● ●  
● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 3.2.1 | Ensure an organization-wide dependency usage policy is enforced (Manual) | Enforce a policy for dependency usage across the organization. For example, disallow the use of packages less than 60 days old. | Enforce policies for dependency usage across the organization. |
| 3.2.2 | Ensure packages are automatically scanned for known vulnerabilities (Manual) | Automatically scan every package for vulnerabilities. | 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. |
| 3.2.3 | Ensure packages are automatically scanned for license implications (Manual) | 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. | 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. |
| 3.2.4 | Ensure packages are automatically scanned for ownership change (Manual) | Scan every package automatically for ownership change. | Set automatic scanning of packages for ownership change. |

## 3.3 Configure Data Access Control Lists

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.  
● ● ●

## 3.10 Encrypt Sensitive Data in Transit

Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).

## 3.11 Encrypt Sensitive Data at Rest

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.

## 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.  
● ●

## 3.14 Log Sensitive Data Access

Log sensitive data access, including modification and disposal.

# 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 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.  
● ● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 4.1.1 | Ensure all artifacts are signed by the build pipeline itself (Manual) | Configure the build pipeline to sign every artifact it produces and verify that each artifact has the appropriate signature. | Sign every artifact produced with the build pipeline that created it. Configure the build pipeline to sign each artifact. |
| 4.1.2 | Ensure artifacts are encrypted before distribution (Manual) | Encrypt artifacts before they are distributed and ensure only trusted platforms have decryption capabilities. | Encrypt every artifact before distribution. |
| 4.1.3 | Ensure only authorized platforms have decryption capabilities of artifacts (Manual) | Grant decryption capabilities of artifacts only to trusted and authorized platforms. | Grant decryption capabilities of the organization's artifacts only for trusted and authorized platforms. |

## 4.2 Change Default Passwords

Before deploying any new asset, change all default passwords to have values consistent with administrative level accounts.  
● ● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 4.2.1 | Ensure the authority to certify artifacts is limited (Manual) | 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. | Limit which artifact can be certified by which authority. |
| 4.2.2 | Ensure number of permitted users who may upload new artifacts is minimized (Manual) |  |  |
| 4.2.4 | Ensure user management of the package registry is not local (Manual) | Manage users and their access to the package registry with an external authentication server and not with the package registry itself. | For each package registry, use the main authentication server of the organization for user management and do not manage locally. |
| 4.2.5 | Ensure anonymous access to artifacts is revoked (Manual) | For GitLab projects anonymous access is not available. Verify that all that require access controls are Private or Internal. | 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. |
| 4.2.6 | Ensure minimum number of administrators are set for the package registry (Manual) | Ensure the package registry has a minimum number of administrators. | 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. |

## 4.3 Ensure the Use of Dedicated Administrative Accounts

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.  
● ● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 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) | Validate the signature of all versions of an existing artifact. | For each artifact, sign and validate each version before uploading or using the artifact. |
| 4.3.3 | Ensure changes in package registry configuration are audited (Manual) | Audit changes of the package registry configuration. | Audit the changes to the package registry configuration. |
| 4.3.4 | Ensure webhooks of the repository are secured (Manual) | Use secured webhooks to reduce the possibility of malicious payloads. | For each webhook in use, change it to secured (over HTTPS). |

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 4.4.1 | Ensure artifacts contain information about their origin (Manual) | 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. | For each artifact supplied, supply information about its origin. For each artifact in use, ask for information about its origin. |

## 4.5 Use Multifactor Authentication For All Administrative

Access Use multi-factor authentication and encrypted channels for all administrative account access.  
● ●  
● ●

## 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.

## 4.8 Uninstall or Disable Unnecessary Services on

Enterprise Assets and Software Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.  
● ●

# 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 Establish Secure Configurations

Maintain documented, standard security configuration standards for all authorized operating systems and software.  
● ● ●  
v8  
v7

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 5.1.1 | Ensure deployment configuration files are separated from source code (Manual) | Deployment configurations are often stored in a version control system. Separate deployment configuration files from source code repositories. | Store each deployment configuration file in a dedicated repository separately from source code. |
| 5.1.2 | Ensure changes in deployment configuration are audited (Manual) | Audit and track changes made in deployment configuration. | For each deployment configuration, track and audit changes made to it. |
| 5.1.4 | Limit access to deployment configurations (Manual) | Restrict access to the deployment configuration to trusted and qualified users only. | Restrict access to the deployment configuration to trusted and qualified users. |
| 5.1.5 | Scan Infrastructure as Code (IaC) (Manual) | Detect and prevent misconfigurations or insecure instructions in Infrastructure as Code (IaC) files, such as Terraform files. | For every Infrastructure as Code (IaC) instructions file, set scanners to identify and prevent misconfigurations and insecure instructions. |
| 5.1.6 | Ensure deployment configuration manifests are verified (Manual) | Verify the deployment configuration manifests. | Verify each deployment configuration manifest in use. |

### 5.1.7 Ensure deployment configuration manifests are pinned to a specific, verified version

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

## 5.2 Use Unique Passwords

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.  
● ● ●

|  |  |  |  |
| --- | --- | --- | --- |
| Control Reference ID | Control Name & Description | Description | Remediation |
| 5.2.1 | Ensure deployments are automated (Manual) | Automate deployments of production environment and application. | Automate each deployment process of the production environment and application. |
| 5.2.2 | Ensure the deployment environment is reproducible (Manual) | 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. | Adjust the process that deploys the deployment/production environment to build the same environment each time when the configuration has not changed. |
| 5.2.3 | Ensure access to production environment is limited (Manual) | Restrict access to the production environment to a few trusted and qualified users only. | Restrict access to the production environment to trusted and qualified users. |
| 5.2.4 | Ensure default passwords are not used (Manual) | Do not use default passwords of deployment tools and components. | 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. |

## 5.3 Disable Dormant Accounts

Delete or disable any dormant accounts after a period of 45 days of inactivity, where supported.  
● ● ●

## 5.4 Restrict Administrator Privileges to Dedicated

Administrator Accounts 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.  
● ● ●

## 6.1 Establish an Access Granting Process

Establish and follow a process, preferably automated, for granting access to enterprise assets upon new hire, rights grant, or role change of a user.  
● ● ●

## 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.  
● ●  
● ●

## 6.4 Require MFA for Remote Network Access

Require MFA for remote network access.

## 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.

## 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.  
●

## 7.5 Perform Automated Vulnerability Scans of Internal

Enterprise Assets 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.

## 7.6 Perform Automated Vulnerability Scans of ExternallyExposed Enterprise Assets

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.

## 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.

## 12.5 Centralize Network Authentication, Authorization,

and Auditing (AAA) Centralize network AAA.

## 12.6 Use of Secure Network Management and

Communication Protocols Use secure network management and communication protocols (e.g., 802.1X, Wi-Fi Protected Access 2 (WPA2) Enterprise or greater).  
● ●

## 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.

## 12.12 Manage All Devices Remotely Logging into Internal

Network 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.  
● ●  
●

## 13.2 Remove Sensitive Data or Systems Not Regularly

Accessed by Organization 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.  
● ● ●

## 13.5 Manage Access Control for Remote Assets

Manage access control for assets remotely connecting to enterprise resources. Determine amount of access to enterprise resources based on: up-to-date antimalware software installed, configuration compliance with the enterprise’s secure configuration process, and ensuring the operating system and applications are upto-date.

## 14.4 Encrypt All Sensitive Information in Transit

Encrypt all sensitive information in transit.  
● ●  
● ●  
● ●

## 14.5 Utilize an Active Discovery Tool to Identify Sensitive

Data 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.  
●  
●

## 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.  
● ● ●

## 14.7 Enforce Access Control to Data through Automated

Tools 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.  
●

## 14.8 Encrypt Sensitive Information at Rest

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.  
v8  
v7  
● ●  
●

## 14.9 Enforce Detail Logging for Access or Changes to

Sensitive Data 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).  
●  
● ●  
●

## 16.1 Establish and Maintain a Secure Application

Development Process 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.  
● ●

## 16.2 Establish and Maintain a Process to Accept and

Address Software Vulnerabilities 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.  
● ●

## 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.  
● ●  
● ● ●  
● ● ●  
● ●

## 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.

## 16.7 Use Standard Hardening Configuration Templates for

Application Infrastructure 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.

## 16.8 Separate Production and Non-Production Systems

Maintain separate environments for production and non-production systems.

## 16.9 Disable Dormant Accounts

Automatically disable dormant accounts after a set period of inactivity.  
● ● ●

## 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.  
● ● ●  
● ●

## 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.

## 18.1 Establish Secure Coding Practices

Establish secure coding practices appropriate to the programming language and development environment being used.  
● ●  
v8  
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.  
● ●  
● ●  
● ●  
● ●  
● ●

## 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.  
●  
● ●

## 18.8 Establish a Process to Accept and Address Reports of

Software Vulnerabilities Establish a process to accept and address reports of software vulnerabilities, including providing a means for external entities to contact your security group.  
● ●  
v8  
v7

## 18.9 Separate Production and Non-Production Systems

Maintain separate environments for production and nonproduction systems. Developers should not have unmonitored access to production environments.  
● ●  
● ●

## 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.  
● ●  
● ●